Computer Aids for Mathematical Model Building

by

Jae Hyun Cho

February 1997

A thesis submitted for the degree of
Doctor of Philosophy
of the
University of London
and for the
Diploma of Membership of Imperial College

Department of Chemical Engineering and Chemical Technology,
Imperial College of Science, Technology and Medicine,
Prince Consort Road,
London SW7 2BY,
United Kingdom.

Abstract

Mathematical models of processes are now widely used at all levels, from process synthesis and design, through operations planning, to process control and monitoring. With the increasing sophistication of applications and the resulting models, it has become clear that the formulation of appropriate consistent models is a major, time–consuming and error–prone task. A number of modelling and simulation packages have been developed in recent years to aid process modelling. In these systems the basic building blocks are the equations representing the physico–chemical relations, however there has been little work on aids for generating such equations.

This thesis is concerned with the development of a system for automatically generating lumped–parameter models from a purely physical description of process systems, the majority of which routinely experience discontinuous physical behaviour such as phase transitions (e.g. presence or absence of phases), flow regime transitions (between laminar and turbulent) and discrete changes resulting from the geometry of individual process units.

The approach described in this work is based on the conceptual view of process systems as a set of inter–*connected vessels* containing interacting *phases*. A physical modelling language supporting the description of process systems in a purely physical manner has been designed, with a special emphasis on the hierarchical description of the inter–connections between phases or vessels. A methodology for formulating mathematical models from the physical description represented in the language has been developed, focusing on the mathematical description of discontinuous physico–chemical behaviour as mentioned above. The current version of this package generates mathematical models in the format of the *gPROMS* (Barton, 1992) input language. The implementation of this methodology (written in C) has led to a prototype of a new model generation package. The ability of this package to automatically generate mathematical models is demonstrated by several case studies.

Acknowledgements

I would like to express my sincere thanks to my supervisors, Professor Roger Sargent and Professor John Perkins. Their intellectual supervision, guidance and encouragement during the whole period of my Ph.D research have been invaluable. I would also like to thank Professor Costas Pantelides for his contributions to the design of the physical modelling language during the early stages of the project and Dr. Steve Walsh for his sincere and comforting advice.

I would especially like to thank Johnathan Heath, David Rowe, Erick Groep, Ben Keeping, Vince White and Nefyn Jones for their friendship which gave me a great deal in my life. I am greatly indebted to Edward Smith for his crucial help with the computer implementation. Special thanks go to Jamie Barber for our useful discussions and his sincere help in writing up the thesis. I must also thank my other friends and colleages in the Centre for Process Systems Engineering for making my Ph.D period so enjoyable.

There are not enough words to thank my wife Young Mee Lee and my son Gi Chul Cho as well as my parents, brothers and sisters for their continuous love and support without which it would have been impossible to finish my Ph.D succesfully.

Contents

Abstract **2**

Acknowledgements **3**

List of Figures **7**

List of Tables **9**

1 Introduction **10**
1.1 Motivation and Objective 11
1.2 Important Aspects of Process Modelling 13
1.2.1 Multi–faceted Character of Models 13
1.2.2 Model Structuring Concept 14
1.2.3 Process Discontinuity 17
1.3 Recent Computer–Aided Model Generation Techniques 18
1.4 Thesis Outline 22

2 Process Representation **23**
2.1 Summary 27

3 Physical Modelling Language **29**
3.1 Vessel Entity 30
3.1.1 Phase attributes 31
3.1.2 Aggregation attributes 34
3.1.3 Transfer Law attributes 35
3.1.4 Port attributes 36
3.1.5 Geometry attributes 41
3.2 Reservoir Entity 44
3.2.1 Compound attributes 45
3.2.2 Reservoir Port attributes 46
3.3 Connection Entity 47
3.4 Physical Modelling Examples 48
3.4.1 Flash drum 49
3.4.2 Two Flash Drums with reversible flow 50
3.4.3 Decanter 53
3.5 Summary 55

4 Mathematical Model Formulation **57**
4.1 Basic Model Building Strategy 58
4.2 Generic Model Formalism 59
4.3 Model Generation Algorithms 62
4.3.1 Vessel Sub–Model Generation 63
4.3.2 Reservoir Sub–Model Generation 79
4.3.3 Connection Sub–Model Generation 80
4.4 Notation used in Mathematical Models 82
4.5 Summary 84

5 Implementation **85**
5.1 Software Architecture 85
5.2 The Translator 86
5.2.1 The Scanner 86
5.2.2 The Parser 88
5.2.3 The Semantic Routines 88
5.3 The Model Generation Engine 89
5.3.1 The Mathematical Symbol Table 90
5.4 The Transfer Law Library 91
5.5 The Code Generator 91
5.6 Summary 93

6 Simulation Examples **95**
6.1 Flash Drum 95
6.2 Two Flash Drums with reversible flow 99
6.3 Decanter : Settling tank 102

7 Conclusions and Future Work **107**
7.1 Conclusions 107
7.2 Future Work 109
7.2.1 Suggestions on "Phase" and "Aggregation" statement 109
7.2.2 Broadening the scope of the physical description 109
7.2.3 System environment 110
7.2.4 Refinement of transfer law library 110
7.2.5 Generation of distributed parameter models 110

References **112**

A Lex Input Specification **118**

B Yacc Input Specification **121**

C Transfer Law Library **130**
C.1 Phase Equilibrium 130
C.2 Bubble Rise 132
C.3 Containing Phase Transfer 132
C.4 Irreversible Laminar Flow 133
C.5 Irreversible Turbulent Flow 134
C.6 Irreversible Pressure Driven Flow 134

C.7 Pressure Driven Flow 136
C.8 Weir Over Flow 137
C.9 Static Pressure Driven Flow 138

D Simulation Input Files **139**
D.1 Flash Drum 139
D.2 Two Flash Drums with reversible flow 155
D.3 Decanter : Settling Tank 185

List of Figures

2.1 Conceptual diagram of simple process 25
2.2 Multiple possibilities in flash drum 26

3.1 Example the declaration of VESSEL entity 31
3.2 Vessel containing two immiscible liquids 32
3.3 Example **Phase** section 34
3.4 Example **Aggregation** section 35
3.5 Example **Transfer Law** section 36
3.6 Example **Port** section 38
3.7 The Association of inlet aggregate stream by the aggregation type 39
3.8 The Association of inlet aggregate stream by the phase type 39
3.9 The Association of inlet single phase by the phase level 40
3.10 The Association of inlet single phase by the phase type 40
3.11 Example **Shape** section 42
3.12 Example **Orientation** section 42
3.13 Example **Dimension** section 43
3.14 Example **Port Position** section 44
3.15 Example the declaration of reservoir entity 45
3.16 Example Compound section 46
3.17 Example reservoir entity 46
3.18 Example connection entity 48
3.19 Conceptual diagram of Flash drum 49
3.20 Physical Modelling Example Flash Drum 50
3.21 Conceptual diagram of Flash drum 51
3.22 Physical Modelling Example Two Flash Drums with reversible flow 52
3.23 Schematic Diagram of Decanter 53
3.24 Physical Modelling Example Decanter 54
3.25 Hierarchical structure of physical modelling elements 55

5.1 The Software Architecture of the Prototype Package 87
5.2 The Internal Hierarchical Structure of the Mathematical Symbol Table . . 92

6.1 Holdup variation in flash drum 96
6.2 Level variation in flash drum 97
6.3 Driving force variation accross of connection C2 between flash drum and the reservoir 98
6.4 Variation of total mass rate flowing through port Po1 in the flash drum . 98

6.5 Level variation in two flash drums 100
6.6 Holdup variation in flash drum1 101
6.7 Holdup variation in flash drum2 101
6.8 Flow reversibility between two flash drums 102
6.9 Level variation in decanter 104
6.10 Holdup variation in decanter 105
6.11 Flow over weir in decanter 106

List of Tables

4.1 Standard Notation Table 83
4.2 Non–Standard Notation Table 83

Chapter 1

Introduction

This thesis is concerned with the development of a software tool for automatically generating a mathematical model from a purely physical description for a process unit or system.

Models have been recognised as indispensable for describing and predicting the behaviour of the real world. Many modelling methodologies have been applied to the abstraction of the real world for a long time. Among them, mathematical models expressed in sets of differential, algebraic, integral equations etc., are most commonly used in the wide range of science and engineering field.

Mathematical modelling essentially represents the translation of the real world problems into mathematical problems. It is the art of devising a mathematical structure that takes into account the essence of a given situation. A real world problem, in all its generality, can seldom be translated into a mathematical domain, and even if it can be so translated, it may not be possible to solve the mathematical equations. This fact quite often necessitates the idealisation or approximation of an original problem to get it translated and to obtain its solution.

In the area of process systems engineering mathematical models are essential in tackling a variety of problems such as process synthesis and design, process simulation and optimisation, planning and scheduling of process operation, and process control. They provide the formal representation for describing the relevant physico–chemical dynamic behaviour which is composed of the relationships between mathematical terms. Only well formulated mathematical models enable us to successfully perform process systems engineering tasks. If an incorrect mathematical model is applied, valid results of process engineering work can not be expected. An important aspect of the modelling problem is thus the identification of the significant features of the behaviour for the purpose in

hand, and the construction of a model which adequately reproduces those features without irrelevant complexities (Sargent, 1983).

Recently the rapid development of computer hardware and software has led to the emergence of more powerful computational modelling tools in a wide variety of fields. This trend could be seen in the field of process systems engineering in such a way that flowsheeting whose application range was limited to simple unit modules has been extended to be able to deal with dynamic modelling and simulation for complex process systems involving several hundreds of thousands of variables.

At the same time, as the complexities of process interactions increase due to the tighter specification of plant performance, the strengthening of the legislation on environment and safety, and flexible and just–in–time production systems to meet the rapid changes of market demands, higher–fidelity process models are required to adequately cope with them. Actually in the course of specific process engineering activities, the cost of developing the requisite models represents a significant part of the overall budget (Perkins and Barton, 1987).

Those general trends mentioned above have therefore been stimulating the development of a number of dynamic modelling packages, most of which are based on an equation–oriented flowsheeting architecture.

This chapter is composed of four sections. Section 1.1 describes the background to this research involving the motivation and the objective. In the following two sections the important aspects for process models and an overview on the current state of art of computer aids for automatic model generation are illustrated. Finally the thesis outline will be introduced.

1.1 Motivation and Objective

Considerable effort has been made in the development of modelling facilities which allow the user to concentrate on the correct mathematical formulation of the process model, as opposed to the numerical algorithms and coding required to solve it. From the earliest days flowsheeting packages provided a means of building whole process models from knowledge of the flowsheet and models for the individual units. SpeedUp (Sargent and Westerberg, 1964; Perkins and Sargent, 1982; Pantelides, 1988) provides a system which separates computation from the models enabling these to be provided essentially as a set of equations, together with a simple macro facility for building complex models from simpler ones. This has already been commercialised and being widely used in the pro-

cess industries. ASCEND (Piela, 1989; Piela *et al.*, 1991) uses object–oriented programming methodologies to provide a more general facility using combination and inheritance, while DESIGN-KIT (Stephanopoulos *et al.*, 1990a), and MODEL.LA (Stephanopoulos *et al.*, 1987; Stephanopoulos *et al.*, 1990b) uses object-oriented programming both to build models in this way, and to tailor them for different uses. Omola integrated with OmSim (Andersson, 1990; Mattsson and Andersson, 1992; Nilsson, 1993; Andersson, 1994) and gPROMS (Barton, 1992; Barton and Pantelides, 1994; Oh, 1995; Oh and Pantelides, 1996) provide similar general facilities to model both the process and the operations. It is clear that current powerful modelling environments help modellers to alleviate the time–consuming and error–prone task of formulating appropriate consistent mathematical models for complex process systems.

Even so, in the majority of all these packages, the correct formulation of the mathematical model is the responsibility of the user. That is, users are required to mathematically formulate process models using the high–level modelling language which has its own syntax and semantics, consequently the correct mathematical modelling is the responsibility of the user. The formulation of correct, complete and non–redundant mathematical models is not always an easy task. The difficulties result from a number of situations such as the under–specification of the underlying system or vice versa, the formulation of inconsistent equations, and the inconsistent formulation of boundary and initial conditions. At this point, there is a crucial requirement for the development of more advanced modelling tool to provide the extensive support for model construction activities. In all the modelling packages mentioned above, the basic building blocks are the equations representing physico–chemical relations, and there has been little work on aids for generating such equations. The challenge in this area is to provide a modelling facility which allows the user to describe his system purely in terms of the elementary physico–chemical processes involved, with the appropriate mathematical model generated automatically by the package (Sargent, 1990).

There has been little work on exploring the possibility of computer–aided mathematical model generation for a process system, for example, by defining it in a more purely physical and elementary behaviour–oriented fashion. Meyssami and Åsbjørnsen describe a prototype expert system for modelling from first principles (Meyssami and Åsbjørnsen, 1989) while Preisig *et. al* have developed a computer–aided model generation tool for physical–chemical–biological systems using object–oriented model representation (Preisig *et al.*, 1990; Preisig, 1995; Preisig, 1996) close to our approach in terms of the basic concept for process representation.

With the above in mind, it is thought to be of considerable importance to develop such an automatic model generation environment.

The major objective of this research, which is a continuation of earlier work (Vázquez-Román, 1992) resulting in a prototype computer program (written in the object–oriented language SMALLTALK), as a vehicle for ideas, is to develop a computer software package which generates a mathematical model from a purely physico–chemical description of process systems, focusing on possible situations likely to arise from discontinuous behaviour.

1.2 Important Aspects of Process Modelling

This section describes several important aspects of process models and their systematic formulation. This section has been organised as follows : §1.2.1 briefly describes the multi–faceted character of models with several applications of this nature, §1.2.2 explains basic concepts required to represent process models which are essential to systematic model–building and in §1.2.3 process discontinuities routinely arising in process systems will be discussed.

1.2.1 Multi–faceted Character of Models

As stated earlier, the modelling activity represents the abstraction of a real world problem by encapsulating the knowledge considered essential to the specific modelling purpose. Models do not exist in isolation and though they may at times be considered in their own terms, models are never fully understood except in relation to other members of the model family to which they belong (Aris, 1978). Simple models are required early in a process design but more complex ones are needed as the design process proceeds. We thus often need very different kinds of models at different modelling stage, and an important part of the modelling process is the tailoring of the model for the particular purpose in hand (Sargent, 1990). There are different models in terms of their levels of detail for the same process system depending on the modelling purpose at each modelling stages, resulting in an hierarchy structured into the model family. This is termed *multi–facet* of process models and the facility for supporting it is provided in the majority of recently developed modelling languages.

SpeedUp (Sargent and Westerberg, 1964; Perkins and Sargent, 1982; Pantelides. 1988) provides a fixed number of modelling levels comprising hierarchy for supporting top–down or bottom–up modelling approach. Alternatively, Omola (Mattsson and Andersson,

1992; Nilsson, 1993), ASCEND (Piela, 1989) and gPROMS (Barton and Pantelides, 1994) have adopted the concept of *hierarchical sub–model decomposition* (Elmqvist, 1978) which enables to describe the formal definition of models in a recursive manner, thus permitting an effectively unlimited number of hierarchical levels (Pantelides and Barton, 1992). In MODEL.LA (Stephanopoulos *et al.*, 1990b) multi–faceted modelling has been incorporated as a key feature, and a provision for supporting multi–level modelling of the same process, to automate a hierarchical sub–model description.

1.2.2 Model Structuring Concept

The objective of all modelling languages is to eliminate the modelling bottleneck of engineering applications by providing a computer–aided environment which can support: (a) expeditious construction of models by the human user, and/or (b) automatic generation of models by another program (Stephanopoulos and Han, 1996). Recently developed modelling languages, the majority of which have adopted the object–oriented concept, in common provide the user with a high level declarative representation coupled with highly structured formalism in order to support the construction of consistent mathematical models for complex process systems. To achieve these objectives the majority of the modelling packages use a software architecture whereby the model description is completely decoupled from the mathematical solution method. This feature has resulted in a shift away from purely dynamic simulation packages towards more general–purpose modelling environments, in which a common process model is used in a number of different modes such as steady–state simulation and design, dynamic simulation, steady–state and dynamic optimisation, data reconciliation *etc.* (Pantelides and Barton, 1992). The basic conceptual point of view of process systems in order to support facilities for representing the process models in a more structural way will be briefly reviewed on recently emerged modelling languages in the following text.

The conceptual essence of Omola (Andersson, 1990; Mattsson and Andersson, 1992; Andersson, 1994) is based on object–oriented programming ideas. The key concept in Omola is the *class*, a general data aggregation which is the basis for representing different modelling concepts. Omola is structured into two separate layers: the *data representation layer* and the *model representation layer*. The former represents the description of the internal behaviour within a model. It defines a set of syntactic, semantic, and pragmatic rules for representing general data such as model variables and equations as well as the class. The model representation layer represents the interfaces of models for communicating with its environment. It consists of a set of classes previously defined

in the data representation layer and structural components such as *models*, *parameters*, *terminals* and *connections*. The connections represent topological linkages of involved models through the terminals which may transfer a set of information about the media. The parameters normally represent some kind of time–invariant design variables.

ASCEND (Piela, 1989; Piela *et al.*, 1991) is a domain–independent object–oriented computer environment for analysing and modelling complex process systems in terms of large sets of simultaneous nonlinear algebraic equations. The basic conceptual elements of the ASCEND language are *model*, *elementary* and *atom* types. The *models* are structured types built hierarchically from instances of other models, instances of atoms, and relationships between instances. The *elementary* types are primitive data types such as real, string, unit, etc. The *atoms* are primitive structured types for representing physical quantities. The atoms and models are organised into inheritance hierarchies comprising networks of connected parts which are themselves instances of models.

gPROMS (Barton, 1992; Barton and Pantelides, 1994; Oh, 1995; Oh and Pantelides, 1996) is a general–purpose software package designed for modelling and simulation of combined discrete and continuous processes, supporting combined lumped and distributed parameter mathematical models. The conceptual framework of the gPROMS language is based on three distinct categories of entities, *models*, *tasks* and *processes*. Model entities encapsulate the description of the physico–chemical behaviour, while task entities encapsulate the description of the external control actions or disturbances imposed on the system. A process entity is formed by the application of tasks to instances of model entities in order to define a complete simulation of the process system. gPROMS is discussed in more detail later since it is closely related to our work.

MODEL.LA (Stephanopoulos *et al.*, 1990a; Stephanopoulos *et al.*, 1990b) is a high level and completely declarative language especially constructed for the interactive and automatic definition of models of process systems, Based on the following fundamental requirement of modelling language:

> "A modeling language in process engineering should be fully declarative and in no way its generality should be compromised by the specificity of the methodologies of the process engineering tasks, themselves".

The language structure is based on six modelling elements: three for modelling the "structural characteristics" of any processing system and three for describing the "functional characteristics" as follows.

- *Generic Unit* (*GU*) : an isolated spatial region coupled with well defined system

boundary at any level of abstraction.

- *Port* : special purpose entities through which *GUs* transfer information among each other.

- *Stream* : the connections between *GUs* in association with their *ports*.

- *Modeling–Scope* : a consistent set of declarative relationships, which apply to all the components of a model including the variables and parameters of a single *GU* or a network consisting of *GUs*, *Ports* and *Streams*.

- *Constraint* : unsolved relations among quantities such as variables and terms, also containing the information on the scope of the relationship, its meaning and significance, and range of its applicability.

- *Generic Variable* : basic building block for constructing modelling relationships, encapsulating the information on physical significance, value, possible range of values, units, trends, etc.

Based on the six modelling elements given above and eleven semantic relationships obeying basic axioms of transitivity, monotonicity, commutativity and merging, process models can be interactively or automatically generated from the process representation. This representation is completely modularised using object–oriented formalism at various levels of abstraction and gives complete documentation of the modelling context (assumptions, simplifications, process engineering task) capturing qualitative, semi-quantitative and quantitative knowledge. The structure of process models is depicted by specific digraphs, which are symbolically constructed by algorithmic procedures driven by the context of the modelling activity.

VEDA (Marquardt *et al.*, 1993; Bogusch and Marquardt, 1995; Marquardt, 1996) is an application specific object–oriented data model especially designed for supporting the object–oriented representation of chemical process systems in a structured way. It is currently being implemented to be integrated with DIVA (Holl *et al.*, 1988; Kroner *et al.*, 1990). It is the basic concept of VEDA that chemical process systems are described in terms of two basic entities; *structure* and *behaviour*. The structures of chemical processes are envisaged as sets of *devices* linked through *connections* which transform a driving force determined by the known states of two adjacent devices into a flux. VEDA supports hierarchical sub–model decomposition in a recursive fashion as devices and connections entities are structured into hierarchical taxonomy. The elementary device *generalised phase* is the key concept in the structural description of chemical processes.

It represents any delimitable – but not necessarily homogeneous – non–decomposable material entity in a process. The structural description of a complex process system is complemented by its behavioural description which characterises the system in terms of *process quantities* and *model equations* consisting of *balance equations*, *constitutive equations* and *constraints*.

1.2.3 Process Discontinuity

Process systems cannot be considered to proceed in a completely continuous way. Most continuous processes undergo appreciable discrete changes overlaid on their mainly continuous behaviour. These discrete events result from, for example, the digital process control, plant equipment maintenance, plant shut–down and start–up. In addition to process operational discontinuities, other types of discontinuities may arise due to the intrinsic process mechanisms such as phase appearance/disappearance, reverse flow and the transition of flow regime between laminar and turbulent, etc. Of course, some process systems are designed to exhibit discontinuous behaviour, such as batch or semi–continuous processes.

In order to encompass such discontinuities, mathematical models of process systems should switch between different structures in terms of equations and variables whenever discrete events occur. Future simulation packages must support the analysis of arbitrarily operated processes within a unified framework (Marquardt, 1991).

Considerable efforts have been made in the development of specific–purpose simulation packages for batch and semi–continuous processes such as BOSS/BATCHES (Joglekar and Reklatis, 1984), UNIBATCH (Czulek, 1988) and DYNSIM (Gani *et al.*, 1992; Perregaard *et al.*, 1992).

However a few general–purpose modelling and simulation packages fully supporting the application of process models to combined discrete and continuous processes involving general discontinuities have emerged in academia in recent years.

Omola, for example, has been extended to provide facilities for modelling combined discrete and continuous process systems, namely *hybrid systems* (Andersson, 1992; Andersson, 1994). In order to deal with hybrid systems, a general mathematical and logical framework to deal with the mechanisms of transitions between discrete states, called OHM (Omola Hybrid Model) has been developed as an intermediate representation. The formalism consists of sets of variables, parameters, equations, event conditions, and event actions. The OHM representation can be automatically translated into more specialised representation.

A key focus of gPROMS (Barton, 1992; Barton and Pantelides, 1994) was to support process modelling of hybrid systems with general discontinuities. These are classified into two primitive categories; *physico–chemical discontinuities* and *external actions.* The former represents those inherently embedded in physics of the process including thermodynamic (e.g. appearance and absence of phase) and mechanical (e.g. flow regime change between laminar and turbulent) transitions and those resulting from the geometry of an individual process unit (e.g. the switch of the phase flowing through outlet pipe depending on its level and the location of the pipe). The description of intrinsic discontinuities is incorporated into the *model entity.* The external actions represent those imposed on a process by its environment such as disturbances or control actions, and are incorporated in the *task entity.* Consequently, instead of decomposing a process model into a continuous subsystem and a discrete subsystem, the description of a process model is decomposed into the underlying combined discrete/continuous physical behaviour of the plant and the external actions. This has the consequence that all the knowledge concerning the physical behaviour of a particular system is not only encapsulated in a single *model entity* but is also completely decoupled from the external actions that are applied during a particular operation. In order to provide a sufficiently general representation of the discontinuous behaviour, physico–chemical discontinuities are classified into three categories according to the mechanisms that result in transitions between the discrete states, namely *reversible, irreversible* and *asymmetric and reversible discontinuities.* A general formalism to represent these mechanisms is provided. gPROMS deserves to receive attention on supporting the modelling and simulation of arbitrarily operated processing systems within a unified framework.

1.3 Recent Computer–Aided Model Generation Techniques

As illustrated, it is clear that the majority of recent modelling packages provide advanced modelling facilities such as highly structured and declarative representation of models, hierarchical sub–model decomposition for supporting the recursively modularised representation, the reusability and inheritance of models, documentation for specifying assumptions and the scope of models, user–friendly interfaces with modelling environments and the combined lumped/distributed parameter modelling of combined discrete/continuous processes within a single framework, where the physico–chemical process behaviour is coupled with complex sequences of control actions.

It should be noted however that the current state–of–the–art of modelling pack-

ages is still in need to explore practically applicable computer aids for generating appropriate mathematical equations describing physical process behaviour. The key reason for this necessity is that the basic building blocks of most existing packages are a set of equations representing the physico-chemical conservation principles and closure equations. The mathematical models built by the user using these packages are often badly posed, thus causing structural, functional or numerical singularity during the simulation. We note that the fundamental research on the automatic generation of correct, complete and non–redundant mathematical models by a computer facility is of considerable importance in the area of the development of leading edge modelling packages. Little work has been carried out in this direction. Recent research progress in the field of computer–aided model generation are briefly discussed although not all are related directly.

Several research projects on automatic model generation from natural language have recently been undertaken in the field of system simulation. Austin and Khoshnevis (Austin and Khoshnevis, 1989) have developed an intelligent simulation environment for automatically generating models of production–distribution systems from a description written in natural language. Perhaps more interestingly, Beck and Fishwick (Beck and Fishwick, 1989) have explored an approach to merge simulation and natural language. This is achieved using a conceptual framework for representing mathematical equations, the syntactical and semantical structure of sentences. Sentences are translated and transformed into the knowledge representation language, CANDIDE. This step results in a collection of CANDIDE objects whose structure represents the meaning of sentences. Mathematical equations are then generated from the information encapsulated in the structural description by using a language generation facility for differential equations. These procedures are undertaken in a system environment called NATSIM which accepts natural language descriptions of a model, generates equations, accepts questions about the model and then solves the equations using simulation or analytical techniques to answer these questions.

PROFIT (Telnes, 1992) is a knowledge–based modelling tool for generating mathematical models from a menu–driven textual description of process systems based on first principles. The basic building blocks for constructing mathematical equations are a set of simple physical terms such as *volume*, *surfaces*, *phases*, *chemical reactions* and *transport phenomena and forces*. The important classes defined in the system are OMPR (Object Model for Process Representation) and OMMK (Object Model for Modeling Knowledge). The former is employed for describing process systems, the latter for constructing the structural knowledge base of the mathematical models. The procedure

of mapping the process description stored in OMPR into mathematical models structured in the form of OMMK is driven by an inference engine where each term comprising the process description is associated with a physical phenomenon by an appropriate rule. The internal representation of the mathematical models is converted into a readable format by a translator. The current version of the translator produces a LaTeX (Lamport, 1986) format.

The knowledge base in the current system consists of the following three parts:

- A dictionary containing features, global constants, approximating functions and other objects representing modelling knowledge in various forms.
- A set of equations used to construct models.
- Several sets of rules designed for different purposes such as phase interaction identification and selection of conservation equations.

In addition to the modelling aids above, PROFIT provides useful modelling facilities such as documentation of all relevant assumptions relating to the mathematical models under consideration and the modification of mathematical models according to changes in the process description. It should be noted that PROFIT supports a provision allowing the detailed description of the process equipment geometry.

A computer–aided system for automatically generating problem specific process models from the physical description for process systems has been developed, which is intended to be integrated within the Integrated Computer Aided System (ICAS) (Jensen and Gani, 1996). The model generation methodology is based on two basic concepts: *control shells* representing a region of space delimited by its boundaries and a *reference model* containing all the possible terms arising in a chemical process model. The former is defined in such a way that within the boundaries the partial gradients (with respect to temperature, pressure and fugacity) are negligible, or can be incorporated in an interphase or overall flux model. The physical description of the underlying process system is undertaken in the control shell involving a geometric description, balance and boundary specifications and equilibrium specifications. A specific mathematical model is generated by applying the model generation algorithm procedures which simplify a reference model from the information given in the physical description of the control shells. One feature of this approach is the ability to generate distributed parameter models, although the usefulness and practicality of using a distributed parameter control shell approach is open to question.

MODELLER (Preisig *et al.*, 1990; Preisig, 1995; Preisig, 1996) is a object–oriented computer–aided modelling tool for generating mathematical models from a physical description of physical–chemical–biological process systems. Process systems are viewed as being composed of a communicating network of simple thermodynamic systems. Based on this general point of view two principal conceptual elements are identified to support a physical description at any level of details in a hierarchically structured fashion, namely: *systems* and *connections*. The former represents any spatial capacities containing mass and is defined as consisting of a single phase or a pseudo–phase (an average of several phases such that it appears as a single phase). The latter represents communication paths between parts of the overall system and is employed for describing the transfer of extensive quantities through boundaries assuming a pseudo–steady state for the physical system associated with actual transfer. The transfer law is described as a function of the state variables of the two connected systems with its directionality. Connections also incorporate all effects associated with system surfaces (e.g. change of phase across the boundary). The topological structures of process systems are described in a graphical and textual manner and allows hierarchical decomposition of complex process systems into physical subsystems. In order to deal with the distribution of species in the physical topology of a given process, a set of species are specified including a set of species participating in chemical or biological reactions and a set of species representing permeability for allowing the passage through a connection. The final step in the physical description procedure is to incorporate mechanistic details such as transfer laws, kinetic laws for chemical or biological reactions, physical property relations, geometrical properties, etc. It is intended that all the information required to incorporate these mechanistic details will be supplied from a knowledge base, although this has not yet been implemented in the current version of MODELLER. Mathematical models are generated in a textual output file containing both the list for the hierarchical structure of physical systems and a set of equations involving the total mass balance, the species balances and the energy balances. The development of information processing units to deal with control systems has been recently undertaken. It could be noted that MODELLER fully supports physical description in an easy–to–use graphical interface with the hierarchical decomposition of the topology of physical systems in a recursive way.

1.4 Thesis Outline

This thesis is composed of 7 chapters. Chapter 1 describes the background to this research including the motivation and objectives, several significant aspects of process models and the current state–of–the–art in computer aids for generating mathematical models. Chapter 2 introduces the basic concepts for the representation of process systems. In chapter 3 the formal definition of the language for representing process systems in a purely physical fashion in terms of its syntactical structure and semantics is presented. Chapter 4 is mainly concerned with the development of the algorithms for formulating the mathematical model of a given process system from the physical description represented in the language defined in chapter 3. This includes the basic model building strategy, a generic mathematical formalism to deal with combined discrete/continuous behaviour and also gives the notation used in the mathematical models generated. Chapter 5 describes the software architecture and the details of the internal data structure of the current package. In chapter 6 The ability of the prototype of the package to automatically generate mathematical models is tested through several simulations to which the mathematical models generated by the package are applied. This thesis concludes in chapter 7 with some suggestions for future works.

Chapter 2

Process Representation

In the previous chapter we discussed the motivation and the ultimate goal of this work, which is the development of the package for automatically generating lumped–parameter models from a purely physical description of process systems, focusing on the incorporation of intrinsically physical discontinuous behaviour embedded in a given process system into the mathematical model. In order to achieve this objective first we need a conceptual framework for structuring process systems into basic elements. This chapter describes the basic concept for representing a process system in a purely physical fashion.

We start with the concept for process representation developed in earlier work (Vázquez-Román, 1992) followed by the argument with possible situations arising in a given process system, namely physical discontinuities, then finally construct the concept for process representation, which is general enough to deal with such discontinuities, identifying basic conceptual elements and their meanings in physical representation of process systems. It should be noted that the essence of the concept is still based on the previous one.

Recently the formal description of the concept for process representation has been made as follows (Perkins *et al.*, 1996):

> "A basic premise is that all material undergoing processing is instantaneously in a stable thermodynamic state, which implies that this state is describable in terms of a finite set of state variables, for instance, two independent variable properties plus the masses of each chemical species (Feinberg, 1979). It further implies that all material is present in well defined thermodynamic phases, and hence that any *process* can ultimately be defined as a collection of interacting *phases*".

These interactions may involve transfers of material and/or energy between pairs of phases, which are described in terms of the state–variables of the two phases through well defined physico–chemical *transfer laws.*

Other interactions arise from the fact that the *phases* are contained in *vessels.* We assume uniform pressure throughout each phase and across any free surfaces between phases. We note that a *vessel* is considered here simply as a containing surface; the mass of the containing walls and other relevant parts of actual vessels can, if desired, be taken into account by defining these as appropriate phases.

Using this concept, a *process* is a set of *vessels* linked through *connections.* These are normally pipes through which material flows, but we extend the concept to cover transfer through permeable membranes or transmission of energy through pistons in pumps, compressors or engines. Again a *connection* is an idealised topological concept, merely defining a link for transfer of material and/or energy, and itself containing no material or energy holdup. These transfers are also described by physico–chemical *transfer laws.* By use of an appropriate law it is thus possible to include idealised valves, pumps, compressors etc., which merely provide flow–resistance, or cause pressure–changes, without involving material holdup. Again if desired these additional elements can be modelled more realistically by defining appropriate "vessels" and "phases" if desired.

Of course the plant does not exist in isolation, but receives feeds from elsewhere, delivers products and also uses or generates utilities such as steam or cooling agents. There may also be heat losses to the atmosphere. These interactions with the environment are described in terms of exchanges with *reservoirs* of material or energy in appropriate states, which are assumed to be of infinite extent so that their states are not affected by transfers to or from them.

The states of the phases are affected not only by interactions with other phases but also by chemical reactions occurring within them. Thus we need to define the chemical components present, and the relevant stoichiometry and kinetic laws.

In figure 2.1 we give an example of the use of these concepts to describe part of a simple process, consisting of three *vessels*: a flash-drum, an absorber and a reactor, with feed, solvent, and cooling water provided from the "environment", a product to the environment and other streams for further processing.

Thus far several key concepts for the representation of a process system have been described, however these are insufficient to deal with potential situations routinely arising in a given process system. For example, phases may appear and disappear even in the presence of only finite–rate transfers, as illustrated by the flash–drum in figure 2.2.

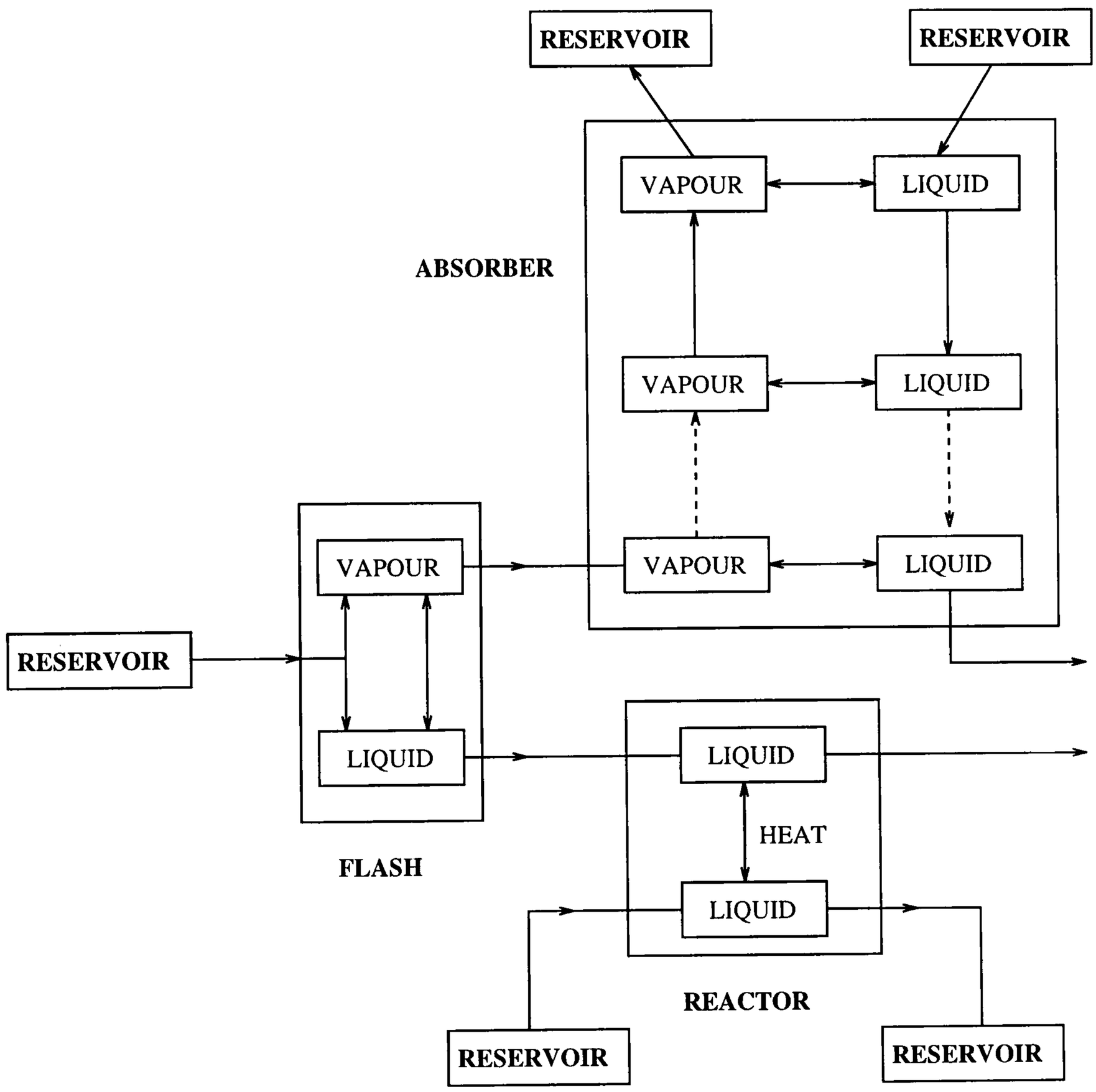

Figure 2.1: Conceptual diagram of simple process

Case(a) is the normal situation in which a vapour–liquid feed is separated into vapour and liquid streams. Case(b) shows that a third "bubble phase" will appear if the liquid level is above the feed point, and with finite mass transfer rate its composition will differ from the vapour phase. Case(c) shows that this may still occur if vapour bubbles in the feed are entrained in the liquid, while case(d) illustrates the case when feed or flow conditions cause the liquid to disappear. In this case the bottom connection may include a float–valve which only allows liquid to pass, or vapour may flow through the connection. Similar complications arise if the drum fills with liquid. Finally, a pressure–rise downstream may cause a reverse flow (in the absence of a non–return valve) and, if this contains vapour, there will again be a bubble-phase, as in case(e).

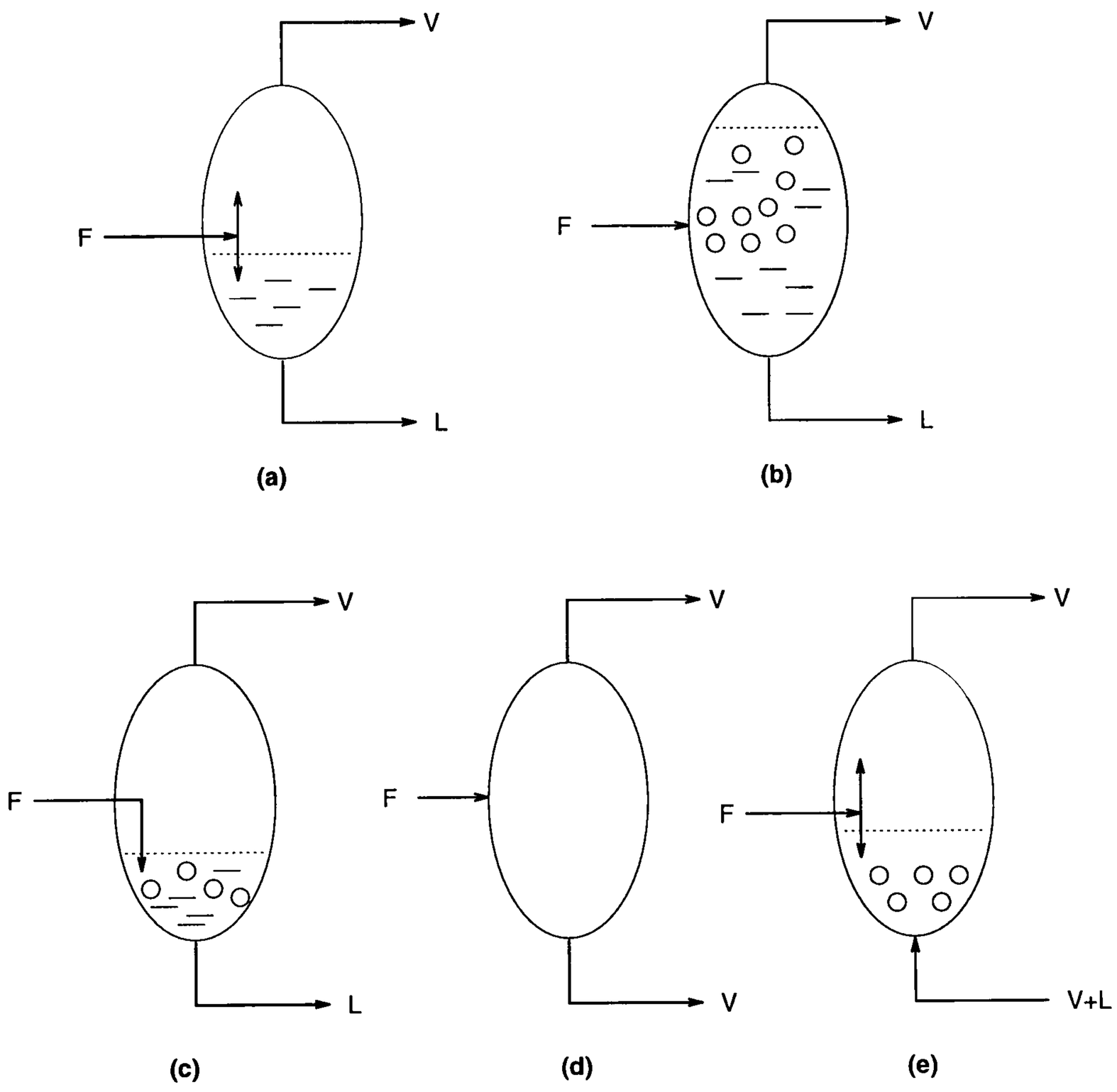

Figure 2.2: Multiple possibilities in flash drum

The tool for mathematical models generation should deal with all the possible situations likely to arise in a given process system, providing a computer environment which enables the user to describe these situations in a compact and easy manner. We therefore need to provide a language for the user to describe the relevant assumptions and provisions and hence introduce two more concepts.

First the phases in a vessel are assumed to be segregated into distinct layers in an order determined by their relative densities, unless otherwise specified. For a lumped–parameter system, the only other option is a uniform mixture of several phases, for example a suspension of a solid in a liquid, or a dispersion of one fluid in another in the form of bubbles, droplets, or an emulsion. We describe such a uniform mixture as an *aggregation* of phases, and again its relative position in the vessel is determined by its mean density.

Normally connections to vessels are simply pipes, and what flows out through the connection depends on what phase or aggregation of phases is covering the outlet. We therefore define a *port* as the position of a connection to a vessel, which can be identified with respect to the geometry of the vessel. We also allow flow through a port to be restricted to one direction (implying, for example, the existence of a non–return valve). As already noted, what flows through a connection depends on the phases in contact with the upstream port, but to provide greater flexibility in process description, a phase or aggregation of phases associated with the upstream can be independently specified as an attribute of the entry port.

2.1 Summary

The process representation in terms of the concepts given above is sufficiently general to describe the arbitrary complex nature of process systems. However it should be noted that the arbitrary complexity above is within the scope of lumped-parameter system since the generated model is limited to a lumped parameter system. As illustrated earlier, these situations include phase appearance and disappearance, those resulting from the geometry of vessels, the change of the direction for reversible flow and a vast variety of other factors, for example, the change of flow regime in a pipe between laminar and turbulent. The conceptual elements identified are as follows:

- *vessels*
- *phases*
- *transfer laws*

- *aggregation*
- *port*
- *connections*
- *reservoirs*

Over the following chapters we will illustrate how these arbitrary complex situations are incorporated into mathematical models, focusing on hierarchical descriptions of the inter–connections between phases or vessels. In the next chapter the language required to represent process systems in purely physical fashion will be proposed in detail, based on the newly refined process representation concept.

Chapter 3

Physical Modelling Language

In the preceding chapter, it was a basic point of view for process representation that chemical process systems are envisaged as a set of inter-connected *vessels* in which *phases* interact. Then basic process structuring elements were identified: (*phase*, *vessel* and *connection*). Also it was argued through a simple flash drum example that process systems routinely experience intrinsic discontinuities which arise typically from discontinuous physical behaviour such as phase transitions (e.g. phase appearance or absence), flow regime transitions (e.g. between laminar and turbulent), discrete changes from geometry of individual process units and other factors. In order to embody the basic process representation concept, a formal provision which enables users to describe process systems under investigation in a purely physical fashion is needed. A textual description in an elegantly designed language, which is completely oriented toward physical behaviour, has been chosen as the formal provision.

This chapter is concerned with the introduction to the conceptual design of a language in terms of its syntax and semantics, in order to enable users to describe process systems in question in a purely physical fashion with the relevant assumptions and provisions. The proposed language therefore must not only possess syntactic structures and semantics to keep description consistencies but also provide a purely physical behaviour-oriented formalism. Furthermore, it must contain syntax and semantics by which process discontinuities can be identified in order for generated mathematical models to encompass physical discontinuities as noted above. In realising the basic concept with this language, an attempt to let conceptual elements correspond to language syntax structures as well as semantics has been made. As a consequence, the language structure has three basic primitive physical process entities: *vessel*, *reservoir* and *connection*.

Drawing on these ideas, the elements of the proposed language and its structure

that enable the detailed description of three primitive process entities are introduced and several physical description examples written in the proposed language are presented, which will be utilised as input files to simulations.

3.1 Vessel Entity

This section is concerned with the development of language structures for the declaration of one of the three primitive physical process entities - *vessel* entity.

A vessel entity is defined as the surface enclosing one or more phases, which must fill the volume of a *vessel*; the mass of the containing walls and other relevant parts of actual vessels can if desired be taken into account by defining these as requisite *phases*. For lumped parameter systems, uniform pressure throughout each *phase* and across any free surfaces between *phases* can be assumed.

A vessel entity captures all the knowledge regarding the physical system of a vessel, including all possible phases present, their aggregation status, their physico-chemical interactions, and the things related to the vessel geometries. As a consequence, vessel entity forms a complex data structure that encapsulates a declaration of the following information regarding its structures and the physico-chemical behaviour likely to arise within the vessel:

- A set of all possible existing phases within the vessel.
- A set of possible uniform mixtures (aggregations) of several phases.
- A set of phase interactions between pairs of phases, which are defined by physico-chemical transfer laws.
- A set of terminals (ports) that represent the vessel's interface with its environment. The terminals will subsequently be utilised in the construction of a topological connection between vessels.
- A set of the vessel's geometries including its shape, dimension, orientation, and the terminal (port) positions.

Each item of information in the list above is called an *attribute* of the vessel entity. The identifier of an attribute must be unique, by which the attribute may be referenced in the vessel entity. The set of attributes encapsulate all the information about the physical representation of the vessel entity.

The declaration of a vessel entity commences with the keyword **VESSEL** and a colon followed by an identifier by which it may be referenced globally. The formalism of the declaration of a vessel entity is shown below.

Formalism of the declaration of **VESSEL** entity

VESSEL : <name> [1]

It is possible for users to declare more than one identifier in one vessel entity. Each identifier is distinguished by a comma. An illustrative example of multiple declarations of vessel identifiers is a series of CSTR (Continuous Stirred Tank Reactors), the formal description of which is shown in figure 3.1.

```
VESSEL : CSTR1, CSTR2, CSTR3, CSTR4, CSTR5
```

Figure 3.1: Example the declaration of VESSEL entity

The remainder of the declaration of a vessel entity is decomposed into a set of *sections* so that all the attributes belonging to a particular set of the declaration are collected in the corresponding section, which makes it easy to document the physical system under question. The details of how to declare each category of attributes will now be introduced.

3.1.1 Phase attributes

Phase attributes represent *thermodynamic phases* the state of which is describable in terms of a finite set of state variables, which implies that all material is present in well defined thermodynamic phases, as discussed in the previous chapter.

A phase instance is defined by two attributes; its name and type. A phase must be named uniquely within a vessel in the approach presented here. However, the name of identical phase instances in different vessels need not be unique and hence users are completely free to name phases arbitrarily. It is believed that this allows greater flexibility to support the physical model development through hierarchical sub–model decomposition (Barton, 1992).

[1] Language key–words are written in **bold** or *italics* and user–defined names are enclosed with angle brackets

In order to deal with inter–vessel connections, the phase attribute information is required to associate the phases in each vessel with those in the inlet and outlet streams. This association cannot be made by the phase name, since the implementation presented here allows the user to define different phase names in each vessel for a particular phase flowing between two or more connected vessels (through ports), as mentioned above. Therefore, the association must be made by phase type.

Conceptually, the phase type could be defined in analogy with those that exist in reality by its fundamental states, namely *vapour*, *liquid* and *solid*. However, this approach is not sufficient to give an unambiguous phase association as illustrated by the following example. Consider a vessel containing two immiscible liquids, A and B. The vessel has an inlet flow of one of the two liquids (say A) from an upstream vessel, but where it was named C by the user (see figure 3.2).

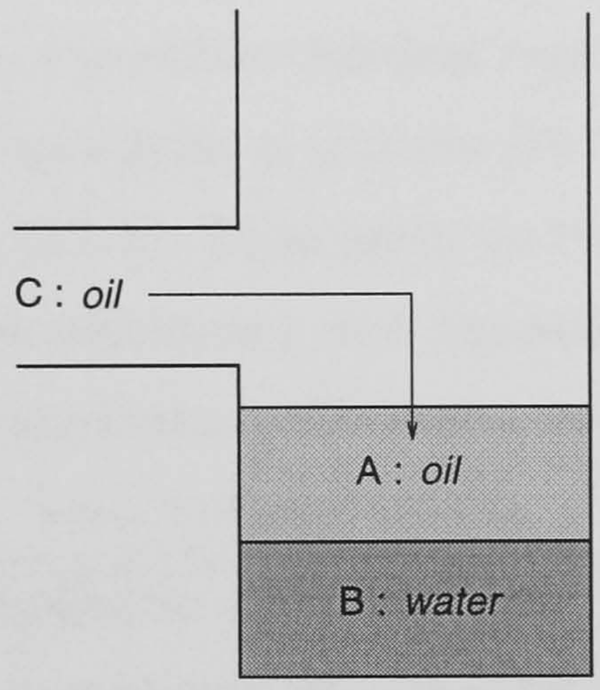

Figure 3.2: Vessel containing two immiscible liquids

Using the concept of only three phase types (*vapour*, *liquid* and *solid*) it is not possible to associate phase C with phase A since the phase types of A, B and C are all *liquid*. Additional information would be required to infer the correct phase association based on this conceptual view of the phase type.

Instead, the phase association is made by extending the concept of phase type to allow the user to declare more detailed instances than merely liquid or solid. In this way a particular phase type (the state of which is liquid or solid) may be defined uniquely across vessels, for example as liquid1, liquid2, solid1, solid2, etc so that the information about the state of the phase can be encapsulated in the declaration of the phase type. This allows the inter–vessel phase connections to be determined unambiguously. Using this

extended concept phase A and C in the above example would be defined with the same phase type (say liquid1) and this would be different from that of phase B (say liquid2).

It is the prerequisite for the use of this package that a user has enough knowledge of the physical process system to be able to describe the system. This requirement for the user includes the ability to declare all possible phases in vessels. For example, consider the flash vessel with two outlets and one inlet through which a pressurised gas flows from a container into the vessel. In the flash vessel, bubbles in the liquid must be created if the current liquid level is higher than the inlet point. Hence the possible phases in the flash vessel are separate vapour and bubbles in liquid. It is a user's responsibility in this example to declare the three phases in the flash vessel. If the user declares merely the separate vapour and the liquid containing no bubbles, the package will guide the user to declare the bubbles dispersed uniformly in the liquid before generating the appropriate mathematical model.

In order to deal with chemical reactions, we need to provide the language to enable the user to specify a set of possible chemical reactions which could occur in each phase in each vessel, similar to specifying a transfer law for interactions between phases (which will be demonstrated in §3.1.3). Thus again we can envisage a library of chemical reactions, which contains the stoichiometry and kinetics for each reaction. This could easily be added to the Phase attributes. However, the provision of the language for chemical reactions has not yet been implemented in the present package. Of course, absence of declared reactions implies no reactions.

In addition, we need a provision in the language for what compounds are present in the system. Since matter is present only in phases, this again implies a statement in the **Phase** section, and again one could require the user to give a list of compounds which could be present in each phase in each vessel. However this is clumsy, instead we note that it suffices to provide a list of compounds present in each *feed reservoir*; then the package can deduce from the flow type in connections (see §3.3), declared reactions and transfer laws what compounds could be present in all phases and all vessels.

The **Phase** section is employed for the declaration of all the *possible* phase attributes of a vessel entity. It must contain all the possible existing phase attributes. Phase attributes must be declared as instances of a phase type. The formalism of the phase section is presented below and an example phase section is shown in figure 3.3.

Formalism of **Phase** section

```
Phase : <name> : <type> : <a list of reactions>
```

Again it should be noted that the provision of the language for chemical reactions is not available in the present package. It is recommended that this language provision be considered in association with the specification of the identity of the compounds in a feed reservoir (see §3.2.1).

```
Phase :    B, V    : vapour
           L1      : liquid1
           L2      : liquid2
           L3      : liquid3
           L4      : liquid4
           S1      : solid1
           S2      : solid2
```

Figure 3.3: Example **Phase** section

3.1.2 Aggregation attributes

As introduced in chapter 2, the phases in a vessel are assumed to be segregated into distinct layers, in an order by their densities. For lumped systems, the other option is a uniform mixture of several phases, for example a dispersion of one fluid in another form of bubbles, droplets, or an emulsion. Such a uniform mixture is defined as an *aggregation* of phases enclosed in square brackets, and again its relative position in the vessel is determined by its mean density.

Aggregation attributes are used to describe a set of aggregations of phases listed in an increasing order of relative density. Of course, it is unnecessary to declare an aggregation attribute if a user assumed that there was only one single phase in a vessel.

As will be discussed in the port attributes section, in order to provide greater flexibility in physical description an aggregation attribute can be independently specified as an attribute of a port for an inlet stream associated with the aggregation in a vessel.

An **Aggregation** section is employed to declare aggregation attributes and uses the phase instances declared already in the phase section. The formalism of the aggregation section is presented below. An example of an aggregation section is shown in figure 3.4. Assume that figure 3.3 is the description of the phase attributes for these aggregation attributes.

Formalism of **Aggregation** section

Aggregation : `<a list of aggregations of phases>`

```
Aggregation :  V, [B,L1], [L2,L3], [S1,L4], S2
```

Figure 3.4: Example **Aggregation** section

3.1.3 Transfer Law attributes

Transfer laws represent the interactions of phases within a vessel. These interactions may involve transfers of material and/or energy between pairs of phases, which are described in terms of the *state* variables of the two phases through well defined physico-chemical *transfer laws*. A transfer law attribute is composed of a pair of relevant phases and the corresponding transfer law which must be already installed in a library.

Users are responsible for the requisite specification of transfer laws. Once a transfer law is specified by a user, the appropriate set of equations corresponding to the transfer law are invoked from a library where a set of equations for each transfer law have already been installed. Some transfer laws may include a subset of physical discontinuity equations depending on the nature of the transfer between the pair of phases. As will be described in the *connection entity* section, transfer law attributes will also be used in the transfer law declaration for transfers between vessels through their *ports*.

The **Transfer Law** section is employed for the declaration of transfer law attributes. The specification of a transfer law attribute is optional. If no transfer law is specified between a pair of phases, the phases do not interact. The first phase of a pair of phases must be declared as a source of a transfer if the transfer was *irreversible*. The formalism of the transfer law section is as follows.

Formalism of **Transfer Law** section

Transfer Law : `<phase name>` , `<phase name>` : *transfer law*

An example of a transfer law section is shown in figure 3.5, which represents a set of phase interactions in the flash drum discussed in chapter 2. This transfer law section contains two phase interactions: the physical thermodynamic equilibrium between the

liquid (L) and bubbles (B) dispersed in it, and the bubbles rising to the separate vapour (V) where the rising rate is determined by a BubbleRise transfer law. Note that there is no direct interaction between L and V.

```
Transfer Law :   B, L   : PhaseEquilibrium
                 B, V   : BubbleRise
```

Figure 3.5: Example **Transfer Law** section

3.1.4 Port attributes

Thus far there have been introductions of language syntax and semantics of three sections such as **Phase**, **Aggregation** and **Transfer Law** section, all of which have been centred around the intra-vessel physical description. This section is concerned with a vessel's interface with its environment, by which it is useful to identify and elaborate physical discontinuities. These intrinsic discontinuities may include phase transitions due to their thermodynamic states as well as transient behaviour within the vessel, and physical discontinuities resulting from the vessel's geometry.

As introduced in chapter 2, normally connections to vessels are simply pipes, and what flows out through the connection depends on what phase or aggregation of phases is covering the outlet. A *port* therefore is defined as the position of a connection to a vessel, which can be identified with respect to the geometry of the vessel. We also allow flow through a port to be restricted to one direction (implying for example the existence of a non-return valve), which is termed *irreversible*. In addition, material can flow into or out of a vessel through a port, depending on a flow driving potential (for example pipe flow driven by pressure difference between vessels), so called *reversible*.

We have the convention that what flows out of a port is determined by the layer covering the port, as mentioned above, though we can impose directionality (assuming a non–return valve) and could provide for selectivity (e.g.. via a filter or membrane), allowing passage of only specified phases or aggregations. For the latter, the language could then require identity between port specifications at each end of a connection, though it would be better for the package to deduce the logical intersection of the conditions.

All mass streams passing through ports as well as interphase transfers within a vessel carry an accompanying energy flow. To allow users to describe heat transfer

between vessels where there should be only energy transfer through ports of the vessels without any material flow, we introduce a special kind of port, an "energy port" through which only energy transfers.

A port attribute is composed of its name, type and optional specification of stream association. The optional stream association allows the user to specify explicitly the source and/or destination phase of material flowing through the port. For an energy port, there is no need to declare its type and the specification of stream association, but only its name. Three port types are available in terms of flow directionality through the port as follows;

- *in* : entry port
- *out* : exit port
- *both* : a port having *reversible* flow directionality

A port section is employed for the declarations of port attributes. A port name must be declared as an instance of its port type (except energy port). The port section begins with a keyword, **Port**, and has three options as defined in the formalisms as follow;

Formalism 1 of Mass Stream Port

Port : <name> : *type*

Formalism 2 of Mass Stream Port

Port : <name> : *type* : <specification of stream association>

Formalism of Energy Port

Port : <name>

An example of a port section is shown in figure 3.6 where there are six ports in a vessel. The flow of P1 and P2 must be in, and the flow of P2 can only be associated with the aggregate [B,L1]. The flow of P3 and P4 must be out, and only the phase L2 flows out of port P4. Finally P6 is an energy port, allowing only energy transfer.

For cases where the stream association is not specified, consistent rules for associating the inlet stream with a phase or aggregation in a vessel should be established.

```
Port :  P1  :  in
        P2  :  in   :  [B,L1]
        P3  :  out
        P4  :  out  :  L2
        P5  :  both
        P6
```

Figure 3.6: Example **Port** section

In principle the rules are classified into two categories depending on whether the inlet stream is a mixed phase (an aggregation) or single phase.

When a stream forming the aggregation is entering a vessel, the inlet stream association with a phase or an aggregation in the vessel depends on the availability of a suitable target aggregation in the vessel. If a suitable target aggregation was declared in the vessel, the inlet stream is associated with it regardless of the current levels of phases or aggregates in the vessel. Otherwise, the phases entering are distributed to the appropriate phases in the vessel. The formal description of this first category is as follows:

If there is a suitable target aggregation in the vessel,

- Associate the inlet stream with the target aggregation in the vessel.

Else

- Associate the components of the inlet stream with the same phase types.

For example, consider a vessel containing a separate vapour, an aggregate (bubbles dispersed into a liquid) and a liquid, each of which is distributed into its own distinct layer as shown in figure 3.7, an inlet aggregate (bubbles dispersed into a liquid) stream is entering through an entry port. All possible situations arising from the relative positions of the inlet port level and each level of phases or aggregate are given as cases (a), (b) and (c) in figure 3.7. By the rule above (corresponding to the If statement), since the aggregate in the vessel has been declared as a suitable target for the inlet stream association, the inlet stream is associated with the aggregate in the vessel in all cases.

Now consider a vessel containing only two separate phases (a vapour and liquid) without any aggregation and an aggregate stream (bubbles dispersed into a liquid) is entering the vessel through the entry port. This example is illustrated in figure 3.8 . By

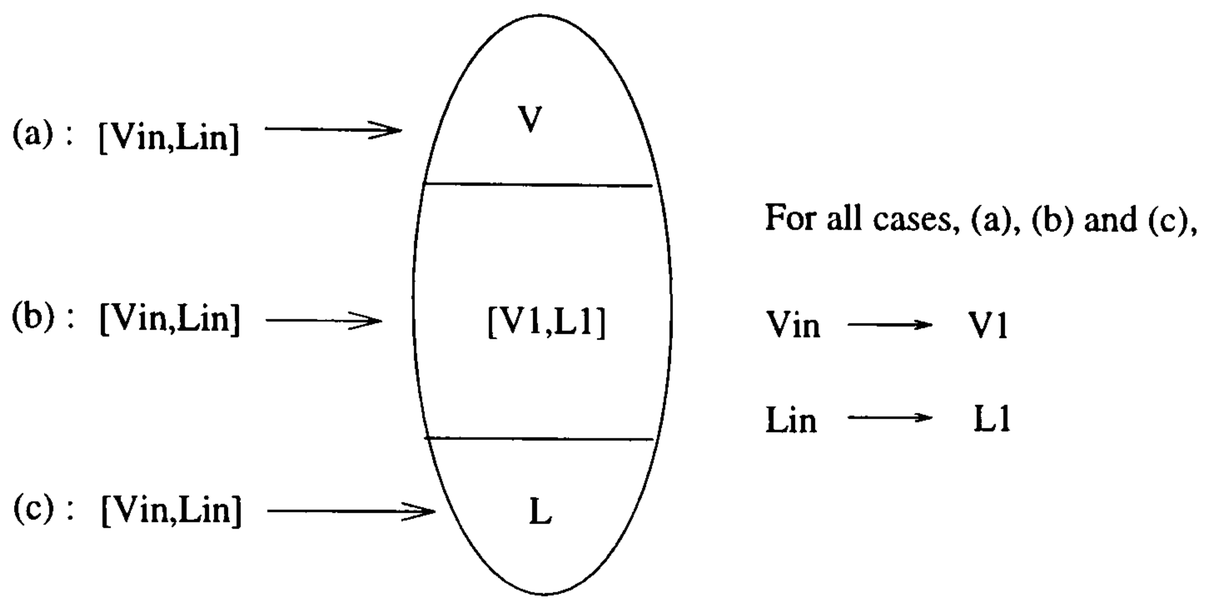

Figure 3.7: The Association of inlet aggregate stream by the aggregation type

the rule above (corresponding to the Else statement), since there is no suitable target aggregation in the vessel, the bubbles in the inlet stream are associated with the vapour in the vessel, the type of which is the same as that of the bubble phase and the inlet liquid phase is associated with the liquid in the vessel, in both cases (a) and (b). Note that this association represents "instantaneous separation" of the phases comprising the inlet aggregate stream.

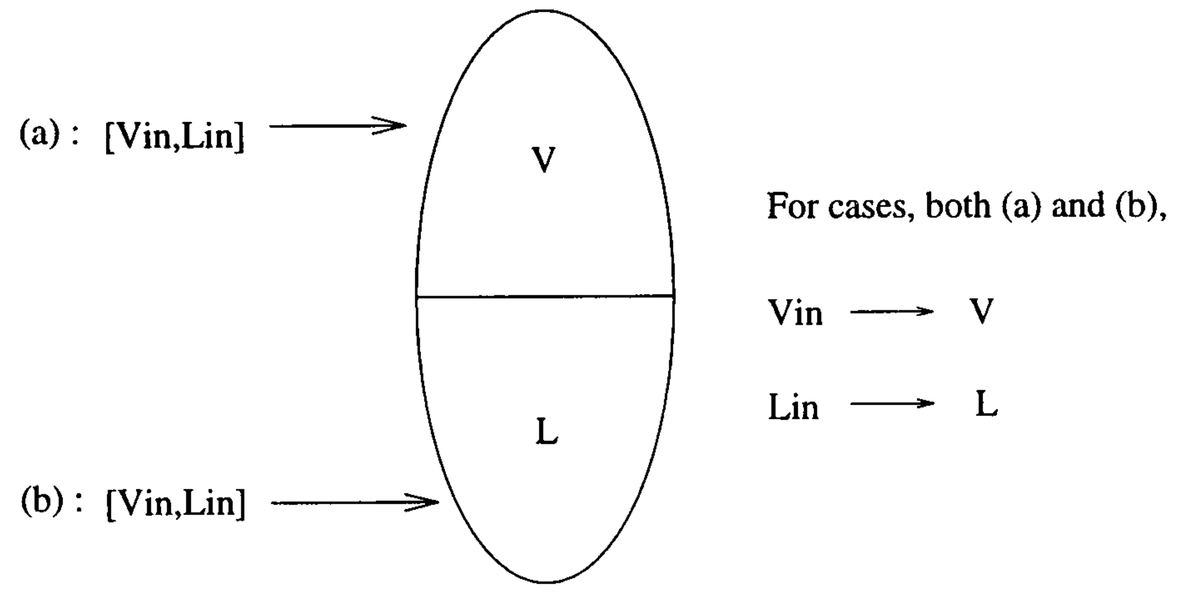

Figure 3.8: The Association of inlet aggregate stream by the phase type

When a stream entering a vessel is a single phase, the inlet stream association with a phase in the vessel depends on whether or not more than one suitable target phase (i.e. of the same phase type as the inlet stream) have been declared. In the former case the association of the inlet stream with a phase in the vessel is determined by the levels of the suitable target phases, whereas, in the latter case the association is determined by the same phase type. The formal description of this second category is as follows:

If there is more than one suitable target phase in the vessel,

- Associate the inlet stream with a target phase in the vessel, depending on the levels of the target phases in the vessel.

Else

- Associate the inlet stream with the same phase type.

For example, consider that a single vapour phase is entering the same vessel as given in figure 3.7. All the possibilities of associating the inlet stream with a phase in the vessel are given in figure 3.9, by applying the rule above (corresponding to the If statement). In the absence of ta suitable target phase covering the entry port the inlet stream is associated with the nearest target phase to the entry port (case (c)). It should be noted that this reflects the physical situation that when an inlet single vapour phase is entering into a liquid covering an entry port, it creates the bubbles dispersed into the liquid.

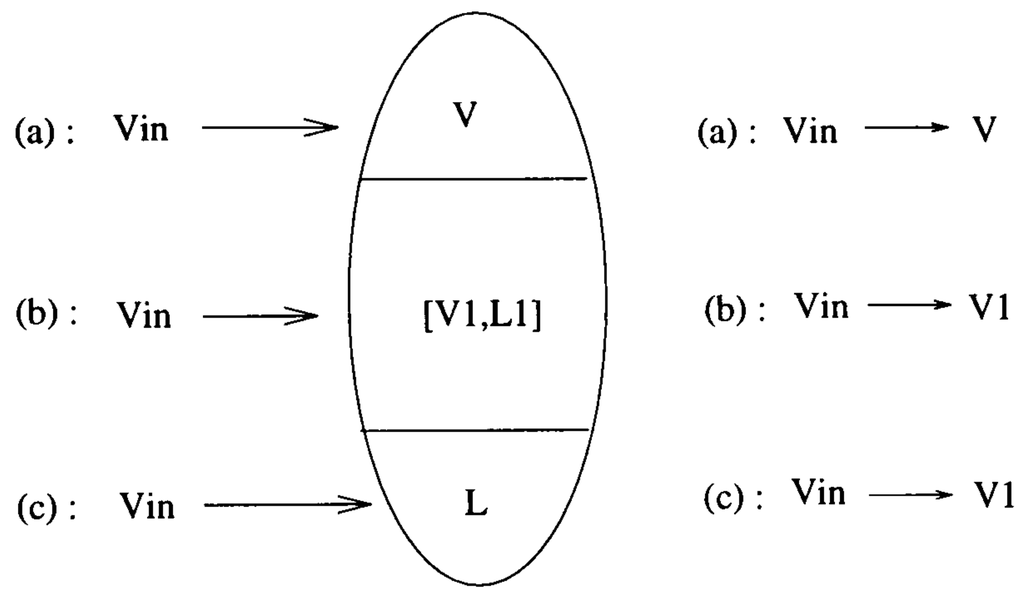

Figure 3.9: The Association of inlet single phase by the phase level

Again consider that a single liquid phase is entering the same vessel as given in figure 3.8. The resulting inlet stream association with a phase in the vessel is shown in figure 3.10 by applying the rule above (corresponding to the Else statement). The inlet single liquid phase is associated with the same phase type in the vessel regardless of its level. This represents an "instantaneous separation" of the inlet stream.

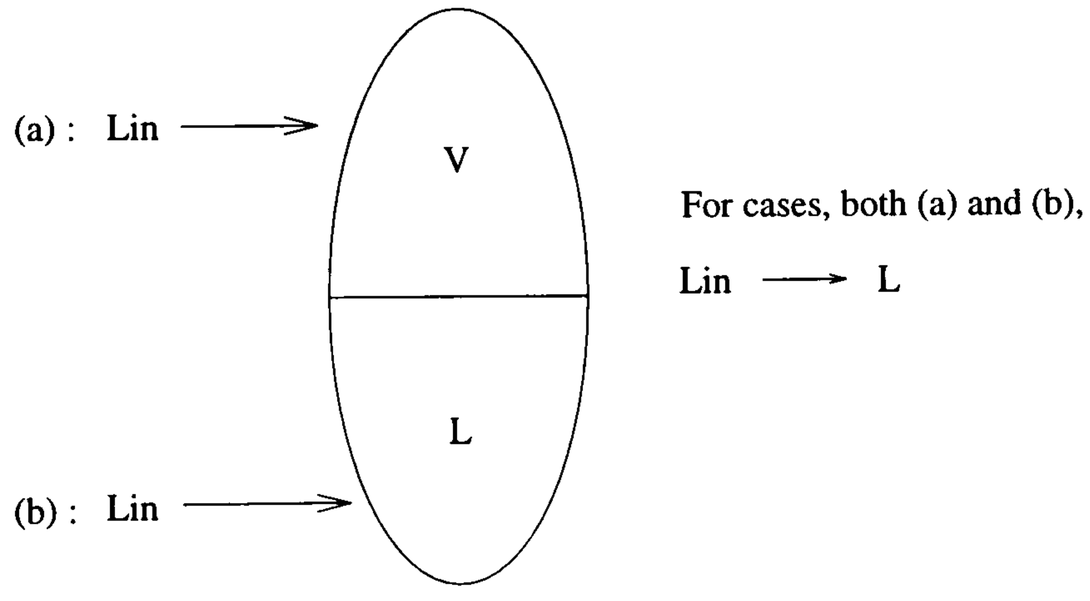

Figure 3.10: The Association of inlet single phase by the phase type

Thus far the rule for associating an inlet stream with a phase or an aggregation in a vessel has been presented with illustrative examples. It should be noted that although these examples include a system consisting of a vapour, liquid and its aggregation, the same rule can be symmetrically applied to immiscible liquid systems forming an aggregation (droplets dispersed into a containing phase).

In the early stages of design the vessel geometry will not be known, so the concept of a port can be extended to be an entry/exit point to a phase(or aggregation of phases), a vessel, or a process. This allows a *top–down* hierarchical approach to building up a model of a process. For example, a process can be first decomposed into a reaction system, separation systems etc. A separation system can in turn be decomposed successively into several columns, then each column into plates, condenser, reboiler, etc. finishing eventually with phases. At each level of decomposition, ports and connections are introduced to link the ports for the subprocess in question to the newly defined elements within it.

3.1.5 Geometry attributes

A geometry attribute is concerned with the physical description of a vessel geometry itself. It encapsulates all the knowledge about a vessel's shape, orientation and the relevant dimensions including the positions of ports already declared in the port section.

A geometry section is employed for the declaration of a geometry attribute and commences with keyword **Geometry**. It has the following four subsections and each will now be introduced in detail.

- Shape
- Orientation
- Dimension
- Port Position

3.1.5.1 Shape attributes

The shape of a vessel is described by a shape attribute. In reality there is a vast variety of shape types, however for simplicity only three kinds of primitive shapes are allowed in the present version of the language, which are *cylinder*, *sphere* and *box*. In addition to the shape types, we need the information about whether the vessel is *closed* or *open*. A **Shape** section is employed for the declaration of a shape attribute. The formalism of the shape section is as follows:

Formalism of **Shape** section

Shape : *type1* : *type2*

In the above formalism *type1* represents the type of the shape of a vessel and *type2* is the specification on whether the vessel is closed or open. The example of the declaration of the shape section is shown in figure 3.11.

```
Shape : cylinder : closed
```

Figure 3.11: Example **Shape** section

3.1.5.2 Orientation attributes

A geometrical orientation of a vessel is described by its orientation attribute. Only two primitive types of orientation have been considered; *vertical* and *horizontal.* Since the "orientation" for a *sphere* has no geometrical meaning, it applies only to a *cylinder.* In the present prototype package, only the *vertical* type is available due to geometrical complexities resulting from the various types of orientation of a vessel and also

An **Orientation** section is employed for the declaration of an orientation attribute. The formalism of the orientation section is as follows and an example of this is shown in figure 3.12.

Formalism of **Orientation** section

Orientation : *type*

```
Orientation : vertical
```

Figure 3.12: Example **Orientation** section

3.1.5.3 Dimension attributes

Dimension attributes are concerned with the declaration of relevant dimensions of a vessel. Selection of an appropriate set of dimension attributes depends on its shape.

Hence *height* and *diameter* are necessarily typical set of dimension attributes for a vertical *cylinder*, *diameter* for *sphere*, and *width*, *depth* and *height* for *box*.

A dimension attribute is described by its type followed by its numeric value. A **Dimension** section is used to describe dimension attributes and must contain all the requisite dimension attributes according to the vessel shape. A formalism of the dimension section is as follows;

Formalism of **Dimension** section

Dimension : *type* : `numeric value`

An example of the description of the **Dimension** section for a cylindrical vessel is given in figure 3.13.

```
Dimension : diameter : 1.0
            height   : 5.0
```

Figure 3.13: Example **Dimension** section

3.1.5.4 Port Position attributes

Port position attributes are concerned with physical locations of ports already declared in the port section. For all ports including energy ports, their locations must be declared in port position attributes. The vertical location of a port can be specified with either a numeric value or one of the relevant dimension types of a vessel, which has already been declared in the dimension section.

A **Port Position** section is employed for the declaration of port attributes and must contain all the ports previously declared in the port section. The formalism of the port position section is below, in which `identifier` represents the specification of the location of a port as mentioned above and `Z` is a language key–word to denote the position of a port.

Formalism of **Port Position** section

Port Position : **Z(**`<name>`**)** : `identifier`

An example of the declaration of a port position section is shown in figure 3.14. The vessel has three ports; P1 located at the bottom, P2 at 2.0 high from the bottom and P3 at the top, namely **height**.

```
Port Positions : Z(P1) : 0.0
                 Z(P2) : 2.0
                 Z(P3) : height
```

Figure 3.14: Example **Port Position** section

3.2 Reservoir Entity

In the previous section one of the three primitive physical process entities (*vessel* entity) has been demonstrated in terms of its syntax and semantics in detail. This section is concerned with the development of language structure for the *reservoir* entity.

As already described in the preceding chapter, a plant does not exist in isolation, but receives feeds from elsewhere and delivers products, and also uses or generates utilities such as steam or cooling agents. A reservoir entity is defined as a source or sink of infinite extent so that its state is not affected by transfer to or from it.

A *sink* reservoir should take whatever is delivered by the plant, and we need nothing but a name. However, a *source* reservoir is to allow the user to *specify* a stream from the environment to the plant, which in general is time–varying both qualitatively (different phases or aggregations at different times) and quantitatively (different ratios of phases, different state variables for each phase). We need language to enable the user to make these specifications, in addition to **Phase**, **Aggregation** (and possibly **Transfer Law**) sections, just as for a vessel – it is a vessel of infinite extent.

As illustrated in 3.1.1, in order to deal with the identities of the compounds involved, we need to provide the language for users to specify a list of compounds present in each *source* reservoir, then the package can deduce what compounds could be present in all phases and all vessels, from the connection arrows (see §3.3) and a set of chemical reactions declared in the phase section and transfer laws. However, a language provision for specifying a list of compounds in each source reservoir has not yet been implemented in the present package.

A reservoir entity begins with key–word, **RESERVOIR** followed by a unique

identifier by which it may be referenced globally. It is possible for users to declare more than one identifier in a reservoir entity, as shown in figure 3.15.

Formalism of the declaration of **RESERVOIR** entity

RESERVOIR : <name>

```
RESERVOIR : R1, R2, R3
```

Figure 3.15: Example the declaration of reservoir entity

A reservoir entity is composed of five optional sections as follows:

- Compound
- Phase
- Aggregation
- Transfer Law
- Port

Since the description of Phase, Aggregation and Transfer Law sections are identical to those of a vessel in terms of their syntax and semantics, Compound and Port sections will now be discussed.

3.2.1 Compound attributes

As stated above, compounds attributes represent the specification of a list of compounds present in each source reservoir. This information is also used for the package to deduce the requisite compounds in all phases and all vessels, as demonstrated in §3.1.1. So these attributes are limited to a source reservoir.

A **Compound** section is used for the specification of compound attributes and must contain all the compounds present in a source reservoir. The formalism of the compound section is as follows:

Formalism of the **Compound** section

Compound : <a list of compounds>

An example of compound section is given in figure 3.16 where propane and butane are specified as feeding chemical compounds.

```
Compound : C3H8, C4H10
```

Figure 3.16: Example Compound section

Again note that these attributes have not yet been implemented in the present package. Further, the implementation for chemical reactions has not been made. As discussed in §3.1.1 the provision of the language for compound attributes is recommended to be implemented, taking account of the language for chemical reactions.

3.2.2 Reservoir Port attributes

It is sufficient to declare only the port name for the reservoir port attributes (note that this is different from the port declaration for vessel ports where additional information is required, such as port type and position). The state variables required to define transfers between vessels and reservoirs are then automatically invoked from the appropriate transfer law in the library.

A **Port** section is employed for the declaration of port attributes and must contain all the ports needed to connect between reservoirs and vessels. The formalism of the port section of a reservoir is given below.

Formalism of the **Port** section of reservoir entity

Port : <name>

The example of a reservoir entity is shown in figure 3.17 where steam is consumed as a utility for a plant through the reservoir ports, P1, P2, P3.

```
RESERVOIR : steam
  Port : P1, P2, P3
```

Figure 3.17: Example reservoir entity

3.3 Connection Entity

In the previous section two of three primitive physical process entities; *vessel* and *reservoir* entity have been demonstrated in terms of its syntax and semantics in detail. This section is concerned with the development of language structure for the *connection* entity.

As introduced in chapter 2, a process system is a set of *vessels* linked through *connections* which may include the transfer of material through pipes or permeable membranes, or the transmission of energy by heat transfer or through pistons in pumps, compressors or engines.

A connection entity is defined as an idealised topological link for inter-vessel transfer of material and/or energy, and itself containing no material and/or energy. These transfers are described by physico–chemical *transfer laws.* Such a topological link is represented by the pair of ports of two mutually connected vessels. A completion of the description of these connections results in a whole flowsheet for a given process.

A connection attribute consists of its name, the topological link and the transfer law. A connection entity begins with key–word, **CONNECTION** followed by connection attributes. It must contain all the connection attributes required to describe the inter–vessel links (including a link between a reservoir and a vessel) for a given process. The formalism of the connection entity is as follows:

Formalism of **CONNECTION** entity

CONNECTION :

`<name> : <vessel1>.<port1>` *arrow* `<vessel2>.<port2>` ; *transfer law*

In the formalism `port1` and `port2` are user–defined port names of the two vessels mutually connected. The *arrow* in the formalism represents the types of flow through a connection and two types are available as follows, depending on whether the flow type is *irreversible* or *reversible*:

- an arrow with right end : $\longrightarrow$
- an arrow with both ends : $\longleftrightarrow$

The arrow with right end ($\longrightarrow$) represents *irreversible flow* through a connection (for example flow through a pipe installed with some kind of non–return valve), where the

type of `port1` and `port2` must be the exit and entry, declared to be *out* and *in*, respectively. The arrow with both ends ($\longleftrightarrow$) represents *reversible flow* through a connection where the types of the two ports must be reversible. This arrow is also used for energy transfer in which the types of the two ports must be those of energy port as previously introduced in §3.1.4.

Note that the transfer law for *reversible flow* of a connection can contain a pair of two different sets of equations, according to the flow directionality. This is termed *asymmetrically reversible.* On the other hand, if the set of equations for forward directed-flow are same as those for reverse flow, it is termed *symmetrically reversible.*

An example of a connection entity is shown in figure 3.18. The transfer law `IrreversiblePressureDrivenFlow` in `C1` describes pipe flow with non-return valve, which contains flow regime transitions between laminar and turbulent depending on the currently active *Reynolds Number.* the transfer law of `C2` is *symmetrically reversible* since the transfer law `PressureDrivenFlow` has the same set of equations for both directions containing flow transitions between laminar and turbulent, determined by the pressure difference between `FlashDrum` and `Receiver`. The details of the description of the library will be demonstrated in appendix C.

```
CONNECTION :
   C1 : Feeder.P1     ---> FlashDrum.P1 ; IrreversiblePressureDrivenFlow
   C2 : FlashDrum.P2 <--> Receiver.P1   ; PressureDrivenFlow
```

Figure 3.18: Example connection entity

As far as the physical description of "controllers" is concerned, the method suggested in earlier work (Vázquez-Román, 1992) can be used to provide a transfer law through a connection, using "measurements" from "sensors" providing variables from phases in vessels.

3.4 Physical Modelling Examples

In the previous section the details of the design of the proposed language for describing purely physical behaviour were described in terms of syntax and semantics. This section shows how process systems are described in the proposed language through several examples, all of which will then be simulated.

3.4.1 Flash drum

This example considers a flash drum where there exist three phases such as vapour, bubbles and liquid, and aims at validating how multiple possibilities of physical behaviour are encompassed.

Consider a simple process system composed of a cylindrical flash drum, one reservoir (source) and two reservoirs (sink). The conceptual diagram of this example is shown in figure 3.19. The dashed lines drawn in the flash drum represent all possible cases in which one of three phases existing in the flash drum participates in the inter–vessel connections through the ports. The two arrows linking phases `B` and `L`, and linking from `B` to `V` represent transfer laws `PhaseEquilibrium` and `BubbleRise` respectively. The flash drum named `FlashDrum` is fed by reservoir `R1` through the connection `C1` with transfer law `IrreversiblePressureDrivenFlow`. The flash drum has two outlet pipes through which one or more of three phases flows out depending on which phase covers the exit points of the outlet pipes. Within the flash drum, bubbles dispersed into a liquid are in equilibrium with the liquid and simultaneously rise to a vapour through the liquid. Note that all connections(`C1, C2, C3`) are physically some sort of pipes installed with non–return valves which do not allow reverse flow through the pipes. The physical description of this example is represented in figure 3.20.

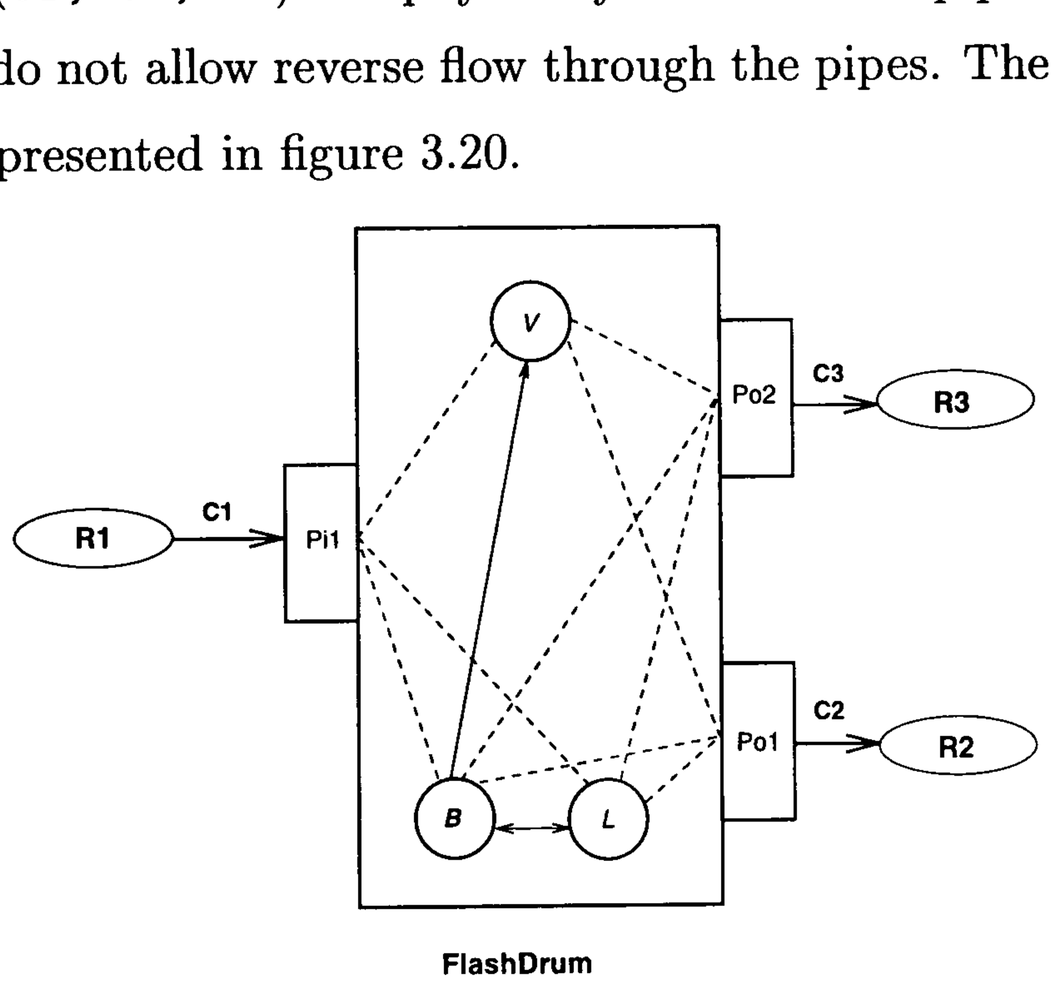

Figure 3.19: Conceptual diagram of Flash drum

```
VESSEL : FlashDrum
  Phase        : B, V : vapour
                 L    : liquid

  Aggregation  : V, [B,L]
```

```
Transfer Law : B, V : BubbleRise
               B, L : PhaseEquilibrium

Port         : Pi1 : in
               Po1 : out
               Po2 : out

  Geometry   : Shape          : cylinder : closed

               Orientation    : vertical

               Dimension      : height   : 3
                                diameter : 1

               Port Position  : Z(Pi1)   : 1.5
                                Z(Po1)   : 0.5
                                Z(Po2)   : 2.5

RESERVOIR : R1, R2, R3
  Port : P


CONNECTION :
  C1 : R1.P           ---> FlashDrum.Pi1 ; IrreversiblePressureDrivenFlow
  C2 : FlashDrum.Po1 ---> R2.P           ; IrreversiblePressureDrivenFlow
  C3 : FlashDrum.Po2 ---> R3.P           ; IrreversiblePressureDrivenFlow
```

Figure 3.20: Physical Modelling Example Flash Drum

3.4.2 Two Flash Drums with reversible flow

As in the representative diagram (Figure 3.21), there are two flash drums linked through a pipe allowing reversible flow, representing that there is no non–return valve in it. The flow direction is determined by the sign (*positive, negative*) of the pressure difference between the two flash drums. The Physical description of this example is presented in Figure 3.22. The reversible flow through the pipe is described in connection `C2` through the two ports `Po1` of `FlashDrum1` and `Pi` of `FlashDrum2`, with transfer law `PressureDrivenFlow`. Note that the types of these two ports have been declared as `both`

in each vessel and that the phase types declared in the two flashdrums must be the same since the connection C2 is reversible flow between the two flash drums.

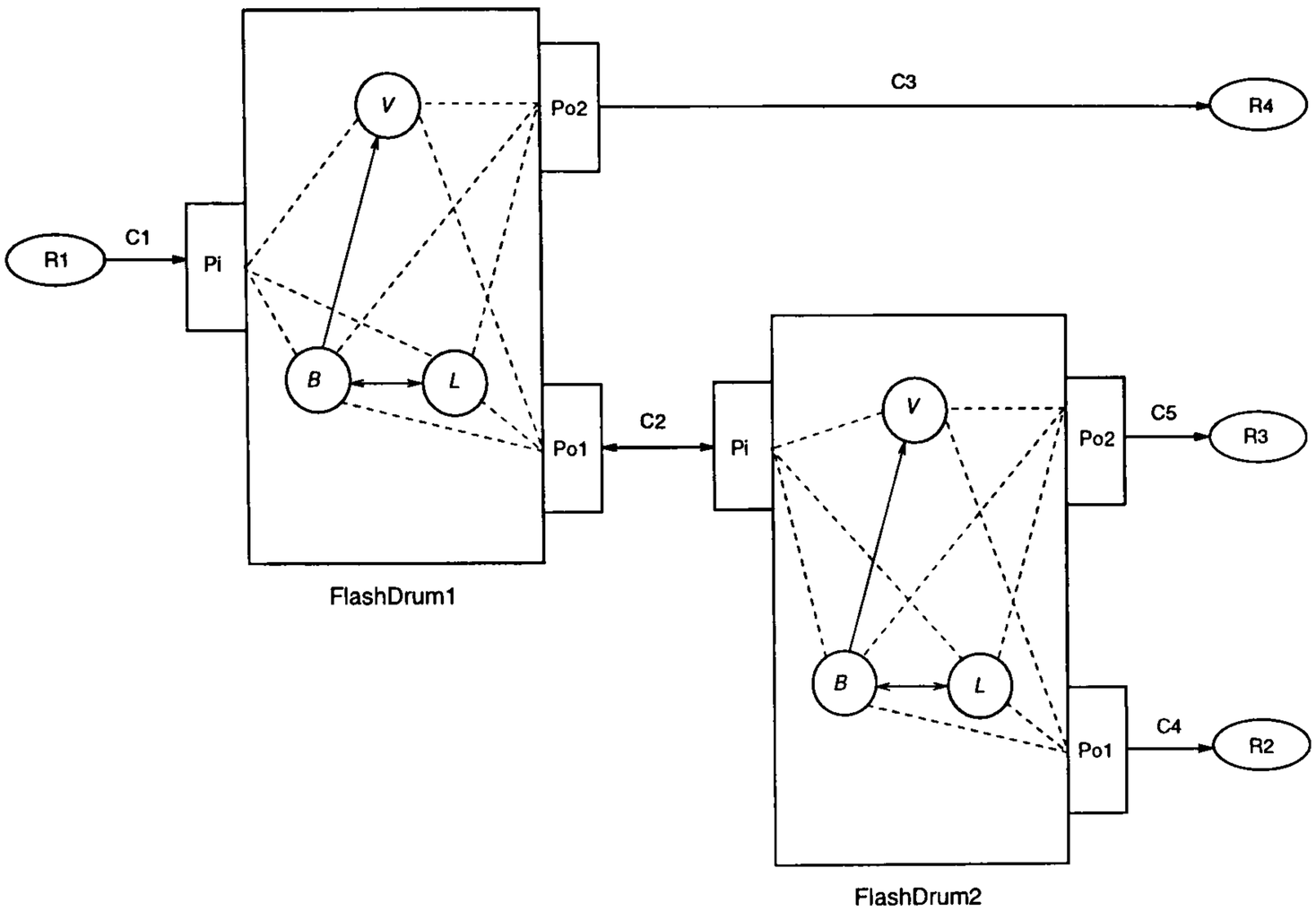

Figure 3.21: Conceptual diagram of Flash drum

```
VESSEL : FlashDrum1
  Phase          : B, V : vapour
                   L    : liquid

  Aggregation    : V, [B,L]

  Transfer Law   : B, V : BubbleRise
                   B, L : PhaseEquilibrium

  Port           : Pi  : in
                   Po1 : both
                   Po2 : out

Geometry      : Shape          : cylinder : closed

                Orientation    : vertical

                Dimension      : height   : 3
                                 diameter : 1

                Port Position  : Z(Pi)    : 1.5
```

```
                                    Z(Po1)    : 0.5
                                    Z(Po2)    : 2.5

VESSEL : FlashDrum2
  Phase           : B, V : vapour
                   L     : liquid

  Aggregation     : V, [B,L]

  Transfer Law    : B, V : BubbleRise
                   B, L : PhaseEquilibrium

  Port            : Pi       : both
                   Po1, Po2 : out

  Geometry        : Shape          : cylinder : closed

                   Orientation    : vertical

                   Dimension      : height   : 3
                                    diameter : 1

                   Port Position  : Z(Pi)    : 1.5
                                    Z(Po1)   : 0.5
                                    Z(Po2)   : 2.5

RESERVOIR : R1, R2, R3, R4
  Port : P

CONNECTION :
  C1 : R1.P            --> FlashDrum1.Pi ; IrreversiblePressureDrivenFlow
  C2 : FlashDrum1.Po1 <-> FlashDrum2.Pi ; PressureDrivenFlow
  C3 : FlashDrum1.Po2 --> R4.P          ; IrreversiblePressureDrivenFlow
  C4 : FlashDrum2.Po1 --> R2.P          ; IrreversiblePressureDrivenFlow
  C5 : FlashDrum2.Po2 --> R3.P          ; IrreversiblePressureDrivenFlow
```

Figure 3.22: Physical Modelling Example Two Flash Drums with reversible flow

3.4.3 Decanter

A decanter is a typical example of an open vessel. A schematic diagram of the decanter is shown in Figure 3.23. There are two immiscible liquids such as butanol and water in the decanter. Note that port `Po2` represents the rim of the decanter over which phases may flow out depending on their levels. The physical description of this example is given in figure 3.24. The flow mechanisms are described as `WeirOverFlow` in connection `C3`.

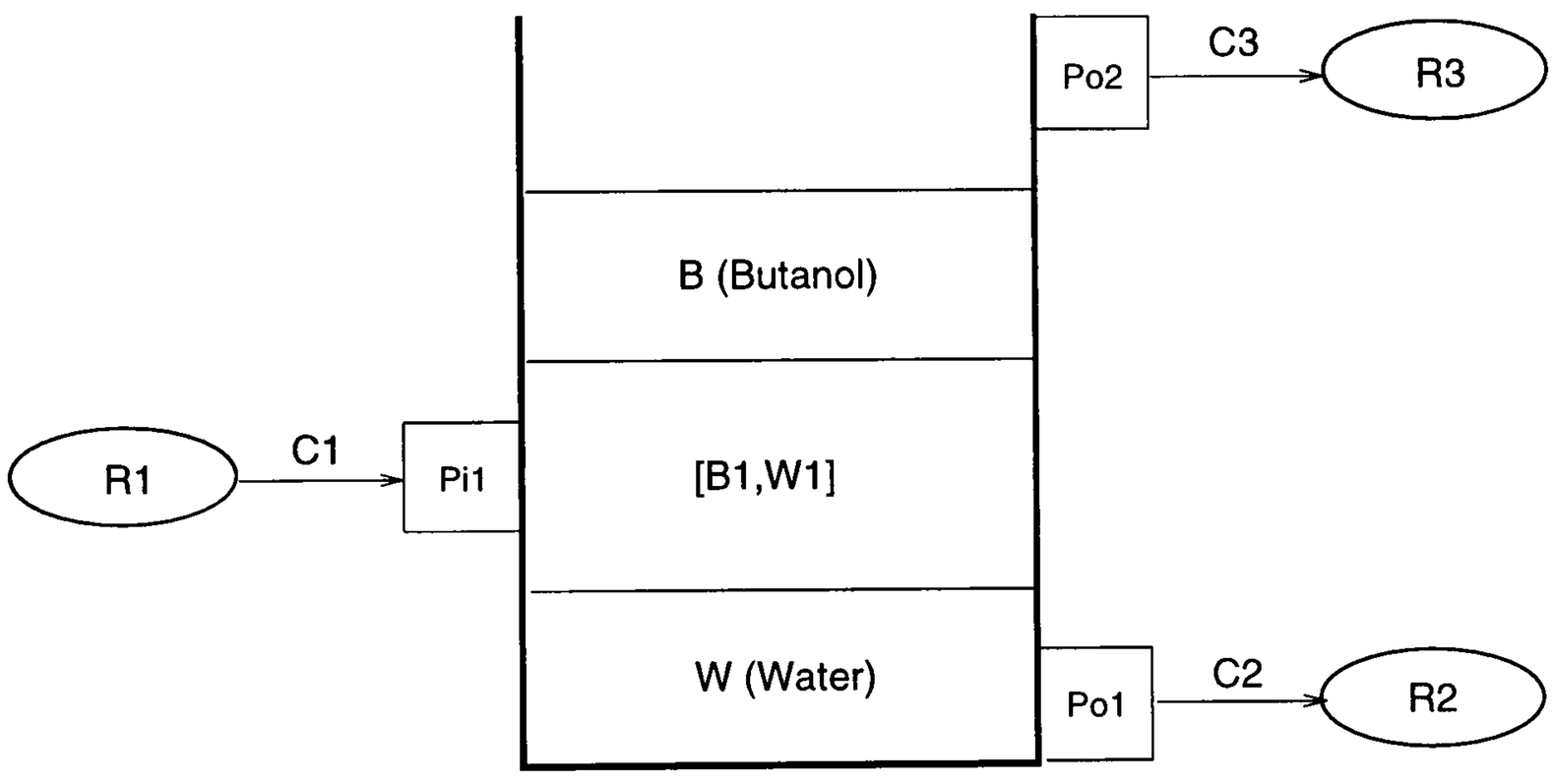

Figure 3.23: Schematic Diagram of Decanter

```
VESSEL : Decanter
  Phase          : B, B1 : liquid1
                   W, W1 : liquid2

  Aggregation    : B,[B1,W1],W

  Transfer Law   : B1,B : BubbleRise
                   W1,W : ContainingPhaseTransfer

  Port           : Pi1 : in : [B1,W1]
                   Po1 : out
                   Po2 : out

  Geometry       : Shape           : cylinder : open

                   Orientation     : vertical

                   Dimension       : diameter : 1
                                     height   : 5

                   Port Position   : Z(Pi1)   : 1.5
                                     Z(Po1)   : 0
                                     Z(Po2)   : height

RESERVOIR : R1, R2, R3
  Port : P

CONNECTION :
  C1 : R1.P          --> Decanter.Pi1 ; IrreversiblePressureDrivenFlow
  C2 : Decanter.Po1 --> R2.P          ; StaticPressureDrivenFlow
  C3 : Decanter.Po2 --> R3.P          ; WeirOverFlow
```

Figure 3.24: Physical Modelling Example Decanter

3.5 Summary

Based on the concept of the representation of process systems as already demonstrated in chapter 2, the language which enables us to describe chemical processes in a completely physical fashion has been designed and the details of its philosophy has been introduced in terms of its syntax and semantics by which physical discontinuities can be identified in the generated mathematical models. As a consequence, the three primitive physical process entities have been identified and the language has been structured into the hierarchy as shown in figure 3.25.

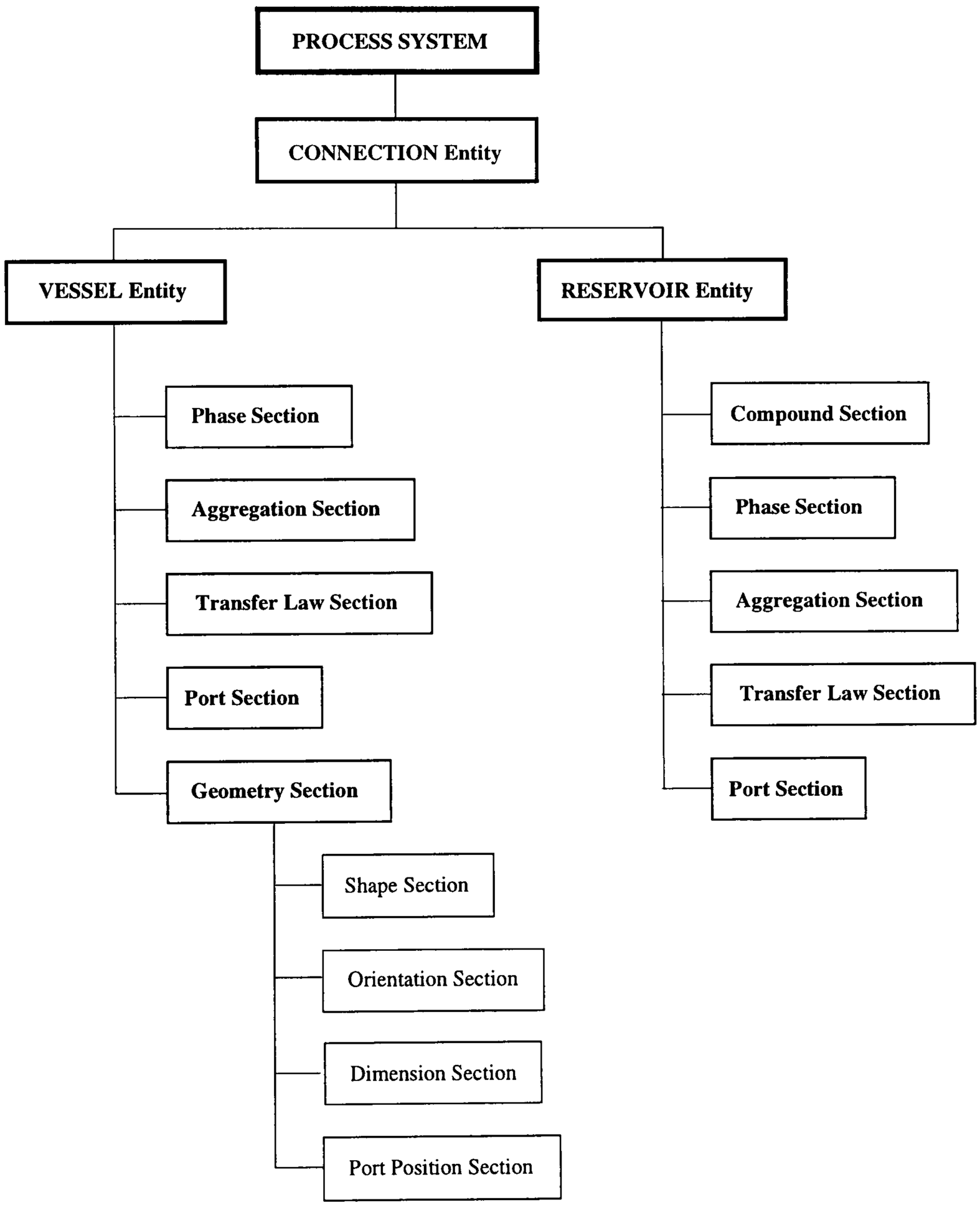

Figure 3.25: Hierarchical structure of physical modelling elements

Finally, three illustrative examples of physical description represented in the language have been introduced, all of which will be simulated later.

In the next chapter the methodology for generating the mathematical models from the information of physical systems described in the language developed will be demonstrated in terms of detailed algorithms for the model generation.

Chapter 4

Mathematical Model Formulation

In the preceding chapter, the design of the language for describing process systems in a purely physical manner has been demonstrated in terms of its syntactical structure and semantics, and several examples described in the language have been introduced.

This chapter demonstrates the methodology for formulating a lumped mathematical model from the information about a physical process system. Of course, as previously stressed, the mathematical models should encompass a set of physical discontinuities which process systems routinely experience in their dynamic behaviour. To deal with these key requirements, firstly the basic strategy for building mathematical models will be set up. Secondly the consistent and general formalism of the mathematical model required to encapsulate a set of physical discontinuities likely to arise for a given process, will be identified. Based on the model–building strategy and the generic model formalism, the detailed algorithms for generating the mathematical model from the information about physical description of a process system will be established as a final step of the mathematical model formulation method.

In the remaining sections of this chapter we will deal with the convention of making the notations used in the mathematical models and finally the construction of transfer law libraries. The mathematical model consists of a large number of parameters and equations expressed in variables and relevant mathematical operators. The consistency in making notation of those parameters and variables comprising a mathematical model must be maintained through the whole generation procedure. Thus the notation used in the generated mathematical model will be introduced.

4.1 Basic Model Building Strategy

This section describes the basic strategy for formulating mathematical models from the physical process representation described in the proposed language.

As previously stated, we consider only lumped parameter models, for which each phase has a uniform state throughout its extent, though of course distributed–parameter systems can be modelled in these terms through the use of cell–type models.

For each phase the masses of each chemical component present, the internal energy and pressure are taken as our basic state–variables. From the process description it is then possible to write down conservation laws for the mass of each component and the energy for each phase, containing typically an accumulation term, a rate of creation due to chemical reaction, and a term for each defined transfer.

Using a library of transfer–laws, we can then write an equation for each transfer, and we need to add means of computing relevant physical properties and relations between other thermodynamic variables (such as temperature) and the basic state variables, finally the volume and pressure relations for vessels.

This in general yields a dynamic model as a set of differential–algebraic equations (DAEs), but there are a number of complications. First, if the transfer law between two or more phases is equilibration, the relevant fluxes create a high–index DAE system; moreover the number of phases present may depend on the state. For these reasons the concept of "region" is introduced, the definition of which is a particular subdivision of the system being modelled composed of a single phase or multiple phases in thermodynamic equilibrium (Vázquez-Román, 1992). Then the package automatically associates phases defined to be in thermodynamic equilibrium in a single *region* and assumes the availability of an appropriate thermodynamic subroutine to determine the number of phases and the state of each from the basic state–variables for the region.

Now a complete strategy for formulating mathematical models in lumped–parameter systems in the form of a set of DAEs has been set up. However, in the course of building this strategy, the physical discontinuous behaviour that processes experience routinely in normal operation has not been considered. Hence in the next section we will discuss the method for dealing with these discontinuities and the generic formalism of mathematical models required to describe the discontinuities.

4.2 Generic Model Formalism

In the previous section a basic model building strategy has been demonstrated without a consideration of physical discontinuities. As has been emphasised in the previous chapters, one of the key requirements of the generated mathematical model is to describe the physically discontinuous behaviour. This behaviour includes phase transitions (e.g. the presence or absence of phases), flow regime transitions (e.g. between laminar and turbulent flow), those resulting from the geometry of individual process units and a variety of other factors. In order to deal with these physical discontinuities we need to identify a sufficiently generalised and consistent mathematical modelling formalism covering these discontinuities. We can then establish the detailed algorithms for generating mathematical models from a purely physical representation of process systems.

Recently the important modelling issues and special modelling requirements for combined discrete/continuous process systems have been identified and used as the conceptual basis of *gPROMS* (general–purpose PROcess Modelling System), which is a general–purpose software package for the modelling and simulation of combined discrete/continuous process systems (Barton, 1992). In fact we intend to generate mathematical models in the form of *gPROMS* input language. Hence it is essential to discuss the conceptual framework of *gPROMS*, and how it deals with physical discontinuities.

The fundamental model structuring concept in *gPROMS* is that process models are decomposed into two main entities; a *model entity* (a combined discrete/continuous model of physical behaviour of a process) and a *task entity* (external actions imposed on processes such as disturbances and control actions). These two entities are completely decoupled in order to aid the representation of process systems in a natural way. Again all the information stored in the two entities is encapsulated in a single entity, namely a *process entity*, which represents a dynamic simulation experiment. The process model representation contained in the model entities may include any possible discontinuities arising from the physico–chemical mechanisms governing the dynamic behaviour of process systems. Thus we need to focus on the mathematical formalism, which enables us to represent these discontinuities in a consistent manner.

In order to describe physical discontinuities their nature has been considered in terms of the transition mechanisms between discrete states. Then any discontinuities are categorised into the following three classes:

- **reversible discontinuity** : The condition for one state transition is the negation of the condition for the other. Hence the two transitions are describable with one

transition rule (`IF`), e.g. flow through a pipe.

- **irreversible discontinuity** : There is only one state transition which never returns to the other state, e.g. a burst–out of pressurised gas in a vessel fitted with a bursting disc.

- **asymmetric and reversible discontinuity** : The transition between two states is reversible but the transition conditions are not directly related. The `CASE` structure is used to describe this transition mechanism, instead of `IF` structure, e.g. a gas flow out of a tank controlled by a safety relief valve.

Mathematical modelling of dynamic physico-chemical behaviour of process systems in terms of lumped parameters yields a mixed set of differential and algebraic equations(DAEs), of which the natural mathematical formalism is expressed by a set of DAEs of the form (Pantelides *et al.*, 1988):

$$\mathbf{f}(\mathbf{x}, \dot{\mathbf{x}}, \mathbf{y}, \mathbf{u}, t) = 0 \tag{4.1}$$

$$\mathbf{u} = \mathbf{u}(t) \tag{4.2}$$

where: $\mathbf{x} \in X \subseteq \Re^n$, $\mathbf{y} \in Y \subseteq \Re^m$, $\mathbf{u} \in U \subseteq \Re^l$, $t \in T = [t^{(0)}, t^{(f)}]$ and

$\mathbf{f} : X \times \Re^n \times Y \times U \times T \mapsto \Re^{n+m}$.

The unknowns $\mathbf{x}$ and $\mathbf{y}$ are usually referred to as the differential variables and algebraic variables respectively, $\mathbf{u}$ are the known system inputs, and t is the independent variable time.

In general a process model involving physico–chemical discontinuities is represented in terms of several discrete states, each described by a potentially different set of variables and/or equations. The general formalism of a process model for representing these discontinuities in a consistent way has been suggested as follows (Barton, 1992):

- A set of variables $\mathbf{x}$, $\dot{\mathbf{x}}$, $\mathbf{y}$ and $\mathbf{u}$.

- A set of equations $\mathbf{f}(\mathbf{x}, \dot{\mathbf{x}}, \mathbf{y}, \mathbf{u}, t) = 0$.

- A set of *transitions* to other states (possibly empty).

A transition is described by:

- An initial state S^I.

- A terminal state S^T.
- A scalar logical expression $l(\mathbf{x}^I, \dot{\mathbf{x}}^I, \mathbf{y}^I, \mathbf{u}^I, t)$ to describe each possible transition between states.
- A set of relationships allowing the determination of consistent initial values for the variables in S^T from the final values of the variables in S^I.

However, the mere existence and completeness of a mathematical formulation does not automatically guarantee that it will be an "easy" or "natural" means of modelling process systems of realistic complexity due to the following items (Pantelides, 1995):

- The number of states in practical systems involving a number of discontinuous phenomena can be quite large.
- Although practical systems are often described by tens of thousands of variables and equations, the sets of such variables and equations describing different system states typically differ only by a few elements. Having to specify the entire mathematical description of each state separately could be unnecessarily tedious.
- The mathematical formulation makes no distinction between the description of the intrinsic physics of the process, and that of the external actions, manipulations and disturbances imposed on it.

A sophisticated modelling technique for dealing with the mechanisms of a large number of the transitions of discontinuous states has been developed by Pantelides (Pantelides, 1995). The basic view of physical discontinuities is that the large number of states arises because of the combination of several interacting discontinuous phenomena, the current state of which can be defined *independently*. Thus the current state of a system with discontinuous behaviour is described by the combination of a number of discontinuous subsystems, each described by its own *state–transition network*. In the flash drum example illustrated in chapter 2, these discontinuous subsystems include the equilibrium relationship between bubbles and liquid, the transition between laminar and turbulent flow regimes as well as the flow directionality resulting from the flow driving potential, and the transition of exit phases flowing through the outlet pipe depending on the liquid level. As a consequence, the mathematical description of process models including physical discontinuities is composed of a set of case *invariant* and *variant* equations, each characterised independently by its own discrete state and transition condition. We note

that this modelling principle is vital to develop our strategy for generating a mathematical model encompassing a set of physical discontinuities, hence it will be used as a basis for the development of the model generation algorithms. The detailed example as to how these independent transitions are combined, forming a set of case invariant and variant equations is given in the paper (Pantelides, 1995).

The algorithms for generating mathematical models will be illustrated in the next section, based on the basic model building strategy and the generic mathematical modelling formalism.

4.3 Model Generation Algorithms

Until now, we have introduced the basic model building strategy and identified the mathematical formalism required to deal with physical discontinuities. This section is concerned with the development of the complete algorithms for generating appropriate mathematical models, based on the model building strategy and the generic mathematical formalism demonstrated in the previous sections.

In the course of generating models the syntax analysis and semantic checking for the physical representation described in the proposed language are carried out in advance of the applications of the model generation algorithms. That is, the correct physical description in terms of the language syntax and semantics of language is an essential prerequisite for initiating the model generation algorithms.

The model generation algorithms are structured into the same hierarchy as that of the conceptual elements for the purely physical representation of process systems as shown in figure 3.25. As a consequence of the exact correspondence between physical model structuring elements identified from the conceptualisation of process systems, and the mathematical model generation algorithms with respect to the hierarchical structure, the generated mathematical models also have the same hierarchical structure. It should be recognised that this consistent hierarchical structure has been maintained during the course of the entire model generation procedure, from the conceptualisation of physical process systems, through the establishment of model generation algorithms, to the generation of appropriate mathematical models.

The method of describing the whole algorithm for generating a mathematical model is based on hierarchical sub–algorithm decomposition (top–down description).

Algorithm 4 given below is the master algorithm for generating a mathematical model from the physical representation of a given process system written in the proposed

physical modelling language. It contains the fixed set of key sub–model generation tasks at top level.

Algorithm 4 GENERATE PROCESS SYSTEM MODEL

1. GENERATE VESSEL SUB–MODEL. (see **algorithm 4.1**)
2. GENERATE RESERVOIR SUB–MODEL. (see **algorithm 4.2**)
3. GENERATE CONNECTION SUB–MODEL. (see **algorithm 4.3**)

Vessel, reservoir and connection sub–model terms used at each step of algorithm 4 represent the mathematical sub–models generated from the physical information encapsulated into the three basic entities of *vessel, reservoir* and *connection.* The background and subsequent derivation of each procedure of algorithm 4 will now be discussed in detail.

4.3.1 Vessel Sub–Model Generation

A vessel sub–model is generated by applying a relevant model generation algorithm to the physical information encapsulated into the *vessel entity.* To do this, for each vessel, we need to initiate the tasks concerned with the generation of the mathematical sub–model from the information stored into the sections comprising the vessel entity (phase, aggregation, transfer law, port and geometry sections). These tasks are therefore composed of a subset of procedures of calling algorithms, each of which corresponds to the algorithm for generating the mathematical sub–model from the physical information stored in the corresponding section. The formal algorithm for generating vessel sub–model is given algorithm 4.1 below.

Algorithm 4.1 GENERATE VESSEL SUB–MODEL

- For each vessel,
  1. GENERATE PHASE SECTION SUB–MODEL. (see **algorithm 4.1.1**)
  2. GENERATE AGGREGATION SECTION SUB–MODEL. (see **algorithm 4.1.2**)
  3. GENERATE TRANSFER LAW SECTION SUB–MODEL. (see **algorithm 4.1.3**)

4. GENERATE PORT SECTION SUB–MODEL. (see **algorithm 4.1.4**)

5. GENERATE CONSERVATION EQUATIONS. (see **algorithm 4.1.5**)

6. SPECIFY DISCONTINUITIES IN A VESSEL. (see **algorithm 4.1.6**).

7. GENERATE GEOMETRY SECTION SUB–MODEL. (see **algorithm 4.1.7**)

From the implementational point of view, each step of algorithm 4.1 is completed by formulating the physical information corresponding to each section of the vessel into abstract data types made suitable for storing the information about the mathematical model. As a consequence, once the application of the algorithm 4.1 is finished successfully, the mathematical sub–model corresponding to the vessel entity is obtained, encapsulating all the information about the model in an appropriate data structure. If more than one vessel were declared in one vessel entity, all the information stored in the above data structure for the one vessel are copied to others.

Subsequent derivation of the subset of algorithms comprising algorithm 4.1 will now be presented in detail.

4.3.1.1 Phase Section Sub–Model

This section is concerned with the generation of the mathematical sub–model from the physical information about the phase attributes declared in the phase section.

As mentioned in §3.1.1, the provision of the language for chemical reactions has been suggested as a phase attribute, but has not yet been implemented in the present package. Consequently there is no algorithm for dealing with chemical reactions.

Recall that the phase attributes described in the phase section includes the phase types for a set of all possible existing phases within the vessel, and that the phase types are employed for associating a stream passing through a port with a phase in a vessel. We thus create the variables required to represent the types of all the phases specified to be present in the vessel and declare all the phase types defined by the user. We also need to formulate the required constitutive equations. Finally the requirement that the pressure for all the phases present in a vessel is uniform is expressed in the form of equations. The formal procedures are described in algorithm 4.1.1 below.

Algorithm 4.1.1 GENERATE PHASE SECTION SUB–MODEL

1. Create phase type variables.

2. Declare user–defined phase types.

3. Generate the required constitutive equations.

4. Equate the pressure of the vessel to that of phases present.

At step 2 the phase type defined for each phase in the physical description is declared as parameters in the mathematical model for later use to provide data for dealing with a set of discontinuities resulting from the selection of a phase or an aggregate passing through ports, the details of which will be described in §4.3.1.4.

The constitutive equations (step 3) involve those relating total mass with density and volume as well as those relating mass of each component with total mass and mass fraction.

As already stated, one of the basic assumptions for lumped systems is that pressure is uniform through all phases present within a vessel. This is embodied at step 4 by equalising the pressure of the vessel to that of all phases present.

4.3.1.2 Aggregation Section Sub–Model

This section describes the procedures for generating the mathematical sub–model from the physical description of the aggregation section.

By the similar rationale discussed in the previous section, for each aggregate, we create the variables representing the aggregation type and declare the user–defined phase type of the aggregate. Also total mass ratio of the dispersed phase should be defined for each aggregate. Finally the required constitutive equations and the equations describing the uniform pressure for each aggregate are then formulated.

As demonstrated in the previous chapter, the information about which phase in a vessel is covering a port is required to determine the association of an entry stream with a phase in the vessel as well as the selection of a phase flowing out of an exit port. To do this, we need to create the variables for describing volume bounds of each element of the aggregation list and then relate them to their level variables. The formal procedures for the formulation of aggregation section sub–model is given in algorithm 4.1.2 below.

Algorithm 4.1.2 GENERATE AGGREGATION SECTION SUB–MODEL

1. For each aggregation of phases,
   (a) Create the variable representing the aggregation type.
   (b) Declare the aggregation type.
   (c) Define the total mass ratio of the dispersed phase.
   (d) Generate the required constitutive equations.
   (e) Equate the pressure of the vessel to that of the aggregate.
2. For each element of the aggregation list,
   (a) Create variables representing lower and upper volume bounds.
   (b) Create a variable representing level for the element.
   (c) Equalise the upper volume bound to its level.
   (d) Express inter–element boundaries by equalising the lower volume bound of one element, to the upper volume bound of the element located below.
3. Specify numeric values starting with "0" at the bottom of the vessel, to the lower volume bounds of each element.

The phase type variable for the aggregation is declared by combining the two strings of the user–defined phase types for the set of the aggregated phases through under–bar "_" (step 1.b). As an example of step 1.b, a phase type for an aggregation phase is `vapour_liquid` where `vapour` and `liquid` are user–defined phase types for the aggregation. Total mass ratio of the dispersed phase (step 1.c) is used to calculate the physical properties of an aggregation from the physical properties of each phase in the aggregation.

The constitutive equations (step 1.d) include the relation of total mass with density and volume and that of each component mass with total mass and mass fraction as well as the expression of the aggregation variables in terms of the corresponding variables of the two aggregated phases with respect to mass, density, enthalpy, internal energy, etc.

Step 2 and 3 are concerned with the specification of the lower/upper volume bounds for all the elements of the list of aggregation of phases. The notation of the variables used in mathematical models are expressed as "Bot" and "Top" to denote the lower and upper volume bound respectively.

4.3.1.3 Transfer Law Section Sub–Model

This section presents the algorithmic procedures for generating the mathematical sub–model from the physical description of the transfer law section. Recall that in the transfer law section a possible set of the mechanisms of the transfer between phases present in a vessel are described by specifying the relevant phases with the names of transfer laws which have been already installed in the library. From this information a set of appropriate equations belonging to transfer laws will then be invoked from the library. Whence the set of equations the required additional properties are created. The formal procedures for the generation of the transfer law section sub–model are given in algorithm 4.1.3 below.

Algorithm 4.1.3 GENERATE TRANSFER LAW SECTION SUB–MODEL.

1. For each transfer law,

   (a) Search the transfer law entry in the library.

   (b) Transform the set of equations in the library into the appropriate form of equations by taking account of the relevant phases.

From the implementational point of view once a transfer law is invoked, its availability is checked by searching its entry in the library table with its name (step 1.a) and the set of equations comprising the transfer law are then transformed into the appropriate form of equations by passing the pair of phases participating in the transfer law as the parameters for the invoking function (step 1.b).

4.3.1.4 Port Section Sub–Model

This section describes the procedure for creating the variables for the streams passing through ports from the information stored in the port section.

We note that it is reasonable that the information about the material flowing out of a port is encapsulated into a mass stream instance containing a set of its attributes; mass rate, enthalpy flow, dispersion ratio and phase type. In order to deal with energy transfer with no mass flow, we build an energy stream instance, which contains only an enthalpy flow as its attribute. The formal procedure for creating stream variables is given in algorithm 4.1.4 below.

Algorithm 4.1.4 GENERATE PORT SECTION SUB–MODEL

1. For each mass stream port,

   (a) Create the set of variables as the mass stream attributes of the port; mass rate, enthalpy flow, dispersion ratio and phase type.

   (b) Build the MassStream instance containing the set of the above attributes.

2. For each energy port,

   (a) Create the enthalpy flow variable as the energy stream attribute of the port.

   (b) Build the EnergyStream instance of the port with the above attribute.

4.3.1.5 Conservation Equations

This section is concerned with the algorithm for detecting regions and for formulating the conservation equations in terms of mass and energy.

Recall that the flux between the phases in thermodynamic equilibrium creates a high index DAE system as incorporated into the conservation equations. To avoid this problem a *region* has been defined as a particular subdivision of the system being modelled consisting of a single phase or multiple phases in thermodynamic equilibrium (Vázquez-Román, 1992) and the pair of phases in thermodynamic equilibrium is then incorporated into a single region. The appropriate set of conservation equations is formulated for each region instead of each phase present in the vessel. Therefore, the procedure for detecting regions are required in advance of formulating the conservation equations. The phases in thermodynamic equilibrium are detected from the transfer law attributes specified in the transfer law section.

The formal procedure for formulating conservation equations is given in algorithm 4.1.5 below.

Algorithm 4.1.5 GENERATE CONSERVATION EQUATIONS

1. DETECT REGIONS. (see **algorithm 4.1.5.1**)

2. For each element of regions,

(a) GENERATE MASS CONSERVATION EQUATION. (see **algorithm 4.1.5.2**)

(b) GENERATE ENERGY CONSERVATION EQUATION. (see **algorithm 4.1.5.3**)

As the task for detecting regions is completed by step 1, the information about a fixed set of elements of regions will then be saved into the proper data structure for later use to formulate conservation equations.

The regions are first taken to be the set of phases present in the vessel. The set of phases in thermodynamic equilibrium are extracted from the regions and then included into the regions as a single element. The formal procedure for detecting regions is given in algorithm 4.1.5.1 below.

Algorithm 4.1.5.1 DETECT REGIONS

1. Define *regions* as a set of phases in the vessel.
2. For each pair in phase equilibrium,
   (a) Remove the pair of phases from *regions*.
   (b) Define a new element representing the pair of the phases.
   (c) Incorporate the element into *regions*.

A set of mass and energy conservation equations are formulated for each region having been already detected by the completion of algorithm 4.1.5.1. Recall that the physical discontinuities routinely arising in the vessel include appearance and disappearance of phases present in the vessel, and those resulting from the selection of the phase passing through the ports. To enable us to deal with these discontinuities later, the conservation equations are formulated in such a way that each element of the regions has a potential to pass through all the mass stream ports declared in the vessel. This has a consequence that each conservation equation has a set of potential mass input/output rate terms for the flows through all the mass stream ports.

In order to determine the appropriate sign for the transfer terms in the conservation equations, we introduce the convention that the first of the two phases specified in the transfer law section is taken as the source, and the other as the sink. For example, in

fig 3.5 the `B` phase is taken as the source and the `V` phase as the sink in the `BubbleRise` transfer law.

The formal procedure for generating a mass conservation equation is given in algorithm 4.1.5.2 below.

Algorithm 4.1.5.2 GENERATE MASS CONSERVATION EQUATION

1. Write the differential term followed by equal sign.
2. For each mass stream port,
   (a) Create the mass rate term.
   If the port type is an entry then
   (b) Add the mass rate term.
   Else
   (b) Subtract the mass rate term.
3. For each defined inter–phase *transfer*,
   (a) Create the mass rate term.
   If the current region is a sink for the transfer then
   (b) Add the mass rate term.
   Else
   (b) Subtract the mass rate term.

A conservation equation for the mass of each component comprising a region contains typically an accumulation term, the terms for the rates of inter–vessel mass flow, the terms for each defined transfer between interacting phases, and the rates of creation or consumption due to chemical reactions.

The accumulation term is expressed with a time derivative (denoted `$`) followed by the mass holdup term of each component, for instance `Mass_V` represents the mass holdup of components present in the region `V` (step 1) and `$Mass_V` its time derivative.

As previously mentioned, each element of regions has a possibility of flowing through the mass stream ports present in a vessel, though the selection of a phase or an aggregation flowing through a mass stream port will be made in algorithm 4.1.6. We then incorporate into the mass conservation law the mass rates of flow through all the mass stream ports present in a vessel. The notation of a mass rate term is the combination of mass rate (`Rate`), the name of the element of the region and the name of the port, for instance `Rate_V_P` where `V` and `P` denote the name of element and the name of the port respectively. The sign of a mass rate term for a port depends on the type of the port specified by a user. By default, if a port type was declared as an entry, the sign becomes positive, otherwise (including an exit and *reversible* port) it becomes negative (step 2.b)[1].

The notation of a mass rate term for an intra–vessel defined transfer is made up of the combination of the rate symbol (`Rate`) with the names of phases participating in the transfer (e.g. `Rate_B_L` where `B` and `L` are phases of a transfer) (step 3.b).

Although the provision for dealing with chemical reactions has not been implemented, the method for this will now be mentioned briefly. The reaction rate included in component mass conservation equations is composed of the following three terms:

- stoichiometric coefficients
- molecular weights
- reaction kinetics

In order to keep the dimension of the reaction rate terms consistent in the mass conservation equations, the dimension of the vector of stoichiometric coefficients is same as that of molecular weight, which is the number of components of the system. The signs of the corresponding stoichiometric coefficients of reactants and products are negative or positive respectively, since the reactants and products are consumed and produced during the chemical reaction. Of course, the reaction rate invoked by a user must have been already installed in the library for chemical reactions.

As with a mass conservation law, an energy conservation law for a phase contains typically an accumulation term, the terms for the inter–vessel energy flows and the terms for the transfers specified between interacting phases. The procedures for formulating the energy conservation law will now be illustrated in algorithm 4.1.5.3 below.

[1]In the case of reverse flow through a *reversible port* since the relevant inter–vessel transfer law contains the appropriate sign representing the directionality of the flow, there is no inconsistency for the sign of rate terms, for example, the sign of DrivingForce in appendix C.7 (`SGN(DrivingForce)`) is negative in the case of reverse flow.

Algorithm 4.1.5.3 GENERATE ENERGY CONSERVATION EQUATION

1. Write a differential term followed by equal sign.

2. For each port,

   (a) Create the enthalpy term.

   If the port type is an entry then

   (b) Add the enthalpy flow term.

   Else

   (b) Subtract the enthalpy flow term.

3. For each defined inter–phase *transfer*,

   (a) Create the enthalpy flow term.

   If the current region is a sink for the transfer then

   (b) Add the enthalpy flow term.

   Else

   (b) Subtract the enthalpy flow term.

In principle the total energy of a phase should be defined as the summation of its kinetic, potential and internal energy in the absence of electric and magnetic fields. However we note that the kinetic and potential energy can be often neglected in process systems. The accumulation (differential) term therefore represents the total internal energy of the current region. The total internal energy is described by the mass specific internal energy multiplied by the total mass of components for the region (step 1).

It should be noted that an energy conservation law for a phase concerns the total energy and total mass, as opposed to a mass conservation law where the mass of each component within the phase is conserved. The reaction term therefore is included implicitly in the internal energy. This has a consequence that there is no need to incorporate a term for a chemical reaction into the energy conservation law.

All the mass streams as well as the mass transfer between interacting phases carry an accompanying energy flow. The rates of energy flows required to formulate a conservation equation for energy therefore involve not only all the declared energy flows but also the energy flows associated with all the inter–vessel mass streams and by the transfer between interacting phases within a vessel (step 2 and 3).

4.3.1.6 Phase Selection Rule

As stated earlier, in order for a mathematical model to encompass physical discontinuities such as the appearance and absence of phases present in a vessel and the discontinuities resulting from a vessel geometry as well as to associate the stream passing through ports with a single phase or an aggregate, we introduce the rule for selecting a relevant phase or an aggregate passing through a port. This rule will therefore be used as a basis on formulating the algorithms for dealing with physical discontinuities. That is, by this rule a fixed set of discrete transitions are identified and each state can then be described.

The phase selection rule is categorised into two classes based on the type of port (exit or entry). In fact the basic principle of this rule has been introduced in §3.1.4, however the complete formal statement will be made in this section. Firstly, we consider the rule for exit port, which is concerned with the rule for selecting a phase or an aggregation of phases flowing out of the port. Since what flows out of the port is determined by the layer covering the outlet, the phase or aggregation of phases in contact with the port is selected as the outlet stream. By applying this rule to each element of aggregation list specified in the "Aggregation" statement, sets of all possible discrete transitions are identified and each state is then described. If a phase or an aggregation of phases associated with the port is independently specified by a user for selectivity, allowing passage of only the specified phase or aggregation through the port, it is selected as the outlet stream passing through the port. The formal statement of the phase selection rule for an exit port is given in rule 4.1 below.

Rule 4.1 PHASE SELECTION RULE FOR AN EXIT PORT

Select the phase or aggregate covering the exit port as the outlet stream passing through the outlet unless specified differently in "Port" statement.

Now consider the phase selection rule for an entry port which allows association of an inlet phase with a phase in a vessel. As introduced in §3.1.4, the inlet stream association rule is classified into the two categories depending on whether the inlet stream is an aggregation or a single phase. We can get this information about the upstream from the application of the phase selection rule for the upstream port (exit port). This information will be used as a basis for determining the association of an inlet stream with a phase in a vessel. Of course, if a phase or an aggregate in a vessel is explicitly specified by the user as the entry port attributes, it will be taken as the inlet stream association without applying the inlet stream association rule introduced in §3.1.4. The formal statement of the phase selection rule for an entry port is given in the rule 4.2.

Rule 4.2 PHASE SELECTION RULE FOR AN ENTRY PORT

What enters through the entry port is determined by the rule in §3.1.4 unless specified differently in the "Port" statement.

4.3.1.7 Vessel Discontinuities

As stated in §4.2, in order to deal with the mechanisms of a large number of discontinuous phenomena in a consistent and systematic manner, a sophisticated mathematical modelling technique was adopted (Pantelides, 1995), decomposing a mathematical model into two main groups: case invariant group and a set of physical discontinuities, each characterised independently by its own state–transition. Based on this formalism, the method for handling physical discontinuities will now be introduced in detail.

The required physical discontinuities are specified in such a way that based on the phase selection rule defined in §4.3.1.6, a set of possible discrete states for a given vessel are identified and each state is then described in terms of a set of relevant variables.

As emphasised at the beginning of §4.3, the hierarchy of the conceptual elements of the physical modelling language, as shown in figure 3.25, will be maintained throughout the whole model generation procedure. As a consequence the method of dealing with physical discontinuities is based on this hierarchical structure. From this hierarchical point of view physical discontinuities are categorised into two classes; a vessel and connection class. In the vessel class the specification of a set of physical discontinuities is concerned with the selection of the phase or aggregate passing through a port, including

the appearance and disappearance of phases present in a vessel, on the other hand in the connection class it is concerned with the selection of the relevant port of the two mutually connected vessels. The latter is in fact carried out in the course of invoking appropriate inter–vessel transfer law libraries, the details of which will be presented at §4.3.3. The formal procedure for the specification of physical discontinuities in a vessel is given in algorithm 4.1.6 below.

Algorithm 4.1.6 SPECIFY DISCONTINUITIES IN A VESSEL

If the port type is *reversible* then

1. Identify discrete cases in terms of flow directionality, each identified in accordance with PHASE SELECTION RULE.
2. Specify discrete states in terms of *relevant variables.*

Else

1. Identify discrete cases in accordance with PHASE SELECTION RULE.
2. Specify discrete states in terms of *relevant variables*

Recall that a type of a mass stream port has the following three categories; an entry, exit and reversible port, as demonstrated in §3.1.4. The above algorithm is mainly decomposed into two cases depending on whether a port type is reversible or not. In the case of the port allowing a reversible flow, firstly the pair of transitions between flow directions (forward and reverse) are identified, and then each transition is again decomposed into a subset of the transitions resulting from the identification of a currently active phase or an aggregate passing through the reversible port. In the other cases (an entry or exit port), a set of transitions are identified in terms of a phase or an aggregate participating in the flow through the port. In order to identify these discrete cases, we suggest the rule for both selecting a phase or an aggregate through an exit port and associating an inlet stream with a phase or an aggregate in a vessel. The details of this rule will be illustrated in rule 4.1. Let us assume that a fixed set of discrete cases for a port have been identified by the application of rule 4.1. In accordance with the identified discrete cases, an appropriate set of equations is generated corresponding to each discrete state and if necessary, the condition for a set of transitions to the other states.

The basic method for specifying discrete states in mathematical terms will now be introduced. Once a set of states are identified by applying phase selection rule, the specification of a phase or aggregate passing through a port is mathematically formulated in such a way that a set of relevant variables is equated with the *corresponding variables* of the port. In order to maintain the same set of the variables across a given set of discrete cases, a set of the corresponding variables for a set of phases not passing through the port should be made dummy by assigning numeric value, "0" to this set of the variables. Note that *relevant variables* being able to be used here are categorised into the following three classes:

- state–variables characterising the thermodynamic state of a phase or an aggregation of phases (e.g. viscosity, density, enthalpy, mass fraction and etc.)
- some of the attributes of a mass stream (e.g. mass rate, enthalpy flow, dispersion ratio and phase type of the port)
- port type variables in the case of a reversible port (e.g. outlet and inlet).

From the variables categorised in the above list those that are used to specify discrete states depend on the type of a port. First, in the case of an exit port, (as stated earlier) what flows out through the port depends on what phase or aggregate is covering the port. We therefore need to specify the discrete states in terms of all the attributes of the mass stream for the port as well as the appropriate set of state–variables. In the course of invoking from a library a set of equations representing the inter–vessel transfer law, these state–variables are determined by those contained in the library, which will be demonstrated in §4.3.3.

In the case of an entry port, suppose that the discrete cases resulting from the association of the inlet stream with a phase in a vessel has been identified by applying the phase selection rule for the entry port. The set of variables required to specify each discrete state are mass rate and enthalpy flow.

Finally in the case of a reversible port, as already noted, it is decomposed into two cases in terms of the flow directionality, and each is then identified by applying the appropriate phase selection rule. The flow directionality determines the type of the reversible port (an exit or entry port). This enables us to apply the appropriate method for specifying discrete states according to the flow directionality. That is, in the case of forward flow we apply the method for specifying discrete states for an exit port, in the other case (reverse flow) apply the method for an entry port. In the case of forward flow,

as stated above, we need to specify discrete states in terms of all the attributes of the mass stream for an exit port.

However, as far as the specification of discrete states in terms of the dispersion ratio and phase type of the reversible mass stream attributes is concerned, the two vessel identifiers mutually connected are needed to describe the logical expression of the condition for each state. This will be dealt with in §4.3.3.

As introduced in §3.1.4, a user could provide for selectivity (e.g. via a filter or membrane) allowing passages of only specified phase or aggregate. In this case, instead of specifying discrete states, we describe the fixed state in terms of a set of relevant variables of the explicitly specified phase or aggregate.

Thus far we have demonstrated how to deal with a set of physical discontinuities through ports in a vessel. By the completion of algorithm 4.1.6, it is possible to generate an appropriate set of discontinuous equations through the explicit specifications of the *relevant variables*. It should be recognised that all possible physical discontinuities likely to arise through ports have been incorporated into a set of *case variant* equations in the mathematical model. Consequently these discontinuities encompass transitions resulting from vessel geometry (port positions). However, the discontinuities embedded intrinsically in a transfer law itself are beyond the scope of algorithm 4.1.6. For instance, the relationship of phase equilibrium where the transitions among the states of only sub–cooled liquid, only super–heated vapour and coexistence of two phases, and pipe flow containing the transition between flow regimes (e.g. laminar and turbulent). The transfer law library should be constructed for these discontinuities to be embedded in it, the details of which will be illustrated in appendix C.

In the next section the procedures for generating mathematical equations from the physical description of the geometry of a vessel will be introduced.

4.3.1.8 Geometry Section Sub–Model

This section is concerned with the generation of the mathematical sub– model from the physical description of the geometry of a vessel. The generation procedure is decomposed into the generation procedures for a set of subsections of the geometry such as the section of shape, dimension and port position. Recall that as far as the orientation section is concerned, only the "vertical" orientation is available in the present package due to its simplicity. Consequently there is no generation procedure for the orientation section. The master procedure for a geometry section model is given in algorithm 4.1.7 below.

Algorithm 4.1.7 GENERATE GEOMETRY SECTION SUB–MODEL

1. For shape section,
   (a) Apply vessel volume constraint.
   (b) For each element of the aggregation list,
      i. Generate an equation describing its volume.
2. For dimension section,
   (a) Declare dimension variables as real type parameters.
   (b) Set all numerically specified values to the dimension variables.
3. For port position section,
   (a) Declare port position variables as real type parameters.
   (b) Set all numerically specified values to the port position variables.
   (c) If a port position is specified by a dimension attribute then
      i. Equalise the port position variable to the dimension variable.

Recall the basic premise that all the phases present in a vessel must fill the vessel. This is the case for a closed vessel the volume of which is fixed. The volume of a closed vessel therefore is equal to the summation of the volumes of all phases present in the vessel (step 1.a). Even the volume of an open vessel in fact cannot be varied. The difference between open and closed vessels is the fact that since the volumes of phases present in the open vessel may be varied and the remaining space is taken up by the atmosphere, which is the part of the environment, the vessel content need not fill an open vessel, but the vessel cannot be over–filled. We therefore assume that the remaining space is taken up by the pseudo–volume, namely `Vol_Empty` in order to relate the vessel volume to the volumes of the phases present in an open vessel, and then this volume is included into the volume constraint of the vessel. That is, the open vessel volume is equal to the summation of phases present plus the pseudo–volume (step 1.a).

The equations describing the volume occupied by each element of the aggregation list are appropriately generated according to the shape of the vessel (step 1.b.i). For an open vessel, the equation describing the pseudo–volume is generated additionally.

The attributes of the geometrical dimension of a vessel are declared as the geometrical design parameters which are time invariant quantities, as opposed to the variables of equations governing dynamic behaviour (step 2).

Similar to dimension attributes, the attributes of port positions are declared as time invariant design parameters if specified by numeric value (step 3.a and 3.b). However, if the attribute of a port position was specified by a dimension attribute (e.g. height, diameter for cylinder) instead of a numeric value, the equation equalising the port position attribute and the dimension attribute is generated (step 3.c.i).

Thus far we have described the procedures for generating the mathematical model of a vessel (algorithm 4.1). The remaining sections will be concerned with the procedure for generating mathematical model from the information encapsulated into a reservoir and connection entity.

4.3.2 Reservoir Sub–Model Generation

This section describes how to generate the mathematical model of a reservoir from the physical information declared in a *reservoir* entity.

As illustrated in §3.2, the package can deduce which compounds may be present in the phases within a vessel from the connection arrows and declared reactions and transfer laws and the list of compounds specified in each source reservoir (described in the reservoir entity). As the language provision for specifying a list of compounds in each reservoir is not available in the present package, this section does not include dealing with the identities of compounds involved.

Therefore only the port section in a reservoir entity is specified since all the requisite state–variables in each reservoir are automatically detected from the information about the inter–vessel transfer law in library whether it is a source or sink (see §4.3.3). The procedure for the generation of the mathematical models of a reservoir is given in algorithm 4.2 below.

Algorithm 4.2 GENERATE RESERVOIR SUB–MODEL

1. For each mass stream port,
   (a) Create a set of the mass stream attributes; mass rate, enthalpy flow, dispersion ratio and phase type.
   (b) Build the mass stream instance composed of the set of the above attributes.

2. For each energy port,

   (a) Create the enthalpy flow attribute.

   (b) Build an energy stream instance which has the above attribute.

This algorithm is mainly decomposed into two parts according to whether mass or energy flows through a port of a reservoir, which the package automatically can deduce from the information about the port of the vessel connected to the reservoir.

As already noted in algorithm 4.1.4, the currently active status of any material flowing through a mass stream port is represented by a set of the mass stream attributes; its mass rate, enthalpy flow, ratio and phase type (step 1).

Energy flow through an energy port is characterised by the enthalpy flow, since this is its only attribute (step 2).

In the next section the detailed procedures for the generation of the mathematical model from the physical description of a connection entity is described.

4.3.3 Connection Sub–Model Generation

This section deals with the generation of the mathematical sub–model from the information about physical description of a connection entity. The procedures for generating the mathematical sub–model from a connection entity are given in algorithm 4.3 below.

Algorithm 4.3 GENERATE CONNECTION SUB–MODEL

1. Deduce the connectivities from the CONNECTION statement.

2. Invoke appropriate transfer laws from library.

3. If *reversible* mass flow then

   (a) SPECIFY CONNECTION DISCONTINUITIES. (see **algorithm 4.3.1**)

Recall that the physical description of an inter–vessel connection attribute consists of a pair of ports from two vessels (one of the two vessels may be a reservoir), the connection arrow to specify the flow directionality, and the inter–vessel transfer law.

From the information about the pair of the ports, the topological inter–vessel connections are deduced in terms of the relevant stream instances (step 1). The completion of this step results in whole process flow-sheeting for a given process system.

As the inter–vessel transfer law is specified by a user, the set of transfer law equations will be invoked from the library (step 2). This procedure in fact proceeds to transform the set of the transfer law equations in the library table into an appropriate form by considering the pair of the ports through which the transfer is occurring, after having searched the specified transfer law to check whether or not it has been installed in the library table (step 2). As mentioned in algorithm 4.1.6, in the course of invoking a transfer law the set of state–variables embedded in the transfer law are identified and then discrete states arising through the port of the source vessel in terms of the set of state–variables.

As mentioned in §4.3.1.7, if the directionality of the inter–vessel flow is reversible, it will be impossible to specify discrete states in terms of the dispersion ratio and phase type of the attributes of the mass stream passing through the reversible port within the scope of a vessel due to the impossibility of accessing vessels. These discrete states therefore are specified at connection entity (step 3.a). The method for this specification is that discrete cases are identified in terms of the flow directionality for a reversible connection and each is identified by applying the phase selection rule described in §4.3.1.6 and then the state for each identified case is described in terms of the two attributes of the mass stream flowing through the reversible connection; its dispersion ratio and phase type. The formal procedure for specifying connection discontinuities is given in algorithm 4.3.1 below.

Algorithm 4.3.1 SPECIFY DISCONTINUITIES IN CONNECTION

1. Identify discrete cases in terms of flow directionality, each identified in accordance with PHASE SELECTION RULE.
2. Specify discrete states in terms of the two mass stream attributes; dispersion ratio and phase type.

Thus far the algorithms required to formulate mathematical models from physical description represented in the physical modelling language for a given process system

have been presented in detail. In the end the automatically formulated mathematical models encompass the discontinuous behaviour resulting from the discontinuities arising routinely in process systems, which includes the appearance and disappearance of phases, the flow regime transition, the discontinuities from vessel geometry and other factors.

This concludes the algorithms required to generate the mathematical models. The remaining section describe the conventions for notation used in mathematical models.

4.4 Notation used in Mathematical Models

Before implementing the algorithms for generating mathematical models, a consistent method of notation should be identified. This section describes the notation used in the present implementation. The Notation described in this section is limited to that of variables and parameters comprising equations in mathematical models. As *gPROMS* language is the form of the generated mathematical models, the other symbols required to formulate a set of equations are based on the syntax of the *gPROMS* input language, which includes logical expressions, arithmetic operators and a set of built–in functions.

In order to ensure a consistent notation, we take account of the symbols commonly used in physical descriptions and for physical properties. There are several methods for formulating the notation. We note that the most significant aspect of notation is the legibility to users, prompting to adopt the names of physical properties and symbols in physical description literally. On the other hand we must control the length of a notation made in this way. Too long a notation may inhibit its legibility in itself and even the ease of understanding. These aspects lead to a balance between the literal adoption and compactness, depending on a specific notation. The standard notation used in this package is given in table 4.1, the lower part of which represents parameters used in mathematical models, and non–standard notation given in table 4.2.

As stated earlier, all variables and parameters used in mathematical models should be unique within each vessel, reservoir and connection. To be so, a notation, in most cases, is made from the symbol of physical property followed by a additional identifier taken from a name of phase, aggregation, port, vessel, reservoir or connection as described in the physical process description. In addition to these, an under–bar, "_" is inserted for the ease of readability between the symbol of the physical property and the supplementary identifier. Examples are `Mass_V`, `Enth_Po1`, `Vol_FlashDrum` and `Velocity_C1`, where `V`, `Po1`, `FlashDrum` and `C1` denote identifiers of a phase, port, vessel and connection respectively. The under–bar is also employed as a delimiter in various cases. It should

standard items	variables/parameters	types
mass	Mass	Array(NoComp) of Mass
mole	Mole	Array(NoComp) of Mole
mass fraction	MassFrac	Array(NoComp) of Fraction
mole fraction	MoleFrac	Array(NoComp) of Fraction
mass flow rate	Rate	Array(NoComp) of Mass_Rate
velocity	Velocity	Array(NoComp) of Velocity
enthalpy	Enth	Enthalpy
internal energy	IntEnergy	Int_Energy
enthalpy flow	EnthFlow	Enthalpy_Flow
entropy	Entropy	Entropy
pressure	Press	Pressure
temperature	Temp	Temperature
volume	Vol	Volume
density	Den	Density
viscosity	Viscosity	Viscosity
ratio	Ratio	Array(NoComp) of Fraction
total ratio	TotalRatio	Fraction
phase type	PhaseType	Positive
molecular weight	MoleWeight	Array(NoComp) of Positive
equilibrium constant	EquilConst	Array(NoComp) of Positive
phase bounds	Top, Bot	Positive
level	Level	Positive
area	area	Positive
number of components	NoComp	INTEGER
port position	Z	REAL
diameter	diameter	REAL
height	height	REAL
length	length	REAL
width	width	REAL
constant	Const	REAL
port type	OUTLET, INLET	INTEGER

Table 4.1: Standard Notation Table

non–standard items	variables/parameters	types
dew temperature	DewTemp	Temperature
bubble temperature	BubTemp	Temperature
driving force	DrivingForce	NoType
reynolds number	ReynoldsNo	Positive
reynolds constant	ReynoldsConst	Positive
...	...	...

Table 4.2: Non–Standard Notation Table

be noted that users are banned from using an under–bar in the physical description of a process system.

4.5 Summary

This chapter was concerned with the generation of mathematical models exhibiting discontinuous behaviour potentially arising in a given system, from the purely physical description represented by the proposed language.

The basic strategy for formulating mathematical models was established and then the consistent and general mathematical formalism required to encompass the physical discontinuities in an "easy" and "natural" manner was identified on the basis of the *gPROMS* model structuring concept. Based on this model generation strategy and generic formalism, the complete algorithms for generating mathematical models from the physical description written in the physical modelling language designed in the previous chapter were developed. Finally, in the last section, the notation used in the mathematical models was discussed in detail.

As the methodology for model generation has now been developed the computer implementation of it will be discussed in the next chapter.

Chapter 5

Implementation

In the previous two chapters a language supporting the formal physical description of process systems and the methodology for formulating mathematical models from this physical description have been presented.

This chapter is concerned with the implementation of the package for automatically generating mathematical models, based on the model formulation methodology presented in chapter 4. The current version of this package has been implemented in C programming language (Kernighan and Richie, 1988) on UNIX system. It begins with an overview of the model generation package in terms of the software architecture. The implementation of each of architectural sub–systems will then be discussed in detail in the subsequent sections, including the transfer law library and the two symbol tables for the internal storage of information about the physical description and mathematical models.

5.1 Software Architecture

The model generation package is composed of three major sub–systems: the *translator*, the *model generation engine* and the *code generator*. In addition there are the library containing appropriate transfer laws and the two symbol tables for the internal dynamic storages of the physical description and mathematical models: *physical* and *mathematical symbol table*. A schematic diagram of the software architecture of the package is given in figure 5.1.

In order to store the information about physical description in an appropriate data structure, we need to set up the physical symbol table, which is internally represented in the same hierarchical structure as shown in that of figure 3.25. Prior to building the physical symbol table, the translator takes an input file containing the physical description

and then checks the syntax and semantics of the language. If the input file is successfully translated, the translator creates the physical symbol table, which is dynamically allocated in memory, in which it stores all the information contained in the input file. The physical symbol table is used by the model generation engine to formulate mathematical models, based on the model generation algorithms developed in the previous chapter. In the course of the model formulation process the model generation engine may invoke transfer laws which have already been installed in the library. All the mathematical information generated by the model generation engine is encapsulated in the mathematical symbol table which uses the same structured hierarchy as shown in figure 3.25. Finally the mathematical symbol table is employed by the code generator to convert the internal mathematical representation into the form of a specific simulator input language.

5.2 The Translator

The translator performs two main tasks:

- checks whether or not the physical description coded in the physical modelling language is correct in terms of its syntax and semantics.
- stores the information about the physical description into the physical symbol table for later use by the model generation engine.

In order to complete these tasks the translator is decomposed into the three structural sub–systems; the *scanner*, the *parser* and the *semantic routines* as shown 5.1. These sub–systems and the physical symbol table are discussed in the following sections.

5.2.1 The Scanner

The scanner reads the input character stream and identifies the *tokens* which include keywords, operators, identifiers, constants, literal strings and punctuation symbols such as parenthesis, commas and semicolons. The tokens are passed to the parser for syntax analysis.

Although the scanner is the simplest sub–system of the translator enough to implement in any programming language, the scanner of this package has been easily implemented by *lex* (Levine *et al.*, 1995) (installed on the UNIX operating system) which is a tool for automatically generating the scanner in C code from a specification based on regular expressions. The *lex specification* consists of the declarations section (which sets up the execution environments in terms of variables, manifest constants, and regular

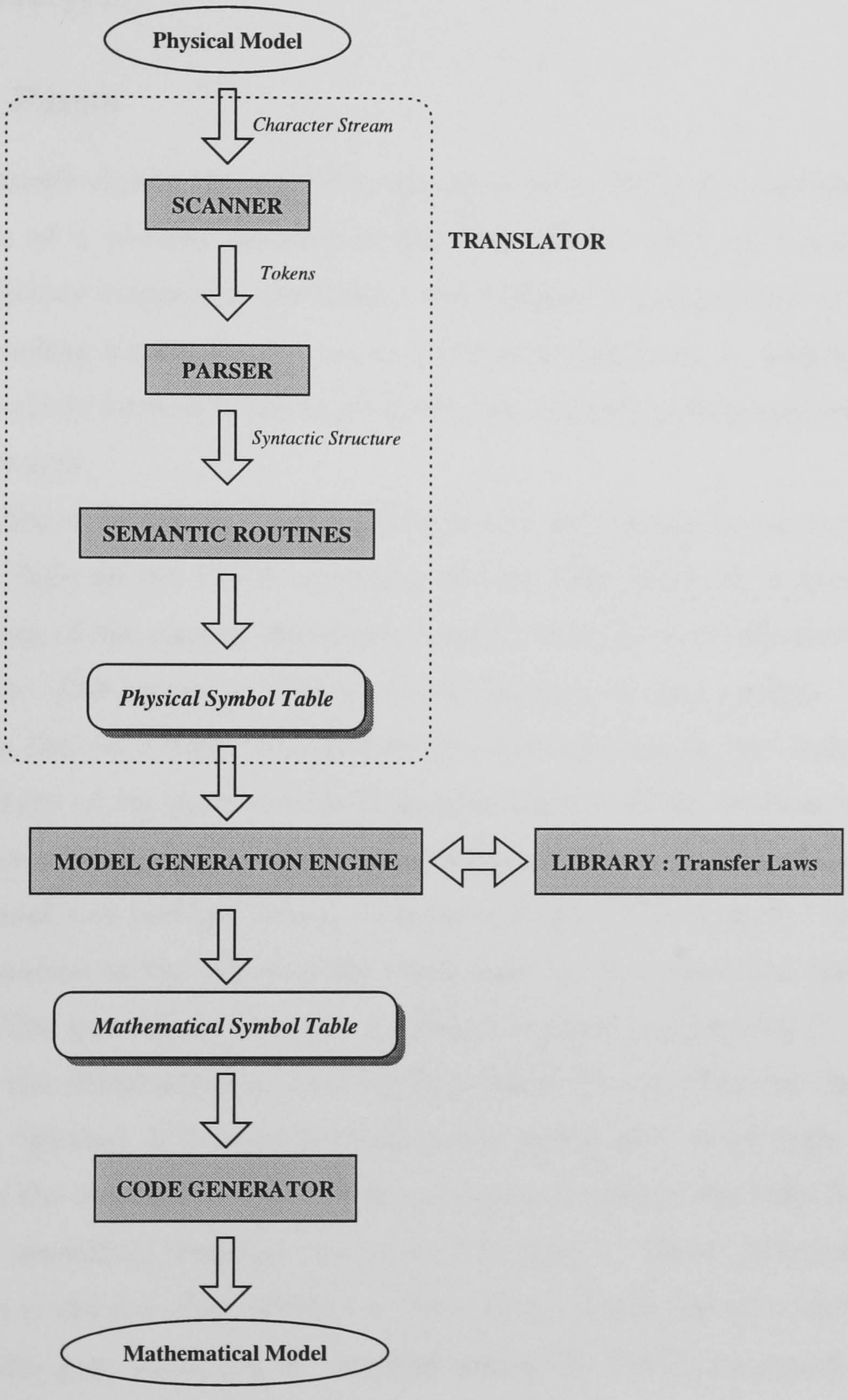

Figure 5.1: The Software Architecture of the Prototype Package

definitions), the translation rules section, (which includes a list of regular expressions composed of the patterns and actions) and C supporting–subroutines section (there is no subroutine in the present package). The *lex* input file for the this package (appendix A) is compiled by using the UNIX command `lex` with the option `ll` to get the scanner coded in C (namely `lex.yy.c`).

5.2.2 The Parser

The parser checks the syntactic structure of an input file representing the physical description of a process systems in the form of the physical modelling language. Whenever the parser requires a new token, the scanner is invoked and returns the token as the corresponding integer to the parser. The returned token is used by the parser to check if it is correctly formed in terms of the syntax and the appropriate semantic routines are then undertaken.

In addition to the *lex* utility, there is also an automatic parser generator *yacc* (Levine *et al.*, 1995) on the UNIX operating system. The adoption of *yacc* has led to the rapid prototyping of the parser, otherwise, considerable efforts would have been required to implement it. The parser is thus generated by *yacc* in this package. To do this we need to specify the *yacc* input which is in the form of regular expressions. Similar to the basic structure of *lex*, yacc specification is composed of the declarations section, the translation rules section and C supporting–subroutines section. The declarations section has three optional sub–sections which includes a literal C block code enclosed in `%{ %}` lines, the declaration of the types of the stack used in the parser and the definition of a list of tokens. The *yacc* input file in this package is given in appendix B, where the first two options of the declarations section corresponds to the inclusion of the global header file (`#include "global.h"`) and the declaration of the *yacc* stack type (`union{...}`), respectively. In the translation rules section consists of a list of the rules for the grammar of the physical modelling language discussed in chapter 3. The C subroutine specified in the last section is the function calling *lex* (`lex.yy.c`). Once the *yacc* input specification is completed, the *yacc* input file is compiled using the UNIX command `yacc` with the option `ly` to get the parser (namely `y.tab.c`).

5.2.3 The Semantic Routines

Adoption of *yacc* has the consequence that coding of the semantic routines represents most of the effort required in the implementation of the translator. The semantic routines undertake two main tasks as follows:

- checks that the semantics of the physical description contained in the input file are valid.

- creates the *physical symbol table*, the appropriate internal data structure that will store the information about this physical description.

The details of the semantics of the language for physical description have been presented in the chapter 3 and the following section deals with the data structure of the *physical symbol table*.

5.2.3.1 The Physical Symbol Table

As stated in chapter 3, there are three primitive physical process entities (vessel, reservoir and connection). The physical symbol table is thus categorised into three classes of tables corresponding to the three primitive physical process entities, i.e. the vessel, reservoir and connection tables. The physical symbol table for each entity is structured in the form of hash tables consisting of a fixed array, each element pointing to an entity instance currently created in memory. Again a set of attributes declared in each section belonging to an entity is structured into the corresponding hash table[1]. The hash table structure enables to speed up a search of the table, which will be used for the model generation engine to construct the mathematical symbol table.

The detailed procedure for constructing the physical symbol table involves the following:

1. create the space available to each attribute in memory by use of the dynamic storage allocation facilities of the programming language.

2. store the attribute information into the space in memory.

3. insert the entry into the table if it is not already present.

5.3 The Model Generation Engine

The model generation engine embodies the algorithms presented in chapter 4. That is, the model generation engine transforms the information encapsulated in the physical symbol table into a mathematical model by applying the generation algorithms and then stores the new information into the mathematical symbol table. In the course

[1]The structure of geometry attributes declared in geometry section of a vessel entity is not based on the hash table but based on a linked list because there is only one geometry instance for the vessel entity.

of this process, transfer laws in the library are invoked as required and then converted into an appropriate set of equations.

To do this, the first task of the model generation engine is to design the abstract data type for the structure of the mathematical symbol table. The data type is basically composed of a set of complex combinations between linked lists and binary trees. Recall that the structure of a mathematical model is mainly decomposed into a *case invariant* group and a set of *physical discontinuities* to encompass the combined discrete/continuous behaviour of a given process system. The linked list data type is employed for the case invariant group, whilst, the data type composed of the combined linked lists and binary trees is employed for storing mathematical sub–model representing the discontinuities.

Based on a well designed abstract data type for the mathematical symbol table, all activities performed by the model generation engine, like those of the semantic routines, are concerned with handling the data structure, which includes the dynamic creation of an abstract data type, the intermediate storage of the information into the structure and its insertion into the mathematical symbol table. This is then used by the code generator to construct the mathematical model which is represented in a specific simulation input language. The details of internal data structure for the mathematical symbol table will now be presented.

5.3.1 The Mathematical Symbol Table

The internal representation of the mathematical symbol table is structured into the hierarchy as shown figure 3.25, in terms of the three primitive physical entities. The internal hierarchical structure of the mathematical symbol table is shown in figure 5.2. The table is decomposed into the three hierarchical levels, each corresponding to the sub–model (vessel, reservoir or connection sub–model) formulated from the corresponding physical process entity. The two dotted lines in figure 5.2 delimit the hierarchical levels of the three sub–models, each of which are composed of a set of abstract data sub–structures based on combinations of the data types of linked lists and/or binary trees. The declaration section in each sub–model represents a set containing the parameter, variable, set and selector sections. It should be noted that, as stated in §4.2, the `equation` section in the vessel and connection sub–model are decomposed into *case_invariant* and *discontinuity* sections.

Each sub–model is structured in the form of its own hash table. The following set of attributes are also structured in the form of hash tables:

- parameter, variable and set attributes in each sub–model

- stream attributes in both the vessel sub–model and the reservoir sub–model stream attributes
- unit attributes in the connection sub–model

The internal structure used to store the *discontinuity* section in the connection and vessel sub–models are a set of abstract data types mainly built by the combination of linked lists and/or binary trees. The internal structure for the other sections are based on the derivatives of linked lists.

5.4 The Transfer Law Library

When a transfer law is invoked, its availability is checked by searching for it by name in the transfer law library table. The set of equations comprising the transfer law is then transformed into an appropriate form by passing the set of phases participating in the transfer law as the parameters of this invoking procedure.

The data type required to represent a transfer law in the library is structured into a set containing its name, the type of flow (mass or energy flow) and a fixed set of equations providing a mathematical description of the transfer. As discussed in §4.3.1.7, the set of equations comprising a transfer law are categorised into two main classes: *case invariant group* and a set of *physical discontinuities*. The data types required to install the equations representing a transfer law are thus based on combinations of linked lists and/or binary trees. In order to improve the speed of searching for a transfer law in the library table, the transfer law instances are installed into a hash table structure.

Also included in the library are a set of functions which manage the construction and installation of relevant variables and equations required to represent a transfer law, thus allowing easy maintenance and updating of the library. The task of installing a new transfer law is thus comparatively simple.

5.5 The Code Generator

Once the task of the model generation engine is completed by storing all the information about a mathematical model into the mathematical model symbol table, the final model generation procedure is undertaken by the code generator, which converts the information stored in the mathematical symbol table into a specific target language. In the present package the target language is the *gPROMS* (Barton, 1992) input language.

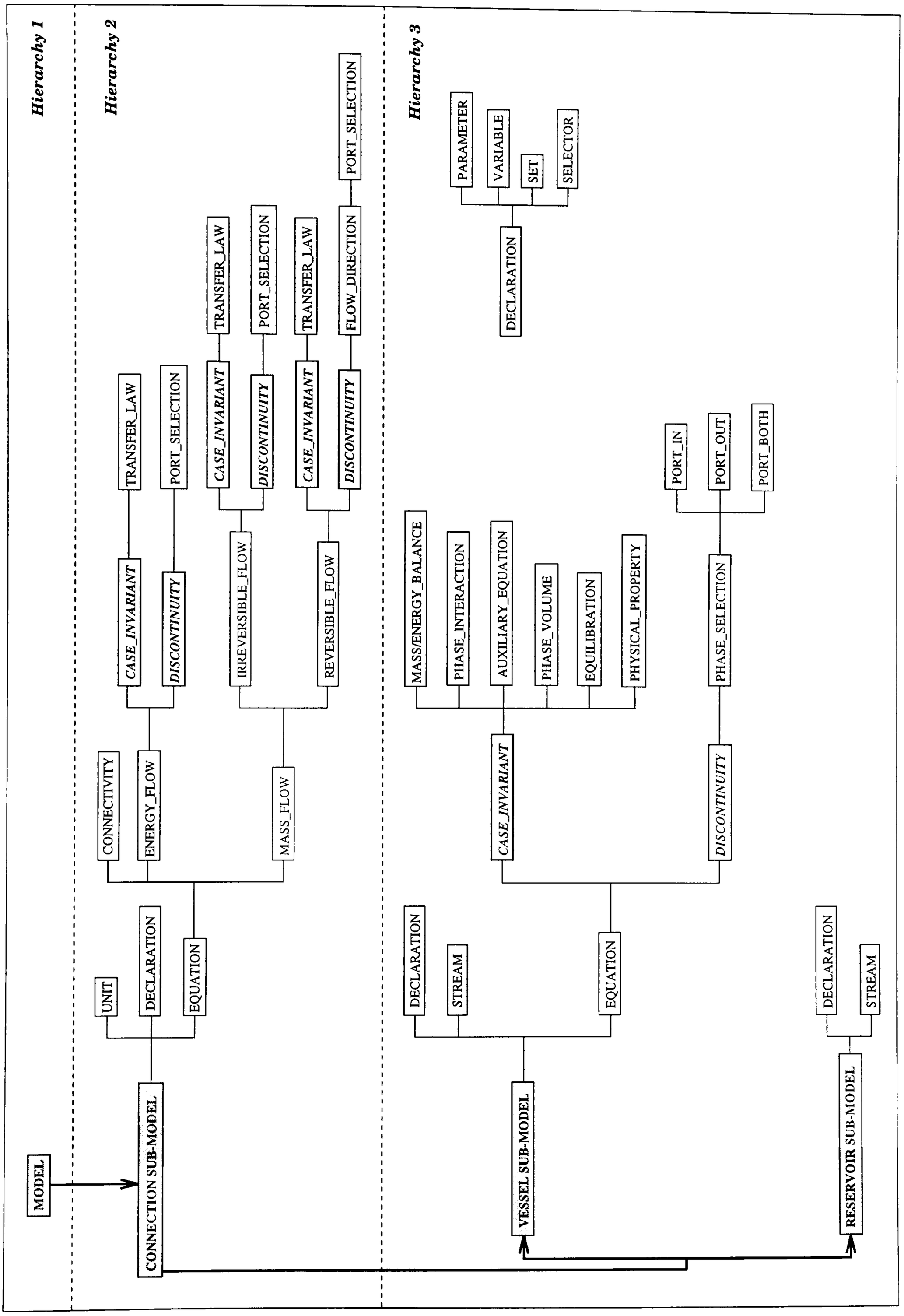

Figure 5.2: The Internal Hierarchical Structure of the Mathematical Symbol Table

From the point of view of programming, the conversion procedures simply utilise formatting functions to fit the output to the form of target language and hence the implementation is in itself not a complex job. The task of updating the package to formulate a mathematical model in the form of a new target language therefore only requires the additional implementation in the code generator, rather than upgrading the whole model generation engine. This has a consequence that the model generation package provides the potential of multi–functionalities for generating different forms of mathematical models as required.

5.6 Summary

Based on the model generation algorithms presented in §4.3, the implementation of the current version of the package has been discussed in this chapter. The software architecture of the model generation package is similar to that of a compiler which is a program that reads a program written in one language – the *source* language – and translates it into an equivalent program in another language – the *target* language (which is usually a machine code) (Aho *et al.*, 1986). The process of translating a program written in a source language (the physical description) into a target language (the simulation input language) is decomposed into two stages; the intermediate storage of the information about the mathematical models formulated by the model generation engine into the mathematical symbol table (instead of direct generation of the mathematical models from the information stored in the physical symbol table) and the conversion of the information encapsulated in the mathematical symbol table into a specific target language.

The decoupling of the model generation process into the two stages provides a potential for the versatility of the code generator in this package, that is, it is possible to easily improve the code generator to generate mathematical models in the various forms of simulator input language (e.g. *gPROMS*, *SpeedUp*, etc.). Furthermore, it has a potential of enhancing the compatibility of this package which means the ease with which the package may be interfaced with other modelling and simulation environments.

Inevitably the language designed for the physical description and facilities evolve as more complex processes and a wider range of applications are considered. Due to the adoption of the automatic parser generator (*yacc*), changes to the syntax of the language require only that the *yacc* input specification be updated. This process does not affect the semantic routines since the process of generating the parser by use of *yacc* is completely independent to the semantic routines. This has a consequence that the task of upgrading

the package resulting from the evolution of the language syntax is straightforward, which means high extendibility of software products.

Chapter 6

Simulation Examples

This chapter describes the simulation results of the examples introduced in §3.4.

6.1 Flash Drum

Here we consider the flash drum described in §3.4.1. Each phase is composed of two components; propane and butane. The operating and intial conditions used in this simulation are as follows:

Operating Conditions

- source reservoir, R1

Pressure	=	4x1.013E2	kPa
Density	=	330	kg/m3
Mass Fraction	=	[0.4,0.6]	
Enthalpy	=	3.5E2	kJ/kg
Dispersion Ratio	=	0.001	
PhaseType	=	3	(aggregation type)
Viscosity	=	5E-5	Pa.s

- sink reservoir, R2 and R3

Pressure	=	1.013E2	kPa

Initial Conditions

Temperature of vapour	=	300	K
Mass of each component in separate vapour	=	[1,1]	kg
Mass of each component in aggregation	=	[15,15]	kg
dispersion ratio	=	0.001	

Figure 6.1 and 6.2 show the change with time of the total mass holdup of each phase and the horizontal mixture level of the aggregate respectively.

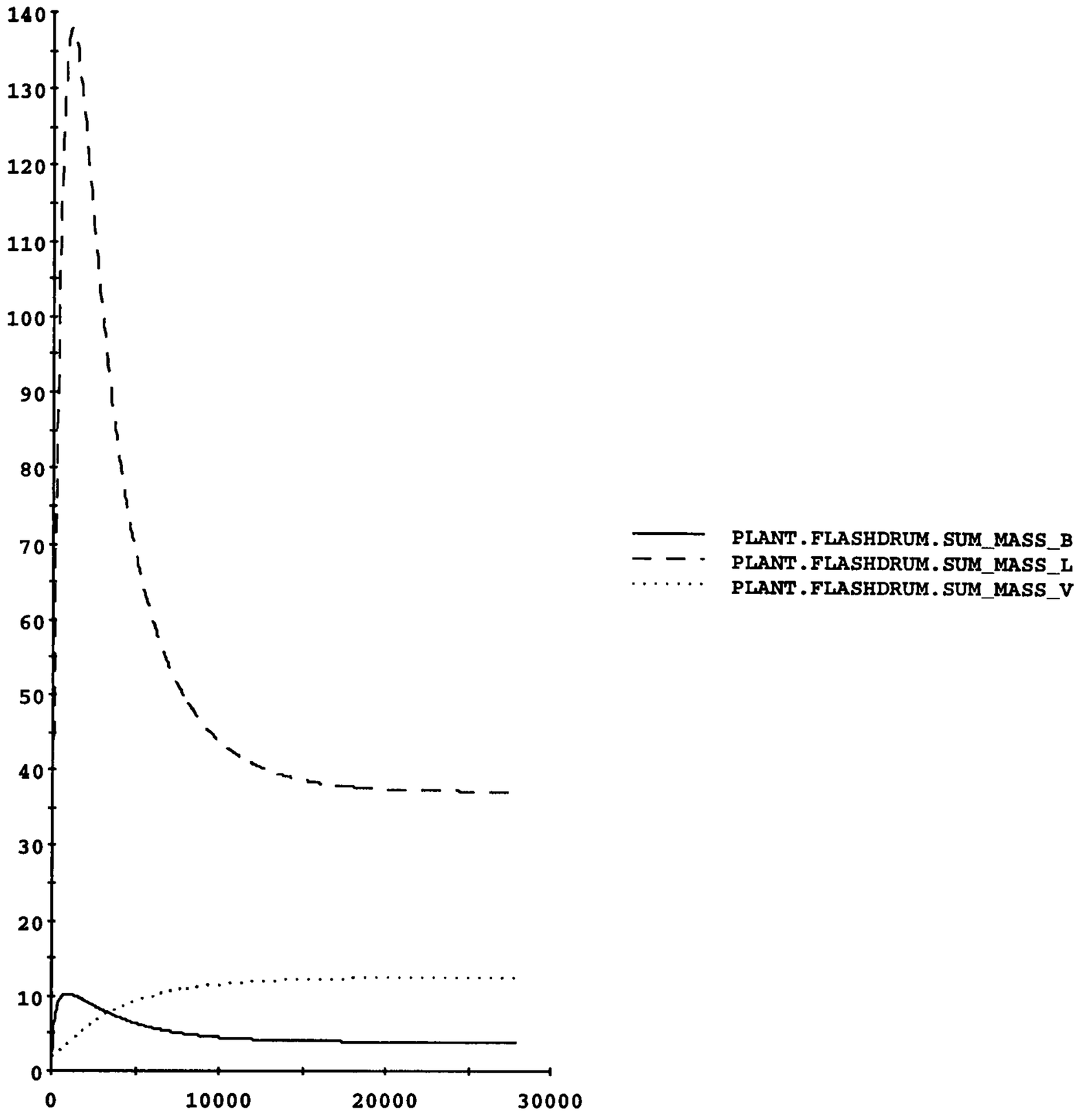

Figure 6.1: Holdup variation in flash drum

The connection between the flash drum and reservoir is defined to be irreversible flow (i.e. through the use of a pipe installed with a non–return valve). Since the pressure of the flash drum is initially lower that that of reservoir R2 there is no flow through the connection and the pressure of the flash drum varies with time. When the driving force (the pressure difference between the flash drum and reservoir) becomes positive (at t $\cong$ 56 sec.), the non–return valve opens and flow commences out of the flash drum through port Po1. Figure 6.3 and 6.4 shows this discontinuous behaviour.

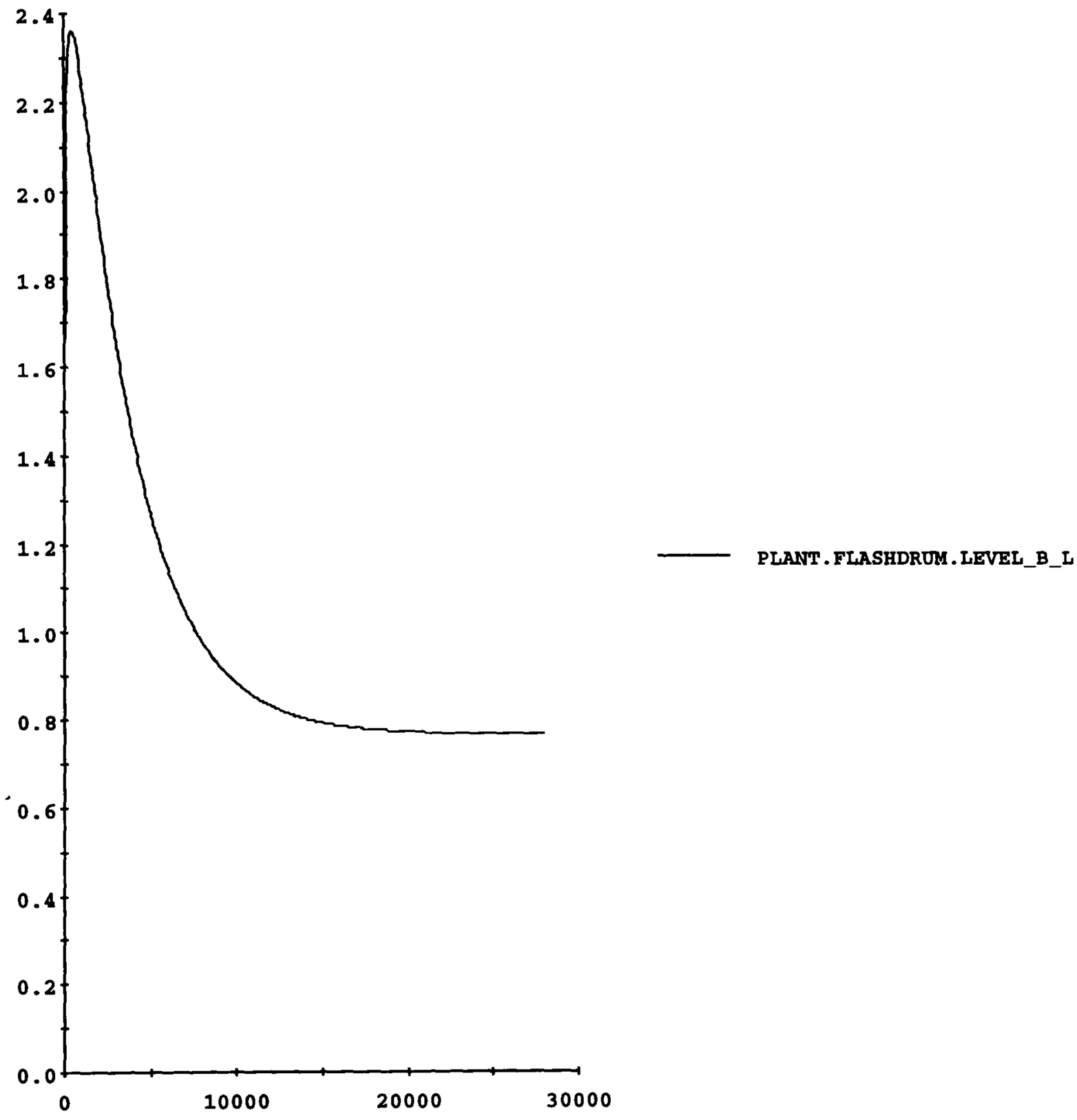

Figure 6.2: Level variation in flash drum

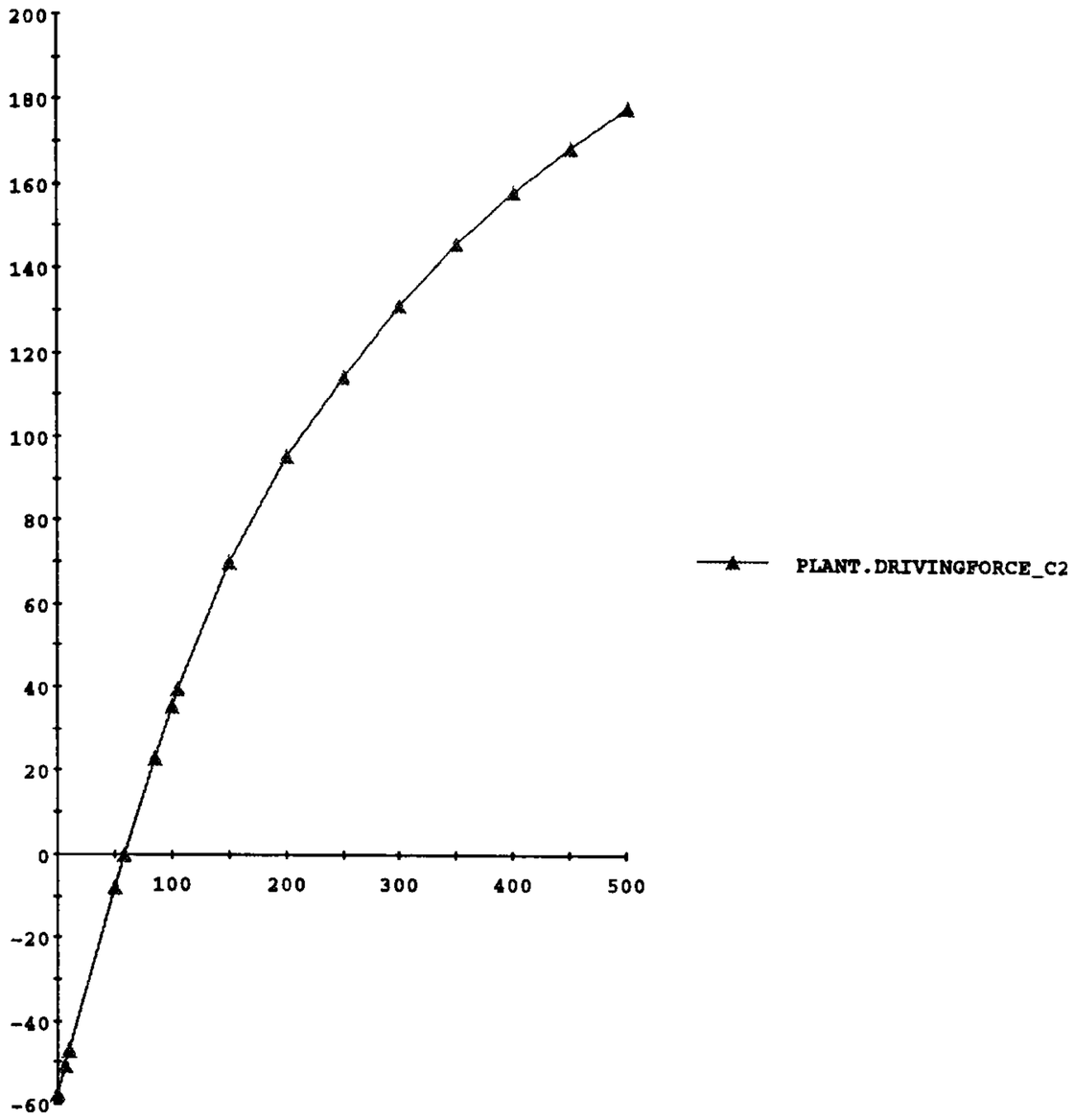

Figure 6.3: Driving force variation accross of connection C2 between flash drum and the reservoir

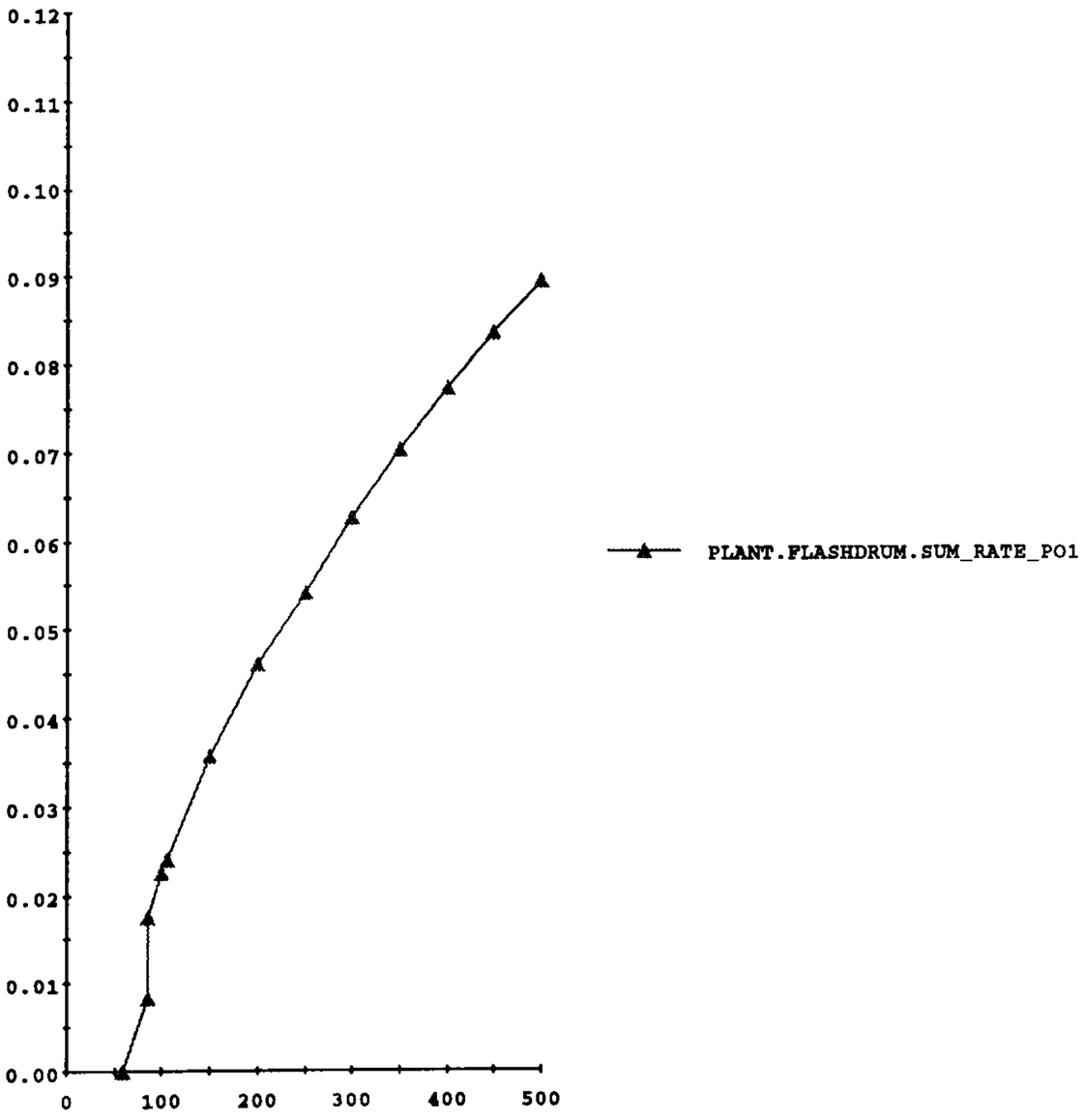

Figure 6.4: Variation of total mass rate flowing through port Po1 in the flash drum

6.2 Two Flash Drums with reversible flow

This example considers the two flash drums linked together via. a connection allowing flow in either direction. The operating and initial conditions used in this simulation are as follows:

Operating Conditions

- source reservoir, R1

Pressure	=	1.013E2	kPa
Density	=	330	kg/m3
Mass Fraction	=	[0.4,0.6]	
Enthalpy	=	3.5E2	kJ/kg
Dispersion Ratio	=	0.01	
PhaseType	=	3	(aggregation type)
Viscosity	=	5E-5	Pa.s

- sink reservoirs, R2, R3 and R4

Pressure	=	1.013E2	kPa

Initial Conditions

- FlashDrum1

Temperature of vapour	=	300	K
Mass of each component in separate vapour	=	[1,1]	kg
Mass of each component in aggregation	=	[15,15]	kg
dispersion ratio	=	0.001	

- FlashDrum2

Temperature of vapour	=	280	K
Mass of each component in separate vapour	=	[1,1]	kg
Mass of each component in aggregation	=	[15,15]	kg
dispersion ratio	=	0.002	

Figure 6.5, 6.6 and 6.7 shows the change with time of the horizontal mixture levels and the total mass holdup in each flash drum respectively.

This example illustrates the change in flow direction in the connection between the two flash drums. This is shown in in figure 6.8 which represents the change in port type with type. In this figure a port type of 1 represents an outlet flow whilst a port type of 2 represents a flow into the vessel.

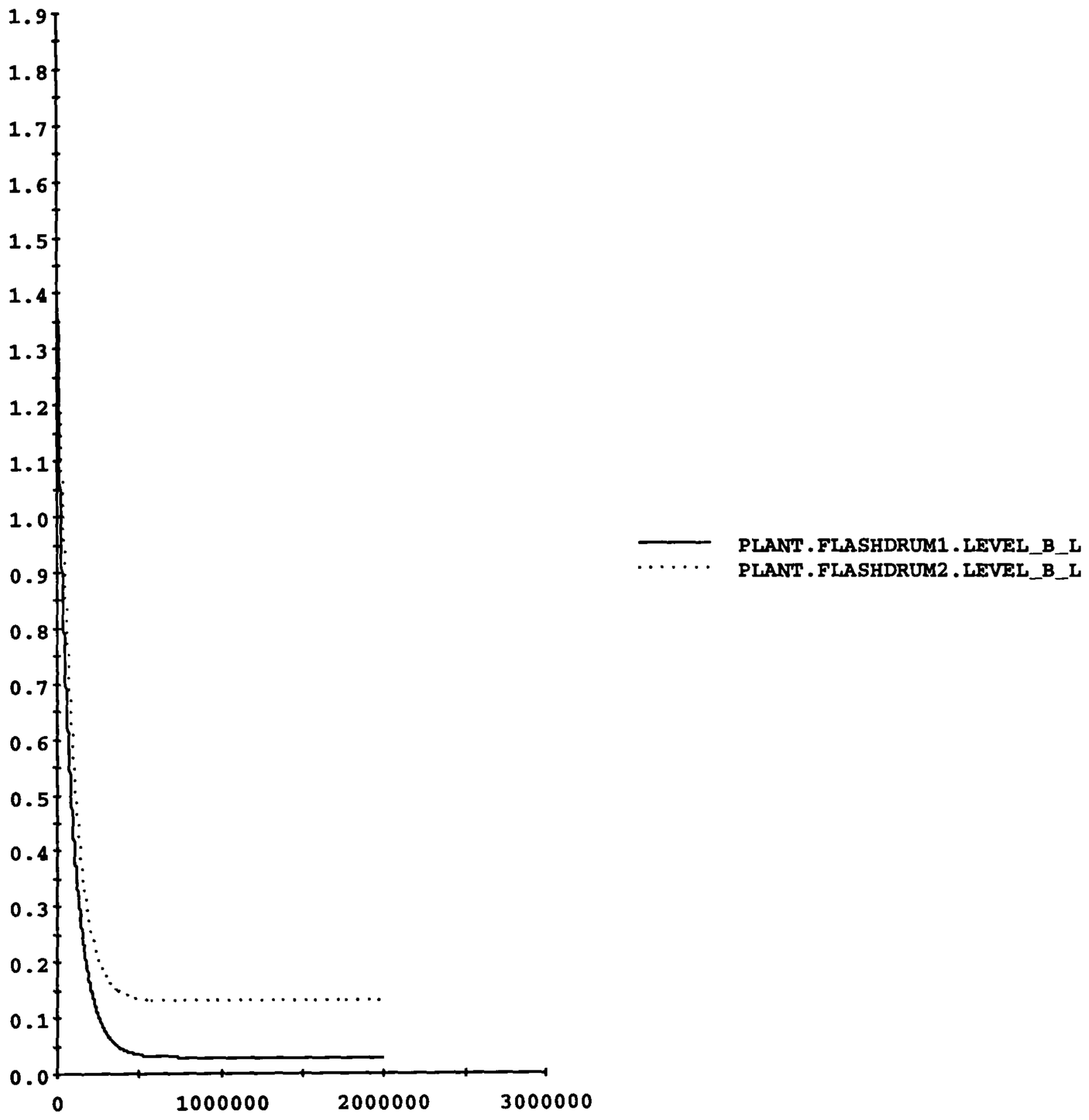

Figure 6.5: Level variation in two flash drums

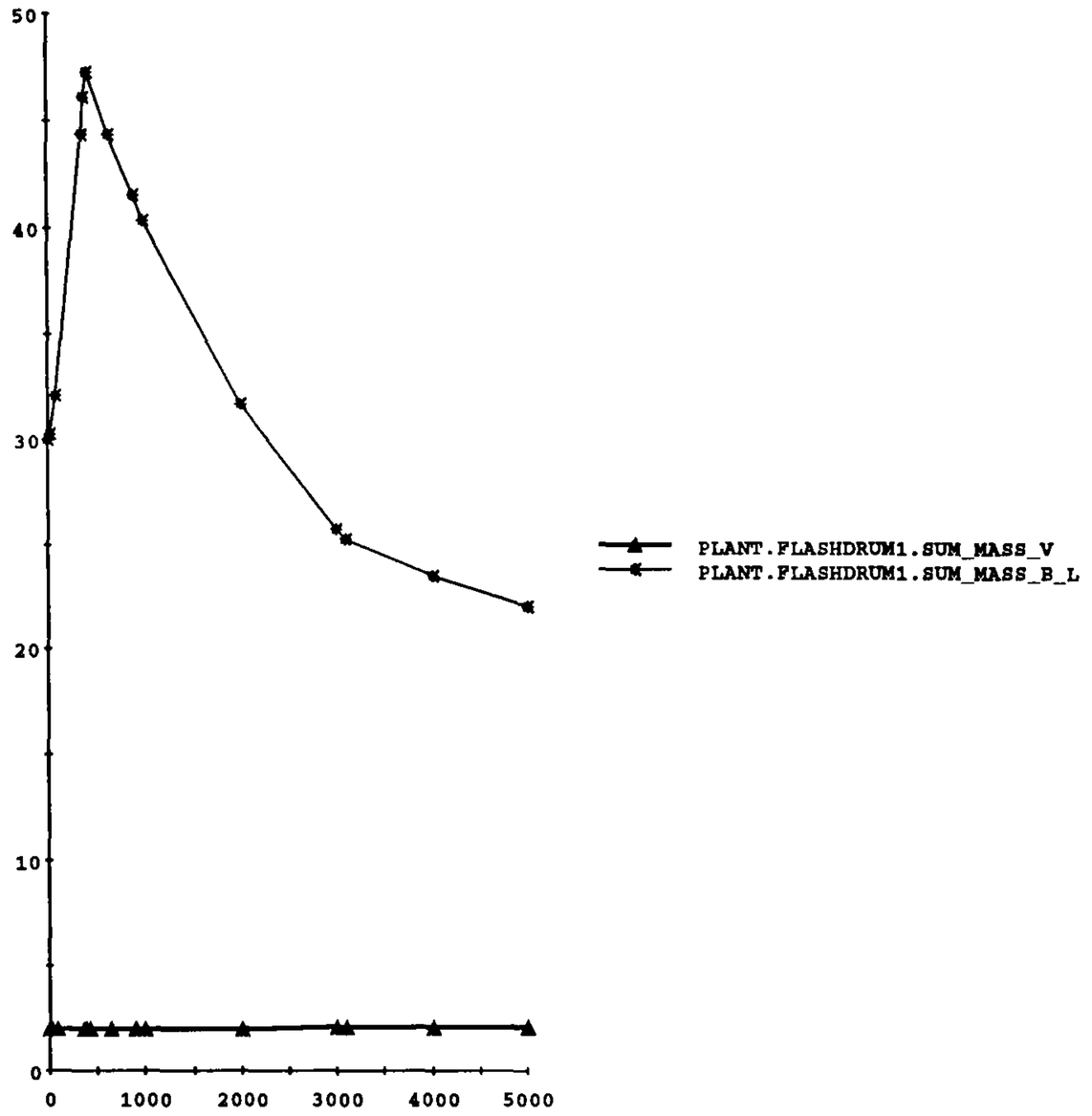

Figure 6.6: Holdup variation in flash drum1

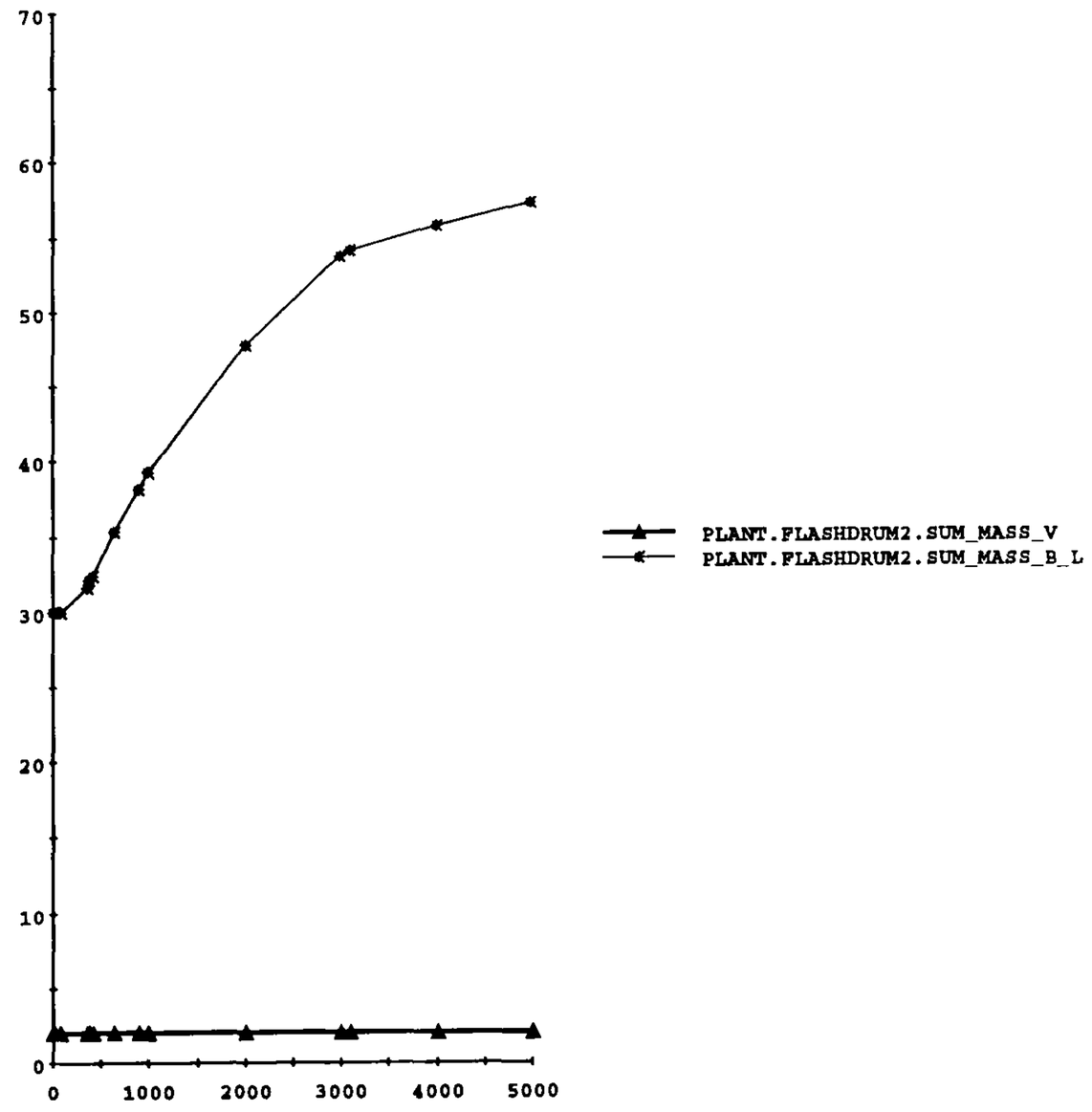

Figure 6.7: Holdup variation in flash drum2

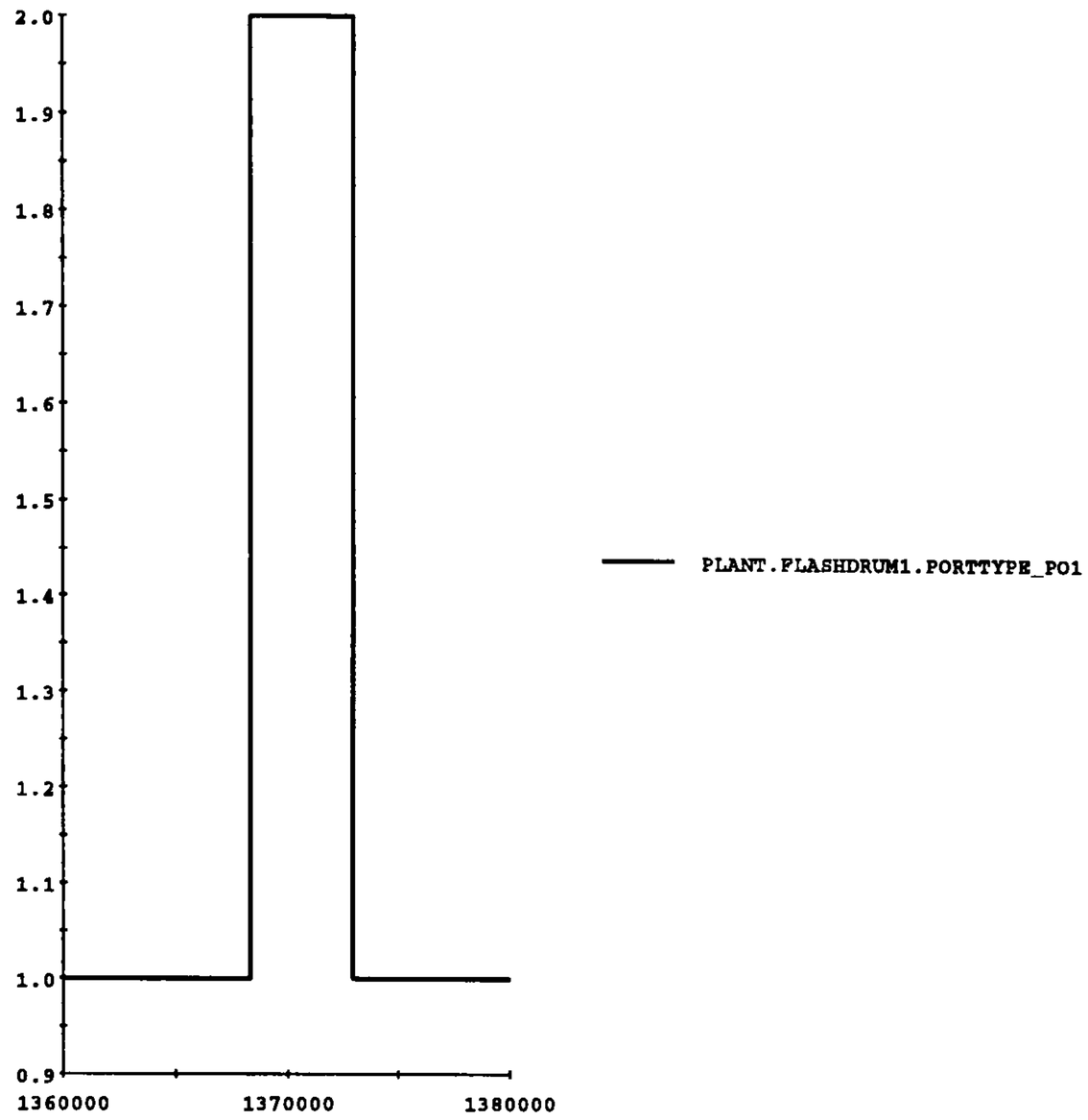

Figure 6.8: Flow reversibility between two flash drums

6.3 Decanter : Settling tank

We now consider the simulation of the decanter introduced in §3.4.3. In this example we considered a single component multi–phase system. The operating and initial conditions are as follows:

Operating Conditions

- source reservoir, R1

Pressure	=	2*1.013E2	kPa
Density	=	902.5	kg/m3
Mass Fraction	=	1.0	
Enthalpy	=	0	kJ/kg
Dispersion Ratio	=	0.3	
PhaseType	=	3	(aggregation type)
Viscosity	=	0.00178	Pa.s

- sink reservoirs, R2, R3

Pressure	=	1.013E2	kPa

- decanter

Enthalpy of Butanol	=	0	kJ/kg
Enthalpy of Butanol in aggregation	=	0	
Enthalpy of Water in aggregation	=	0	
Enthalpy of Water	=	0	
Temperature of Butanol	=	298	K
Temperature of Butanol in aggregation	=	298	
Temperature of Water in aggregation	=	298	
Temperature of Water	=	298	
Density of Butanol	=	805	kg/m3
Density of Butanol in aggregation	=	805	
Density of Water in aggregation	=	1000	
Density of Water	=	1000	
Viscosity of Butanol	=	2.65E-3	Pa.s
Viscosity of Butanol in aggregation	=	2.65E-3	
Viscosity of Water in aggregation	=	9.0E-4	
Viscosity of Water	=	9.0E-4	
Pressure in decanter	=	1.013E2	kPa

Initial Conditions

- decanter

Mass of Butanol	=	0	kg
Mass of Water	=	0	kg
Mass of the aggregate	=	0	kg
Mass of Water in aggregation	=	0	kg
IntEnergy of Butanol	=	0	kJ/kg
IntEnergy of Butanol in aggregation	=	0	kJ/kg
IntEnergy of Water in aggregation	=	0	kJ/kg
IntEnergy of Water	=	0	kJ/kg

Figure 6.9 and 6.10 shows change with time of the butanol level and the mass holdup of each phase respectively. Figure 6.11 illustrates the fact that there is no flow over the weir until the butanol level as reached the top of the weir, at approximately 1200 sec.

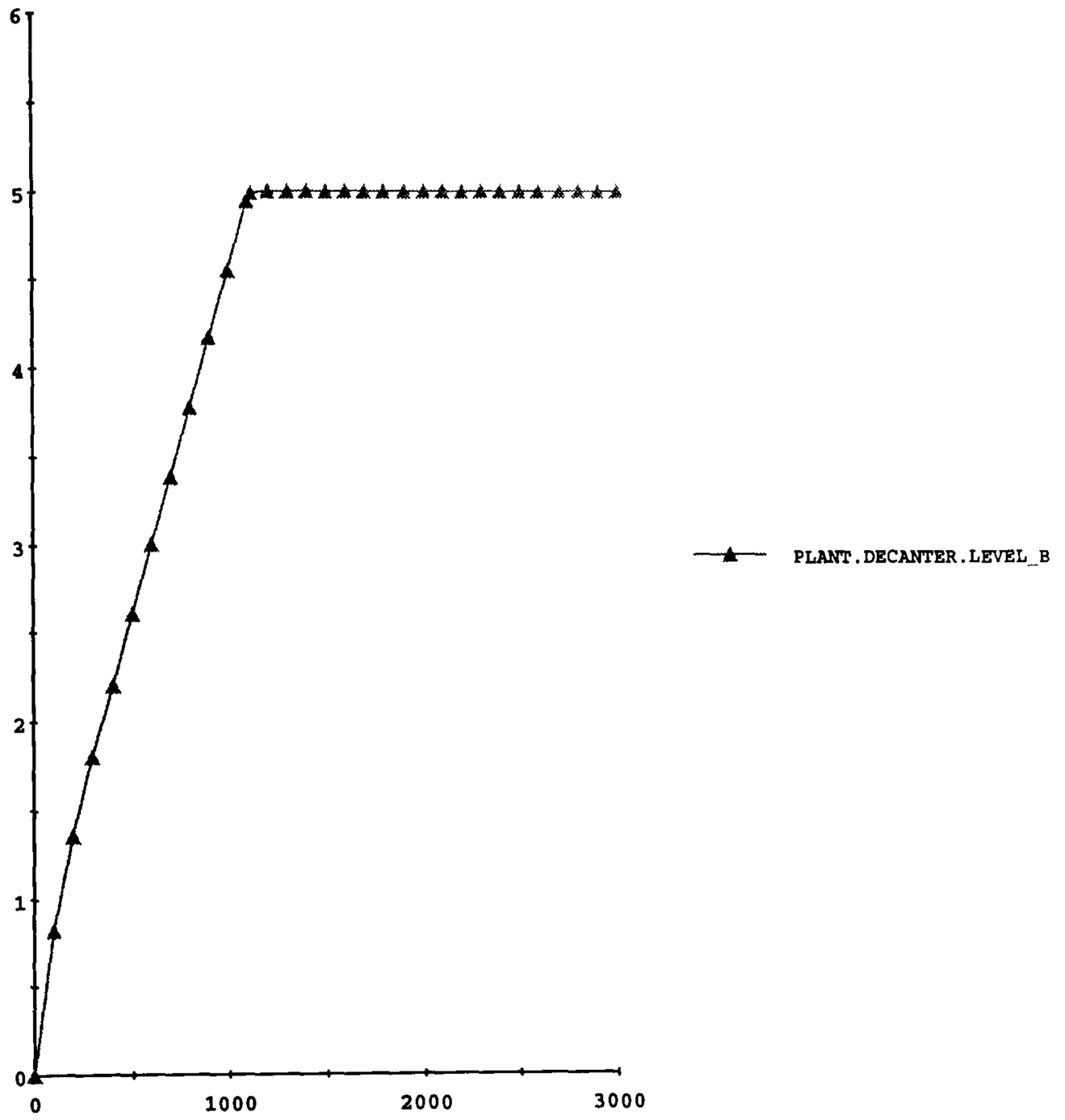

Figure 6.9: Level variation in decanter

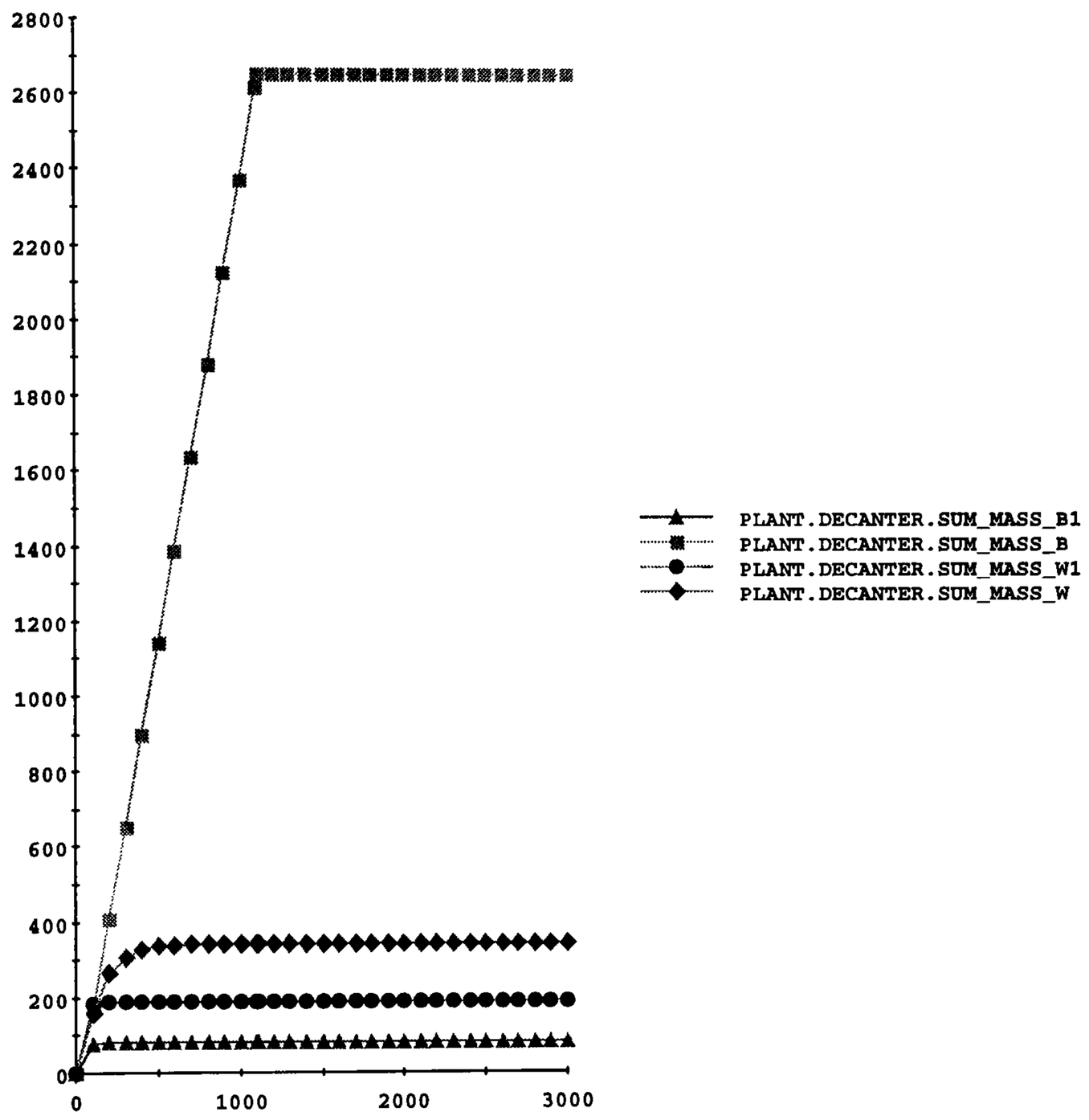

Figure 6.10: Holdup variation in decanter

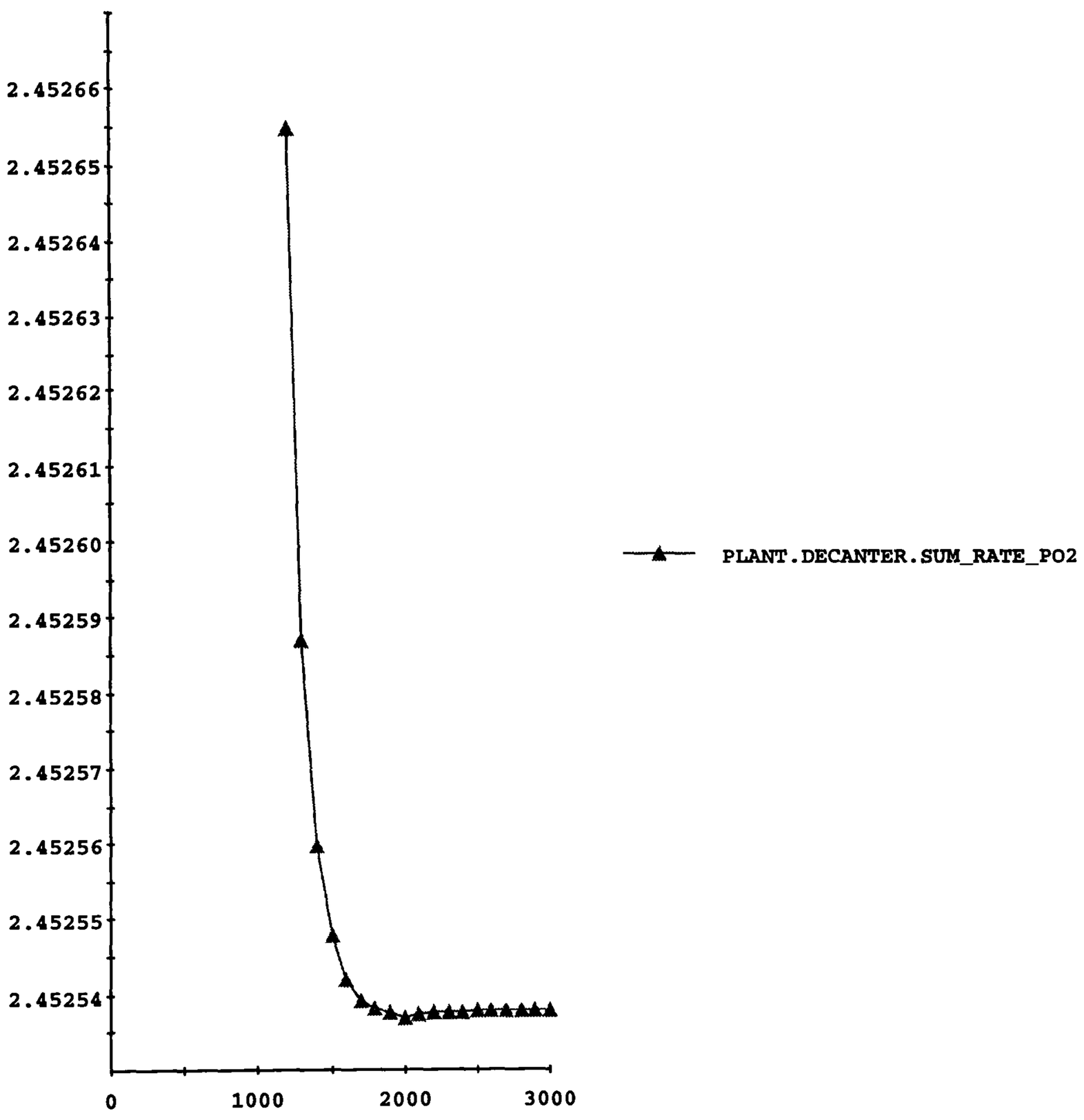

Figure 6.11: Flow over weir in decanter

Chapter 7

Conclusions and Future Work

7.1 Conclusions

The software package implemented on the basis of the methodology described in this thesis allows the automatic generation of lumped–parameter models taking account of discontinuous behaviour routinely arising in process systems. The package has been successfully tested through the simulation of several examples using the general–purpose simulation package, *gPROMS* to prove the feasibility of the proposed model formulation approach.

The starting point of this research was the construction of the concept for process representation from the previous work (Vázquez-Román, 1992), which has been extended to encompass combined discrete/continuous process systems, although the underlying concept remains unchanged. The basic premise is well defined *thermodynamic phases* where the state of all material present may be described in terms of a finite set of state variables and hence any process system may be viewed as a set of *inter–connected vessels* where the *phases* present interact. These interactions may involve transfers of material and/or energy between pairs of phases or vessels through well defined physico–chemical *transfer laws.*

The textual description in a high level language specific to process systems has been chosen as a method for supporting the description of process systems in a purely physical fashion based on the conceptual process representation. The formal language for this purpose has been designed with a special emphasis on process representation for phase distribution within a vessel and on the geometry of process equipment and their connections. This has a consequence that the formalism of the language satisfies the key requirements of our ultimate research objective:

- highly declarative description
- hierarchically structured formalism
- purely physical behaviour–oriented feature
- provisions enabling identification of the physical discontinuities routinely arising in a given process system

The most important feature of the methodology for formulating mathematical models presented in this thesis is the capability for dealing with the physical discontinuities arising in a given process system. This discontinuous behaviour arises routinely due to the phase appearance or disappearance, transitions between fluid flow regimes and flow direction and those resulting from the geometry of individual process equipment. No other computer–aided model generation package designed to automatically identify these discontinuities and then to generate appropriate mathematical models from purely physical fashion has yet been reported.

The current version of this package generates mathematical models in the form of *gPROMS* input language, which makes it possible for the models to be directly simulated within the *gPROMS* environment. The procedure of storing the information about mathematical models in internal data structure is completely decoupled from that of generating its output format, instead of directly producing its output format without its intermediate storage. From the point of view of a software development this suggests a potential for high compatibility of a software product, producing multiple output formats as desired within a unified software framework, thus making it possible to easily interface with other modelling and simulation packages.

Research into the development of general–purpose modelling environments, which, in this context, provide users with a facility for describing the underlying dynamic behaviour of process systems in terms of a set of variables and equations, may be thought of as being somewhat in its maturity, though further work is required in some aspects, for example, the modelling and simulation of combined lumped and distributed parameter systems. In contrast, the research into automatically generating appropriate mathematical models, partly or totally, with the objective of ultimately freeing users from formulating the consistent set of equations comprising the model for a desired process system, from author's point of view, is still in its infancy. It is hoped that the work described in this thesis will make some contribution towards enhancing the practical feasibility of sophisticated computer–aided modelling tools.

7.2 Future Work

In this section a number of issues for future work from a both theoretical and practical point of view will briefly be discussed.

7.2.1 Suggestions on "Phase" and "Aggregation" statement

The phase selection rule for entry ports in this research has been developed in §4.3.1.6. However, further refinement of this rule is still required to deal with the wide range of real situations relating to the behaviour of interacting phases in a vessel and their inter–vessel connections.

The basic idea is to predict the ratios of the phases present as a function of height, using simple mixing/separation laws (in general "dispersion law"), thus their solution yields volume–fractions of phases present in a vessel as a function of height and time. By this idea it is clear that what flows out of a port at a given height is the mixture of phases at this height. Similarly, the injection of a given mixture of phases through a feed–port will be dispersed according to these dispersion laws. These dispersion laws can take account of the presence of stirrers, and are thus defined for each *vessel* (not for transfers between phases), in principle to cover all possible phases which can be simultaneously present in the vessel.

Strictly, this sweeps away the need for our "Aggregation" statement, since in principle all phases involved *may* be simultaneously present, and mixed to different extents at different heights. Instead the user is allowed to simply define the dispersion laws.

However it may be useful to retain the Aggregation statement so that the user can override this general principle, restricting the possible combinations assumed to occur and specifying perhaps different dispersion laws for different combinations.

7.2.2 Broadening the scope of the physical description

The model formulation methodology developed in this thesis has been proved to be successful as shown through the simulation of several examples involving flash drums and a decanter. However the current version of the proposed language and the implemented software package based on this methodology requires some extensions to deal with a wide range of applications such as membrane, packed beds, fluidised beds.

In addition to this requirement, a more general treatment for describing irregular geometry of process equipments is needed. One possible way to cope with this is an integration of the model generation package with a commercialised facility fully supporting

the geometrical description of individual process units in 3–dimensions.

7.2.3 System environment

This thesis has focussed on the core part of computer–aided process modelling, the automatic mathematical model generation, rather than the design of a sophisticated system environment (Stephanopoulos *et al.*, 1987; Sorlie, 1990; Bar and Zeitz, 1990; Westerberg *et al.*, 1991; Andersson, 1994; Vázquez-Román *et al.*, 1996) which is an issue of considerable importance in process engineering. The current version of the package is thus designed to interface with those environments above. As far as the development of such an environment is concerned, it is recommended that such development should be undertaken *within* the scope of the development of an integrated process engineering environment.

7.2.4 Refinement of transfer law library

As illustrated in §4.1, the transfers of material and/or energy between pairs of interacting phases and inter–vessels are described in terms of the state–variables of the two phases through well defined physico–chemical *transfer laws*. The set of equations and variables comprising a given transfer law are invoked from the transfer law library. In order for the library to hold a vast variety of transfer laws and to allow rapid access to search for a given transfer law entry, it is desirable to construct the library whose internal structure is designed by the technology employed for building efficient database management systems.

7.2.5 Generation of distributed parameter models

At present the prototype package is limited to the generation of lumped–parameter models. To broaden the range of application, the extension of this package to generate distributed–parameter models (or combined with lumped–parameter models) as a long term research project is recommended. Current contributions made in this direction are (Dieterich and Eigenberger, 1995; Lohmann and Marquardt, 1996; Jensen and Gani, 1996; Barber, 1997). A number of issues of considerable importance remain unresolved, these include facilities for handling complex geometries, the automatic generation of appropriate boundary conditions and a consistent methodology for modelling multi–phase systems.

As a continuation of this work the development of a computer package for the automatic generation of combined lumped/distributed parameter mathematical models

is in progress at Imperial College (Barber, 1997). This work will include the design of a conceptual framework for representing process systems based on a completely physical description and the development of a mathematical modelling strategy to deal with, in general, 3 dimensional multi–phase systems.

It is also worth noting that the second version of *gPROMS* (Oh, 1995) is the only currently available modelling environment with facilities for modelling and simulation of combined lumped and distributed parameter systems.

References

A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers: principles, techniques, and tools.* Addison–Wesley Publishing Company, 1986.

M. Andersson. Omola – an object–oriented language for model representation. Licentiate Thesis TFRT–3208, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden, 1990.

M. Andersson. Discrete event modelling and simulation in omola. In *IEEE Symp. Computer–Aided Control System Design*, CACSD '92, California, March 17–19 1992.

M. Andersson. Object–oriented modelling and simulation of hybrid systems. Doctoral dissertation, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden, 1994.

R. Aris. *Mathematical Modelling Techniques.* Pitmans, London, 1978.

W. M Austin and B. Khoshnevis. Automatic model generation for production–distribution systems using natural language. *SIMULATION*, 52(5):207–211, 1989.

M. Bar and M. Zeitz. A knowledge–based flowsheet–oriented user interface for a dynamic simulator. *Computers & Chemical Engineering*, 14:1275–1283, 1990.

J. A. Barber. Towards computer generation of combined lumped and distributed parameter process models. Second quarterly progress report, The Centre for Process Systems Engineering, Imperial College, London, 1997.

P. I. Barton and C. C. Pantelides. The modelling of combined discrete/continuous processes. *AIChE J*, 40:966–979, 1994.

P. I. Barton. *The modelling and simulation of combined discrete/continuous processes.* PhD thesis, Imperial College, University of London, 1992.

H. W. Beck and P. A. Fishwick. Incorporating natural language descriptions into modeling and simulation. *SIMULATION*, 52(3):102–109, 1989.

R. Bogusch and W. Marquardt. A formal representation of process model equations. *Computers & Chemical Engineering*, 19, Suppl.:S211–S216, 1995.

A. J. Czulek. An experimental simulator for batch chemical processes. *Computers & Chemical Engineering*, 12(2/3):253–259, 1988.

E. E. Dieterich and G. Eigenberger. Bimap – a tool for comuter aided modeling in chemical reaction engineering. *Computers & Chemical Engineering*, 19, Suppl.:S773–S778, 1995.

H. Elmqvist. *A structured model language for large continuous systems.* PhD thesis, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden, 1978.

M. Feinberg. On Gibb's phase rule. *Archives Rational Mechanics and Analysis*, 70:219–234, 1979.

R. Gani, E. L. Sorensen, and J. Perregaard. Design and analysis of chemical processes through DYNSIM. *Ind. Eng. Chem. Res.*, 31(1):244–254, 1992.

P. Holl, W. Marquardt, and E. D. Gilles. DIVA – a powerful tool for dynamic process simulation. *Computers & Chemical Engineering*, 12(5):421–426, 1988.

A. K. Jensen and R. Gani. A computer aided system for generation of problem specific process models. *Computers & chemical Engineering*, 20, Suppl.:S145–S150, 1996.

G. S. Joglekar and G. V. Reklatis. A simulator for batch and semi–continuous processes. *Computers & Chemical Engineering*, 8(6):315–327, 1984.

B. W. Kernighan and D. M. Richie. *The C programming language.* Prentice Hall, second edition, 1988.

A Kroner, P. Holl, W. Marquardt, and E. D. Gilles. DIVA – an open architecture for dynamic bimulation. *Computers & Chemical Engineering*, 14(11):1289–1295, 1990.

L. Lamport. *Latex user's guide & reference manual.* Addison–Wesley Publishing Company, 1986.

J. R. Levine, T. Mason, and D. Brown. *lex & yacc.* O'Reilly & Associates, Inc., 1995.

B. Lohmann and W. Marquardt. On the systematization of the process of model development. *Computers & Chemical Engineering*, 20, Suppl.:S213–S218, 1996.

W. Marquardt, A. Gerstlauer, and E. D. Gilles. Modeling and representation of complex objects: chemical engineering perspective. In *Proc. of 6th Int. Conf. on IEA/AIE*, pages 219–228, Edinbrugh, Scotland, June 1–4 1993.

W. Marquardt. Dynamic process simulation – recent progress and future challenges. In Y. Arkun and W.H. Ray, editors, *Chemical Process Control*, pages 131–180. CACHE-AICHE Publications, 1991.

W. Marquardt. Trends in computer–aided porcess modeling. *Computers & Chemical Engineering*, 20(6/7):591–609, 1996.

S. E. Mattsson and M. Andersson. Omola – an object–oriented modeling language. In M. Jamshidi and C.J. Herget, editors, *Recent Advances in Computer–Aided Control Engineering*, pages 291–310, Amsterdam, 1992. Elsevier.

B Meyssami and O. A. Åsbjørnsen. Process modelling from first principles–method and automation. In *Proc. 1989 Summer Computer Simulation Conference*, pages 292–297, Austin, TX, July 24–27 1989.

B. Nilsson. *Object–oriented modeling of chemical processes.* Phd thesis, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden, 1993.

M. Oh and C. C. Pantelides. A modelling and simulation language for combined lumped and distributed parameter systems. *Computers & Chemical Engineering*, 20(6/7):611–633, 1996.

M. Oh. *Modelling and simulation of combined lumped and distributed processes.* PhD thesis, Imperial College, University of London, 1995.

C. C. Pantelides and P. I. Barton. Equation–oriented dynamic simulation: current status and future perspectives. *Computers & Chemical Engineering*, 17, Suppl.:S263–S285, 1992.

C. C. Pantelides, D. Gritsis, K. R. Morison, and R. W. H. Sargent. The mathematical modeling of transient systems using differential–algebraic equations. *Computers & Chemical Engineering*, 12:745–755, 1988.

C. C. Pantelides. SpeedUp – recent advances in process simulation. *Computers & Chemical Engineering*, 12(7):745–755, 1988.

C. C. Pantelides. Modelling, simulation and optimisation of hybrid processes. In *Workshop on Analysis and Design of Event–Driven Operations in Process Systems*, London, 10–11 April 1995. Centre for Process Systems Engineering, Imperial College.

C. C. Pantelides. *Dynamic Behaviour of Process Systems: Lecture Note*, chapter 8. Imperial College, London, March 1996.

J. D. Perkins and G. W. Barton. Modelling and simulation in process operation. In G.V. Reklatis and H.D. Spriggs, editors, *Computer Aided Process Operations*, pages 287–316, 1987.

J. D. Perkins and R. W. H. Sargent. SPEEDUP: a computer program for steady–state and dynamic simulation and design of chemical processes. *AIChE Symposium Series*, 78(214):1–11, 1982.

J. D. Perkins, R. W. H. Sargent, R. Vázquez-Román, and J. H. Cho. Computer generation of process models. *Computers & Chemical Engineering*, 20(6/7):635–639, 1996.

J. Perregaard, B. S. Pedersen, and R. Gani. Steady state and dynamic simulation of complex chemical processes. *Trans IChemE*, 70:99–109, March 1992.

P. C. Piela, T. G. Epperly, K. M. Westerberg, and A. W. Westerberg. ASCEND : an objected–oriented computer environment for modeling and analysis : the modeling language. *Computers & Chemical Engineering*, 15(1):53–72, 1991.

P. Piela. *ASCEND – an object–oriented environment for the development of quantitative models.* PhD thesis, Carnegie Mellon University, Pittsburgh, PA, 1989.

H. A. Preisig, T. Y. Lee, and F. Little. A prototype computer–aided modelling tool for life–support system models. Technical Paper 901269, SAE International, July 9–12 1990.

H. A. Preisig. Modeller – an object–oriented computer–aided modelling tool. In L.T. Biegler and M. F. Doherty, editors, *Foundations of Computer–Aided Process Design*, volume 91 of *AIChE Symp. Series No. 304*, pages 328–331. CACHE and AIChE, 1995.

H. A. Preisig. Computer–aided modelling – two paradigms on control. *Computers & Chemical Engineering*, 20, Suppl.:S981–S986, 1996.

W. F. Ramirez. *Computational methods for process simulation.* Butterworths Series in Chemical Engineering, 1989.

R. W. H. Sargent and A. W. Westerberg. SpeedUp in chemical engineering design. *Trans. Instn. Chem. Engrs.*, 42:T190–T197, 1964.

R. W. H. Sargent. Advances in modelling and analysis of chemical process systems. *Computers & Chemical Engineering*, 7(4):219–237, 1983.

R. W. H. Sargent. Process design – what next ? In J.J. Siirola, I.E. Grossmann, and G. Stephanopoulos, editors, *Foundations of Computer–Aided Process Design*, pages 529–553, Amsterdam, 1990. Elsevier.

R. W. H. Sargent. *Technical suggestion for future work*, October 1996.

C. F. Sorlie. *A computer environment for process modeling.* Phd thesis, University of Trondheim, Norway, 1990.

G. Stephanopoulos and C. Han. Intelligent systems in process engineering: a review. *Computers & Chemical Engineering*, 20(6/7):743–791, 1996.

G. Stephanopoulos, J. Johnston, T. Kriticox, R. Lakshmanan, M. Mavrovouniotis, and C. Siletti. Design–Kit : an object–oriented environment for process engineering. *Computers & Chemical Engineering*, 11(6):655–674, 1987.

G. Stephanopoulos, G. Henning, and H. Leone. MODEL.LA A modeling language for process engineering – I. the formal framework. *Computers & Chemical Engineering*, 14(8):813–869, 1990.

G. Stephanopoulos, G. Henning, and H. Leone. MODEL.LA A modeling language for process engineering – II. multifaceted modeling of process systems. *Computers & Chemical Engineering*, 14(8):847–869, 1990.

K. Telnes. *Computer aided modeling of dynamic processes based on elementary physics.* PhD thesis, the Norwegian Institute of Technology, Trondheim University, Norway, 1992.

R. Vázquez-Román, J. M. P. King, and R. Ba nares Alcántara. KBMoSS: A process engineering modelling support system. *Computers & Chemical Engineering*, 20, Suppl.:S309–S314, 1996.

R. Vázquez-Román. *Computer aids for process model–building.* PhD thesis, Imperial College, University of London, 1992.

A. W. Westerberg, P. C. Piela, R. D. McKelvey, and T. G. Epperly. The ASCEND modeling environment and its implications. In *Proceedings 4th International Symposium on Process Systems Engineering*, volume I, 1991.

Appendix A

Lex Input Specification

```
%{
/* ======================== */
/* Lex Input Specification */
/* ======================== */

#include <stdio.h>
#include <ctype.h>
%}
/* ==================== */
/* regular definitions */
/* ==================== */

comment              "#".*
delim                [ \t\n\r]
ws                   {delim}+
letter               [A-Za-z]
digit                [0-9]
irreversible_flow    "-"+\>
reversible_flow      "<"{irreversible_flow}
id                   {letter}({letter}|{digit})*
number               {digit}+(\.{digit}+)?(E[+\-]?{digit}+)?
underline            "_"
%%
{comment}       { }
```

```
{ws}                {  }
{irreversible_flow}    {return IRREVERSIBLE_FLOW;}
{reversible_flow}      {return REVERSIBLE_FLOW;}
RESERVOIR           {return RESERVOIR;}
VESSEL              {return VESSEL;}
CONNECTION          {return CONNECTION;}
Phase               {return PHASE;}
vapour              {return VAPOUR;}
Aggregation         {return AGGREGATION;}
Transfer\ Law       {return TRANSFER_LAW;}
Port                {return PORT;}
in                  {return IN;}
out                 {return OUT;}
both                {return BOTH;}
Geometry            {return GEOMETRY;}
Port\ Position      {return PORT_POSITION;}
Phase\ Position     {return PHASE_POSITION;}
Shape               {return SHAPE;}
cylinder            {return CYLINDER;}
box                 {return BOX;}
sphere              {return SPHERE;}
open                {return OPENVESSEL;}
closed              {return CLOSEDVESSEL;}
Dimension           {return DIMENSION;}
height              {return HEIGHT;}
length              {return LENGTH;}
width               {return WIDTH;}
depth               {return DEPTH;}
diameter            {return DIAMETER;}
area                {return AREA;}
Orientation         {return ORIENTATION;}
vertical            {return VERTICAL;}
horizontal          {return HORIZONTAL;}
\Z                  {return  'Z';}
{id}                {return Identifier;}
```

```
{number}        {return Number;}
\,              {return  ',';}
\;              {return  ';';}
\(              {return  '(';}
\)              {return  ')';}
\:              {return  ':';}
\.              {return  '.';}
\[              {return  '[';}
\]              {return  ']';}
%%
```

Appendix B

Yacc Input Specification

```
/* ================================================== */
/* yacc_input.y : Yacc Input Specification for the Current */
/*                Implemenatation of a model generator     */
/* ================================================== */

%{
#include "global.h"
%}

/* ================================== */
/* data structure for YACC stack type */
/* ================================== */
%union
{
  VesselSymb                    *vessel_symb;
  ReservoirSymb                 *reservoir_symb;
  Symb                          *symb;
  PhaseDec                      *phase_dec;
  AggPair                       *agg_pair;
  TransferLawDec                *transfer_law_dec;
  PortDec                       *port_dec;
  PortPhaseSpec                 *port_phase_spec;
  GeometrySec                   *geometry_sec;
```

```
PortPositionDec          *port_position_dec;
Parameter                *parameter;
DimensionDec             *dimension_dec;
ConnectionSection        *connection_section;
Path                     *path;
double                   fval;
  int                    ival;
  char                   *string;
}

/* ====== */
/* Tokens */
/* ====== */

%token   Identifier
%token   Number
%token   '.'
%token   ':'
%token   ';'
%token   '('
%token   ')'
%token   '['
%token   ']'
%token   ','
%token   'Z'
%token   RESERVOIR
%token   VESSEL
%token   PHASE
%token   VAPOUR
%token   AGGREGATION
%token   TRANSFER_LAW
%token   PORT
%token   IN
%token   OUT
%token   BOTH
```

```
%token    GEOMETRY
%token    PORT_POSITION
%token    PHASE_POSITION
%token    SHAPE
%token    CYLINDER
%token    BOX
%token    SPHERE
%token    OPENVESSEL
%token    CLOSEDVESSEL
%token    DIMENSION
%token    HEIGHT
%token    LENGTH
%token    WIDTH
%token    DEPTH
%token    DIAMETER
%token    AREA
%token    ORIENTATION
%token    VERTICAL
%token    HORIZONTAL
%token    CONNECTION
%token    IRREVERSIBLE_FLOW
%token    REVERSIBLE_FLOW

%%

/* ================== */
/* Grammar Definition */
/* ================== */

/* A Process System */
yyModelBlock : yyVesselBlock
               yyReservoirBlock
               yyConnectionBlock
             ;
```

```
/* Vessel Entities */
yyVesselBlock : yyVesselSection
              | yyVesselBlock yyVesselSection
              ;

yyReservoirBlock : yyReservoirSection
                 | yyReservoirBlock yyReservoirSection
                 ;

yyReservoirSection : RESERVOIR ':'
                     yyReservoirIdentifierList
                     PORT ':'
                     yyIdentifierList
                   ;

yyReservoirIdentifierList : Identifier
                          | yyReservoirIdentifierList ',' Identifier
                          ;

yyVesselSection : VESSEL ':' yyVesselIdentifierList
                  yyPhaseSection
                  yyAggregationSection
                  yyTransferLawSection
                  yyPortSection
                  yyGeometrySection
                ;

yyVesselIdentifierList : Identifier
                       | yyVesselIdentifierList ',' Identifier
                       ;

yyPhaseSection : PHASE ':' yyPhaseList
               ;

yyPhaseList : yyPhaseDec
```

```
            | yyPhaseList yyPhaseDec
            ;

yyPhaseDec : yyIdentifierList ':' yyPhaseType
           ;

yyPhaseType : Identifier
            ;

yyAggregationSection : AGGREGATION ':' yyAggregationList
                     |
                     ;

yyAggregationList : yyAggregationDec
                  | yyAggregationList ',' yyAggregationDec
                  ;

yyAggregationDec : Identifier
                 | yyAggIdentifier
                 ;

yyAggIdentifier : '[' Identifier ',' Identifier ']'
                ;

yyTransferLawSection : TRANSFER_LAW ':' yyTransferLawList
                     |
                     ;

yyTransferLawList : yyTransferLawDec
                  | yyTransferLawList yyTransferLawDec
                  ;

yyTransferLawDec : Identifier ',' Identifier ':' Identifier
                 ;
```

```
yyPortSection : PORT ':' yyPortList
              ;

yyPortList : yyPortDec
           | yyPortList yyPortDec
           ;

yyPortDec : yyIdentifierList
          | yyIdentifierList ':' yyPortType
          | yyIdentifierList ':' yyPortType ':' yyPortPhaseSpecification
          ;

yyPortType : IN
           | OUT
           | BOTH
           ;

yyPortPhaseSpecification : Identifier
                         | yyPortAggIdentifier
                         ;

yyPortAggIdentifier : '[' Identifier ',' Identifier ']'
                    ;

yyGeometrySection : GEOMETRY ':'
                    yyShapeSection
                    yyOrientationSection
                    yyDimensionSection
                    yyPortPositionSection
                  ;

yyPortPositionSection : PORT_POSITION ':' yyPortPositionList
                      ;
```

```
yyPortPositionList : yyPortPositionDec
                   | yyPortPositionList yyPortPositionDec
                   ;

yyPortPositionDec : 'Z' '(' Identifier ')' ':' yyParameter
                  ;

yyParameter : Number
            | HEIGHT
            | DIAMETER
            | Identifier
            ;

yyShapeSection : SHAPE ':' yyShapeType ':' yyOpenClosed
               ;

yyShapeType : CYLINDER
            | BOX
            | SPHERE
            ;

yyOpenClosed : OPENVESSEL
             | CLOSEDVESSEL
             ;

yyDimensionSection : DIMENSION ':' yyDimensionList
                   ;

yyDimensionList : yyDimensionDec
                | yyDimensionList yyDimensionDec
                ;

yyDimensionDec : yyDimensionType ':' Number
               ;
```

```
yyDimensionType : Identifier
                | HEIGHT
                | DEPTH
                | WIDTH
                | DIAMETER
                ;

yyOrientationSection : ORIENTATION ':' yyOrientationType
                     ;

yyOrientationType : VERTICAL
                  | HORIZONTAL
                  ;


yyConnectionBlock : CONNECTION ':' yyConnectionList
                  ;

yyConnectionList : yyConnectionSection
                 | yyConnectionList yyConnectionSection
                 ;

yyConnectionSection : Identifier ':'
                        yyPathName yyConnectionType yyPathName ';'
                        Identifier
                    ;

yyPathName : Identifier '.' Identifier
           ;

yyConnectionType : IRREVERSIBLE_FLOW
                 | REVERSIBLE_FLOW
                 ;

yyIdentifierList : Identifier
```

```
                | yyIdentifierList ',' Identifier
                ;
%%
#include "lex.yy.c"
```

Appendix C

Transfer Law Library

As illustrated earlier, a transfer law is a set of equations describing the mechanism of the transfer between phases in a vessel or through a connection. Such a law should be installed in the library table so that whenever invoked by the user, the package searches the library table and then proceeds to transform the set of equations in the transfer law into an appropriate form by taking account of the relevant phases (intra–vessel phase interaction) or ports (inter–vessel connection). In general the mathematical formalism of transfer laws is the same as that discussed in §4.2 in terms of dealing with physical discontinuities, which are decomposed into case invariant equations and variant parts.

Illustrative transfer laws installed in the library table in order to deal with the examples presented will now be introduced in detail.

C.1 Phase Equilibrium

Recall that if the package detects a region where the pair of phases are in phase equilibrium, a new element for the region will be created and incorporated into *regions*, assuming the availability of an appropriate thermodynamic subroutine to determine the number of phases and the state of each. However, as there is no utility for interfacing the subroutine to a package for predicting physical properties in the present implementation (and in the present version of *gPROMS*), we have constructed a library phase equilibrium routine, the name of which is *PhaseEquilibrium*.

From this law, we obtain not only the relevant physical properties of two phases in thermodynamic equilibrium, but also their distribution. This law is therefore decomposed into three discontinuous states; sub–cooled liquid (the temperature below the

bubble point), super–heated vapour (the temperature above the dew point) and the equilibrated two phases.

As the state transitions are reversible but the conditions of transitions are not directly related (*asymmetric and reversible*, see §4.2), a set of equations for describing this discontinuity should be expressed into the logical statement of CASE instead of IF.

A set of equations for this library are given below, where we assume that vapour `A` and liquid `B` are declared to be in phase equilibrium if both are present and the mechanism for this equilibration follows Raoult's law.

```
Temp_A    = Temp_B ;                                              (c.1)
Mass_A_B = Mass_A + Mass_B ;                                      (c.2)
Mass_A_B = MoleWeight * Mole_A_B ;                                (c.3)
Mass_B    = MoleWeight * Mole_B ;                                 (c.4)
Mole_A_B = Mole_A + Mole_B ;                                      (c.5)
Mole_A    = MoleFrac_A * SIGMA(Mole_A) ;                          (c.6)
Mole_B    = MoleFrac_B * SIGMA(Mole_B) ;                          (c.7)

CASE Phase_A_B OF
  WHEN LPhase :
    Mole_A = 0 ;                                                  (c.8)
    EquilConst_A_B = 0 ;                                          (c.9)
    SWITCH TO TwoPhase IF Temp_B > BubTemp_A_B
  WHEN TwoPhase :
    VapPress_B * MoleFrac_B = MoleFrac_A * Press_A_B ;            (c.10)
    MoleFrac_B = EquilConst_B_L * MoleFrac_L ;                    (c.11)
    SWITCH TO LPhase IF SIGMA(Mole_A) <= 0 ;
    SWITCH TO VPhase IF SIGMA(Mole_B) <= 0 ;
  WHEN VPhase :
    Mole_B = 0 ;                                                  (c.12)
    EquilConst_A_B = 1 ;                                          (c.13)
    SWITCH TO TwoPhase IF Temp_A < DewTemp_A_B ;
END
```

Note that the set of equations (c.1 – c.7) are categorised as a set of case invariant equations and those within the `CASE` statement (c.8 – c.13) are categorised as a set of case variant equations describing a set of three discrete states; sub–cooled liquid (c.8 and c.9),

equilibrium state (c.10 – c.11) and super–heated vapour (c.12 and c.13). Also note that the same number of equations in the CASE statement are maintained through the discrete states so as to make it possible to simulate the problem.

C.2 Bubble Rise

This law (namely *BubbleRise*) concerns the transfer between the vapour phase (dispersed into the liquid in the form of bubbles) and another vapour phase separated from the aggregated mixture. The driving force for this transfer is the density difference between the bubbles and the liquid. Several factors, in reality, have an effect on this transfer; the size of the bubbles enlarge gradually and the rising rate is accelerated as it rises through the liquid, wall effects, also bubble break–up, and coalescence. In other words, the rising rate of bubbles is a function of several variables of both phases; mass holdups, densities, etc. However, for simplicity we assume that the transfer rate is proportional to the mass of the bubble phase. The set of library equations, where `A` and `B` denote the bubble phase and liquid respectively, are given in equation c.14 and c.15 which describes the energy flow accompanied by the bubble phase.

```
Rate_A_B     = Const_A_B * Mass_B ;                    (c.14)
EnthFlow_A_B = SIGMA(Rate_A_B) * Enth_A ;              (c.15)
```

C.3 Containing Phase Transfer

This transfer law (namely *ContainingPhaseTransfer*) describes the natural production of the separate phase from the containing phase in an aggregate as the dispersed phase leaves the aggregate (for example, transfer law "BubbleRise").

Let's assume that uniform dispersion of `A1` in `B1`, containing phase and all bubbles (or droplets) of `A1` rise at the *same* rate.

Suppose the instantaneous volume of the aggregate is `Vol_A1_B1`, with volume fractions `VolFrac_A1` of `A1` and `VolFrac_B1` of `B1` (`VolFrac_A1 + VolFrac_B1 = 1`), and the volume rate of rise of `A1` is `VolRate_A1`.

Since all bubbles rise at the same rate, over interval `[t, t+dt]` a volume `VolRate_A1*dt` of `A1` will leave the aggregate and join separate `A1` (namely `A`). The same volume of `A1` will leave the bottom layer of the aggregate, leaving `B1` to join clear `B1` (namely `B`).

If the volume of this bottom layer is `dVol_A1_B1`, mass balance gives:

```
VolRate_A1*dt = VolFrac_A1*d(Vol_A1_B1)
```

and the amount of `B1` joining `B` is

```
VolFrac_B1*dVol_A1_B1 = (VolFrac_B1/VolFrac_A1)*VolRate_A1*dt
```

Note also that the volume fractions `VolFrac_A1`, `VolFrac_B1` left in the aggregate do not change with time. Thus the dynamic mass balances are :

```
dVol_A1/dt = VolRate_A1 = - VolFrac_A1*dVol_A1_B1/dt
dVol_B1/dt = (VolFrac_B1/VolFrac_A1)*VolRate_A1
           = - VolFrac_B1*dVol_A1_B1/dt
```

If this relation is expressed in mass rate terms, the equation c.16 is constructed. This transfer law has not yet been implemented in the current package.

```
Rate_B1_B * Vol_A1 * Den_A1 = Rate_A1_A * Vol_B1 * Den_B1 ;        (c.16)
EnthFlow_A_B = SIGMA(Rate_A_B) * Enth_A ;                          (c.17)
```

C.4 Irreversible Laminar Flow

Consider that fluid flows through a pipe where a non–return valve has been installed to prevent reverse flow. We assume that the transfer rate is slow enough so that the flow regime should be laminar. In order to take the irreversible flow directionality into account we need the fixed set of physical discontinuities described in terms of two discrete states, corresponding to whether or not fluid flows through the pipe. The transition between states depends on the sign of the value of the pressure difference between the mutually connected vessels; we can set up the fixed set of discrete cases describing that the fluid will flow if the sign is positive, otherwise there is no flow (reflecting the existence of the non–return valve).

Since the conditions for the transitions of the two discrete states can be characterised by a single logical expression, a set of equations structured into an `IF` statement is suitable for dealing with this discontinuity.

The set of equations for this law (namely *IrreversibleLaminarFlow*) is given in equation c.18 – c.21, where `S1` and `S2` denote the vessels connected with the pipe respectively. The laminar flow mechanism is described in equation c.19 (Ramirez, 1989).

```
DrivingForce = Press_S1 - Press_S2 ;                                    (c.18)
IF DrivingForce > 0 THEN
  Rate = Const * Den * MassFrac * DrivingForce ;                        (c.19)
ELSE
  Rate = 0 ;                                                            (c.20)
END
EnthFlow = SIGMA(Rate) * Enth ;                                         (c.21)
```

C.5 Irreversible Turbulent Flow

The set of equations for this law (namely *IrreversibleTurbulentFlow*) are essentially the same as those of the previous law, except for the fact that the pipe–flow regime is turbulent (Ramirez, 1989). The symbol `SQRT` in equation c.23 denotes square root, which is one of the built–in function in *gPROMS*. Note that the sign of the numeric value of `DrivingForce` is guaranteed to be positive due to the condition of the `IF` expression (`DrivingForce > 0`).

```
DrivingForce = Press_S1 - Press_S2 ;                                    (c.22)
IF DrivingForce > 0 THEN
  Rate = Const * Den * MassFrac * SQRT(DrivingForce) ;                  (c.23)
ELSE
  Rate = 0 ;                                                            (c.24)
END
EnthFlow = SIGMA(Rate) * Enth ;                                         (c.25)
```

C.6 Irreversible Pressure Driven Flow

This law (namely *IrreversiblePressureDrivenFlow*) is concerned with the transfer law for describing the flow mechanism of fluid through a pipe fitted with a non–return

valve, encompassing the two pipe–flow regimes; laminar and turbulent.

In order to allow transition between the two regimes, the conditions for the transitions between the flow regimes are described by a logical expression in terms of dimensionless Reynolds number defined in equation c.27 where `Const` represents the diameter of the pipe. The two transition conditions, in reality, are not directly related, so the equations for dealing with this discontinuity are structured into a CASE statement. The transition from turbulent to laminar regime is detected from the logical condition (the value of current Reynolds number less than `ReynoldsConst1`) and the reverse transition detected from the other logical condition (the value of current Reynolds number greater than `ReynoldsConst2`). For generality, these two constants may be specified by users to take the reality of individual situation into account, rather than equations c.28 and c.29.

In addition to this, flow irreversibility should be taken into account. Consequently, this law becomes a merged version of the two previous laws (§C.4 and §C.5). The set of discontinuities in this law (*case variant group*) are organised in such a way that the flow irreversibility is structured into an `IF` statement, which again embraces a `CASE` statement to deal with the transitions between the flow regimes.

```
DrivingForce    = Press_S1 - Press_S2 ;                                  (c.26)
(4/3.14) * SIGMA(Rate) = ReynoldsNo * Const * Viscosity                  (c.27)
ReynoldsConst1 = 2100 ;                                                  (c.28)
ReynoldsConst2 = 4000 ;                                                  (c.29)
EnthFlow        = SIGMA(Rate) * Enth ;                                   (c.30)
IF DrivingForce > 0 THEN
  CASE FlowType OF
    WHEN Turbulent :
      Rate = Const2 * Den * MassFrac * SQRT(DrivingForce) ;              (c.31)
      SWITCH TO Laminar IF ReynoldsNo < ReynoldsConst1 ;
    WHEN Laminar :
      Rate = Const3 * Den * MassFrac * DrivingForce ;                    (c.32)
      SWITCH TO Turbulent IF ReynoldsNo > ReynoldsConst2 ;
  END
ELSE
  Rate = 0 ;                                                             (c.33)
END
```

C.7 Pressure Driven Flow

This law (namely *PressureDrivenFlow*) concerns the same physical situation with the previous library, except the reverse flow, considering the possibility of a pressure–rise downstream through the pipe not fitted with a non–return valve. It is composed of three main parts as follows:

- a set of case invariant equations: c.34 – c.36,
- a set of discontinuities for the transitions between flow regimes, structured into a `CASE` statement: c.37 and c.38,
- a set of discontinuities for the selection of the relevant port of the mutually linked vessels according to the flow directionality, structured into an `IF` statement: c.39 – c.54.

In equation c.37 the absolute quantity of `DrivingForce` within the square root is taken to guarantee its positive value (`SQRT(ABS(DrivingForce))`) and then the flow directionality is determined by the sign of `DrivingForce` (`SGN(DrivingForce)`).

The suffix `P1` and `P2` used in the `IF` statements denote the name of the ports of the mutually connected vessels (denoted `S1` and `S2`).

It should be recognised in third part of this library that the selection of the relevant ports is embodied by specifying each discrete state in terms of port type, *relevant state variables* (viscosity, density, mass fraction and enthalpy) and the mass stream attributes (mass rate and enthalpy flow).

```
DrivingForce = Press_S1 - Press_S2 ;                                    (c.34)
(4/3.14) * SIGMA(Rate) = ReynoldsNo * Const * Viscosity                 (c.35)
EnthFlow = SGN(DrivingForce) * SIGMA(ABS(Rate)) * Enth ;                (c.36)
CASE FlowType OF
  WHEN Turbulent :
    Rate = Const2 * Den * MassFrac * SGN(DrivingForce) *
           SQRT(ABS(DrivingForce)) ;                                    (c.37)
    SWITCH TO Laminar IF ReynoldsNo < 2100 ;
  WHEN Laminar :
    Rate = Const3 * Den * MassFrac * DrivingForce ;                     (c.38)
    SWITCH TO Turbulent IF ReynoldsNo > 4000 ;
```

```
END
IF DrivingForce > 0 THEN
  PortType_P1 = OUTLET ;                                    (c.39)
  PortType_P2 = INLET ;                                     (c.40)
  Rate        = Rate_P1 ;                                   (c.41)
  EnthFlow    = EnthFlow_P1 ;                               (c.42)
  Viscosity   = Viscosity_P1 ;                              (c.43)
  Den         = Den_P1 ;                                    (c.44)
  MassFrac    = MassFrac_P1 ;                               (c.45)
  Enth        = Enth_P1 ;                                   (c.46)
ELSE
  PortType_P2 = OUTLET ;                                    (c.47)
  PortType_P2 = INLET ;                                     (c.48)
  Rate        = Rate_P2 ;                                   (c.49)
  EnthFlow    = EnthFlow_P2 ;                               (c.50)
  Viscosity   = Viscosity_P2 ;                              (c.51)
  Den         = Den_P2 ;                                    (c.52)
  MassFrac    = MassFrac_P2 ;                               (c.53)
  Enth        = Enth_P2 ;                                   (c.54)
END
```

C.8 Weir Over Flow

This law (namely *WeirOverFlow*) describes the behaviour of the flow over a weir. The mechanism of this law is based on the modified Francis formula (Pantelides, 1996). The discontinuity for this behaviour can be specified in terms of two states, corresponding to whether or not fluid flows over the weir. However, as the weir may be considered as a type of port, this discontinuity has been already considered in the course of specifying a set of discontinuities for a port (see §4.3.1.4). This law therefore only describes the flow mechanism itself. The symbol `Level` and `Z` in equation c.55 denote the level of the phase flowing over the weir and the position of the port (weir).

```
Rate     = Const * Den * ABS(Level - Z)^1.5 ;             (c.55)
EnthFlow = SIGMA(Rate) * Enth ;                           (c.56)
```

C.9 Static Pressure Driven Flow

This law (namely *StaticPressureDrivenFlow*) describes the flow mechanism driven from static liquid head pressure. The velocity is $\sqrt{2gh}$ where g is gravitational constant and h represent the liquid level. This relation is expressed in terms of mass rate at equation c.57.

```
Rate     = Const * Den * SQRT(2 * 9.8 * ABS(Level)) ;          (c.57)
EnthFlow = SIGMA(Rate) * Enth ;                                (c.58)
```

Appendix D

Simulation Input Files

D.1 Flash Drum

```
DECLARE

TYPE
  Mass_rate        =  50     :  -1E-1     :  1E4     UNIT = "kg/sec"
  Temperature      =  100    :  -1E-1     :  1E4     UNIT = "K"
  Length           =  15     :  -1E-1     :  1E2     UNIT = "m"
  Enthalpy         =  700    :  -1E7      :  1E4     UNIT = "kJ/kg"
  Int_Energy       =  600    :  -1E9      :  1E4     UNIT = "kJ/kg"
  Volume           =  0.5    :  -1E-1     :  1E1     UNIT = "m3"
  Pressure         =  43     :  -1E-1     :  1E4     UNIT = "kPa"
  Enthalpy_Flow    =  1E3    :  -1E9      :  1E7     UNIT = "kJ/sec"
  Mass             =  5      :  -1E-1     :  1E9     UNIT = "kg"
  Mole             =  0.1    :  -1E-1     :  1E2     UNIT = "kmole"
  Density          =  100    :  -1E-1     :  1E5     UNIT = "kg/m3"
  Viscosity        =  5E-5   :  -1E-1     :  1E2     UNIT = "Pa.s"
  Velocity         =  1E1    :  -1E-1     :  1E4     UNIT = "m/s"
  Fraction         =  0.5    :  -1E-1     :  10
  NoType           =  200    :  -1E9      :  1E9
  Positive         =  5      :  -1E-3     :  1E9


STREAM
 MassStream    IS Mass_Rate, Enthalpy_Flow, Fraction, Positive
 EnergyStream  IS Enthalpy_Flow

END

# ===================================================================

# ##############################
# BEGINNING of generated model #
# ##############################


MODEL  m_R1

PARAMETER
  NoComp                              AS  INTEGER

VARIABLE
  Ratio_P                             AS  Fraction
  Rate_P                              AS  Array(NoComp) of Mass_Rate
  Den_P                               AS  Density
  Press_P                             AS  Pressure
```

```
  EnthFlow_P                              AS  Enthalpy_Flow
  PhaseType_P                             AS  Positive
  MassFrac_P                              AS  Array(NoComp) of Fraction
  Viscosity_P                             AS  Viscosity
  Enth_P                                  AS  Enthalpy

STREAM
  P : Rate_P, EnthFlow_P, Ratio_P, PhaseType_P  AS  MassStream

END  # end of MODEL m_R1


MODEL  m_R2

PARAMETER
  NoComp                                  AS  INTEGER

VARIABLE
  Ratio_P                                 AS  Fraction
  Rate_P                                  AS  Array(NoComp) of Mass_Rate
  Press_P                                 AS  Pressure
  EnthFlow_P                              AS  Enthalpy_Flow
  PhaseType_P                             AS  Positive

STREAM
  P : Rate_P, EnthFlow_P, Ratio_P, PhaseType_P  AS  MassStream

END  # end of MODEL m_R2


MODEL  m_R3

PARAMETER
  NoComp                                  AS  INTEGER

VARIABLE
  Ratio_P                                 AS  Fraction
  Rate_P                                  AS  Array(NoComp) of Mass_Rate
  Press_P                                 AS  Pressure
  EnthFlow_P                              AS  Enthalpy_Flow
  PhaseType_P                             AS  Positive

STREAM
  P : Rate_P, EnthFlow_P, Ratio_P, PhaseType_P  AS  MassStream

END  # end of MODEL m_R3


MODEL  m_FlashDrum

PARAMETER
  NoComp                                  AS  INTEGER
  vapour                                  AS  INTEGER
  vapour_liquid                           AS  INTEGER
  liquid                                  AS  INTEGER
  Z_Pi1                                   AS  REAL
  Z_Po1                                   AS  REAL
  Z_Po2                                   AS  REAL
  height                                  AS  REAL
  Const_B_V                               AS  Array(NoComp) of REAL
  diameter                                AS  REAL

VARIABLE
  Mass_B_L                                AS  Array(NoComp) of Mass
  Mass_B                                  AS  Array(NoComp) of Mass
  Mass_V                                  AS  Array(NoComp) of Mass
  Mass_L                                  AS  Array(NoComp) of Mass
  Rate_B_V                                AS  Array(NoComp) of Mass_Rate
```

```
Rate_B_L_Po1                AS  Array(NoComp) of Mass_Rate
Rate_B_L_Po2                AS  Array(NoComp) of Mass_Rate
Rate_B_L_Pi1                AS  Array(NoComp) of Mass_Rate
Rate_V_Pi1                  AS  Array(NoComp) of Mass_Rate
Rate_Pi1                    AS  Array(NoComp) of Mass_Rate
Rate_V_Po1                  AS  Array(NoComp) of Mass_Rate
Rate_V_Po2                  AS  Array(NoComp) of Mass_Rate
Rate_Po1                    AS  Array(NoComp) of Mass_Rate
Rate_Po2                    AS  Array(NoComp) of Mass_Rate
Mole_B                      AS  Array(NoComp) of Mole
Mole_L                      AS  Array(NoComp) of Mole
Mole_B_L                    AS  Array(NoComp) of Mole
Den_B_L                     AS  Density
Den_B                       AS  Density
Den_V                       AS  Density
Den_L                       AS  Density
Den_Po1                     AS  Density
Den_Po2                     AS  Density
Viscosity_B_L               AS  Viscosity
Viscosity_V                 AS  Viscosity
Viscosity_Po2               AS  Viscosity
Viscosity_Po1               AS  Viscosity
Press_B                     AS  Pressure
Press_V                     AS  Pressure
Press_FlashDrum             AS  Pressure
Press_L                     AS  Pressure
Press_B_L                   AS  Pressure
VapPress_L                  AS  Array(NoComp) of Pressure
DewTemp_B_L                 AS  Temperature
Temp_B                      AS  Temperature
Temp_V                      AS  Temperature
Temp_L                      AS  Temperature
BubTemp_B_L                 AS  Temperature
Vol_B                       AS  Volume
Vol_B_L                     AS  Volume
Vol_V                       AS  Volume
Vol_L                       AS  Volume
Vol_FlashDrum               AS  Volume
Enth_Po1                    AS  Enthalpy
Enth_Po2                    AS  Enthalpy
Enth_B                      AS  Enthalpy
Enth_V                      AS  Enthalpy
Enth_L                      AS  Enthalpy
Enth_B_L                    AS  Enthalpy
IntEnergy_B_L               AS  Int_Energy
IntEnergy_L                 AS  Int_Energy
IntEnergy_V                 AS  Int_Energy
IntEnergy_B                 AS  Int_Energy
EnthFlow_B_V                AS  Enthalpy_Flow
EnthFlow_V_Po2              AS  Enthalpy_Flow
EnthFlow_B_L_Po2            AS  Enthalpy_Flow
EnthFlow_Po2                AS  Enthalpy_Flow
EnthFlow_V_Po1              AS  Enthalpy_Flow
EnthFlow_B_L_Po1            AS  Enthalpy_Flow
EnthFlow_Po1                AS  Enthalpy_Flow
EnthFlow_V_Pi1              AS  Enthalpy_Flow
EnthFlow_B_L_Pi1            AS  Enthalpy_Flow
EnthFlow_Pi1                AS  Enthalpy_Flow
Ratio_B                     AS  Fraction
Ratio_V                     AS  Fraction
Ratio_Pi1                   AS  Fraction
Ratio_Po2                   AS  Fraction
Ratio_Po1                   AS  Fraction

MoleFrac_L                  AS  Array(NoComp) of Fraction
MassFrac_B_L                AS  Array(NoComp) of Fraction
MassFrac_L                  AS  Array(NoComp) of Fraction
MassFrac_V                  AS  Array(NoComp) of Fraction
```

```
MassFrac_Po2                    AS  Array(NoComp) of Fraction
MoleFrac_B                      AS  Array(NoComp) of Fraction
MassFrac_B                      AS  Array(NoComp) of Fraction
MassFrac_Po1                    AS  Array(NoComp) of Fraction
Top_B_L                         AS  Positive
Level_V                         AS  Positive
Bot_V                           AS  Positive
Level_B_L                       AS  Positive
PhaseType_B_L                   AS  Positive
PhaseType_L                     AS  Positive
PhaseType_V                     AS  Positive
PhaseType_Po2                   AS  Positive
PhaseType_B                     AS  Positive
PhaseType_Po1                   AS  Positive
PhaseType_Pi1                   AS  Positive
Bot_B_L                         AS  Positive
area                            AS  Positive
Top_V                           AS  Positive
EquilConst_B_L                  AS  Array(NoComp) of Positive
MoleWeight                      AS  Array(NoComp) of Positive

# ================================ #
# Variables for Physical Properties #
# ================================ #

Enth_Vo                         AS ARRAY(NoComp) OF Enthalpy
Enth_Bo                         AS ARRAY(NoComp) OF Enthalpy
Enth_Lo                         AS ARRAY(NoComp) OF Enthalpy
Heat_Lo                         AS ARRAY(NoComp) OF Enthalpy

Temp_ro                         AS ARRAY(NoComp) OF Fraction
Den_Vo                          AS ARRAY(NoComp) OF Density
Den_Bo                          AS ARRAY(NoComp) OF Density
Temp_co                         AS ARRAY(NoComp) OF Temperature

Viscosity_B                     As Viscosity
Viscosity_L                     As Viscosity
Viscosity_Vo                    As ARRAY(NoComp) OF Viscosity
Viscosity_Bo                    As ARRAY(NoComp) OF Viscosity
Viscosity_Lo                    As ARRAY(NoComp) OF Viscosity
Av, Bv, Cv, Dv                  AS ARRAY(NoComp) OF NoType
Al, Bl, Cl, Dl, El              AS ARRAY(NoComp) OF NoType

Temp_r                          AS Fraction
Press_c                         AS Pressure

Press_co                        AS ARRAY(NoComp) OF Positive
Temp_c                          AS Temperature
M                               AS Positive
Z                               AS NoType
Zo                              AS ARRAY(NoComp) OF NoType

R                               AS NoType
A, B, C, D, E                   AS ARRAY(NoComp) OF NoType
#  F, G                          AS ARRAY(NoComp) OF NoType
vap1, vap2, vap3, vap4, vap5    AS ARRAy(NoComp) OF NoType
Wo                              AS ARRAY(NoComp) OF NoType


# ================================ #
# dummy variable for visualisation #
# ================================ #

Sum_Mass_V,
Der_IntEnergy_V,
Sum_Mass_B_L,
Der_Mass_B_L,
Sum_Mass_B,
```

```
  Sum_Mass_L,
  Sum_Rate_B_V,
  Sum_Rate_Pi1,
  Sum_Rate_Po1,
  Sum_Rate_Po2,
  VapPress_1,
  VapPress_2,
  Equil_1,
  Equil_2                                    AS NoType

STREAM
  Pi1 : Rate_Pi1, EnthFlow_Pi1, Ratio_Pi1, PhaseType_Pi1   AS  MassStream
  Po1 : Rate_Po1, EnthFlow_Po1, Ratio_Po1, PhaseType_Po1   AS  MassStream
  Po2 : Rate_Po2, EnthFlow_Po2, Ratio_Po2, PhaseType_Po2   AS  MassStream

SELECTOR
  Phase_B_L    AS   (LPhase, TwoPhase, VPhase)

SET
  vapour                          := 1 ;
  Z_Pi1                           := 1.500000 ;
  Z_Po1                           := 0.500000 ;
  Z_Po2                           := 2.500000 ;
  height                          := 3.000000 ;
  vapour_liquid                   := 3 ;
  liquid                          := 2 ;
  diameter                        := 1.000000 ;

EQUATION

  # case invariant mass balance
  $Mass_V = Rate_V_Pi1 - Rate_V_Po1 - Rate_V_Po2 +  Rate_B_V ;

  $Mass_B_L = Rate_B_L_Pi1 - Rate_B_L_Po1 - Rate_B_L_Po2  - Rate_B_V ;


  # case invariant energy balance
  $IntEnergy_V * SIGMA(Mass_V) + IntEnergy_V * SIGMA($Mass_V) =
    EnthFlow_V_Pi1 - EnthFlow_V_Po1 - EnthFlow_V_Po2 + EnthFlow_B_V ;

  $IntEnergy_B_L * SIGMA(Mass_B_L) + IntEnergy_B_L * SIGMA($Mass_B_L) =
    EnthFlow_B_L_Pi1 - EnthFlow_B_L_Po1 - EnthFlow_B_L_Po2 - EnthFlow_B_V ;

  # ratio of dispersed phase
  Ratio_V = 1 ;
  SIGMA(Mass_B) = Ratio_B * SIGMA(Mass_B_L) ;

  # Equilibrium :
  Temp_B = Temp_L ;
  Mass_B_L = Mass_B + Mass_L ;
  Mass_B_L = MoleWeight * Mole_B_L ;
  Mole_B_L = Mole_B + Mole_L ;
  Mass_B = MoleWeight * Mole_B ;
  Mole_B = MoleFrac_B * SIGMA(Mole_B) ;
  Mole_L = MoleFrac_L * SIGMA(Mole_L) ;
  CASE Phase_B_L OF
    WHEN LPhase :
      Mole_B = 0 ;
      EquilConst_B_L = 0 ;
      SWITCH TO TwoPhase IF Temp_L > BubTemp_B_L ;
    WHEN TwoPhase :
      VapPress_L * MoleFrac_L =  MoleFrac_B * Press_B_L ;
      MoleFrac_B = EquilConst_B_L * MoleFrac_L ;
      SWITCH TO LPhase IF SIGMA(Mole_B) <= 0 ;
      SWITCH TO VPhase IF SIGMA(Mole_L) <= 0 ;
    WHEN VPhase :
      Mole_L = 0 ;
```

```
    EquilConst_B_L = 1 ;
    SWITCH TO TwoPhase IF Temp_B < DewTemp_B_L ;
END

# BubbleRise :
Rate_B_V = Const_B_V * Mass_B ;
EnthFlow_B_V = SIGMA(Rate_B_V) * Enth_B ;

# mass = mass fraction * total mass
Mass_B_L = MassFrac_B_L * SIGMA(Mass_B_L) ;
Mass_L = MassFrac_L * SIGMA(Mass_L) ;
Mass_B = MassFrac_B * SIGMA(Mass_B) ;
Mass_V = MassFrac_V * SIGMA(Mass_V) ;

# total mass = density * volume
SIGMA(Mass_B_L) = Den_B_L * Vol_B_L ;
SIGMA(Mass_L) = Den_L * Vol_L ;
SIGMA(Mass_B) = Den_B * Vol_B ;
SIGMA(Mass_V) = Den_V * Vol_V ;

# phase type
PhaseType_B_L = vapour_liquid ;
PhaseType_L = liquid ;
PhaseType_B = vapour ;
PhaseType_V = vapour ;

# volume relationship
Vol_B_L = Vol_B + Vol_L ;
Vol_FlashDrum = Vol_V + Vol_B_L ;

# uniform pressure within vessel
Press_FlashDrum = Press_B_L ;
Press_FlashDrum = Press_L ;
Press_FlashDrum = Press_B ;
Press_FlashDrum = Press_V ;

# phase bound : upper/low bound of phase volume = level
Top_V = Level_V ;
Top_B_L = Level_B_L ;
Top_V = height ;
Bot_B_L = 0 ;
Bot_V = Top_B_L ;

# phase volume : volume = area * (top - bottom)
area = (3.14/4) * diameter^2 ;
Vol_B_L = area * (Top_B_L - Bot_B_L) ;
Vol_V = area * (Top_V - Bot_V) ;

# discontinuity on input port, "Pi1"
IF PhaseType_Pi1 = vapour  THEN
  IF Bot_V < Z_Pi1 AND Z_Pi1 <= Top_V  THEN
    Rate_V_Pi1 = Rate_Pi1 ;
    EnthFlow_V_Pi1 = EnthFlow_Pi1 ;
    Rate_B_L_Pi1 = 0 ;
    EnthFlow_B_L_Pi1 = 0 ;
  ELSE
    Rate_B_L_Pi1 = Rate_Pi1 ;
    EnthFlow_B_L_Pi1 = EnthFlow_Pi1 ;
    Rate_V_Pi1 = 0 ;
    EnthFlow_V_Pi1 = 0 ;
  END
ELSE
  Rate_B_L_Pi1 = Rate_Pi1 ;
  EnthFlow_B_L_Pi1 = EnthFlow_Pi1 ;
  Rate_V_Pi1 = 0 ;
  EnthFlow_V_Pi1 = 0 ;
END
# end of discontinuity on input port, "Pi1"
```

```
# discontinuity on output port, "Po1"
IF Bot_V < Z_Po1 AND Z_Po1 <= Top_V  THEN
  MassFrac_V = MassFrac_Po1 ;
  Viscosity_V = Viscosity_Po1 ;
  Den_V = Den_Po1 ;
  Enth_V = Enth_Po1 ;
  Rate_V_Po1 = Rate_Po1 ;
  EnthFlow_V_Po1 = EnthFlow_Po1 ;
  Ratio_V = Ratio_Po1 ;
  PhaseType_V = PhaseType_Po1 ;
  Rate_B_L_Po1 = 0 ;
  EnthFlow_B_L_Po1 = 0 ;
ELSE
  MassFrac_B_L = MassFrac_Po1 ;
  Viscosity_B_L = Viscosity_Po1 ;
  Den_B_L = Den_Po1 ;
  Enth_B_L = Enth_Po1 ;
  Rate_B_L_Po1 = Rate_Po1 ;
  EnthFlow_B_L_Po1 = EnthFlow_Po1 ;
  Ratio_B = Ratio_Po1 ;
  PhaseType_B_L = PhaseType_Po1 ;
  Rate_V_Po1 = 0 ;
  EnthFlow_V_Po1 = 0 ;
END
# end of discontinuity on output port, "Po1"


# discontinuity on output port, "Po2"
IF Bot_V < Z_Po2 AND Z_Po2 <= Top_V  THEN
  MassFrac_V = MassFrac_Po2 ;
  Viscosity_V = Viscosity_Po2 ;
  Den_V = Den_Po2 ;
  Enth_V = Enth_Po2 ;
  Rate_V_Po2 = Rate_Po2 ;
  EnthFlow_V_Po2 = EnthFlow_Po2 ;
  Ratio_V = Ratio_Po2 ;
  PhaseType_V = PhaseType_Po2 ;
  Rate_B_L_Po2 = 0 ;
  EnthFlow_B_L_Po2 = 0 ;
ELSE
  MassFrac_B_L = MassFrac_Po2 ;
  Viscosity_B_L = Viscosity_Po2 ;
  Den_B_L = Den_Po2 ;
  Enth_B_L = Enth_Po2 ;
  Rate_B_L_Po2 = Rate_Po2 ;
  EnthFlow_B_L_Po2 = EnthFlow_Po2 ;
  Ratio_B = Ratio_Po2 ;
  PhaseType_B_L = PhaseType_Po2 ;
  Rate_V_Po2 = 0 ;
  EnthFlow_V_Po2 = 0 ;
END
# end of discontinuity on output port, "Po2"


  # ================== #
  # Physical Properties #
  # ================== #

  # vapour pressure of L
#  VapPress_L * 0.145 = 10^(F - G/((1.8 * Temp_L - 459.4) + 382)) ;
   VapPress_L = 1E-3 * EXP(vap1 + vap2/Temp_L + vap3*LOG(Temp_L) +
                vap4*Temp_L^vap5) ;

  # Application Range :
  # Propane : 228K ~ 366K
  # Butane  : 228K ~ 421K
```

```
# pure enthalpies of V
Enth_Vo * 0.43 = A * ((Temp_V * 9/5)/100) + B * ((Temp_V * 9/5)/100)^2 +
  C * 1E-2 * ((Temp_V * 9/5)/100)^3 + D * (100/(Temp_V * 9/5)) + E ;

# enthalpy of V
Enth_V = SIGMA(MassFrac_V * Enth_Vo) ;

# pure enthalpies of B
Enth_Bo * 0.43 = A * ((Temp_B * 9/5)/100) + B * ((Temp_B * 9/5)/100)^2 +
  C * 1E-2 * ((Temp_B * 9/5)/100)^3 + D * (100/(Temp_B * 9/5)) + E ;

# enthalpy of B
Enth_B = SIGMA(MassFrac_B * Enth_Bo) ;

# heats of vapourisation of pure L
Heat_Lo * MoleWeight = (R * Temp_co) * (7.08 * (1 - Temp_ro)^0.354 +
  10.95 * Wo * (1-Temp_ro)^0.456) ;
Temp_ro * Temp_co = Temp_L ;

# pure enthapies of L
Enth_Lo = Enth_Bo - Heat_Lo ;

# enthalpy of L
Enth_L = SIGMA(MassFrac_L * Enth_Lo) ;

# enthalpy of B_L
Enth_B_L = Ratio_B * Enth_B + (1 - Ratio_B) * Enth_L ;

# internal energy of V
Enth_V * SIGMA(Mass_V) = IntEnergy_V * SIGMA(Mass_V) +
                         SIGMA(Mass_V / MoleWeight) * R * Temp_V ;

  # internal energy of B_L
  Enth_B_L = IntEnergy_B_L + Press_B_L / Den_B_L ;
#  IntEnergy_B_L = IntEnergy_B * Ratio_B +
#                  (1 - Ratio_B) * IntEnergy_L ;

  # internal energy of B
  Enth_B * SIGMA(Mass_B) = IntEnergy_B * SIGMA(Mass_B) +
                           SIGMA(Mass_B / MoleWeight) * R * Temp_B ;

  # internal energy of L
  Enth_L  = IntEnergy_L + Press_L / Den_L ;

  # density of V
  Den_Vo * R * Temp_V = Press_V * MoleWeight ; # for pure component
  Den_V * SIGMA(MassFrac_V / Den_Vo) = 1 ;     # for mixture

  # density of B
  Den_Bo * R * Temp_L = Press_B * MoleWeight;  # for pure component
  Den_B * SIGMA(MassFrac_B / Den_Bo) = 1;      # for mixture

  # density of L
  Den_L * R * Temp_c * Z^(1 + (1 - Temp_r)^(2/7)) = Press_c * M ;
  Press_c = SIGMA(MoleFrac_L * Press_co) ;
  Temp_c = SIGMA(MoleFrac_L * Temp_co) ;
  Z = SIGMA(MoleFrac_L * Zo) ;
  Temp_r * Temp_c = Temp_L ;
  M = SIGMA(MoleFrac_L * MoleWeight) ;

  # viscosity of V
  Viscosity_Vo = Av * Temp_V^Bv / (1 + Cv / Temp_V + Dv / Temp_V^2) ;
  Viscosity_V = SIGMA(MassFrac_V * Viscosity_Vo) ;

  # viscosity of B_L
  Viscosity_Bo = (Av * Temp_B^Bv) / (1 + Cv / Temp_B + Dv / Temp_B^2) ;
  Viscosity_B = SIGMA(MassFrac_V * Viscosity_Bo) ;
```

```
Viscosity_Lo = EXP(A1 * (B1 / Temp_L) + C1 * LOG(Temp_L) +
               D1 * Temp_L^E1) ;
Viscosity_L = SIGMA(MassFrac_L * Viscosity_Lo) ;
Viscosity_B_L = Ratio_B * Viscosity_B +
                (1 - Ratio_B) * Viscosity_L ;


# ================================= #
# dummy variables for visualisation #
# ================================= #

  Sum_Mass_V = SIGMA(Mass_V) ;
  Der_IntEnergy_V = $IntEnergy_V;
  Sum_Mass_B_L = SIGMA(Mass_B_L) ;
  Der_Mass_B_L = SIGMA($Mass_B_L);
  Sum_Mass_B = SIGMA(Mass_B) ;
  Sum_Mass_L = SIGMA(Mass_L) ;
  Sum_Rate_B_V = SIGMA(Rate_B_V) ;
  Sum_Rate_Pi1 = SIGMA(Rate_Pi1);
  Sum_Rate_Po1 = SIGMA(Rate_Po1);
  Sum_Rate_Po2 = SIGMA(Rate_Po2);
  VapPress_1 = VapPress_L(1);
  VapPress_2 = VapPress_L(2);
  Equil_1 = EquilConst_B_L(1);
  Equil_2 = EquilConst_B_L(2);


END  # end of MODEL m_FlashDrum


MODEL  Flowsheet

PARAMETER
  NoComp                          AS  INTEGER
  Const2_C3                       AS  REAL
  Const2_C2                       AS  REAL
  Const2_C1                       AS  REAL
  Const3_C3                       AS  REAL
  Const3_C2                       AS  REAL
  Const3_C1                       AS  REAL
  Const_C1                        AS  REAL
  Const_C2                        AS  REAL
  Const_C3                        AS  REAL

VARIABLE
  DrivingForce_C3                 AS  NoType
  ReynoldsNo_C3                   AS  Positive
  ReynoldsConst1_C3               AS  Positive
  ReynoldsConst2_C3               AS  Positive
  DrivingForce_C2                 AS  NoType
  ReynoldsNo_C2                   AS  Positive
  ReynoldsConst1_C2               AS  Positive
  ReynoldsConst2_C2               AS  Positive
  DrivingForce_C1                 AS  NoType
  ReynoldsNo_C1                   AS  Positive
  ReynoldsConst1_C1               AS  Positive
  ReynoldsConst2_C1               AS  Positive

UNIT
  R1                              AS  m_R1
  R2                              AS  m_R2
  R3                              AS  m_R3
  FlashDrum                       AS  m_FlashDrum

SELECTOR
  FlowType_C1    AS  (Turbulent, Laminar)
  FlowType_C2    AS  (Turbulent, Laminar)
```

```
  FlowType_C3   AS  (Turbulent, Laminar)

EQUATION

                #==================================#
                # stream connections through ports #
                #==================================#
  R1.P                  IS      FlashDrum.Pi1 ;
  FlashDrum.Po1         IS      R2.P ;
  FlashDrum.Po2         IS      R3.P ;


                #==================================#
                # transfer law of each connection #
                #==================================#
# "IrreversiblePressureDrivenFlow" in connection, "C1"
DrivingForce_C1 = R1.Press_P - FlashDrum.Press_FlashDrum ;
(4/3.14) * SIGMA(R1.Rate_P) = ReynoldsNo_C1 * Const_C1 * R1.Viscosity_P ;
ReynoldsConst1_C1 = 2100 ;
ReynoldsConst2_C1 = 4000 ;
R1.EnthFlow_P = SIGMA(R1.Rate_P) * R1.Enth_P ;
IF DrivingForce_C1 > 0 THEN
  CASE FlowType_C1 OF
    WHEN Turbulent :
      R1.Rate_P = Const2_C1 * R1.Den_P * R1.MassFrac_P *
                  SQRT(DrivingForce_C1) ;
      SWITCH TO Laminar IF ReynoldsNo_C1 < ReynoldsConst1_C1 ;
    WHEN Laminar :
      R1.Rate_P = Const3_C1 * R1.Den_P * R1.MassFrac_P * DrivingForce_C1 ;
      SWITCH TO Turbulent IF ReynoldsNo_C1 > ReynoldsConst2_C1 ;
  END
ELSE
  R1.Rate_P = 0;
END


# "IrreversiblePressureDrivenFlow" in connection, "C2"
DrivingForce_C2 = FlashDrum.Press_FlashDrum - R2.Press_P ;
(4/3.14) * SIGMA(FlashDrum.Rate_Po1) = ReynoldsNo_C2 * Const_C2 *
                                       FlashDrum.Viscosity_Po1 ;
ReynoldsConst1_C2 = 2100 ;
ReynoldsConst2_C2 = 4000 ;
FlashDrum.EnthFlow_Po1 = SIGMA(FlashDrum.Rate_Po1) * FlashDrum.Enth_Po1 ;
IF DrivingForce_C2 > 0 THEN
  CASE FlowType_C2 OF
    WHEN Turbulent :
      FlashDrum.Rate_Po1 = Const2_C2 * FlashDrum.Den_Po1 *
                           FlashDrum.MassFrac_Po1 * SQRT(DrivingForce_C2) ;
      SWITCH TO Laminar IF ReynoldsNo_C2 < ReynoldsConst1_C2 ;
    WHEN Laminar :
      FlashDrum.Rate_Po1 = Const3_C2 * FlashDrum.Den_Po1 *
                           FlashDrum.MassFrac_Po1 * DrivingForce_C2 ;
      SWITCH TO Turbulent IF ReynoldsNo_C2 > ReynoldsConst2_C2 ;
  END
ELSE
  FlashDrum.Rate_Po1 = 0;
END


# "IrreversiblePressureDrivenFlow" in connection, "C3"
DrivingForce_C3 = FlashDrum.Press_FlashDrum - R3.Press_P ;
(4/3.14) * SIGMA(FlashDrum.Rate_Po2) = ReynoldsNo_C3 * Const_C3 *
                                       FlashDrum.Viscosity_Po2 ;
ReynoldsConst1_C3 = 2100 ;
ReynoldsConst2_C3 = 4000 ;
FlashDrum.EnthFlow_Po2 = SIGMA(FlashDrum.Rate_Po2) * FlashDrum.Enth_Po2 ;
IF DrivingForce_C3 > 0 THEN
  CASE FlowType_C3 OF
```

```
    WHEN Turbulent :
      FlashDrum.Rate_Po2 = Const2_C3 * FlashDrum.Den_Po2 *
                           FlashDrum.MassFrac_Po2 * SQRT(DrivingForce_C3) ;
      SWITCH TO Laminar IF ReynoldsNo_C3 < ReynoldsConst1_C3 ;
    WHEN Laminar :
      FlashDrum.Rate_Po2 = Const3_C3 * FlashDrum.Den_Po2 *
                           FlashDrum.MassFrac_Po2 * DrivingForce_C3 ;
      SWITCH TO Turbulent IF ReynoldsNo_C3 > ReynoldsConst2_C3 ;
  END
ELSE
  FlashDrum.Rate_Po2 = 0;
  END


END  # end of MODEL Flowsheet
# ================================================================== #

############################
# END of generated model #
############################


# ======= #
# Process #
# ======= #

PROCESS test

UNIT
  Plant AS Flowsheet

SET
WITHIN Plant DO
  NoComp         := 2 ;
  Const_C1       := 0.05 ;
  Const_C2       := 0.05 ;
  Const_C3       := 0.05 ;
  Const2_C1      := 5E-5 ;
  Const2_C2      := 1E-4 ;
  Const2_C3      := 1E-5 ;
  Const3_C1      := 5E-6 ;
  Const3_C2      := 1E-5 ;
  Const3_C3      := 1E-6 ;
  WITHIN R1 DO
    NoComp       := 2 ;
  END
  WITHIN R2 DO
    NoComp       := 2 ;
  END
  WITHIN R3 DO
    NoComp       := 2 ;
  END
  WITHIN FlashDrum DO
    NoComp       := 2 ;
    Const_B_V    := 3E-4 ;
  END
END

ASSIGN
  WITHIN Plant DO
    WITHIN R1 DO
      Press_P        := 4 * 1.013E2 ;
      Den_P          := 330 ;
      Enth_P         := 3.5E2 ;
      MassFrac_P     := [0.4, 0.6] ;
      Ratio_P        := 0.001 ;
      PhaseType_P    := 3 ;
      Viscosity_P    := 5E-5 ;
```

```
END # R1
WITHIN R2 DO
  Press_P        := 1.013E2 ;
END # R2
WITHIN R3 DO
  Press_P        := 1.013E2 ;
END # R3
WITHIN FlashDrum DO
  BubTemp_B_L    := 260 ;
  DewTemp_B_L    := 300 ;
  Temp_co(1)     := 369.82 ;
  Temp_co(2)     := 425.15 ;

  A(1)           := 8.03820 ;
  A(2)           := 8.29348 ;
  B(1)           := 3.49075 ;
  B(2)           := 3.46000 ;
  C(1)           := -3.96060 ;
  C(2)           := -4.02109 ;
  D(1)           := 27.52980 ;
  D(2)           := 30.35096 ;
  E(1)           := 166.170 ;
  E(2)           := 153.044 ;
  R              := 8.31433 ;      # [kJ/kmol K]
  Wo(1)          := 0.1454 ;
  Wo(2)          := 0.1928 ;
  MoleWeight(1)  := 44.09 ;
  MoleWeight(2)  := 58.12 ;

  vap1(1)        := 5.4276E1 ;
  vap1(2)        := 6.2570E1 ;
  vap2(1)        := -3.3680E3 ;
  vap2(2)        := -4.3220E3 ;
  vap3(1)        := -5.2610E0 ;
  vap3(2)        := -6.3640E0 ;
  vap4(1)        := 8.6000E-6 ;
  vap4(2)        := 6.8000E-6 ;
  vap5(1)        := 2.0000E0 ;
  vap5(2)        := 2.0000E0 ;

{
  F(1)           := 4.843 ;
  F(2)           := 5.273 ;
  G(1)           := 1245.3 ;
  G(2)           := 1747.2 ;
}

  Av(1)          := 2.2090E-6 ;
  Av(2)          := 1.0310E-5 ;
  Bv(1)          := 3.8240E-1 ;
  Bv(2)          := 2.0770E-1 ;
  Cv(1)          := 4.0500E2 ;
  Cv(2)          := 1.0055E3 ;
  Dv(1)          := 0 ;
  Dv(2)          := 8.1000E3 ;
  Al(1)          := -1.2832E1 ;
  Al(2)          := 7.5000E-1 ;
  Bl(1)          := 5.6634E2 ;
  Bl(2)          := 2.1870E2 ;
  Cl(1)          := 3.4688E-1 ;
  Cl(2)          := -1.7882 ;
  Dl(1)          := -3.5111E-26 ;
  Dl(2)          := -4.0000E-27 ;
  El(1)          := 1.0000E1 ;
  El(2)          := 1.0000E1 ;

{
  Den_L          := 600 ;
```

```
      Viscosity_V    := 7.9E-6 ;
      Viscosity_B_L  := 5E-5 ;
}

      Press_co(1)    := 4248 ;
      Press_co(2)    := 3795 ;
      Zo(1)          := 0.27664 ;
      Zo(2)          := 0.27331 ;

    END # within FlashDrum
  END # within Plant


PRESET
    PLANT.FLASHDRUM.ENTHFLOW_V_PO2 := 0.00000E+00 : -1.000E+09 : 1.000E+07 ;
    PLANT.FLASHDRUM.ENTHFLOW_B_L_PO1 := 0.00000E+00 : -1.000E+09 : 1.000E+07 ;
    PLANT.FLASHDRUM.ENTHFLOW_B_L_PO2 := 0.00000E+00 : -1.000E+09 : 1.000E+07 ;
    PLANT.FLASHDRUM.VOL_B := 2.80947E-02 : -1.000E-01 : 1.000E+01 ;
    PLANT.FLASHDRUM.PHASETYPE_B := 1.00000E+00 : -1.000E-03 : 1.000E+09 ;
    PLANT.FLASHDRUM.RATE_PI1(1) := 1.25494E-01 : -1.000E-01 : 1.000E+04 ;
    PLANT.FLASHDRUM.RATE_PI1(2) := 1.88241E-01 : -1.000E-01 : 1.000E+04 ;
    PLANT.FLASHDRUM.ENTHFLOW_PI1 := 1.09807E+02 : -1.000E+09 : 1.000E+07 ;
    PLANT.FLASHDRUM.RATIO_PI1 := 6.00000E-01 : -1.000E-01 : 1.000E+01 ;
    PLANT.FLASHDRUM.PHASETYPE_PI1 := 3.00000E+00 : -1.000E-03 : 1.000E+09 ;
    PLANT.FLASHDRUM.RATE_PO1(1) := 0.00000E+00 : -1.000E-01 : 1.000E+04 ;
    PLANT.FLASHDRUM.RATE_PO1(2) := 0.00000E+00 : -1.000E-01 : 1.000E+04 ;
    PLANT.FLASHDRUM.MASSFRAC_B(1) := 8.84494E-01 : -1.000E-01 : 1.000E+01 ;
    PLANT.FLASHDRUM.MASSFRAC_B(2) := 1.15506E-01 : -1.000E-01 : 1.000E+01 ;
    PLANT.FLASHDRUM.ENTH_LO(1) := 1.73454E+02 : -1.000E+07 : 1.000E+04 ;
    PLANT.FLASHDRUM.ENTH_LO(2) := 1.52137E+02 : -1.000E+07 : 1.000E+04 ;
    PLANT.FLASHDRUM.RATE_PO2(1) := 0.00000E+00 : -1.000E-01 : 1.000E+04 ;
    PLANT.FLASHDRUM.RATE_PO2(2) := 0.00000E+00 : -1.000E-01 : 1.000E+04 ;
    PLANT.FLASHDRUM.INTENERGY_B := 5.58220E+02 : -1.000E+09 : 1.000E+04 ;
    PLANT.FLASHDRUM.DEN_BO(1) := 1.03804E+00 : -1.000E-01 : 1.000E+05 ;
    PLANT.FLASHDRUM.DEN_BO(2) := 1.36836E+00 : -1.000E-01 : 1.000E+05 ;
    PLANT.FLASHDRUM.ENTH_PO1 := 7.07255E+02 : -1.000E+07 : 1.000E+04 ;
    PLANT.FLASHDRUM.MOLEFRAC_L(1) := 5.68255E-01 : -1.000E-01 : 1.000E+01 ;
    PLANT.FLASHDRUM.MOLEFRAC_L(2) := 4.31745E-01 : -1.000E-01 : 1.000E+01 ;
    PLANT.FLASHDRUM.ENTH_PO2 := 7.07255E+02 : -1.000E+07 : 1.000E+04 ;
    PLANT.FLASHDRUM.ENTHFLOW_B_V := 5.39479E-03 : -1.000E+09 : 1.000E+07 ;
    PLANT.FLASHDRUM.LEVEL_V := 3.00000E+00 : -1.000E-03 : 1.000E+09 ;
    PLANT.FLASHDRUM.EQUILCONST_B_L(1) := 1.60115E+00 : -1.000E-03 : 1.000E+09 ;
    PLANT.FLASHDRUM.EQUILCONST_B_L(2) := 2.08772E-01 : -1.000E-03 : 1.000E+09 ;
    PLANT.FLASHDRUM.VISCOSITY_BO(1) := 6.20289E-06 : -1.000E-01 : 1.000E+02 ;
    PLANT.FLASHDRUM.VISCOSITY_BO(2) := 5.58908E-06 : -1.000E-01 : 1.000E+02 ;
    PLANT.FLASHDRUM.PRESS_B := 4.36584E+01 : -1.000E-01 : 1.000E+04 ;
    PLANT.FLASHDRUM.LEVEL_B_L := 9.70524E-02 : -1.000E-03 : 1.000E+09 ;
    PLANT.FLASHDRUM.ENTHFLOW_PO1 := 0.00000E+00 : -1.000E+09 : 1.000E+07 ;
    PLANT.FLASHDRUM.RATIO_PO1 := 1.00000E+00 : -1.000E-01 : 1.000E+01 ;
    PLANT.FLASHDRUM.PHASETYPE_PO1 := 1.00000E+00 : -1.000E-03 : 1.000E+09 ;
    PLANT.FLASHDRUM.PRESS_C := 4.05242E+03 : -1.000E-01 : 1.000E+04 ;
    PLANT.FLASHDRUM.ENTHFLOW_PO2 := 0.00000E+00 : -1.000E+09 : 1.000E+07 ;
    PLANT.FLASHDRUM.RATIO_PO2 := 1.00000E+00 : -1.000E-01 : 1.000E+01 ;
    PLANT.FLASHDRUM.PHASETYPE_PO2 := 1.00000E+00 : -1.000E-03 : 1.000E+09 ;
    PLANT.FLASHDRUM.BOT_V := 9.70524E-02 : -1.000E-03 : 1.000E+09 ;
    PLANT.FLASHDRUM.BOT_B_L := 0.00000E+00 : -1.000E-03 : 1.000E+09 ;
    PLANT.FLASHDRUM.MOLE_B(1) := 6.01833E-04 : -1.000E-01 : 1.000E+02 ;
    PLANT.FLASHDRUM.MOLE_B(2) := 5.96210E-05 : -1.000E-01 : 1.000E+02 ;
    PLANT.FLASHDRUM.VOL_L := 4.80914E-02 : -1.000E-01 : 1.000E+01 ;
    PLANT.FLASHDRUM.PHASETYPE_L := 2.00000E+00 : -1.000E-03 : 1.000E+09 ;
    PLANT.FLASHDRUM.MASSFRAC_L(1) := 4.99615E-01 : -1.000E-01 : 1.000E+01 ;
    PLANT.FLASHDRUM.MASSFRAC_L(2) := 5.00385E-01 : -1.000E-01 : 1.000E+01 ;
    PLANT.FLASHDRUM.ENTH_VO(1) := 7.21461E+02 : -1.000E+07 : 1.000E+04 ;
    PLANT.FLASHDRUM.ENTH_VO(2) := 6.93049E+02 : -1.000E+07 : 1.000E+04 ;
    PLANT.FLASHDRUM.VOL_FLASHDRUM := 2.35500E+00 : -1.000E-01 : 1.000E+01 ;
    PLANT.FLASHDRUM.ENTH_B := 5.99106E+02 : -1.000E+07 : 1.000E+04 ;
    PLANT.FLASHDRUM.INTENERGY_L := 1.62717E+02 : -1.000E+09 : 1.000E+04 ;
    PLANT.FLASHDRUM.RATIO_V := 1.00000E+00 : -1.000E-01 : 1.000E+01 ;
```

```
PLANT.FLASHDRUM.TEMP_RO(1) := 6.03079E-01 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM.TEMP_RO(2) := 5.24593E-01 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM.VISCOSITY_LO(1) := 4.55853E-14 : -1.000E-01 : 1.000E+02 ;
PLANT.FLASHDRUM.VISCOSITY_LO(2) := 1.31680E-04 : -1.000E-01 : 1.000E+02 ;
PLANT.FLASHDRUM.DEN_PO1 := 8.77650E-01 : -1.000E-01 : 1.000E+05 ;
PLANT.FLASHDRUM.PRESS_L := 4.36584E+01 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM.MASS_B(1) := 2.65348E-02 : -1.000E-01 : 1.000E+09 ;
PLANT.FLASHDRUM.MASS_B(2) := 3.46517E-03 : -1.000E-01 : 1.000E+09 ;
PLANT.FLASHDRUM.DEN_PO2 := 8.77650E-01 : -1.000E-01 : 1.000E+05 ;
PLANT.FLASHDRUM.VISCOSITY_PO1 := 9.40000E-06 : -1.000E-01 : 1.000E+02 ;
PLANT.FLASHDRUM.PRESS_FLASHDRUM := 4.36584E+01 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM.VISCOSITY_PO2 := 9.40000E-06 : -1.000E-01 : 1.000E+02 ;
PLANT.FLASHDRUM.TEMP_B := 2.23031E+02 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM.MOLE_L(1) := 3.39611E-01 : -1.000E-01 : 1.000E+02 ;
PLANT.FLASHDRUM.MOLE_L(2) := 2.58027E-01 : -1.000E-01 : 1.000E+02 ;
PLANT.FLASHDRUM.VOL_V := 2.27881E+00 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM.PHASETYPE_V := 1.00000E+00 : -1.000E-03 : 1.000E+09 ;
PLANT.FLASHDRUM.TEMP_C := 3.93708E+02 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM.VOL_B_L := 7.61861E-02 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM.MASSFRAC_V(1) := 5.00000E-01 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM.MASSFRAC_V(2) := 5.00000E-01 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM.PHASETYPE_B_L := 3.00000E+00 : -1.000E-03 : 1.000E+09 ;
PLANT.FLASHDRUM.ENTH_L := 1.62787E+02 : -1.000E+07 : 1.000E+04 ;
PLANT.FLASHDRUM.INTENERGY_V := 6.57510E+02 : -1.000E+09 : 1.000E+04 ;
PLANT.FLASHDRUM.MASSFRAC_B_L(1) := 5.00000E-01 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM.MASSFRAC_B_L(2) := 5.00000E-01 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM.TOP_V := 3.00000E+00 : -1.000E-03 : 1.000E+09 ;
PLANT.FLASHDRUM.INTENERGY_B_L := 1.63113E+02 : -1.000E+09 : 1.000E+04 ;
PLANT.FLASHDRUM.TOP_B_L := 9.70524E-02 : -1.000E-03 : 1.000E+09 ;
PLANT.FLASHDRUM.DEN_VO(1) := 7.71719E-01 : -1.000E-01 : 1.000E+05 ;
PLANT.FLASHDRUM.DEN_VO(2) := 1.01729E+00 : -1.000E-01 : 1.000E+05 ;
PLANT.FLASHDRUM.DEN_B := 1.06782E+00 : -1.000E-01 : 1.000E+05 ;
PLANT.FLASHDRUM.VISCOSITY_VO(1) := 8.32490E-06 : -1.000E-01 : 1.000E+02 ;
PLANT.FLASHDRUM.VISCOSITY_VO(2) := 7.58946E-06 : -1.000E-01 : 1.000E+02 ;
PLANT.FLASHDRUM.M := 5.01474E+01 : -1.000E-03 : 1.000E+09 ;
PLANT.FLASHDRUM.RATE_V_PI1(1) := 0.00000E+00 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM.RATE_V_PI1(2) := 0.00000E+00 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM.PRESS_V := 4.36584E+01 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM.VISCOSITY_B := 5.89599E-06 : -1.000E-01 : 1.000E+02 ;
PLANT.FLASHDRUM.MASS_L(1) := 1.49735E+01 : -1.000E-01 : 1.000E+09 ;
PLANT.FLASHDRUM.MASS_L(2) := 1.49965E+01 : -1.000E-01 : 1.000E+09 ;
PLANT.FLASHDRUM.RATE_B_L_PI1(1) := 1.25494E-01 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM.RATE_B_L_PI1(2) := 1.88241E-01 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM.PRESS_B_L := 4.36584E+01 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM.TEMP_L := 2.23031E+02 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM.AREA := 7.85000E-01 : -1.000E-03 : 1.000E+09 ;
PLANT.FLASHDRUM.MOLE_B_L(1) := 3.40213E-01 : -1.000E-01 : 1.000E+02 ;
PLANT.FLASHDRUM.MOLE_B_L(2) := 2.58087E-01 : -1.000E-01 : 1.000E+02 ;
PLANT.FLASHDRUM.RATE_V_PO1(1) := 0.00000E+00 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM.RATE_V_PO1(2) := 0.00000E+00 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM.ENTH_V := 7.07255E+02 : -1.000E+07 : 1.000E+04 ;
PLANT.FLASHDRUM.RATE_B_L_PO1(1) := 0.00000E+00 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM.RATE_B_L_PO1(2) := 0.00000E+00 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM.RATE_V_PO2(1) := 0.00000E+00 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM.RATE_V_PO2(2) := 0.00000E+00 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM.ENTH_B_L := 1.63224E+02 : -1.000E+07 : 1.000E+04 ;
PLANT.FLASHDRUM.RATE_B_L_PO2(1) := 0.00000E+00 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM.RATE_B_L_PO2(2) := 0.00000E+00 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM.DEN_L := 6.23188E+02 : -1.000E-01 : 1.000E+05 ;
PLANT.FLASHDRUM.TEMP_R := 5.66487E-01 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM.VISCOSITY_L := 6.58909E-05 : -1.000E-01 : 1.000E+02 ;
PLANT.FLASHDRUM.MASS_V(1) := 1.00000E+00 : -1.000E-01 : 1.000E+09 ;
PLANT.FLASHDRUM.MASS_V(2) := 1.00000E+00 : -1.000E-01 : 1.000E+09 ;
PLANT.FLASHDRUM.MASS_B_L(1) := 1.50000E+01 : -1.000E-01 : 1.000E+09 ;
PLANT.FLASHDRUM.MASS_B_L(2) := 1.50000E+01 : -1.000E-01 : 1.000E+09 ;
PLANT.FLASHDRUM.HEAT_LO(1) := 4.28858E+02 : -1.000E+07 : 1.000E+04 ;
PLANT.FLASHDRUM.HEAT_LO(2) := 4.22423E+02 : -1.000E+07 : 1.000E+04 ;
PLANT.FLASHDRUM.Z := 2.75202E-01 : -1.000E+09 : 1.000E+09 ;
```

```
PLANT.FLASHDRUM.TEMP_V := 3.00000E+02 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM.RATE_B_V(1) := 7.95751E-06 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM.RATE_B_V(2) := 1.04249E-06 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM.ENTHFLOW_V_PI1 := 0.00000E+00 : -1.000E+09 : 1.000E+07 ;
PLANT.FLASHDRUM.ENTH_BO(1) := 6.02311E+02 : -1.000E+07 : 1.000E+04 ;
PLANT.FLASHDRUM.ENTH_BO(2) := 5.74560E+02 : -1.000E+07 : 1.000E+04 ;
PLANT.FLASHDRUM.ENTHFLOW_B_L_PI1 := 1.09807E+02 : -1.000E+09 : 1.000E+07 ;
PLANT.FLASHDRUM.RATIO_B := 1.0000E-03 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM.MASSFRAC_PO1(1) := 5.00000E-01 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM.MASSFRAC_PO1(2) := 5.00000E-01 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM.DEN_V := 8.77650E-01 : -1.000E-01 : 1.000E+05 ;
PLANT.FLASHDRUM.MASSFRAC_PO2(1) := 5.00000E-01 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM.MASSFRAC_PO2(2) := 5.00000E-01 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM.MOLEFRAC_B(1) := 9.09864E-01 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM.MOLEFRAC_B(2) := 9.01362E-02 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM.DEN_B_L := 3.93772E+02 : -1.000E-01 : 1.000E+05 ;
PLANT.FLASHDRUM.VISCOSITY_V := 7.95718E-06 : -1.000E-01 : 1.000E+02 ;
PLANT.FLASHDRUM.VISCOSITY_B_L := 6.58309E-05 : -1.000E-01 : 1.000E+02 ;
PLANT.FLASHDRUM.VAPPRESS_L(1) := 6.99037E+01 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM.VAPPRESS_L(2) := 9.11464E+00 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM.ENTHFLOW_V_PO1 := 0.00000E+00 : -1.000E+09 : 1.000E+07 ;
PLANT.R3.RATIO_P := 1.00000E+00 : -1.000E-01 : 1.000E+01 ;
PLANT.R3.PHASETYPE_P := 1.00000E+00 : -1.000E-03 : 1.000E+09 ;
PLANT.R3.RATE_P(1) := 0.00000E+00 : -1.000E-01 : 1.000E+04 ;
PLANT.R3.RATE_P(2) := 0.00000E+00 : -1.000E-01 : 1.000E+04 ;
PLANT.R3.ENTHFLOW_P := 0.00000E+00 : -1.000E+09 : 1.000E+07 ;
PLANT.R2.RATIO_P := 1.00000E+00 : -1.000E-01 : 1.000E+01 ;
PLANT.R2.PHASETYPE_P := 1.00000E+00 : -1.000E-03 : 1.000E+09 ;
PLANT.R2.RATE_P(1) := 0.00000E+00 : -1.000E-01 : 1.000E+04 ;
PLANT.R2.RATE_P(2) := 0.00000E+00 : -1.000E-01 : 1.000E+04 ;
PLANT.R2.ENTHFLOW_P := 0.00000E+00 : -1.000E+09 : 1.000E+07 ;
PLANT.R1.RATE_P(1) := 1.25494E-01 : -1.000E-01 : 1.000E+04 ;
PLANT.R1.RATE_P(2) := 1.88241E-01 : -1.000E-01 : 1.000E+04 ;
PLANT.R1.ENTHFLOW_P := 1.09807E+02 : -1.000E+09 : 1.000E+07 ;
PLANT.DRIVINGFORCE_C1 := 3.61542E+02 : -1.000E+09 : 1.000E+09 ;
PLANT.DRIVINGFORCE_C2 := -5.76416E+01 : -1.000E+09 : 1.000E+09 ;
PLANT.DRIVINGFORCE_C3 := -5.76416E+01 : -1.000E+09 : 1.000E+09 ;
PLANT.REYNOLDSCONST1_C1 := 2.10000E+03 : -1.000E-03 : 1.000E+09 ;
PLANT.REYNOLDSCONST1_C2 := 2.10000E+03 : -1.000E-03 : 1.000E+09 ;
PLANT.REYNOLDSCONST2_C1 := 4.50000E+03 : -1.000E-03 : 1.000E+09 ;
PLANT.REYNOLDSCONST1_C3 := 2.10000E+03 : -1.000E-03 : 1.000E+09 ;
PLANT.REYNOLDSCONST2_C2 := 4.50000E+03 : -1.000E-03 : 1.000E+09 ;
PLANT.REYNOLDSCONST2_C3 := 4.50000E+03 : -1.000E-03 : 1.000E+09 ;
PLANT.REYNOLDSNO_C1 := 8.30707E+03 : -1.000E-03 : 1.000E+09 ;
PLANT.REYNOLDSNO_C2 := 0.00000E+00 : -1.000E-03 : 1.000E+09 ;
PLANT.REYNOLDSNO_C3 := 0.00000E+00 : -1.000E-03 : 1.000E+09 ;

{
  WITHIN Plant DO
    ReynoldsNo_C2  := 5000 ;
    ReynoldsNo_C3  := 5000 ;
    WITHIN FlashDrum DO
      PhaseType_V      := 1 ;
      PhaseType_B      := 1 ;
      PhaseType_L      := 2 ;
      PhaseType_B_L    := 3 ;
      PhaseType_Po1    := 3 ;
      PhaseType_Po2    := 1 ;
      Top_V            := 3 ;
      Bot_V            := 0.0535 ;
      Top_B_L          := 0.0535 ;
      Bot_B_L          := 0 ;
      Level_B_L        := 0.0535 ;
      Level_V          := 3 ;
      VapPress_L(1)    := 311.379 ;
      VapPress_L(2)    := 11.018E-3 ;
      EquilConst_B_L(1)  := 1.6 ;
      EquilConst_B_L(2)  := 0.2 ;
```

```
      Vol_FlashDrum    := 2.356 ;
      Press_FlashDrum  := 43 ;
      Temp_B           := 220 ;
      Temp_L           := 220 ;
    END
  END
}


SELECTOR
  WITHIN Plant DO
    FlowType_C1         := Turbulent ;
    FlowType_C2         := Turbulent ;
    FlowType_C3         := Turbulent ;
    WITHIN FlashDrum DO
      Phase_B_L         := TwoPhase  ;
    END
  END

INITIAL
  WITHIN Plant DO
    WITHIN FlashDrum DO
      Temp_V      = 300   ;
      Mass_V      = 1     ;
      Mass_B_L    = 15    ;
      Ratio_B     = 0.001 ;
    END
  END


SOLUTIONPARAMETERS
  BLOCKDECOMPOSITION := OFF ;
  OUTPUTLEVEL := 1 ;


SCHEDULE
  SEQUENCE
    CONTINUE FOR 10
      RESET Plant.R1.Press_P := 4*1.013E2 + (TIME-10) ;
    END
    CONTINUE UNTIL Plant.R1.Press_P > 500
      RESET Plant.R1.Press_P := 600 - TIME ;
    END
    CONTINUE UNTIL Plant.R1.Press_P < 350
      RESET Plant.R1.Press_P := 4*1.013E2 ;
    END
    CONTINUE UNTIL Plant.FlashDrum.Der_IntEnergy_V^2 < 1E-20
  END

END # Process test
```

D.2 Two Flash Drums with reversible flow

```
DECLARE

TYPE
  Mass_rate       =  50     :  -1E-1     :  1E4     UNIT = "kg/sec"
  Temperature     =  100    :  -1E-1     :  1E4     UNIT = "K"
  Length          =  15     :  -1E-1     :  1E2     UNIT = "m"
  Enthalpy        =  700    :  -1E7      :  1E4     UNIT = "kJ/kg"
  Int_Energy      =  600    :  -1E9      :  1E4     UNIT = "kJ/kg"
  Volume          =  0.5    :  -1E-1     :  1E1     UNIT = "m3"
  Pressure        =  43     :  -1E-1     :  1E4     UNIT = "kPa"
  Enthalpy_Flow   =  1E3    :  -1E9      :  1E7     UNIT = "kJ/sec"
  Mass            =  5      :  -1E-1     :  1E9     UNIT = "kg"
  Mole            =  0.1    :  -1E-1     :  1E2     UNIT = "kmole"
  Density         =  100    :  -1E-1     :  1E5     UNIT = "kg/m3"
  Viscosity       =  5E-5   :  -1E-1     :  1E2     UNIT = "Pa.s"
  Velocity        =  10     :  -1E-1     :  1E4     UNIT = "m/s"
  Fraction        =  0.5    :  -1E-1     :  10
  NoType          =  200    :  -1E9      :  1E9
  Positive        =  1E2    :  -1E-3     :  1E9


STREAM
 MassStream    IS Mass_Rate, Enthalpy_Flow, Fraction, Positive
 EnergyStream  IS Enthalpy_Flow

END

# ===================================================================

# ##############################
# BEGINNING of generated model #
# ##############################


MODEL  m_R1

PARAMETER
  NoComp                                  AS  INTEGER


VARIABLE
  Ratio_P                                 AS  Fraction
  Rate_P                                  AS  Array(NoComp) of Mass_Rate
  Den_P                                   AS  Density
  Press_P                                 AS  Pressure
  EnthFlow_P                              AS  Enthalpy_Flow
  PhaseType_P                             AS  Positive
  Viscosity_P                             AS  Viscosity
  MassFrac_P                              AS  Array(NoComp) of Fraction
  Enth_P                                  AS  Enthalpy

STREAM
  P : Rate_P, EnthFlow_P, Ratio_P, PhaseType_P  AS  MassStream

END  # end of MODEL m_R1


MODEL  m_R2

PARAMETER
  NoComp                                  AS  INTEGER

VARIABLE
  Ratio_P                                 AS  Fraction
  Rate_P                                  AS  Array(NoComp) of Mass_Rate
  Press_P                                 AS  Pressure
```

```
  EnthFlow_P                              AS  Enthalpy_Flow
  PhaseType_P                             AS  Positive

STREAM
  P : Rate_P, EnthFlow_P, Ratio_P, PhaseType_P  AS  MassStream

END  # end of MODEL m_R2

MODEL  m_R3

PARAMETER
  NoComp                                  AS  INTEGER

VARIABLE
  Ratio_P                                 AS  Fraction
  Rate_P                                  AS  Array(NoComp) of Mass_Rate
  Press_P                                 AS  Pressure
  EnthFlow_P                              AS  Enthalpy_Flow
  PhaseType_P                             AS  Positive

STREAM
  P : Rate_P, EnthFlow_P, Ratio_P, PhaseType_P  AS  MassStream

END  # end of MODEL m_R3

MODEL  m_R4

PARAMETER
  NoComp                                  AS  INTEGER

VARIABLE
  Ratio_P                                 AS  Fraction
  Rate_P                                  AS  Array(NoComp) of Mass_Rate
  Press_P                                 AS  Pressure
  EnthFlow_P                              AS  Enthalpy_Flow
  PhaseType_P                             AS  Positive

STREAM
  P : Rate_P, EnthFlow_P, Ratio_P, PhaseType_P  AS  MassStream

END  # end of MODEL m_R4

MODEL  m_FlashDrum1

PARAMETER
  vapour                                  AS  INTEGER
  OUTLET                                  AS  INTEGER
  INLET                                   AS  INTEGER
  vapour_liquid                           AS  INTEGER
  NoComp                                  AS  INTEGER
  liquid                                  AS  INTEGER
  Z_Po1                                   AS  REAL
  Z_Po2                                   AS  REAL
  height                                  AS  REAL
  Const_B_V                               AS  REAL
  Z_Pi                                    AS  REAL
  diameter                                AS  REAL

VARIABLE
  Mass_B_L                                AS  Array(NoComp) of Mass
  Mass_B                                  AS  Array(NoComp) of Mass
  Mass_V                                  AS  Array(NoComp) of Mass
  Mass_L                                  AS  Array(NoComp) of Mass
  Rate_B_V                                AS  Array(NoComp) of Mass_Rate
```

```
Rate_V_Pi                AS  Array(NoComp) of Mass_Rate
Rate_B_L_Po1             AS  Array(NoComp) of Mass_Rate
Rate_B_L_Po2             AS  Array(NoComp) of Mass_Rate
Rate_V_Po1               AS  Array(NoComp) of Mass_Rate
Rate_V_Po2               AS  Array(NoComp) of Mass_Rate
Rate_Pi                  AS  Array(NoComp) of Mass_Rate
Rate_Po1                 AS  Array(NoComp) of Mass_Rate
Rate_Po2                 AS  Array(NoComp) of Mass_Rate
Rate_B_L_Pi              AS  Array(NoComp) of Mass_Rate
Mole_B_L                 AS  Array(NoComp) of Mole
Mole_B                   AS  Array(NoComp) of Mole
Mole_L                   AS  Array(NoComp) of Mole
Den_B_L                  AS  Density
Den_B                    AS  Density
Den_V                    AS  Density
Den_L                    AS  Density
Den_Po1                  AS  Density
Den_Po2                  AS  Density
Viscosity_B_L            AS  Viscosity
Viscosity_V              AS  Viscosity
Viscosity_Po2            AS  Viscosity
Viscosity_Po1            AS  Viscosity
Press_B                  AS  Pressure
Press_V                  AS  Pressure
Press_L                  AS  Pressure
Press_FlashDrum1         AS  Pressure
Press_B_L                AS  Pressure
VapPress_L               AS  Array(NoComp) of Pressure
DewTemp_B_L              AS  Temperature
Temp_B                   AS  Temperature
Temp_V                   AS  Temperature
Temp_L                   AS  Temperature
BubTemp_B_L              AS  Temperature
Vol_B                    AS  Volume
Vol_B_L                  AS  Volume
Vol_V                    AS  Volume
Vol_L                    AS  Volume
Vol_FlashDrum1           AS  Volume
Enth_Po1                 AS  Enthalpy
Enth_Po2                 AS  Enthalpy
Enth_B                   AS  Enthalpy
Enth_V                   AS  Enthalpy
Enth_L                   AS  Enthalpy
Enth_B_L                 AS  Enthalpy
IntEnergy_B_L            AS  Int_Energy
IntEnergy_L              AS  Int_Energy
IntEnergy_V              AS  Int_Energy
IntEnergy_B              AS  Int_Energy
EnthFlow_V_Pi            AS  Enthalpy_Flow
EnthFlow_B_L_Pi          AS  Enthalpy_Flow
EnthFlow_Pi              AS  Enthalpy_Flow
EnthFlow_B_V             AS  Enthalpy_Flow
EnthFlow_V_Po2           AS  Enthalpy_Flow
EnthFlow_B_L_Po2         AS  Enthalpy_Flow
EnthFlow_Po2             AS  Enthalpy_Flow
EnthFlow_Po2             AS  Enthalpy_Flow
EnthFlow_V_Po1           AS  Enthalpy_Flow
EnthFlow_B_L_Po1         AS  Enthalpy_Flow
EnthFlow_Po1             AS  Enthalpy_Flow
Ratio_Pi                 AS  Fraction
Ratio_B                  AS  Fraction
Ratio_V                  AS  Fraction
Ratio_Po2                AS  Fraction
Ratio_Po1                AS  Fraction
MoleFrac_L               AS  Array(NoComp) of Fraction
MassFrac_B_L             AS  Array(NoComp) of Fraction
MassFrac_L               AS  Array(NoComp) of Fraction
MassFrac_V               AS  Array(NoComp) of Fraction
MassFrac_Po2             AS  Array(NoComp) of Fraction
```

```
MoleFrac_B                          AS  Array(NoComp) of Fraction
MassFrac_B                          AS  Array(NoComp) of Fraction
MassFrac_Po1                        AS  Array(NoComp) of Fraction
Top_B_L                             AS  Positive
Level_V                             AS  Positive
Bot_V                               AS  Positive
Level_B_L                           AS  Positive
PhaseType_B_L                       AS  Positive
PhaseType_L                         AS  Positive
PhaseType_Pi                        AS  Positive
PhaseType_V                         AS  Positive
PhaseType_Po2                       AS  Positive
PhaseType_B                         AS  Positive
PortType_Po1                        AS  Positive
PhaseType_Po1                       AS  Positive
Bot_B_L                             AS  Positive
area                                AS  Positive
Top_V                               AS  Positive
EquilConst_B_L                      AS  Array(NoComp) of Positive
MoleWeight                          AS  Array(NoComp) of Positive


# ================================= #
# Variables for Physical Properties #
# ================================= #

Enth_Vo                             AS ARRAY(NoComp) OF Enthalpy
Enth_Bo                             AS ARRAY(NoComp) OF Enthalpy
Enth_Lo                             AS ARRAY(NoComp) OF Enthalpy
Heat_Lo                             AS ARRAY(NoComp) OF Enthalpy

Temp_ro                             AS ARRAY(NoComp) OF Fraction
Den_Vo                              AS ARRAY(NoComp) OF Density
Den_Bo                              AS ARRAY(NoComp) OF Density
Temp_co                             AS ARRAY(NoComp) OF Temperature

Viscosity_B                         As Viscosity
Viscosity_L                         As Viscosity
Viscosity_Vo                        As ARRAY(NoComp) OF Viscosity
Viscosity_Bo                        As ARRAY(NoComp) OF Viscosity
Viscosity_Lo                        As ARRAY(NoComp) OF Viscosity
Av, Bv, Cv, Dv                      AS ARRAY(NoComp) OF NoType
Al, Bl, Cl, Dl, El                  AS ARRAY(NoComp) OF NoType

Temp_r                              AS Fraction
Press_c                             AS Pressure

Press_co                            AS ARRAY(NoComp) OF Positive
Temp_c                              AS Temperature
M                                   AS Positive
Z                                   AS NoType
Zo                                  AS ARRAY(NoComp) OF NoType

R                                   AS NoType
  A, B, C, D, E                     AS ARRAY(NoComp) OF NoType
#  F, G                              AS ARRAY(NoComp) OF NoType
  vap1, vap2, vap3, vap4, vap5      AS ARRAy(NoComp) OF NoType
  Wo                                AS ARRAY(NoComp) OF NoType


  # ================================= #
  # dummy variable for visualisation #
  # ================================= #

  Sum_Mass_V,
  Der_IntEnergy_V,
  Sum_Mass_B_L,
  Der_Mass_B_L,
```

```
  Sum_Mass_B,
  Sum_Mass_L,
  Sum_Rate_B_V,
  Sum_Rate_Pi,
  Sum_Rate_Po1,
  Sum_Rate_Po2,
  VapPress_1,
  VapPress_2,
  Equil_1,
  Equil_2                                    AS NoType

STREAM
  Pi  : Rate_Pi,  EnthFlow_Pi,  Ratio_Pi,  PhaseType_Pi   AS  MassStream
  Po1 : Rate_Po1, EnthFlow_Po1, Ratio_Po1, PhaseType_Po1  AS  MassStream
  Po2 : Rate_Po2, EnthFlow_Po2, Ratio_Po2, PhaseType_Po2  AS  MassStream

SELECTOR
  Phase_B_L   AS  (LPhase, TwoPhase, VPhase)

SET
  vapour                          := 1 ;
  Z_Po1                           := 0.500000 ;
  Z_Po2                           := 2.500000 ;
  OUTLET                          := 1 ;
  INLET                           := 2 ;
  height                          := 3.000000 ;
  vapour_liquid                   := 3 ;
  liquid                          := 2 ;
  Z_Pi                            := 1.500000 ;
  diameter                        := 1.000000 ;

EQUATION

  # case invariant mass balance
  $Mass_V = Rate_V_Pi - Rate_V_Po1 - Rate_V_Po2 + Rate_B_V ;

  $Mass_B_L = Rate_B_L_Pi - Rate_B_L_Po1 - Rate_B_L_Po2  - Rate_B_V ;


  # case invariant energy balance
  $IntEnergy_V * SIGMA(Mass_V) + IntEnergy_V * SIGMA($Mass_V) =
    EnthFlow_V_Pi - EnthFlow_V_Po1 - EnthFlow_V_Po2 + EnthFlow_B_V ;

  $IntEnergy_B_L * SIGMA(Mass_B_L) + IntEnergy_B_L * SIGMA($Mass_B_L) =
    EnthFlow_B_L_Pi - EnthFlow_B_L_Po1 - EnthFlow_B_L_Po2 - EnthFlow_B_V ;

  # ratio of dispersed phase
  Ratio_V = 1 ;
  SIGMA(Mass_B) = Ratio_B * SIGMA(Mass_B_L) ;

  # PhaseEquilibrium :
  Temp_B = Temp_L ;
  Mass_B_L = Mass_B + Mass_L ;
  Mass_B = MoleWeight * Mole_B ;
  Mass_B_L = MoleWeight * Mole_B_L ;
  Mole_B_L = Mole_B + Mole_L ;
  Mole_B = MoleFrac_B * SIGMA(Mole_B) ;
  Mole_L = MoleFrac_L * SIGMA(Mole_L) ;
  CASE Phase_B_L OF
    WHEN LPhase :
      Mole_B = 0 ;
      EquilConst_B_L = 0 ;
      SWITCH TO TwoPhase IF Temp_L > BubTemp_B_L ;
    WHEN TwoPhase :
      VapPress_L * MoleFrac_L = MoleFrac_B * Press_B_L ;
      MoleFrac_B = EquilConst_B_L * MoleFrac_L ;
```

```
    SWITCH TO LPhase IF SIGMA(Mole_B) <= 0 ;
    SWITCH TO VPhase IF SIGMA(Mole_L) <= 0 ;
  WHEN VPhase :
    EquilConst_B_L = 0 ;
    Mole_L = 0 ;
    SWITCH TO TwoPhase IF Temp_B < DewTemp_B_L ;
END

# BubbleRise :
Rate_B_V = Const_B_V * Mass_B ;
EnthFlow_B_V = SIGMA(Rate_B_V) * Enth_B ;

# mass = mass fraction * total mass
Mass_B_L = MassFrac_B_L * SIGMA(Mass_B_L) ;
Mass_L = MassFrac_L * SIGMA(Mass_L) ;
Mass_B = MassFrac_B * SIGMA(Mass_B) ;
Mass_V = MassFrac_V * SIGMA(Mass_V) ;

# total mass = density * volume
SIGMA(Mass_B_L) = Den_B_L * Vol_B_L ;
SIGMA(Mass_L) = Den_L * Vol_L ;
SIGMA(Mass_B) = Den_B * Vol_B ;
SIGMA(Mass_V) = Den_V * Vol_V ;

# phase type
PhaseType_B_L = vapour_liquid ;
PhaseType_L = liquid ;
PhaseType_B = vapour ;
PhaseType_V = vapour ;

# volume relationship
Vol_B_L = Vol_B + Vol_L ;
Vol_FlashDrum1 = Vol_V + Vol_B_L ;

# uniform pressure within vessel
Press_FlashDrum1 = Press_B_L ;
Press_FlashDrum1 = Press_L ;
Press_FlashDrum1 = Press_B ;
Press_FlashDrum1 = Press_V ;

# phase bound : upper/low bound of phase volume = level
Top_B_L = Level_B_L ;
Bot_B_L = 0 ;
Top_V = Level_V ;
Bot_V = Level_B_L ;
Top_V = height ;

# phase volume : volume = area * (top - bottom)
area = (3.14/4) * diameter^2 ;
Vol_B_L = area * (Top_B_L - Bot_B_L) ;
Vol_V = area * (Top_V - Bot_V) ;


# discontinuity on input port, "Pi"
IF PhaseType_Pi = vapour  THEN
  IF Z_Pi > Bot_V AND Z_Pi <= Top_V  THEN
    Rate_V_Pi = Rate_Pi ;
    EnthFlow_V_Pi = EnthFlow_Pi ;
    Rate_B_L_Pi = 0 ;
    EnthFlow_B_L_Pi = 0 ;
  ELSE
    Rate_B_L_Pi = Rate_Pi ;
    EnthFlow_B_L_Pi = EnthFlow_Pi ;
    Rate_V_Pi = 0 ;
    EnthFlow_V_Pi = 0 ;
  END
ELSE
  Rate_B_L_Pi = Rate_Pi ;
```

```
  EnthFlow_B_L_Pi = EnthFlow_Pi ;
  Rate_V_Pi = 0 ;
  EnthFlow_V_Pi = 0 ;
END
# end of discontinuity on input port, "Pi"


# discontinuity on both type port, "Po1"
IF PortType_Po1 = OUTLET  THEN
  IF Z_Po1 > Bot_V AND Z_Po1 <= Top_V  THEN
    Viscosity_V = Viscosity_Po1 ;
    Den_V = Den_Po1 ;
    MassFrac_V = MassFrac_Po1 ;
    Enth_V = Enth_Po1 ;
    Rate_V_Po1 = Rate_Po1 ;
    EnthFlow_V_Po1 = EnthFlow_Po1 ;
    Rate_B_L_Po1 = 0 ;
    EnthFlow_B_L_Po1 = 0 ;
  ELSE
    Viscosity_B_L = Viscosity_Po1 ;
    Den_B_L = Den_Po1 ;
    MassFrac_B_L = MassFrac_Po1 ;
    Enth_B_L = Enth_Po1 ;
    Rate_B_L_Po1 = Rate_Po1 ;
    EnthFlow_B_L_Po1 = EnthFlow_Po1 ;
    Rate_V_Po1 = 0 ;
    EnthFlow_V_Po1 = 0 ;
  END
ELSE
  IF PhaseType_Po1 = vapour  THEN
    IF Z_Po1 > Bot_V AND Z_Po1 <= Top_V  THEN
      Viscosity_Po1 = 0 ;
      Den_Po1 = 0 ;
      MassFrac_Po1 = 0 ;
      Enth_Po1 = 0 ;
      Rate_V_Po1 = Rate_Po1 ;
      EnthFlow_V_Po1 = EnthFlow_Po1 ;
      Rate_B_L_Po1 = 0 ;
      EnthFlow_B_L_Po1 = 0 ;
    ELSE
      Viscosity_Po1 = 0 ;
      Den_Po1 = 0 ;
      MassFrac_Po1 = 0 ;
      Enth_Po1 = 0 ;
      Rate_B_L_Po1 = Rate_Po1 ;
      EnthFlow_B_L_Po1 = EnthFlow_Po1 ;
      Rate_V_Po1 = 0 ;
      EnthFlow_V_Po1 = 0 ;
    END
  ELSE
    Viscosity_Po1 = 0 ;
    Den_Po1 = 0 ;
    MassFrac_Po1 = 0 ;
    Enth_Po1 = 0 ;
    Rate_B_L_Po1 = Rate_Po1 ;
    EnthFlow_B_L_Po1 = EnthFlow_Po1 ;
    Rate_V_Po1 = 0 ;
    EnthFlow_V_Po1 = 0 ;
  END
END
# end of discontinuity on both type port, "Po1"


# discontinuity on output port, "Po2"
IF Z_Po2 > Bot_V AND Z_Po2 <= Top_V  THEN
  Viscosity_V = Viscosity_Po2 ;
  Enth_V = Enth_Po2 ;
  MassFrac_V = MassFrac_Po2 ;
```

```
    Den_V = Den_Po2 ;
    Rate_V_Po2 = Rate_Po2 ;
    EnthFlow_V_Po2 = EnthFlow_Po2 ;
    Ratio_V = Ratio_Po2 ;
    PhaseType_V = PhaseType_Po2 ;
    Rate_B_L_Po2 = 0 ;
    EnthFlow_B_L_Po2 = 0 ;
  ELSE
    Viscosity_B_L = Viscosity_Po2 ;
    Enth_B_L = Enth_Po2 ;
    MassFrac_B_L = MassFrac_Po2 ;
    Den_B_L = Den_Po2 ;
    Rate_B_L_Po2 = Rate_Po2 ;
    EnthFlow_B_L_Po2 = EnthFlow_Po2 ;
    Ratio_B = Ratio_Po2 ;
    PhaseType_B_L = PhaseType_Po2 ;
    Rate_V_Po2 = 0 ;
    EnthFlow_V_Po2 = 0 ;
  END
  # end of discontinuity on output port, "Po2"



  # ================== #
  # Physical Properties #
  # ================== #

  # vapour pressure of L
#  VapPress_L * 0.145 = 10^(F - G/((1.8 * Temp_L - 459.4) + 382)) ;
   VapPress_L = 1E-3 * EXP(vap1 + vap2/Temp_L + vap3*LOG(Temp_L) +
                           vap4*Temp_L^vap5) ;

# Application Range :
# Propane : 228K ~ 366K
# Butane  : 228K ~ 421K

# pure enthalpies of V
Enth_Vo * 0.43 = A * ((Temp_V * 9/5)/100) + B * ((Temp_V * 9/5)/100)^2 +
  C * 1E-2 * ((Temp_V * 9/5)/100)^3 + D * (100/(Temp_V * 9/5)) + E ;

# enthalpy of V
Enth_V = SIGMA(MassFrac_V * Enth_Vo) ;

# pure enthalpies of B
Enth_Bo * 0.43 = A * ((Temp_B * 9/5)/100) + B * ((Temp_B * 9/5)/100)^2 +
  C * 1E-2 * ((Temp_B * 9/5)/100)^3 + D * (100/(Temp_B * 9/5)) + E ;

# enthalpy of B
Enth_B = SIGMA(MassFrac_B * Enth_Bo) ;

# heats of vapourisation of pure L
Heat_Lo * MoleWeight = (R * Temp_co) * (7.08 * (1 - Temp_ro)^0.354 +
  10.95 * Wo * (1-Temp_ro)^0.456) ;
Temp_ro * Temp_co = Temp_L ;

# pure enthapies of L
Enth_Lo = Enth_Bo - Heat_Lo ;

# enthalpy of L
Enth_L = SIGMA(MassFrac_L * Enth_Lo) ;

# enthalpy of B_L
Enth_B_L = Ratio_B * Enth_B + (1 - Ratio_B) * Enth_L ;

# internal energy of V
Enth_V * SIGMA(Mass_V) = IntEnergy_V * SIGMA(Mass_V) +
                         SIGMA(Mass_V / MoleWeight) * R * Temp_V ;
```

```
  # internal energy of B_L
  Enth_B_L = IntEnergy_B_L + Press_B_L / Den_B_L ;
#  IntEnergy_B_L = IntEnergy_B * Ratio_B +
#                  (1 - Ratio_B) * IntEnergy_L ;

  # internal energy of B
  Enth_B * SIGMA(Mass_B) = IntEnergy_B * SIGMA(Mass_B) +
                           SIGMA(Mass_B / MoleWeight) * R * Temp_B ;

  # internal energy of L
  Enth_L  = IntEnergy_L + Press_L / Den_L ;

  # density of V
  Den_Vo * R * Temp_V = Press_V * MoleWeight ; # for pure component
  Den_V * SIGMA(MassFrac_V / Den_Vo) = 1 ;     # for mixture

  # density of B
  Den_Bo * R * Temp_L = Press_B * MoleWeight;  # for pure component
  Den_B * SIGMA(MassFrac_B / Den_Bo) = 1;      # for mixture

  # density of L
  Den_L * R * Temp_c * Z^(1 + (1 - Temp_r)^(2/7)) = Press_c * M ;
  Press_c = SIGMA(MoleFrac_L * Press_co) ;
  Temp_c = SIGMA(MoleFrac_L * Temp_co) ;
  Z = SIGMA(MoleFrac_L * Zo) ;
  Temp_r * Temp_c = Temp_L ;
  M = SIGMA(MoleFrac_L * MoleWeight) ;

  # viscosity of V
  Viscosity_Vo = Av * Temp_V^Bv / (1 + Cv / Temp_V + Dv / Temp_V^2) ;
  Viscosity_V = SIGMA(MassFrac_V * Viscosity_Vo) ;

  # viscosity of B_L
  Viscosity_Bo = (Av * Temp_B^Bv) / (1 + Cv / Temp_B + Dv / Temp_B^2) ;
  Viscosity_B = SIGMA(MassFrac_V * Viscosity_Bo) ;
  Viscosity_Lo = EXP(Al * (Bl / Temp_L) + Cl * LOG(Temp_L) +
                     Dl * Temp_L^El) ;
  Viscosity_L = SIGMA(MassFrac_L * Viscosity_Lo) ;
  Viscosity_B_L = Ratio_B * Viscosity_B +
                  (1 - Ratio_B) * Viscosity_L ;


  # ================================ #
  # dummy variables for visualisation #
  # ================================ #

  Sum_Mass_V = SIGMA(Mass_V) ;
  Der_IntEnergy_V = $IntEnergy_V;
  Sum_Mass_B_L = SIGMA(Mass_B_L) ;
  Der_Mass_B_L = SIGMA($Mass_B_L);
  Sum_Mass_B = SIGMA(Mass_B) ;
  Sum_Mass_L = SIGMA(Mass_L) ;
  Sum_Rate_B_V = SIGMA(Rate_B_V) ;
  Sum_Rate_Pi = SIGMA(Rate_Pi);
  Sum_Rate_Po1 = SIGMA(Rate_Po1);
  Sum_Rate_Po2 = SIGMA(Rate_Po2);
  VapPress_1 = VapPress_L(1);
  VapPress_2 = VapPress_L(2);
  Equil_1 = EquilConst_B_L(1);
  Equil_2 = EquilConst_B_L(2);

END  # end of MODEL m_FlashDrum1


MODEL  m_FlashDrum2

PARAMETER
  vapour                                    AS  INTEGER
```

```
  OUTLET                    AS  INTEGER
  INLET                     AS  INTEGER
  vapour_liquid             AS  INTEGER
  NoComp                    AS  INTEGER
  liquid                    AS  INTEGER
  Z_Po1                     AS  REAL
  Z_Po2                     AS  REAL
  height                    AS  REAL
  Const_B_V                 AS  REAL
  Z_Pi                      AS  REAL
  diameter                  AS  REAL

VARIABLE
  Mass_B_L                  AS  Array(NoComp) of Mass
  Mass_B                    AS  Array(NoComp) of Mass
  Mass_V                    AS  Array(NoComp) of Mass
  Mass_L                    AS  Array(NoComp) of Mass
  Rate_B_V                  AS  Array(NoComp) of Mass_Rate
  Rate_V_Pi                 AS  Array(NoComp) of Mass_Rate
  Rate_B_L_Po1              AS  Array(NoComp) of Mass_Rate
  Rate_B_L_Po2              AS  Array(NoComp) of Mass_Rate
  Rate_V_Po1                AS  Array(NoComp) of Mass_Rate
  Rate_V_Po2                AS  Array(NoComp) of Mass_Rate
  Rate_Pi                   AS  Array(NoComp) of Mass_Rate
  Rate_Po1                  AS  Array(NoComp) of Mass_Rate
  Rate_Po2                  AS  Array(NoComp) of Mass_Rate
  Rate_B_L_Pi               AS  Array(NoComp) of Mass_Rate
  Mole_B_L                  AS  Array(NoComp) of Mole
  Mole_B                    AS  Array(NoComp) of Mole
  Mole_L                    AS  Array(NoComp) of Mole
  Den_B_L                   AS  Density
  Den_B                     AS  Density
  Den_V                     AS  Density
  Den_L                     AS  Density
  Den_Po1                   AS  Density
  Den_Po2                   AS  Density
  Den_Pi                    AS  Density
  Viscosity_B_L             AS  Viscosity
  Viscosity_Pi              AS  Viscosity
  Viscosity_V               AS  Viscosity
  Viscosity_Po2             AS  Viscosity
  Viscosity_Po1             AS  Viscosity
  Press_B                   AS  Pressure
  Press_V                   AS  Pressure
  Press_L                   AS  Pressure
  Press_FlashDrum2          AS  Pressure
  Press_B_L                 AS  Pressure
  VapPress_L                AS  Array(NoComp) of Pressure
  DewTemp_B_L               AS  Temperature
  Temp_B                    AS  Temperature
  Temp_V                    AS  Temperature
  Temp_L                    AS  Temperature
  BubTemp_B_L               AS  Temperature
  Vol_B                     AS  Volume
  Vol_B_L                   AS  Volume
  Vol_V                     AS  Volume
  Vol_L                     AS  Volume
  Vol_FlashDrum2            AS  Volume
  Enth_Po1                  AS  Enthalpy
  Enth_Po2                  AS  Enthalpy
  Enth_Pi                   AS  Enthalpy
  Enth_B                    AS  Enthalpy
  Enth_V                    AS  Enthalpy
  Enth_L                    AS  Enthalpy
  Enth_B_L                  AS  Enthalpy
  IntEnergy_B_L             AS  Int_Energy
  IntEnergy_L               AS  Int_Energy
  IntEnergy_V               AS  Int_Energy
```

```
IntEnergy_B                    AS  Int_Energy
EnthFlow_V_Pi                  AS  Enthalpy_Flow
EnthFlow_B_L_Pi                AS  Enthalpy_Flow
EnthFlow_Pi                    AS  Enthalpy_Flow
EnthFlow_B_V                   AS  Enthalpy_Flow
EnthFlow_V_Po2                 AS  Enthalpy_Flow
EnthFlow_B_L_Po2               AS  Enthalpy_Flow
EnthFlow_Po2                   AS  Enthalpy_Flow
EnthFlow_V_Po1                 AS  Enthalpy_Flow
EnthFlow_B_L_Po1               AS  Enthalpy_Flow
EnthFlow_Po1                   AS  Enthalpy_Flow
Ratio_Pi                       AS  Fraction
Ratio_B                        AS  Fraction
Ratio_V                        AS  Fraction
Ratio_Po2                      AS  Fraction
Ratio_Po1                      AS  Fraction
MoleFrac_L                     AS  Array(NoComp) of Fraction
MassFrac_B_L                   AS  Array(NoComp) of Fraction
MassFrac_L                     AS  Array(NoComp) of Fraction
MassFrac_Pi                    AS  Array(NoComp) of Fraction
MassFrac_V                     AS  Array(NoComp) of Fraction
MassFrac_Po2                   AS  Array(NoComp) of Fraction
MoleFrac_B                     AS  Array(NoComp) of Fraction
MassFrac_B                     AS  Array(NoComp) of Fraction
MassFrac_Po1                   AS  Array(NoComp) of Fraction
Top_B_L                        AS  Positive
Level_V                        AS  Positive
Bot_V                          AS  Positive
Level_B_L                      AS  Positive
PhaseType_B_L                  AS  Positive
PhaseType_L                    AS  Positive
PortType_Pi                    AS  Positive
PhaseType_Pi                   AS  Positive
PhaseType_V                    AS  Positive
PhaseType_Po2                  AS  Positive
PhaseType_B                    AS  Positive
PhaseType_Po1                  AS  Positive
Bot_B_L                        AS  Positive
area                           AS  Positive
Top_V                          AS  Positive
EquilConst_B_L                 AS  Array(NoComp) of Positive
MoleWeight                     AS  Array(NoComp) of Positive


# ================================ #
# Variables for Physical Properties #
# ================================ #

Enth_Vo                        AS ARRAY(NoComp) OF Enthalpy
Enth_Bo                        AS ARRAY(NoComp) OF Enthalpy
Enth_Lo                        AS ARRAY(NoComp) OF Enthalpy
Heat_Lo                        AS ARRAY(NoComp) OF Enthalpy

Temp_ro                        AS ARRAY(NoComp) OF Fraction
Den_Vo                         AS ARRAY(NoComp) OF Density
Den_Bo                         AS ARRAY(NoComp) OF Density
Temp_co                        AS ARRAY(NoComp) OF Temperature

Viscosity_B                    As Viscosity
Viscosity_L                    As Viscosity
Viscosity_Vo                   As ARRAY(NoComp) OF Viscosity
Viscosity_Bo                   As ARRAY(NoComp) OF Viscosity
Viscosity_Lo                   As ARRAY(NoComp) OF Viscosity
Av, Bv, Cv, Dv                 AS ARRAY(NoComp) OF NoType
Al, Bl, Cl, Dl, El             AS ARRAY(NoComp) OF NoType

Temp_r                         AS Fraction
```

```
  Press_c                          AS Pressure

  Press_co                         AS ARRAY(NoComp) OF Positive
  Temp_c                           AS Temperature
  M                                AS Positive
  Z                                AS NoType
  Zo                               AS ARRAY(NoComp) OF NoType

  R                                AS NoType
  A, B, C, D, E                    AS ARRAY(NoComp) OF NoType
#  F, G                             AS ARRAY(NoComp) OF NoType
  vap1, vap2, vap3, vap4, vap5     AS ARRAy(NoComp) OF NoType
  Wo                               AS ARRAY(NoComp) OF NoType


  # ================================ #
  # dummy variable for visualisation #
  # ================================ #

  Sum_Mass_V,
  Der_IntEnergy_V,
  Sum_Mass_B_L,
  Der_Mass_B_L,
  Sum_Mass_B,
  Sum_Mass_L,
  Sum_Rate_B_V,
  Sum_Rate_Pi,
  Sum_Rate_Po1,
  Sum_Rate_Po2,
  VapPress_1,
  VapPress_2,
  Equil_1,
  Equil_2                          AS NoType


STREAM
  Pi  : Rate_Pi,  EnthFlow_Pi,  Ratio_Pi,  PhaseType_Pi   AS  MassStream
  Po1 : Rate_Po1, EnthFlow_Po1, Ratio_Po1, PhaseType_Po1  AS  MassStream
  Po2 : Rate_Po2, EnthFlow_Po2, Ratio_Po2, PhaseType_Po2  AS  MassStream

SELECTOR
  Phase_B_L   AS  (LPhase, TwoPhase, VPhase)

SET
  vapour                           := 1 ;
  Z_Po1                            := 0.500000 ;
  Z_Po2                            := 2.500000 ;
  OUTLET                           := 1 ;
  INLET                            := 2 ;
  height                           := 3.000000 ;
  vapour_liquid                    := 3 ;
  liquid                           := 2 ;
  Z_Pi                             := 1.500000 ;
  diameter                         := 1.000000 ;

EQUATION

  # case invariant mass balance
  $Mass_V = Rate_V_Pi - Rate_V_Po1 - Rate_V_Po2 + Rate_B_V ;

  $Mass_B_L = Rate_B_L_Pi - Rate_B_L_Po1 - Rate_B_L_Po2 - Rate_B_V ;

  # case invariant energy balance
  $IntEnergy_V * SIGMA(Mass_V) + IntEnergy_V * SIGMA($Mass_V) =
    EnthFlow_V_Pi - EnthFlow_V_Po1 - EnthFlow_V_Po2 + EnthFlow_B_V ;

  $IntEnergy_B_L * SIGMA(Mass_B_L) + IntEnergy_B_L * SIGMA($Mass_B_L) =
    EnthFlow_B_L_Pi - EnthFlow_B_L_Po1 - EnthFlow_B_L_Po2 - EnthFlow_B_V ;
```

```
# ratio of dispersed phase
Ratio_V = 1 ;
SIGMA(Mass_B) = Ratio_B * SIGMA(Mass_B_L) ;

# PhaseEquilibrium :
Temp_B = Temp_L ;
Mass_B_L = Mass_B + Mass_L ;
Mass_B = MoleWeight * Mole_B ;
Mass_B_L = MoleWeight * Mole_B_L ;
Mole_B_L = Mole_B + Mole_L ;
Mole_B = MoleFrac_B * SIGMA(Mole_B) ;
Mole_L = MoleFrac_L * SIGMA(Mole_L) ;
CASE Phase_B_L OF
  WHEN LPhase :
    Mole_B = 0 ;
    EquilConst_B_L = 0 ;
  SWITCH TO TwoPhase IF Temp_L > BubTemp_B_L ;
  WHEN TwoPhase :
    VapPress_L * MoleFrac_L = MoleFrac_B * Press_B_L ;
    MoleFrac_B = EquilConst_B_L * MoleFrac_L ;
  SWITCH TO LPhase IF SIGMA(Mole_B) <= 0 ;
  SWITCH TO VPhase IF SIGMA(Mole_L) <= 0 ;
  WHEN VPhase :
    Mole_L = 0 ;
    EquilConst_B_L = 1 ;
  SWITCH TO TwoPhase IF Temp_B < DewTemp_B_L ;
END

# BubbleRise :
Rate_B_V = Const_B_V * Mass_B ;
EnthFlow_B_V = SIGMA(Rate_B_V) * Enth_B ;

# mass = mass fraction * total mass
Mass_B_L = MassFrac_B_L * SIGMA(Mass_B_L) ;
Mass_L = MassFrac_L * SIGMA(Mass_L) ;
Mass_B = MassFrac_B * SIGMA(Mass_B) ;
Mass_V = MassFrac_V * SIGMA(Mass_V) ;

# total mass = density * volume
SIGMA(Mass_B_L) = Den_B_L * Vol_B_L ;
SIGMA(Mass_L) = Den_L * Vol_L ;
SIGMA(Mass_B) = Den_B * Vol_B ;
SIGMA(Mass_V) = Den_V * Vol_V ;

# phase type
PhaseType_B_L = vapour_liquid ;
PhaseType_L = liquid ;
PhaseType_B = vapour ;
PhaseType_V = vapour ;

# volume relationship
Vol_B_L = Vol_B + Vol_L ;
Vol_FlashDrum2 = Vol_V + Vol_B_L ;

# uniform pressure within vessel
Press_FlashDrum2 = Press_B_L ;
Press_FlashDrum2 = Press_L ;
Press_FlashDrum2 = Press_B ;
Press_FlashDrum2 = Press_V ;

# phase bound : upper/low bound of phase volume = level
Top_B_L = Level_B_L ;
Bot_B_L = 0 ;
Top_V = Level_V ;
Bot_V = Level_B_L ;
Top_V = height ;
```

```
# phase volume : volume = area * (top - bottom)
area = (3.14/4) * diameter^2 ;
Vol_B_L = area * (Top_B_L - Bot_B_L) ;
Vol_V = area * (Top_V - Bot_V) ;


# discontinuity on both type port, "Pi"
IF PortType_Pi = OUTLET  THEN
  IF Z_Pi > Bot_V AND Z_Pi <= Top_V  THEN
    Viscosity_V = Viscosity_Pi ;
    Den_V = Den_Pi ;
    MassFrac_V = MassFrac_Pi ;
    Enth_V = Enth_Pi ;
    Rate_V_Pi = Rate_Pi ;
    EnthFlow_V_Pi = EnthFlow_Pi ;
    Rate_B_L_Pi = 0 ;
    EnthFlow_B_L_Pi = 0 ;
  ELSE
    Viscosity_B_L = Viscosity_Pi ;
    Den_B_L = Den_Pi ;
    MassFrac_B_L = MassFrac_Pi ;
    Enth_B_L = Enth_Pi ;
    Rate_B_L_Pi = Rate_Pi ;
    EnthFlow_B_L_Pi = EnthFlow_Pi ;
    Rate_V_Pi = 0 ;
    EnthFlow_V_Pi = 0 ;
  END
ELSE
  IF PhaseType_Pi = vapour  THEN
    IF Z_Pi > Bot_V AND Z_Pi <= Top_V  THEN
      Viscosity_Pi = 0 ;
      Den_Pi = 0 ;
      MassFrac_Pi = 0 ;
      Enth_Pi = 0 ;
      Rate_V_Pi = Rate_Pi ;
      EnthFlow_V_Pi = EnthFlow_Pi ;
      Rate_B_L_Pi = 0 ;
      EnthFlow_B_L_Pi = 0 ;
    ELSE
      Viscosity_Pi = 0 ;
      Den_Pi = 0 ;
      MassFrac_Pi = 0 ;
      Enth_Pi = 0 ;
      Rate_B_L_Pi = Rate_Pi ;
      EnthFlow_B_L_Pi = EnthFlow_Pi ;
      Rate_V_Pi = 0 ;
      EnthFlow_V_Pi = 0 ;
    END
  ELSE
    Viscosity_Pi = 0 ;
    Den_Pi = 0 ;
    MassFrac_Pi = 0 ;
    Enth_Pi = 0 ;
    Rate_B_L_Pi = Rate_Pi ;
    EnthFlow_B_L_Pi = EnthFlow_Pi ;
    Rate_V_Pi = 0 ;
    EnthFlow_V_Pi = 0 ;
  END
END
# end of discontinuity on both type port, "Pi"


# discontinuity on output port, "Po1"
IF Z_Po1 > Bot_V AND Z_Po1 <= Top_V  THEN
  Viscosity_V = Viscosity_Po1 ;
  Enth_V = Enth_Po1 ;
  MassFrac_V = MassFrac_Po1 ;
  Den_V = Den_Po1 ;
```

```
    Rate_V_Po1 = Rate_Po1 ;
    EnthFlow_V_Po1 = EnthFlow_Po1 ;
    Ratio_V = Ratio_Po1 ;
    PhaseType_V = PhaseType_Po1 ;
    Rate_B_L_Po1 = 0 ;
    EnthFlow_B_L_Po1 = 0 ;
  ELSE
    Viscosity_B_L = Viscosity_Po1 ;
    Enth_B_L = Enth_Po1 ;
    MassFrac_B_L = MassFrac_Po1 ;
    Den_B_L = Den_Po1 ;
    Rate_B_L_Po1 = Rate_Po1 ;
    EnthFlow_B_L_Po1 = EnthFlow_Po1 ;
    Ratio_B = Ratio_Po1 ;
    PhaseType_B_L = PhaseType_Po1 ;
    Rate_V_Po1 = 0 ;
    EnthFlow_V_Po1 = 0 ;
  END
  # end of discontinuity on output port, "Po1"


  # discontinuity on output port, "Po2"
  IF Z_Po2 > Bot_V AND Z_Po2 <= Top_V  THEN
    Viscosity_V = Viscosity_Po2 ;
    Enth_V = Enth_Po2 ;
    MassFrac_V = MassFrac_Po2 ;
    Den_V = Den_Po2 ;
    Rate_V_Po2 = Rate_Po2 ;
    EnthFlow_V_Po2 = EnthFlow_Po2 ;
    Ratio_V = Ratio_Po2 ;
    PhaseType_V = PhaseType_Po2 ;
    Rate_B_L_Po2 = 0 ;
    EnthFlow_B_L_Po2 = 0 ;
  ELSE
    Viscosity_B_L = Viscosity_Po2 ;
    Enth_B_L = Enth_Po2 ;
    MassFrac_B_L = MassFrac_Po2 ;
    Den_B_L = Den_Po2 ;
    Rate_B_L_Po2 = Rate_Po2 ;
    EnthFlow_B_L_Po2 = EnthFlow_Po2 ;
    Ratio_B = Ratio_Po2 ;
    PhaseType_B_L = PhaseType_Po2 ;
    Rate_V_Po2 = 0 ;
    EnthFlow_V_Po2 = 0 ;
  END
  # end of discontinuity on output port, "Po2"


  # ==================== #
  # Physical Properties #
  # ==================== #

  # vapour pressure of L
#  VapPress_L * 0.145 = 10^(F - G/((1.8 * Temp_L - 459.4) + 382)) ;
   VapPress_L = 1E-3 * EXP(vap1 + vap2/Temp_L + vap3*LOG(Temp_L) +
                           vap4*Temp_L^vap5) ;

  # Application Range :
  # Propane : 228K ~ 366K
  # Butane  : 228K ~ 421K

  # pure enthalpies of V
  Enth_Vo * 0.43 = A * ((Temp_V * 9/5)/100) + B * ((Temp_V * 9/5)/100)^2 +
    C * 1E-2 * ((Temp_V * 9/5)/100)^3 + D * (100/(Temp_V * 9/5)) + E ;

  # enthalpy of V
  Enth_V = SIGMA(MassFrac_V * Enth_Vo) ;
```

```
# pure enthalpies of B
Enth_Bo * 0.43 = A * ((Temp_B * 9/5)/100) + B * ((Temp_B * 9/5)/100)^2 +
  C * 1E-2 * ((Temp_B * 9/5)/100)^3 + D * (100/(Temp_B * 9/5)) + E ;

# enthalpy of B
Enth_B = SIGMA(MassFrac_B * Enth_Bo) ;

# heats of vapourisation of pure L
Heat_Lo * MoleWeight = (R * Temp_co) * (7.08 * (1 - Temp_ro)^0.354 +
  10.95 * Wo * (1-Temp_ro)^0.456) ;
Temp_ro * Temp_co = Temp_L ;

# pure enthapies of L
Enth_Lo = Enth_Bo - Heat_Lo ;

# enthalpy of L
Enth_L = SIGMA(MassFrac_L * Enth_Lo) ;

# enthalpy of B_L
Enth_B_L = Ratio_B * Enth_B + (1 - Ratio_B) * Enth_L ;

# internal energy of V
Enth_V * SIGMA(Mass_V) = IntEnergy_V * SIGMA(Mass_V) +
                         SIGMA(Mass_V / MoleWeight) * R * Temp_V ;

# internal energy of B_L
Enth_B_L = IntEnergy_B_L + Press_B_L / Den_B_L ;
#  IntEnergy_B_L = IntEnergy_B * Ratio_B +
#                  (1 - Ratio_B) * IntEnergy_L ;

# internal energy of B
Enth_B * SIGMA(Mass_B) = IntEnergy_B * SIGMA(Mass_B) +
                         SIGMA(Mass_B / MoleWeight) * R * Temp_B ;

# internal energy of L
Enth_L  = IntEnergy_L + Press_L / Den_L ;

# density of V
Den_Vo * R * Temp_V = Press_V * MoleWeight ; # for pure component
Den_V * SIGMA(MassFrac_V / Den_Vo) = 1 ;     # for mixture

# density of B
Den_Bo * R * Temp_L = Press_B * MoleWeight;  # for pure component
Den_B * SIGMA(MassFrac_B / Den_Bo) = 1;      # for mixture

# density of L
Den_L * R * Temp_c * Z^(1 + (1 - Temp_r)^(2/7)) = Press_c * M ;
Press_c = SIGMA(MoleFrac_L * Press_co) ;
Temp_c = SIGMA(MoleFrac_L * Temp_co) ;
Z = SIGMA(MoleFrac_L * Zo) ;
Temp_r * Temp_c = Temp_L ;
M = SIGMA(MoleFrac_L * MoleWeight) ;

# viscosity of V
Viscosity_Vo = Av * Temp_V^Bv / (1 + Cv / Temp_V + Dv / Temp_V^2) ;
Viscosity_V = SIGMA(MassFrac_V * Viscosity_Vo) ;

# viscosity of B_L
Viscosity_Bo = (Av * Temp_B^Bv) / (1 + Cv / Temp_B + Dv / Temp_B^2) ;
Viscosity_B = SIGMA(MassFrac_V * Viscosity_Bo) ;
Viscosity_Lo = EXP(Al * (Bl / Temp_L) + Cl * LOG(Temp_L) +
                   Dl * Temp_L^El) ;
Viscosity_L = SIGMA(MassFrac_L * Viscosity_Lo) ;
Viscosity_B_L = Ratio_B * Viscosity_B +
                (1 - Ratio_B) * Viscosity_L ;


# ================================== #
```

```
# dummy variables for visualisation #
# ================================= #

Sum_Mass_V = SIGMA(Mass_V) ;
Der_IntEnergy_V = $IntEnergy_V;
Sum_Mass_B_L = SIGMA(Mass_B_L) ;
Der_Mass_B_L = SIGMA($Mass_B_L);
Sum_Mass_B = SIGMA(Mass_B) ;
Sum_Mass_L = SIGMA(Mass_L) ;
Sum_Rate_B_V = SIGMA(Rate_B_V) ;
Sum_Rate_Pi = SIGMA(Rate_Pi);
Sum_Rate_Po1 = SIGMA(Rate_Po1);
Sum_Rate_Po2 = SIGMA(Rate_Po2);
VapPress_1 = VapPress_L(1);
VapPress_2 = VapPress_L(2);
Equil_1 = EquilConst_B_L(1);
Equil_2 = EquilConst_B_L(2);

END  # end of MODEL m_FlashDrum2


MODEL  Flowsheet

PARAMETER
  OUTLET                          AS  INTEGER
  INLET                           AS  INTEGER
  NoComp                          AS  INTEGER
  Const2_C3                       AS  REAL
  Const2_C2                       AS  REAL
  Const2_C1                       AS  REAL
  Const2_C5                       AS  REAL
  Const2_C4                       AS  REAL
  Const3_C3                       AS  REAL
  Const3_C2                       AS  REAL
  Const3_C1                       AS  REAL
  Const3_C5                       AS  REAL
  Const3_C4                       AS  REAL
  Const_C1                        AS  REAL
  Const_C2                        AS  REAL
  Const_C3                        AS  REAL
  Const_C4                        AS  REAL
  Const_C5                        AS  REAL

VARIABLE
  Den_C2                          AS  Density
  Enth_C2                         AS  Enthalpy
  ReynoldsConst1_C5               AS  Positive
  ReynoldsConst2_C5               AS  Positive
  ReynoldsNo_C5                   AS  Positive
  DrivingForce_C5                 AS  NoType
  ReynoldsConst1_C4               AS  Positive
  ReynoldsConst2_C4               AS  Positive
  ReynoldsNo_C4                   AS  Positive
  DrivingForce_C4                 AS  NoType
  ReynoldsConst1_C3               AS  Positive
  ReynoldsConst2_C3               AS  Positive
  ReynoldsNo_C3                   AS  Positive
  DrivingForce_C3                 AS  NoType
  ReynoldsNo_C2                   AS  Positive
  EnthFlow_C2                     AS  Enthalpy_Flow
  Viscosity_C2                    AS  Viscosity
  MassFrac_C2                     AS  Array(NoComp) of Fraction
  DrivingForce_C2                 AS  NoType
  ReynoldsConst1_C1               AS  Positive
  ReynoldsConst2_C1               AS  Positive
  ReynoldsNo_C1                   AS  Positive
  DrivingForce_C1                 AS  NoType
  Rate_C2                         AS  Array(NoComp) of Mass_Rate
```

```
UNIT
  R1                                  AS  m_R1
  R2                                  AS  m_R2
  R3                                  AS  m_R3
  R4                                  AS  m_R4
  FlashDrum1                          AS  m_FlashDrum1
  FlashDrum2                          AS  m_FlashDrum2

SELECTOR
  FlowType_C5    AS  (Turbulent, Laminar)
  FlowType_C4    AS  (Turbulent, Laminar)
  FlowType_C3    AS  (Turbulent, Laminar)
  FlowType_C2    AS  (Turbulent, Laminar)
  FlowType_C1    AS  (Turbulent, Laminar)

SET
  OUTLET          := 1 ;
  INLET           := 2 ;

EQUATION

                #==================================#
                # stream connections through ports #
                #==================================#
  R1.P                  IS      FlashDrum1.Pi ;
  FlashDrum1.Po1        IS      FlashDrum2.Pi ;
  FlashDrum1.Po2        IS      R4.P ;
  FlashDrum2.Po1        IS      R2.P ;
  FlashDrum2.Po2        IS      R3.P ;


                #==================================#
                # transfer law of each connection #
                #==================================#
# "IrreversiblePressureDrivenFlow" in connection, "C1"
DrivingForce_C1 = R1.Press_P - FlashDrum1.Press_FlashDrum1 ;
(4/3.14) * SIGMA(R1.Rate_P) = ReynoldsNo_C1 * Const_C1 * R1.Viscosity_P ;
ReynoldsConst1_C1 = 2100 ;
ReynoldsConst2_C1 = 4000 ;
R1.EnthFlow_P = SIGMA(R1.Rate_P) * R1.Enth_P ;
IF DrivingForce_C1 > 0  THEN
  CASE FlowType_C1 OF
    WHEN Turbulent :
      R1.Rate_P = Const2_C1 * R1.Den_P * R1.MassFrac_P *
                  SQRT(DrivingForce_C1) ;
      SWITCH TO Laminar IF ReynoldsNo_C1 < ReynoldsConst1_C1 ;
    WHEN Laminar :
      R1.Rate_P = Const3_C1 * R1.Den_P * R1.MassFrac_P * DrivingForce_C1 ;
      SWITCH TO Turbulent IF ReynoldsNo_C1 > ReynoldsConst2_C1 ;
  END
ELSE
  R1.Rate_P = 0 ;
END

# "PressureDrivenFlow" in connection, "C2"
DrivingForce_C2 = FlashDrum1.Press_FlashDrum1 - FlashDrum2.Press_FlashDrum2 ;
(4/3.14) * SIGMA(Rate_C2) = ReynoldsNo_C2 * Const_C2 * Viscosity_C2 ;
EnthFlow_C2 = SGN(DrivingForce_C2) * SIGMA(ABS(Rate_C2)) * Enth_C2 ;
IF DrivingForce_C2 > 0  THEN
  FlashDrum1.PortType_Po1 = OUTLET ;
  FlashDrum2.PortType_Pi = INLET ;
  Rate_C2 = FlashDrum1.Rate_Po1 ;
  EnthFlow_C2 = FlashDrum1.EnthFlow_Po1 ;
  Viscosity_C2 = FlashDrum1.Viscosity_Po1 ;
  Den_C2 = FlashDrum1.Den_Po1 ;
  MassFrac_C2 = FlashDrum1.MassFrac_Po1 ;
  Enth_C2 = FlashDrum1.Enth_Po1 ;
```

```
ELSE
  FlashDrum1.PortType_Po1 = INLET ;
  FlashDrum2.PortType_Pi = OUTLET ;
  Rate_C2 = FlashDrum2.Rate_Pi ;
  EnthFlow_C2 = FlashDrum2.EnthFlow_Pi ;
  Viscosity_C2 = FlashDrum2.Viscosity_Pi ;
  Den_C2 = FlashDrum2.Den_Pi ;
  MassFrac_C2 = FlashDrum2.MassFrac_Pi ;
  Enth_C2 = FlashDrum2.Enth_Pi ;
END
CASE FlowType_C2 OF
  WHEN Turbulent :
    Rate_C2 = Const2_C2 * Den_C2 * MassFrac_C2 * SGN(DrivingForce_C2) *
              SQRT(ABS(DrivingForce_C2)) ;
    SWITCH TO Laminar IF ReynoldsNo_C2 < 2100 ;
  WHEN Laminar :
    Rate_C2 = Const3_C2 * Den_C2 * MassFrac_C2 * DrivingForce_C2 ;
    SWITCH TO Turbulent IF ReynoldsNo_C2 > 4000 ;
END

# "IrreversiblePressureDrivenFlow" in connection, "C3"
DrivingForce_C3 = FlashDrum1.Press_FlashDrum1 - R4.Press_P ;
(4/3.14) * SIGMA(FlashDrum1.Rate_Po2) = ReynoldsNo_C3 * Const_C3 *
                                        FlashDrum1.Viscosity_Po2 ;
ReynoldsConst1_C3 = 2100 ;
ReynoldsConst2_C3 = 4000 ;
FlashDrum1.EnthFlow_Po2 = SIGMA(FlashDrum1.Rate_Po2) * FlashDrum1.Enth_Po2 ;
IF DrivingForce_C3 > 0  THEN
  CASE FlowType_C3 OF
    WHEN Turbulent :
      FlashDrum1.Rate_Po2 = Const2_C3 * FlashDrum1.Den_Po2 *
                            FlashDrum1.MassFrac_Po2 * SQRT(DrivingForce_C3) ;
      SWITCH TO Laminar IF ReynoldsNo_C3 < ReynoldsConst1_C3 ;
    WHEN Laminar :
      FlashDrum1.Rate_Po2 = Const3_C3 * FlashDrum1.Den_Po2 *
                            FlashDrum1.MassFrac_Po2 * DrivingForce_C3 ;
      SWITCH TO Turbulent IF ReynoldsNo_C3 > ReynoldsConst2_C3 ;
  END
ELSE
  FlashDrum1.Rate_Po2 = 0 ;
END

# "IrreversiblePressureDrivenFlow" in connection, "C4"
DrivingForce_C4 = FlashDrum2.Press_FlashDrum2 - R2.Press_P ;
(4/3.14) * SIGMA(FlashDrum2.Rate_Po1) = ReynoldsNo_C4 * Const_C4 *
                                        FlashDrum2.Viscosity_Po1 ;
ReynoldsConst1_C4 = 2100 ;
ReynoldsConst2_C4 = 4000 ;
FlashDrum2.EnthFlow_Po1 = SIGMA(FlashDrum2.Rate_Po1) * FlashDrum2.Enth_Po1 ;
IF DrivingForce_C4 > 0  THEN
  CASE FlowType_C4 OF
    WHEN Turbulent :
      FlashDrum2.Rate_Po1 = Const2_C4 * FlashDrum2.Den_Po1 *
                            FlashDrum2.MassFrac_Po1 * SQRT(DrivingForce_C4) ;
      SWITCH TO Laminar IF ReynoldsNo_C4 < ReynoldsConst1_C4 ;
    WHEN Laminar :
      FlashDrum2.Rate_Po1 = Const3_C4 * FlashDrum2.Den_Po1 *
                            FlashDrum2.MassFrac_Po1 * DrivingForce_C4 ;
      SWITCH TO Turbulent IF ReynoldsNo_C4 > ReynoldsConst2_C4 ;
  END
ELSE
  FlashDrum2.Rate_Po1 = 0 ;
END

# "IrreversiblePressureDrivenFlow" in connection, "C5"
DrivingForce_C5 = FlashDrum2.Press_FlashDrum2 - R3.Press_P ;
(4/3.14) * SIGMA(FlashDrum2.Rate_Po2) = ReynoldsNo_C5 * Const_C5 *
                                        FlashDrum2.Viscosity_Po2 ;
```

```
ReynoldsConst1_C5 = 2100 ;
ReynoldsConst2_C5 = 4000 ;
FlashDrum2.EnthFlow_Po2 = SIGMA(FlashDrum2.Rate_Po2) * FlashDrum2.Enth_Po2 ;
IF DrivingForce_C5 > 0  THEN
  CASE FlowType_C5 OF
    WHEN Turbulent :
      FlashDrum2.Rate_Po2 = Const2_C5 * FlashDrum2.Den_Po2 *
                            FlashDrum2.MassFrac_Po2 * SQRT(DrivingForce_C5) ;
      SWITCH TO Laminar IF ReynoldsNo_C5 < ReynoldsConst1_C5 ;
    WHEN Laminar :
      FlashDrum2.Rate_Po2 = Const3_C5 * FlashDrum2.Den_Po2 *
                            FlashDrum2.MassFrac_Po2 * DrivingForce_C5 ;
      SWITCH TO Turbulent IF ReynoldsNo_C5 > ReynoldsConst2_C5 ;
  END
ELSE
  FlashDrum2.Rate_Po2 = 0 ;
END

      #============================================================#
      # selection of ratio and phase type on reversible connection #
      #============================================================#
# ratio and phase type on "reversible" connection, "C2"
IF FlashDrum1.PortType_Po1 = OUTLET  THEN
  IF FlashDrum1.Z_Po1 > FlashDrum1.Bot_V AND
     FlashDrum1.Z_Po1 <= FlashDrum1.Top_V  THEN
    FlashDrum1.Ratio_V = FlashDrum1.Ratio_Po1 ;
    FlashDrum1.PhaseType_V = FlashDrum1.PhaseType_Po1 ;
  ELSE
    FlashDrum1.Ratio_B = FlashDrum1.Ratio_Po1 ;
    FlashDrum1.PhaseType_B_L = FlashDrum1.PhaseType_Po1 ;
  END
ELSE
    IF FlashDrum2.Z_Pi > FlashDrum2.Bot_V AND
       FlashDrum2.Z_Pi <= FlashDrum2.Top_V  THEN
      FlashDrum2.Ratio_V = FlashDrum2.Ratio_Pi ;
      FlashDrum2.PhaseType_V = FlashDrum2.PhaseType_Pi ;
    ELSE
      FlashDrum2.Ratio_B = FlashDrum2.Ratio_Pi ;
      FlashDrum2.PhaseType_B_L = FlashDrum2.PhaseType_Pi ;
    END
  END


END  # end of MODEL Flowsheet

############################
# END of generated model #
############################

# ======= #
# Process #
# ======= #

PROCESS test

UNIT
  Plant AS Flowsheet

SET
  WITHIN Plant DO
    NoComp          := 2 ;
    Const_C1        := 0.05 ;
    Const_C2        := 0.05 ;
    Const_C3        := 0.05 ;
    Const_C4        := 0.05 ;
    Const_C5        := 0.05 ;
    Const2_C1       := 1E-5 ;
    Const2_C2       := 5*1E-5 ;
```

```
    Const2_C3       := 1E-5 ;
    Const2_C4       := 1E-5 ;
    Const2_C5       := 1E-5 ;
    Const3_C1       := 1E-6 ;
    Const3_C2       := 5*1E-6 ;
    Const3_C3       := 1E-6 ;
    Const3_C4       := 1E-6 ;
    Const3_C5       := 1E-6 ;
    WITHIN R1 DO
      NoComp        := 2 ;
    END
    WITHIN R2 DO
      NoComp        := 2 ;
    END
    WITHIN R3 DO
      NoComp        := 2 ;
    END
    WITHIN R4 DO
      NoComp        := 2 ;
    END
    WITHIN FlashDrum1 DO
      NoComp        := 2 ;
      Const_B_V     := 1E-5 ;
    END
    WITHIN FlashDrum2 DO
      NoComp        := 2 ;
      Const_B_V     := 1E-5 ;
    END
  END

ASSIGN
  WITHIN Plant DO
WITHIN R1 DO
  Press_P         := 1.013E2 ;
  Den_P           := 330 ;
  Enth_P          := 3.5E2 ;
  MassFrac_P      := [0.4, 0.6] ;
  Ratio_P         := 0.01 ;
  PhaseType_P     := 3 ;
  Viscosity_P     := 5E-5 ;
END # R1
WITHIN R2 DO
  Press_P         := 1.013E2 ;
END # R2
WITHIN R3 DO
  Press_P         := 1.013E2 ;
END # R3
WITHIN R4 DO
  Press_P         := 1.013E2 ;
END # R3
WITHIN FlashDrum1 DO
  BubTemp_B_L     := 240 ;
  DewTemp_B_L     := 300 ;
  Temp_co(1)      := 369.82 ;
  Temp_co(2)      := 425.15 ;

  A(1)            := 8.03820 ;
  A(2)            := 8.29348 ;
  B(1)            := 3.49075 ;
  B(2)            := 3.46000 ;
  C(1)            := -3.96060 ;
  C(2)            := -4.02109 ;
  D(1)            := 27.52980 ;
  D(2)            := 30.35096 ;
  E(1)            := 166.170 ;
  E(2)            := 153.044 ;
  R               := 8.31433 ;      # [kJ/kmol K]
  Wo(1)           := 0.1454 ;
```

```
Wo(2)          := 0.1928 ;
MoleWeight(1)  := 44.09 ;
MoleWeight(2)  := 58.12 ;

vap1(1)        := 5.4276E1 ;
vap1(2)        := 6.2570E1 ;
vap2(1)        := -3.3680E3 ;
vap2(2)        := -4.3220E3 ;
vap3(1)        := -5.2610E0 ;
vap3(2)        := -6.3640E0 ;
vap4(1)        := 8.6000E-6 ;
vap4(2)        := 6.8000E-6 ;
vap5(1)        := 2.0000E0 ;
vap5(2)        := 2.0000E0 ;

{
      F(1)          := 4.843 ;
      F(2)          := 5.273 ;
      G(1)          := 1245.3 ;
      G(2)          := 1747.2 ;
}

      Av(1)         := 2.2090E-6 ;
      Av(2)         := 1.0310E-5 ;
      Bv(1)         := 3.8240E-1 ;
      Bv(2)         := 2.0770E-1 ;
      Cv(1)         := 4.0500E2 ;
      Cv(2)         := 1.0055E3 ;
      Dv(1)         := 0 ;
      Dv(2)         := 8.1000E3 ;
      Al(1)         := -1.2832E1 ;
      Al(2)         := 7.5000E-1 ;
      Bl(1)         := 5.6634E2 ;
      Bl(2)         := 2.1870E2 ;
      Cl(1)         := 3.4688E-1 ;
      Cl(2)         := -1.7882 ;
      Dl(1)         := -3.5111E-26 ;
      Dl(2)         := -4.0000E-27 ;
      El(1)         := 1.0000E1 ;
      El(2)         := 1.0000E1 ;

{
      Den_L         := 600 ;
      Viscosity_V   := 7.9E-6 ;
      Viscosity_B_L := 5E-5 ;
}

      Press_co(1)   := 4248 ;
      Press_co(2)   := 3795 ;
      Zo(1)         := 0.27664 ;
      Zo(2)         := 0.27331 ;
   END # within FlashDrum1
   WITHIN FlashDrum2 DO
      BubTemp_B_L   := 240 ;
      DewTemp_B_L   := 300 ;
      Temp_co(1)    := 369.82 ;
      Temp_co(2)    := 425.15 ;

      A(1)          := 8.03820 ;
      A(2)          := 8.29348 ;
      B(1)          := 3.49075 ;
      B(2)          := 3.46000 ;
      C(1)          := -3.96060 ;
      C(2)          := -4.02109 ;
      D(1)          := 27.52980 ;
      D(2)          := 30.35096 ;
      E(1)          := 166.170 ;
      E(2)          := 153.044 ;
```

```
R               := 8.31433 ;      # [kJ/kmol K]
Wo(1)           := 0.1454 ;
Wo(2)           := 0.1928 ;
MoleWeight(1)   := 44.09 ;
MoleWeight(2)   := 58.12 ;

vap1(1)         := 5.4276E1 ;
vap1(2)         := 6.2570E1 ;
vap2(1)         := -3.3680E3 ;
vap2(2)         := -4.3220E3 ;
vap3(1)         := -5.2610E0 ;
vap3(2)         := -6.3640E0 ;
vap4(1)         := 8.6000E-6 ;
vap4(2)         := 6.8000E-6 ;
vap5(1)         := 2.0000E0 ;
vap5(2)         := 2.0000E0 ;

{
    F(1)        := 4.843 ;
    F(2)        := 5.273 ;
    G(1)        := 1245.3 ;
    G(2)        := 1747.2 ;
}

    Av(1)       := 2.2090E-6 ;
    Av(2)       := 1.0310E-5 ;
    Bv(1)       := 3.8240E-1 ;
    Bv(2)       := 2.0770E-1 ;
    Cv(1)       := 4.0500E2 ;
    Cv(2)       := 1.0055E3 ;
    Dv(1)       := 0 ;
    Dv(2)       := 8.1000E3 ;
    Al(1)       := -1.2832E1 ;
    Al(2)       := 7.5000E-1 ;
    Bl(1)       := 5.6634E2 ;
    Bl(2)       := 2.1870E2 ;
    Cl(1)       := 3.4688E-1 ;
    Cl(2)       := -1.7882 ;
    Dl(1)       := -3.5111E-26 ;
    Dl(2)       := -4.0000E-27 ;
    El(1)       := 1.0000E1 ;
    El(2)       := 1.0000E1 ;

{
    Den_L       := 600 ;
    Viscosity_V := 7.9E-6 ;
    Viscosity_B_L := 5E-5 ;
}

    Press_co(1) := 4248 ;
    Press_co(2) := 3795 ;
    Zo(1)       := 0.27664 ;
    Zo(2)       := 0.27331 ;
  END # within FlashDrum2
END # within Plant



PRESET
  PLANT.FLASHDRUM1.ENTHFLOW_V_P02 := 0.00000E+00 : -1.000E+09 : 1.000E+07 ;
  PLANT.FLASHDRUM1.ENTHFLOW_B_L_P01 := 0.00000E+00 : -1.000E+09 : 1.000E+07 ;
  PLANT.FLASHDRUM1.PORTTYPE_P01 := 1.00000E+00 : -1.000E-03 : 1.000E+09 ;
  PLANT.FLASHDRUM1.ENTHFLOW_B_L_P02 := 0.00000E+00 : -1.000E+09 : 1.000E+07 ;
  PLANT.FLASHDRUM2.ENTHFLOW_V_P02 := 0.00000E+00 : -1.000E+09 : 1.000E+07 ;
  PLANT.FLASHDRUM2.ENTHFLOW_B_L_P01 := 0.00000E+00 : -1.000E+09 : 1.000E+07 ;
  PLANT.FLASHDRUM2.ENTH_PI := 0.00000E+00 : -1.000E+07 : 1.000E+04 ;
  PLANT.FLASHDRUM2.ENTHFLOW_B_L_P02 := 0.00000E+00 : -1.000E+09 : 1.000E+07 ;
```

```
PLANT.FLASHDRUM2.VOL_B := 5.91120E-02 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM2.PHASETYPE_B := 1.00000E+00 : -1.000E-03 : 1.000E+09 ;
PLANT.FLASHDRUM2.RATE_PO1(1) := 0.00000E+00 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM2.RATE_PO1(2) := 0.00000E+00 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM2.MASSFRAC_B(1) := 8.85860E-01 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM2.MASSFRAC_B(2) := 1.14140E-01 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM2.ENTH_LO(1) := 1.70751E+02 : -1.000E+07 : 1.000E+04 ;
PLANT.FLASHDRUM2.ENTH_LO(2) := 1.49773E+02 : -1.000E+07 : 1.000E+04 ;
PLANT.FLASHDRUM2.RATE_PI(1) := 1.03162E-05 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM2.RATE_PI(2) := 1.03162E-05 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM2.ENTHFLOW_PI := 1.45923E-02 : -1.000E+09 : 1.000E+07 ;
PLANT.FLASHDRUM2.RATIO_PI := 1.00000E+00 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM2.PHASETYPE_PI := 1.00000E+00 : -1.000E-03 : 1.000E+09 ;
PLANT.FLASHDRUM2.RATE_PO2(1) := 0.00000E+00 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM2.RATE_PO2(2) := 0.00000E+00 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM2.INTENERGY_B := 5.56946E+02 : -1.000E+09 : 1.000E+04 ;
PLANT.FLASHDRUM2.DER_MASS_B_L := 0.00000E+00 : -1.000E+09 : 1.000E+09 ;
PLANT.FLASHDRUM2.DEN_BO(1) := 9.87056E-01 : -1.000E-01 : 1.000E+05 ;
PLANT.FLASHDRUM2.DEN_BO(2) := 1.30115E+00 : -1.000E-01 : 1.000E+05 ;
PLANT.FLASHDRUM2.ENTH_PO1 := 6.73665E+02 : -1.000E+07 : 1.000E+04 ;
PLANT.FLASHDRUM2.MOLEFRAC_L(1) := 5.67874E-01 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM2.MOLEFRAC_L(2) := 4.32126E-01 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM2.ENTH_PO2 := 6.73665E+02 : -1.000E+07 : 1.000E+04 ;
PLANT.FLASHDRUM2.ENTHFLOW_B_V := 5.37878E-03 : -1.000E+09 : 1.000E+07 ;
PLANT.FLASHDRUM2.LEVEL_V := 3.00000E+00 : -1.000E-03 : 1.000E+09 ;
PLANT.FLASHDRUM2.EQUILCONST_B_L(1) := 1.60416E+00 : -1.000E-03 : 1.000E+09 ;
PLANT.FLASHDRUM2.EQUILCONST_B_L(2) := 2.06052E-01 : -1.000E-03 : 1.000E+09 ;
PLANT.FLASHDRUM2.VISCOSITY_BO(1) := 6.20289E-06 : -1.000E-01 : 1.000E+02 ;
PLANT.FLASHDRUM2.VISCOSITY_BO(2) := 5.58908E-06 : -1.000E-01 : 1.000E+02 ;
PLANT.FLASHDRUM2.PRESS_B := 4.13075E+01 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM2.LEVEL_B_L := 1.36386E-01 : -1.000E-03 : 1.000E+09 ;
PLANT.FLASHDRUM2.ENTHFLOW_PO1 := 0.00000E+00 : -1.000E+09 : 1.000E+07 ;
PLANT.FLASHDRUM2.RATIO_PO1 := 1.00000E+00 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM2.PHASETYPE_PO1 := 1.00000E+00 : -1.000E-03 : 1.000E+09 ;
PLANT.FLASHDRUM2.PRESS_C := 4.05225E+03 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM2.SUM_MASS_B := 6.00000E-02 : -1.000E+09 : 1.000E+09 ;
PLANT.FLASHDRUM2.ENTHFLOW_PO2 := 0.00000E+00 : -1.000E+09 : 1.000E+07 ;
PLANT.FLASHDRUM2.RATIO_PO2 := 1.00000E+00 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM2.PHASETYPE_PO2 := 1.00000E+00 : -1.000E-03 : 1.000E+09 ;
PLANT.FLASHDRUM2.BOT_V := 1.36386E-01 : -1.000E-03 : 1.000E+09 ;
PLANT.FLASHDRUM2.BOT_B_L := 0.00000E+00 : -1.000E-03 : 1.000E+09 ;
PLANT.FLASHDRUM2.MOLE_B(1) := 1.20552E-03 : -1.000E-01 : 1.000E+02 ;
PLANT.FLASHDRUM2.MOLE_B(2) := 1.17832E-04 : -1.000E-01 : 1.000E+02 ;
PLANT.FLASHDRUM2.VOL_L := 4.79511E-02 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM2.PHASETYPE_L := 2.00000E+00 : -1.000E-03 : 1.000E+09 ;
PLANT.FLASHDRUM2.DEN_PI := 0.00000E+00 : -1.000E-01 : 1.000E+05 ;
PLANT.FLASHDRUM2.MASSFRAC_L(1) := 4.99227E-01 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM2.MASSFRAC_L(2) := 5.00773E-01 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM2.ENTH_VO(1) := 6.87779E+02 : -1.000E+07 : 1.000E+04 ;
PLANT.FLASHDRUM2.ENTH_VO(2) := 6.59551E+02 : -1.000E+07 : 1.000E+04 ;
PLANT.FLASHDRUM2.VOL_FLASHDRUM2 := 2.35500E+00 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM2.ENTH_B := 5.97642E+02 : -1.000E+07 : 1.000E+04 ;
PLANT.FLASHDRUM2.INTENERGY_L := 1.60180E+02 : -1.000E+09 : 1.000E+04 ;
PLANT.FLASHDRUM2.RATIO_V := 1.00000E+00 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM2.VISCOSITY_PI := 0.00000E+00 : -1.000E-01 : 1.000E+02 ;
PLANT.FLASHDRUM2.TEMP_RO(1) := 6.00081E-01 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM2.TEMP_RO(2) := 5.21985E-01 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM2.VISCOSITY_LO(1) := 4.55853E-14 : -1.000E-01 : 1.000E+02 ;
PLANT.FLASHDRUM2.VISCOSITY_LO(2) := 1.31680E-04 : -1.000E-01 : 1.000E+02 ;
PLANT.FLASHDRUM2.DEN_PO1 := 8.89705E-01 : -1.000E-01 : 1.000E+05 ;
PLANT.FLASHDRUM2.PRESS_L := 4.13075E+01 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM2.MASS_B(1) := 5.31516E-02 : -1.000E-01 : 1.000E+09 ;
PLANT.FLASHDRUM2.MASS_B(2) := 6.84842E-03 : -1.000E-01 : 1.000E+09 ;
PLANT.FLASHDRUM2.DEN_PO2 := 8.89705E-01 : -1.000E-01 : 1.000E+05 ;
PLANT.FLASHDRUM2.SUM_MASS_L := 2.99400E+01 : -1.000E+09 : 1.000E+09 ;
PLANT.FLASHDRUM2.VISCOSITY_PO1 := 9.40000E-06 : -1.000E-01 : 1.000E+02 ;
PLANT.FLASHDRUM2.PRESS_FLASHDRUM2 := 4.13075E+01 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM2.VISCOSITY_PO2 := 9.40000E-06 : -1.000E-01 : 1.000E+02 ;
```

```
PLANT.FLASHDRUM2.TEMP_B := 2.21922E+02 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM2.MOLE_L(1) := 3.39007E-01 : -1.000E-01 : 1.000E+02 ;
PLANT.FLASHDRUM2.MOLE_L(2) := 2.57969E-01 : -1.000E-01 : 1.000E+02 ;
PLANT.FLASHDRUM2.VOL_V := 2.24794E+00 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM2.PHASETYPE_V := 1.00000E+00 : -1.000E-03 : 1.000E+09 ;
PLANT.FLASHDRUM2.TEMP_C := 3.93730E+02 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM2.VAPPRESS_1 := 6.62637E+01 : -1.000E+09 : 1.000E+09 ;
PLANT.FLASHDRUM2.VOL_B_L := 1.07063E-01 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM2.MASSFRAC_V(1) := 5.00000E-01 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM2.MASSFRAC_V(2) := 5.00000E-01 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM2.PHASETYPE_B_L := 3.00000E+00 : -1.000E-03 : 1.000E+09 ;
PLANT.FLASHDRUM2.VAPPRESS_2 := 8.51150E+00 : -1.000E+09 : 1.000E+09 ;
PLANT.FLASHDRUM2.ENTH_L := 1.60246E+02 : -1.000E+07 : 1.000E+04 ;
PLANT.FLASHDRUM2.INTENERGY_V := 6.27236E+02 : -1.000E+09 : 1.000E+04 ;
PLANT.FLASHDRUM2.MASSFRAC_B_L(1) := 5.00000E-01 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM2.MASSFRAC_B_L(2) := 5.00000E-01 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM2.TOP_V := 3.00000E+00 : -1.000E-03 : 1.000E+09 ;
PLANT.FLASHDRUM2.INTENERGY_B_L := 1.60973E+02 : -1.000E+09 : 1.000E+04 ;
PLANT.FLASHDRUM2.TOP_B_L := 1.36386E-01 : -1.000E-03 : 1.000E+09 ;
PLANT.FLASHDRUM2.DEN_VO(1) := 7.82319E-01 : -1.000E-01 : 1.000E+05 ;
PLANT.FLASHDRUM2.DEN_VO(2) := 1.03126E+00 : -1.000E-01 : 1.000E+05 ;
PLANT.FLASHDRUM2.DEN_B := 1.01502E+00 : -1.000E-01 : 1.000E+05 ;
PLANT.FLASHDRUM2.VISCOSITY_VO(1) := 8.32490E-06 : -1.000E-01 : 1.000E+02 ;
PLANT.FLASHDRUM2.VISCOSITY_VO(2) := 7.58946E-06 : -1.000E-01 : 1.000E+02 ;
PLANT.FLASHDRUM2.M := 5.01527E+01 : -1.000E-03 : 1.000E+09 ;
PLANT.FLASHDRUM2.PRESS_V := 4.13075E+01 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM2.VISCOSITY_B := 5.89599E-06 : -1.000E-01 : 1.000E+02 ;
PLANT.FLASHDRUM2.EQUIL_1 := 1.60416E+00 : -1.000E+09 : 1.000E+09 ;
PLANT.FLASHDRUM2.MASS_L(1) := 1.49468E+01 : -1.000E-01 : 1.000E+09 ;
PLANT.FLASHDRUM2.MASS_L(2) := 1.49932E+01 : -1.000E-01 : 1.000E+09 ;
PLANT.FLASHDRUM2.RATE_V_PI(1) := 0.00000E+00 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM2.RATE_V_PI(2) := 0.00000E+00 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM2.PRESS_B_L := 4.13075E+01 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM2.SUM_MASS_V := 2.00000E+00 : -1.000E+09 : 1.000E+09 ;
PLANT.FLASHDRUM2.EQUIL_2 := 2.06052E-01 : -1.000E+09 : 1.000E+09 ;
PLANT.FLASHDRUM2.RATE_B_L_PI(1) := 1.03162E-05 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM2.RATE_B_L_PI(2) := 1.03162E-05 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM2.SUM_MASS_B_L := 3.00000E+01 : -1.000E+09 : 1.000E+09 ;
PLANT.FLASHDRUM2.TEMP_L := 2.21922E+02 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM2.AREA := 7.85000E-01 : -1.000E-03 : 1.000E+09 ;
PLANT.FLASHDRUM2.SUM_RATE_B_V := 9.00000E-06 : -1.000E+09 : 1.000E+09 ;
PLANT.FLASHDRUM2.MOLE_B_L(1) := 3.40213E-01 : -1.000E-01 : 1.000E+02 ;
PLANT.FLASHDRUM2.MOLE_B_L(2) := 2.58087E-01 : -1.000E-01 : 1.000E+02 ;
PLANT.FLASHDRUM2.RATE_V_PO1(1) := 0.00000E+00 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM2.RATE_V_PO1(2) := 0.00000E+00 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM2.ENTH_V := 6.73665E+02 : -1.000E+07 : 1.000E+04 ;
PLANT.FLASHDRUM2.RATE_B_L_PO1(1) := 0.00000E+00 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM2.RATE_B_L_PO1(2) := 0.00000E+00 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM2.RATE_V_PO2(1) := 0.00000E+00 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM2.RATE_V_PO2(2) := 0.00000E+00 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM2.ENTH_B_L := 1.61121E+02 : -1.000E+07 : 1.000E+04 ;
PLANT.FLASHDRUM2.RATE_B_L_PO2(1) := 0.00000E+00 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM2.RATE_B_L_PO2(2) := 0.00000E+00 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM2.DEN_L := 6.24386E+02 : -1.000E-01 : 1.000E+05 ;
PLANT.FLASHDRUM2.TEMP_R := 5.63640E-01 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM2.VISCOSITY_L := 6.58909E-05 : -1.000E-01 : 1.000E+02 ;
PLANT.FLASHDRUM2.DER_INTENERGY_V := 2.68939E-03 : -1.000E+09 : 1.000E+09 ;
PLANT.FLASHDRUM2.SUM_RATE_PI := 2.06323E-05 : -1.000E+09 : 1.000E+09 ;
PLANT.FLASHDRUM2.MASS_V(1) := 1.00000E+00 : -1.000E-01 : 1.000E+09 ;
PLANT.FLASHDRUM2.MASS_V(2) := 1.00000E+00 : -1.000E-01 : 1.000E+09 ;
PLANT.FLASHDRUM2.MASS_B_L(1) := 1.50000E+01 : -1.000E-01 : 1.000E+09 ;
PLANT.FLASHDRUM2.MASS_B_L(2) := 1.50000E+01 : -1.000E-01 : 1.000E+09 ;
PLANT.FLASHDRUM2.MASSFRAC_PI(1) := 0.00000E+00 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM2.MASSFRAC_PI(2) := 0.00000E+00 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM2.HEAT_LO(1) := 4.30058E+02 : -1.000E+07 : 1.000E+04 ;
PLANT.FLASHDRUM2.HEAT_LO(2) := 4.23293E+02 : -1.000E+07 : 1.000E+04 ;
PLANT.FLASHDRUM2.Z := 2.75201E-01 : -1.000E+09 : 1.000E+09 ;
PLANT.FLASHDRUM2.TEMP_V := 2.80000E+02 : -1.000E-01 : 1.000E+04 ;
```

```
PLANT.FLASHDRUM2.RATE_B_V(1) := 7.95751E-06 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM2.RATE_B_V(2) := 1.04249E-06 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM2.ENTH_BO(1) := 6.00809E+02 : -1.000E+07 : 1.000E+04 ;
PLANT.FLASHDRUM2.ENTH_BO(2) := 5.73066E+02 : -1.000E+07 : 1.000E+04 ;
PLANT.FLASHDRUM2.SUM_RATE_PO1 := 0.00000E+00 : -1.000E+09 : 1.000E+09 ;
PLANT.FLASHDRUM2.ENTHFLOW_V_PI := 0.00000E+00 : -1.000E+09 : 1.000E+07 ;
PLANT.FLASHDRUM2.RATIO_B := 2.0000E-03 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM2.SUM_RATE_PO2 := 0.00000E+00 : -1.000E+09 : 1.000E+09 ;
PLANT.FLASHDRUM2.ENTHFLOW_B_L_PI := 1.45923E-02 : -1.000E+09 : 1.000E+07 ;
PLANT.FLASHDRUM2.MASSFRAC_PO1(1) := 5.00000E-01 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM2.MASSFRAC_PO1(2) := 5.00000E-01 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM2.PORTTYPE_PI := 2.00000E+00 : -1.000E-03 : 1.000E+09 ;
PLANT.FLASHDRUM2.DEN_V := 8.89705E-01 : -1.000E-01 : 1.000E+05 ;
PLANT.FLASHDRUM2.MASSFRAC_PO2(1) := 5.00000E-01 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM2.MASSFRAC_PO2(2) := 5.00000E-01 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM2.MOLEFRAC_B(1) := 9.10959E-01 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM2.MOLEFRAC_B(2) := 8.90405E-02 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM2.DEN_B_L := 2.80209E+02 : -1.000E-01 : 1.000E+05 ;
PLANT.FLASHDRUM2.VISCOSITY_V := 7.95718E-06 : -1.000E-01 : 1.000E+02 ;
PLANT.FLASHDRUM2.VISCOSITY_B_L := 6.58309E-05 : -1.000E-01 : 1.000E+02 ;
PLANT.FLASHDRUM2.VAPPRESS_L(1) := 6.62637E+01 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM2.VAPPRESS_L(2) := 8.51150E+00 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM2.ENTHFLOW_V_PO1 := 0.00000E+00 : -1.000E+09 : 1.000E+07 ;
PLANT.FLASHDRUM1.VOL_B := 2.80947E-02 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM1.PHASETYPE_B := 1.00000E+00 : -1.000E-03 : 1.000E+09 ;
PLANT.FLASHDRUM1.RATE_PO1(1) := 1.03162E-05 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM1.RATE_PO1(2) := 1.03162E-05 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM1.MASSFRAC_B(1) := 8.84494E-01 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM1.MASSFRAC_B(2) := 1.15506E-01 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM1.ENTH_LO(1) := 1.73454E+02 : -1.000E+07 : 1.000E+04 ;
PLANT.FLASHDRUM1.ENTH_LO(2) := 1.52137E+02 : -1.000E+07 : 1.000E+04 ;
PLANT.FLASHDRUM1.RATE_PI(1) := 1.25494E-01 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM1.RATE_PI(2) := 1.88241E-01 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM1.ENTHFLOW_PI := 1.09807E+02 : -1.000E+09 : 1.000E+07 ;
PLANT.FLASHDRUM1.RATIO_PI := 4.00000E-01 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM1.PHASETYPE_PI := 3.00000E+00 : -1.000E-03 : 1.000E+09 ;
PLANT.FLASHDRUM1.RATE_PO2(1) := 0.00000E+00 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM1.RATE_PO2(2) := 0.00000E+00 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM1.INTENERGY_B := 5.58220E+02 : -1.000E+09 : 1.000E+04 ;
PLANT.FLASHDRUM1.DER_MASS_B_L := 3.13726E-01 : -1.000E+09 : 1.000E+09 ;
PLANT.FLASHDRUM1.DEN_BO(1) := 1.03804E+00 : -1.000E-01 : 1.000E+05 ;
PLANT.FLASHDRUM1.DEN_BO(2) := 1.36836E+00 : -1.000E-01 : 1.000E+05 ;
PLANT.FLASHDRUM1.ENTH_PO1 := 7.07255E+02 : -1.000E+07 : 1.000E+04 ;
PLANT.FLASHDRUM1.MOLEFRAC_L(1) := 5.68255E-01 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM1.MOLEFRAC_L(2) := 4.31745E-01 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM1.ENTH_PO2 := 7.07255E+02 : -1.000E+07 : 1.000E+04 ;
PLANT.FLASHDRUM1.ENTHFLOW_B_V := 5.39479E-03 : -1.000E+09 : 1.000E+07 ;
PLANT.FLASHDRUM1.LEVEL_V := 3.00000E+00 : -1.000E-03 : 1.000E+09 ;
PLANT.FLASHDRUM1.EQUILCONST_B_L(1) := 1.60115E+00 : -1.000E-03 : 1.000E+09 ;
PLANT.FLASHDRUM1.EQUILCONST_B_L(2) := 2.08772E-01 : -1.000E-03 : 1.000E+09 ;
PLANT.FLASHDRUM1.VISCOSITY_BO(1) := 6.20289E-06 : -1.000E-01 : 1.000E+02 ;
PLANT.FLASHDRUM1.VISCOSITY_BO(2) := 5.58908E-06 : -1.000E-01 : 1.000E+02 ;
PLANT.FLASHDRUM1.PRESS_B := 4.36584E+01 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM1.LEVEL_B_L := 9.70524E-02 : -1.000E-03 : 1.000E+09 ;
PLANT.FLASHDRUM1.ENTHFLOW_PO1 := 1.45923E-02 : -1.000E+09 : 1.000E+07 ;
PLANT.FLASHDRUM1.RATIO_PO1 := 1.00000E+00 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM1.PHASETYPE_PO1 := 1.00000E+00 : -1.000E-03 : 1.000E+09 ;
PLANT.FLASHDRUM1.PRESS_C := 4.05242E+03 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM1.SUM_MASS_B := 3.00000E-02 : -1.000E+09 : 1.000E+09 ;
PLANT.FLASHDRUM1.ENTHFLOW_PO2 := 0.00000E+00 : -1.000E+09 : 1.000E+07 ;
PLANT.FLASHDRUM1.RATIO_PO2 := 1.00000E+00 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM1.PHASETYPE_PO2 := 1.00000E+00 : -1.000E-03 : 1.000E+09 ;
PLANT.FLASHDRUM1.BOT_V := 9.70524E-02 : -1.000E-03 : 1.000E+09 ;
PLANT.FLASHDRUM1.BOT_B_L := 0.00000E+00 : -1.000E-03 : 1.000E+09 ;
PLANT.FLASHDRUM1.MOLE_B(1) := 6.01833E-04 : -1.000E-01 : 1.000E+02 ;
PLANT.FLASHDRUM1.MOLE_B(2) := 5.96210E-05 : -1.000E-01 : 1.000E+02 ;
PLANT.FLASHDRUM1.VOL_L := 4.80914E-02 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM1.PHASETYPE_L := 2.00000E+00 : -1.000E-03 : 1.000E+09 ;
```

```
PLANT.FLASHDRUM1.VOL_FLASHDRUM1 := 2.35500E+00 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM1.MASSFRAC_L(1) := 4.99615E-01 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM1.MASSFRAC_L(2) := 5.00385E-01 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM1.ENTH_VO(1) := 7.21461E+02 : -1.000E+07 : 1.000E+04 ;
PLANT.FLASHDRUM1.ENTH_VO(2) := 6.93049E+02 : -1.000E+07 : 1.000E+04 ;
PLANT.FLASHDRUM1.ENTH_B := 5.99106E+02 : -1.000E+07 : 1.000E+04 ;
PLANT.FLASHDRUM1.INTENERGY_L := 1.62717E+02 : -1.000E+09 : 1.000E+04 ;
PLANT.FLASHDRUM1.RATIO_V := 1.00000E+00 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM1.TEMP_RO(1) := 6.03079E-01 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM1.TEMP_RO(2) := 5.24593E-01 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM1.VISCOSITY_LO(1) := 4.55853E-14 : -1.000E-01 : 1.000E+02 ;
PLANT.FLASHDRUM1.VISCOSITY_LO(2) := 1.31680E-04 : -1.000E-01 : 1.000E+02 ;
PLANT.FLASHDRUM1.DEN_PO1 := 8.77650E-01 : -1.000E-01 : 1.000E+05 ;
PLANT.FLASHDRUM1.PRESS_L := 4.36584E+01 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM1.MASS_B(1) := 2.65348E-02 : -1.000E-01 : 1.000E+09 ;
PLANT.FLASHDRUM1.MASS_B(2) := 3.46517E-03 : -1.000E-01 : 1.000E+09 ;
PLANT.FLASHDRUM1.DEN_PO2 := 8.77650E-01 : -1.000E-01 : 1.000E+05 ;
PLANT.FLASHDRUM1.PRESS_FLASHDRUM1 := 4.36584E+01 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM1.SUM_MASS_L := 2.99700E+01 : -1.000E+09 : 1.000E+09 ;
PLANT.FLASHDRUM1.VISCOSITY_PO1 := 9.40000E-06 : -1.000E-01 : 1.000E+02 ;
PLANT.FLASHDRUM1.VISCOSITY_PO2 := 9.40000E-06 : -1.000E-01 : 1.000E+02 ;
PLANT.FLASHDRUM1.TEMP_B := 2.23031E+02 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM1.MOLE_L(1) := 3.39611E-01 : -1.000E-01 : 1.000E+02 ;
PLANT.FLASHDRUM1.MOLE_L(2) := 2.58027E-01 : -1.000E-01 : 1.000E+02 ;
PLANT.FLASHDRUM1.VOL_V := 2.27881E+00 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM1.PHASETYPE_V := 1.00000E+00 : -1.000E-03 : 1.000E+09 ;
PLANT.FLASHDRUM1.TEMP_C := 3.93708E+02 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM1.VAPPRESS_1 := 6.99037E+01 : -1.000E+09 : 1.000E+09 ;
PLANT.FLASHDRUM1.VOL_B_L := 7.61861E-02 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM1.MASSFRAC_V(1) := 5.00000E-01 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM1.MASSFRAC_V(2) := 5.00000E-01 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM1.PHASETYPE_B_L := 3.00000E+00 : -1.000E-03 : 1.000E+09 ;
PLANT.FLASHDRUM1.VAPPRESS_2 := 9.11464E+00 : -1.000E+09 : 1.000E+09 ;
PLANT.FLASHDRUM1.ENTH_L := 1.62787E+02 : -1.000E+07 : 1.000E+04 ;
PLANT.FLASHDRUM1.INTENERGY_V := 6.57510E+02 : -1.000E+09 : 1.000E+04 ;
PLANT.FLASHDRUM1.MASSFRAC_B_L(1) := 5.00000E-01 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM1.MASSFRAC_B_L(2) := 5.00000E-01 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM1.TOP_V := 3.00000E+00 : -1.000E-03 : 1.000E+09 ;
PLANT.FLASHDRUM1.INTENERGY_B_L := 1.63113E+02 : -1.000E+09 : 1.000E+04 ;
PLANT.FLASHDRUM1.TOP_B_L := 9.70524E-02 : -1.000E-03 : 1.000E+09 ;
PLANT.FLASHDRUM1.DEN_VO(1) := 7.71719E-01 : -1.000E-01 : 1.000E+05 ;
PLANT.FLASHDRUM1.DEN_VO(2) := 1.01729E+00 : -1.000E-01 : 1.000E+05 ;
PLANT.FLASHDRUM1.DEN_B := 1.06782E+00 : -1.000E-01 : 1.000E+05 ;
PLANT.FLASHDRUM1.VISCOSITY_VO(1) := 8.32490E-06 : -1.000E-01 : 1.000E+02 ;
PLANT.FLASHDRUM1.VISCOSITY_VO(2) := 7.58946E-06 : -1.000E-01 : 1.000E+02 ;
PLANT.FLASHDRUM1.M := 5.01474E+01 : -1.000E-03 : 1.000E+09 ;
PLANT.FLASHDRUM1.PRESS_V := 4.36584E+01 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM1.VISCOSITY_B := 5.89599E-06 : -1.000E-01 : 1.000E+02 ;
PLANT.FLASHDRUM1.EQUIL_1 := 1.60115E+00 : -1.000E+09 : 1.000E+09 ;
PLANT.FLASHDRUM1.MASS_L(1) := 1.49735E+01 : -1.000E-01 : 1.000E+09 ;
PLANT.FLASHDRUM1.MASS_L(2) := 1.49965E+01 : -1.000E-01 : 1.000E+09 ;
PLANT.FLASHDRUM1.RATE_V_PI(1) := 0.00000E+00 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM1.RATE_V_PI(2) := 0.00000E+00 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM1.PRESS_B_L := 4.36584E+01 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM1.SUM_MASS_V := 2.00000E+00 : -1.000E+09 : 1.000E+09 ;
PLANT.FLASHDRUM1.EQUIL_2 := 2.08772E-01 : -1.000E+09 : 1.000E+09 ;
PLANT.FLASHDRUM1.RATE_B_L_PI(1) := 1.25494E-01 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM1.RATE_B_L_PI(2) := 1.88241E-01 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM1.SUM_MASS_B_L := 3.00000E+01 : -1.000E+09 : 1.000E+09 ;
PLANT.FLASHDRUM1.TEMP_L := 2.23031E+02 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM1.AREA := 7.85000E-01 : -1.000E-03 : 1.000E+09 ;
PLANT.FLASHDRUM1.SUM_RATE_B_V := 9.00000E-06 : -1.000E+09 : 1.000E+09 ;
PLANT.FLASHDRUM1.MOLE_B_L(1) := 3.40213E-01 : -1.000E-01 : 1.000E+02 ;
PLANT.FLASHDRUM1.MOLE_B_L(2) := 2.58087E-01 : -1.000E-01 : 1.000E+02 ;
PLANT.FLASHDRUM1.RATE_V_PO1(1) := 1.03162E-05 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM1.RATE_V_PO1(2) := 1.03162E-05 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM1.ENTH_V := 7.07255E+02 : -1.000E+07 : 1.000E+04 ;
PLANT.FLASHDRUM1.RATE_B_L_PO1(1) := 0.00000E+00 : -1.000E-01 : 1.000E+04 ;
```

```
PLANT.FLASHDRUM1.RATE_B_L_P01(2) := 0.00000E+00 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM1.RATE_V_P02(1) := 0.00000E+00 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM1.RATE_V_P02(2) := 0.00000E+00 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM1.ENTH_B_L := 1.63224E+02 : -1.000E+07 : 1.000E+04 ;
PLANT.FLASHDRUM1.RATE_B_L_P02(1) := 0.00000E+00 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM1.RATE_B_L_P02(2) := 0.00000E+00 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM1.DEN_L := 6.23188E+02 : -1.000E-01 : 1.000E+05 ;
PLANT.FLASHDRUM1.TEMP_R := 5.66487E-01 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM1.VISCOSITY_L := 6.58909E-05 : -1.000E-01 : 1.000E+02 ;
PLANT.FLASHDRUM1.DER_INTENERGY_V := -4.59876E-03 : -1.000E+09 : 1.000E+09 ;
PLANT.FLASHDRUM1.SUM_RATE_PI := 3.13735E-01 : -1.000E+09 : 1.000E+09 ;
PLANT.FLASHDRUM1.MASS_V(1) := 1.00000E+00 : -1.000E-01 : 1.000E+09 ;
PLANT.FLASHDRUM1.MASS_V(2) := 1.00000E+00 : -1.000E-01 : 1.000E+09 ;
PLANT.FLASHDRUM1.MASS_B_L(1) := 1.50000E+01 : -1.000E-01 : 1.000E+09 ;
PLANT.FLASHDRUM1.MASS_B_L(2) := 1.50000E+01 : -1.000E-01 : 1.000E+09 ;
PLANT.FLASHDRUM1.HEAT_LO(1) := 4.28858E+02 : -1.000E+07 : 1.000E+04 ;
PLANT.FLASHDRUM1.HEAT_LO(2) := 4.22423E+02 : -1.000E+07 : 1.000E+04 ;
PLANT.FLASHDRUM1.Z := 2.75202E-01 : -1.000E+09 : 1.000E+09 ;
PLANT.FLASHDRUM1.TEMP_V := 3.00000E+02 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM1.RATE_B_V(1) := 7.95751E-06 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM1.RATE_B_V(2) := 1.04249E-06 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM1.ENTH_BO(1) := 6.02311E+02 : -1.000E+07 : 1.000E+04 ;
PLANT.FLASHDRUM1.ENTH_BO(2) := 5.74560E+02 : -1.000E+07 : 1.000E+04 ;
PLANT.FLASHDRUM1.SUM_RATE_P01 := 2.06323E-05 : -1.000E+09 : 1.000E+09 ;
PLANT.FLASHDRUM1.ENTHFLOW_V_PI := 0.00000E+00 : -1.000E+09 : 1.000E+07 ;
PLANT.FLASHDRUM1.RATIO_B := 1.0000E-03 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM1.SUM_RATE_P02 := 0.00000E+00 : -1.000E+09 : 1.000E+09 ;
PLANT.FLASHDRUM1.ENTHFLOW_B_L_PI := 1.09807E+02 : -1.000E+09 : 1.000E+07 ;
PLANT.FLASHDRUM1.MASSFRAC_P01(1) := 5.00000E-01 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM1.MASSFRAC_P01(2) := 5.00000E-01 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM1.DEN_V := 8.77650E-01 : -1.000E-01 : 1.000E+05 ;
PLANT.FLASHDRUM1.MASSFRAC_P02(1) := 5.00000E-01 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM1.MASSFRAC_P02(2) := 5.00000E-01 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM1.MOLEFRAC_B(1) := 9.09864E-01 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM1.MOLEFRAC_B(2) := 9.01362E-02 : -1.000E-01 : 1.000E+01 ;
PLANT.FLASHDRUM1.DEN_B_L := 3.93772E+02 : -1.000E-01 : 1.000E+05 ;
PLANT.FLASHDRUM1.VISCOSITY_V := 7.95718E-06 : -1.000E-01 : 1.000E+02 ;
PLANT.FLASHDRUM1.VISCOSITY_B_L := 6.58309E-05 : -1.000E-01 : 1.000E+02 ;
PLANT.FLASHDRUM1.VAPPRESS_L(1) := 6.99037E+01 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM1.VAPPRESS_L(2) := 9.11464E+00 : -1.000E-01 : 1.000E+04 ;
PLANT.FLASHDRUM1.ENTHFLOW_V_P01 := 1.45923E-02 : -1.000E+09 : 1.000E+07 ;
PLANT.R4.RATIO_P := 1.00000E+00 : -1.000E-01 : 1.000E+01 ;
PLANT.R4.PHASETYPE_P := 1.00000E+00 : -1.000E-03 : 1.000E+09 ;
PLANT.R4.RATE_P(1) := 0.00000E+00 : -1.000E-01 : 1.000E+04 ;
PLANT.R4.RATE_P(2) := 0.00000E+00 : -1.000E-01 : 1.000E+04 ;
PLANT.R4.ENTHFLOW_P := 0.00000E+00 : -1.000E+09 : 1.000E+07 ;
PLANT.R3.RATIO_P := 1.00000E+00 : -1.000E-01 : 1.000E+01 ;
PLANT.R3.PHASETYPE_P := 1.00000E+00 : -1.000E-03 : 1.000E+09 ;
PLANT.R3.RATE_P(1) := 0.00000E+00 : -1.000E-01 : 1.000E+04 ;
PLANT.R3.RATE_P(2) := 0.00000E+00 : -1.000E-01 : 1.000E+04 ;
PLANT.R3.ENTHFLOW_P := 0.00000E+00 : -1.000E+09 : 1.000E+07 ;
PLANT.R2.RATIO_P := 1.00000E+00 : -1.000E-01 : 1.000E+01 ;
PLANT.R2.PHASETYPE_P := 1.00000E+00 : -1.000E-03 : 1.000E+09 ;
PLANT.R2.RATE_P(1) := 0.00000E+00 : -1.000E-01 : 1.000E+04 ;
PLANT.R2.RATE_P(2) := 0.00000E+00 : -1.000E-01 : 1.000E+04 ;
PLANT.R2.ENTHFLOW_P := 0.00000E+00 : -1.000E+09 : 1.000E+07 ;
PLANT.R1.RATE_P(1) := 1.25494E-01 : -1.000E-01 : 1.000E+04 ;
PLANT.R1.RATE_P(2) := 1.88241E-01 : -1.000E-01 : 1.000E+04 ;
PLANT.R1.ENTHFLOW_P := 1.09807E+02 : -1.000E+09 : 1.000E+07 ;
PLANT.MASSFRAC_C2(1) := 5.00000E-01 : -1.000E-01 : 1.000E+01 ;
PLANT.MASSFRAC_C2(2) := 5.00000E-01 : -1.000E-01 : 1.000E+01 ;
PLANT.DRIVINGFORCE_C1 := 3.61542E+02 : -1.000E+09 : 1.000E+09 ;
PLANT.DRIVINGFORCE_C2 := 2.35086E+00 : -1.000E+09 : 1.000E+09 ;
PLANT.DRIVINGFORCE_C3 := -5.76416E+01 : -1.000E+09 : 1.000E+09 ;
PLANT.DRIVINGFORCE_C4 := -5.99925E+01 : -1.000E+09 : 1.000E+09 ;
PLANT.RATE_C2(1) := 1.03162E-05 : -1.000E-01 : 1.000E+04 ;
PLANT.RATE_C2(2) := 1.03162E-05 : -1.000E-01 : 1.000E+04 ;
PLANT.DRIVINGFORCE_C5 := -5.99925E+01 : -1.000E+09 : 1.000E+09 ;
```

```
PLANT.ENTH_C2 := 7.07255E+02 : -1.000E+07 : 1.000E+04 ;
PLANT.REYNOLDSCONST1_C1 := 2.10000E+03 : -1.000E-03 : 1.000E+09 ;
PLANT.REYNOLDSCONST2_C1 := 2.20000E+03 : -1.000E-03 : 1.000E+09 ;
PLANT.DEN_C2 := 8.77650E-01 : -1.000E-01 : 1.000E+05 ;
PLANT.REYNOLDSCONST1_C3 := 2.10000E+03 : -1.000E-03 : 1.000E+09 ;
PLANT.ENTHFLOW_C2 := 1.45923E-02 : -1.000E+09 : 1.000E+07 ;
PLANT.REYNOLDSCONST1_C4 := 2.10000E+03 : -1.000E-03 : 1.000E+09 ;
PLANT.REYNOLDSCONST2_C3 := 2.20000E+03 : -1.000E-03 : 1.000E+09 ;
PLANT.REYNOLDSCONST1_C5 := 2.10000E+03 : -1.000E-03 : 1.000E+09 ;
PLANT.REYNOLDSCONST2_C4 := 2.20000E+03 : -1.000E-03 : 1.000E+09 ;
PLANT.VISCOSITY_C2 := 9.40000E-06 : -1.000E-01 : 1.000E+02 ;
PLANT.REYNOLDSCONST2_C5 := 2.20000E+03 : -1.000E-03 : 1.000E+09 ;
PLANT.REYNOLDSNO_C1 := 8.30707E+03 : -1.000E-03 : 1.000E+09 ;
PLANT.REYNOLDSNO_C2 := 2.79608E+00 : -1.000E-03 : 1.000E+09 ;
PLANT.REYNOLDSNO_C3 := 0.00000E+00 : -1.000E-03 : 1.000E+09 ;
PLANT.REYNOLDSNO_C4 := 0.00000E+00 : -1.000E-03 : 1.000E+09 ;
PLANT.REYNOLDSNO_C5 := 0.00000E+00 : -1.000E-03 : 1.000E+09 ;

{
  WITHIN Plant DO
    ReynoldsNo_C2  := 5000 ;
    ReynoldsNo_C3  := 5000 ;
    WITHIN FlashDrum DO
      PhaseType_V      := 1 ;
      PhaseType_B      := 1 ;
      PhaseType_L      := 2 ;
      PhaseType_B_L    := 3 ;
      PhaseType_Po1    := 3 ;
      PhaseType_Po2    := 1 ;
      Top_V            := 3 ;
      Bot_V            := 0.0535 ;
      Top_B_L          := 0.0535 ;
      Bot_B_L          := 0 ;
      Level_B_L        := 0.0535 ;
      Level_V          := 3 ;
      VapPress_L(1)    := 311.379 ;
      VapPress_L(2)    := 11.018E-3 ;
      EquilConst_B_L(1)  := 1.6 ;
      EquilConst_B_L(2)  := 0.2 ;
      Vol_FlashDrum    := 2.356 ;
      Press_FlashDrum  := 43 ;
      Temp_B           := 220 ;
      Temp_L           := 220 ;
    END
  END
}


SELECTOR
  WITHIN Plant DO
    FlowType_C1         := Turbulent ;
    FlowType_C2         := Laminar ;
    FlowType_C3         := Turbulent ;
    FlowType_C4         := Turbulent ;
    FlowType_C5         := Turbulent ;
    WITHIN FlashDrum1 DO
      Phase_B_L         := TwoPhase ;
    END
    WITHIN FlashDrum2 DO
      Phase_B_L         := TwoPhase ;
    END
  END

INITIAL
  WITHIN Plant DO
    WITHIN FlashDrum1 DO
      Temp_V           = 300 ;
      Mass_V           = 1 ;
```

```
    Mass_B_L          = 15 ;
    Ratio_B           = 0.001 ;
  END
  WITHIN FlashDrum2 DO
    Temp_V            = 280 ;
    Mass_V            = 1 ;
    Mass_B_L          = 15 ;
    Ratio_B           = 0.002 ;
  END
END


SOLUTIONPARAMETERS
  BLOCKDECOMPOSITION := OFF ;
  OUTPUTLEVEL := 1 ;

SCHEDULE
  SEQUENCE
    CONTINUE FOR 10
      RESET Plant.R1.Press_P := 1.013E2 + (TIME-10) ;
    END
    CONTINUE UNTIL Plant.R1.Press_P > 500
      RESET Plant.R1.Press_P := 600 - TIME ;
    END
    CONTINUE UNTIL Plant.R1.Press_P < 350
      RESET Plant.R1.Press_P := 1.013E2 ;
    END
    CONTINUE UNTIL Plant.FlashDrum1.Der_Mass_B_L < 1E-20
      RESET Plant.R1.Press_P := 0 ;       # stop feed
    END
    CONTINUE FOR 2E6
  END

END # Process test
```

D.3 Decanter : Settling Tank

```
DECLARE

TYPE
  Mass_rate       =  50     :  -1E-1     :  1E4     UNIT = "kg/sec"
  Temperature     =  100    :  -1E-1     :  1E4     UNIT = "K"
  Length          =  1      :  -1E-1     :  1E2     UNIT = "m"
  Enthalpy        =  0      :  -1E7      :  1E4     UNIT = "kJ/kg"
  Int_Energy      =  0      :  -1E9      :  1E4     UNIT = "kJ/kg"
  Volume          =  0.5    :  -10       :  1E1     UNIT = "m3"
  Pressure        =  43     :  -1E-1     :  1E4     UNIT = "kPa"
  Enthalpy_Flow   =  10     :  -1E9      :  1E7     UNIT = "kJ/sec"
  Mass            =  5      :  -1E-1     :  1E9     UNIT = "kg"
  Mole            =  0.1    :  -1E-1     :  1E2     UNIT = "kmole"
  Density         =  1000   :  -1E-1     :  1E5     UNIT = "kg/m3"
  Viscosity       =  5E-3   :  -1E-1     :  1E2     UNIT = "Pa.s"
  Velocity        =  1E-1   :  -1E-1     :  1E4     UNIT = "m/s"
  Fraction        =  0.5    :  -1E-1     :  10
  NoType          =  200    :  -1E9      :  1E9
  Positive        =  50     :  -1E-3     :  1E9


STREAM
 MassStream    IS Mass_Rate, Enthalpy_Flow, Fraction, Positive
 EnergyStream  IS Enthalpy_Flow

END


#################################
# BEGINNING of generated model #
#################################


MODEL  m_R1

PARAMETER
  NoComp                                  AS  INTEGER

VARIABLE
  Ratio_P                                 AS  Fraction
  Rate_P                                  AS  Array(NoComp) of Mass_Rate
  Den_P                                   AS  Density
  Press_P                                 AS  Pressure
  EnthFlow_P                              AS  Enthalpy_Flow
  PhaseType_P                             AS  Positive
  Viscosity_P                             AS  Viscosity
  MassFrac_P                              AS  Array(NoComp) of Fraction
  Enth_P                                  AS  Enthalpy

STREAM
  P : Rate_P, EnthFlow_P, Ratio_P, PhaseType_P  AS  MassStream

END  # end of MODEL m_R1


MODEL  m_R2

PARAMETER
  NoComp                                  AS  INTEGER

VARIABLE
  Ratio_P                                 AS  Fraction
  Rate_P                                  AS  Array(NoComp) of Mass_Rate
  EnthFlow_P                              AS  Enthalpy_Flow
  PhaseType_P                             AS  Positive
```

```
STREAM
  P : Rate_P, EnthFlow_P, Ratio_P, PhaseType_P   AS  MassStream

END  # end of MODEL m_R2


MODEL  m_R3

PARAMETER
  NoComp                                    AS  INTEGER

VARIABLE
  Ratio_P                                   AS  Fraction
  Rate_P                                    AS  Array(NoComp) of Mass_Rate
  EnthFlow_P                                AS  Enthalpy_Flow
  PhaseType_P                               AS  Positive

STREAM
  P : Rate_P, EnthFlow_P, Ratio_P, PhaseType_P   AS  MassStream

END  # end of MODEL m_R3

MODEL  m_decanter

PARAMETER
  liquid1_liquid2                           AS  INTEGER
  NoComp                                    AS  INTEGER
  liquid1                                   AS  INTEGER
  liquid2                                   AS  INTEGER
  Z_Po1                                     AS  REAL
  height                                    AS  REAL
  Const_B1_B                                AS  REAL
  Z_Pi                                      AS  REAL
  diameter                                  AS  REAL

VARIABLE
  Mass_B1                                   AS  Array(NoComp) of Mass
  Mass_W1                                   AS  Array(NoComp) of Mass
  Mass_B1_W1                                AS  Array(NoComp) of Mass
  Mass_B                                    AS  Array(NoComp) of Mass
  Mass_W                                    AS  Array(NoComp) of Mass
  Rate_W_Po1                                AS  Array(NoComp) of Mass_Rate
  Rate_W_Po2                                AS  Array(NoComp) of Mass_Rate
  Rate_B1_B                                 AS  Array(NoComp) of Mass_Rate
  Rate_Pi                                   AS  Array(NoComp) of Mass_Rate
  Rate_B1_W1_Po2                            AS  Array(NoComp) of Mass_Rate
  Rate_B1_W1_Po1                            AS  Array(NoComp) of Mass_Rate
  Rate_B_Po1                                AS  Array(NoComp) of Mass_Rate
  Rate_B_Po2                                AS  Array(NoComp) of Mass_Rate
  Rate_Po1                                  AS  Array(NoComp) of Mass_Rate
  Rate_Po2                                  AS  Array(NoComp) of Mass_Rate
  Rate_W1_W                                 AS  Array(NoComp) of Mass_Rate
  Den_W1                                    AS  Density
  Den_B1_W1                                 AS  Density
  Den_B                                     AS  Density
  Den_W                                     AS  Density
  Den_Po1                                   AS  Density
  Den_B1                                    AS  Density
  Den_Po2                                   AS  Density
  Viscosity_W                               AS  Viscosity
  Viscosity_B                               AS  Viscosity
  Viscosity_Po1                             AS  Viscosity
  Viscosity_W1                              AS  Viscosity
  Viscosity_B1                              AS  Viscosity
  Viscosity_B1_W1                           AS  Viscosity
  Press_B1_W1                               AS  Pressure
  Press_B1                                  AS  Pressure
```

```
Press_B                 AS  Pressure
Press_W                 AS  Pressure
Press_decanter          AS  Pressure
Press_W1                AS  Pressure
Temp_B                  AS  Temperature
Temp_W                  AS  Temperature
Temp_B1                 AS  Temperature
Temp_W1                 AS  Temperature
Vol_B1_W1               AS  Volume
Vol_B                   AS  Volume
Vol_W                   AS  Volume
Vol_decanter            AS  Volume
Vol_B1                  AS  Volume
Vol_W1                  AS  Volume
Vol_Empty               AS  Volume
Enth_Po1                AS  Enthalpy
Enth_Po2                AS  Enthalpy
Enth_B1                 AS  Enthalpy
Enth_B1_W1              AS  Enthalpy
Enth_W1                 AS  Enthalpy
Enth_B                  AS  Enthalpy
Enth_W                  AS  Enthalpy
IntEnergy_W             AS  Int_Energy
IntEnergy_B             AS  Int_Energy
IntEnergy_B1_W1         AS  Int_Energy
IntEnergy_W1            AS  Int_Energy
IntEnergy_B1            AS  Int_Energy
EnthFlow_Pi             AS  Enthalpy_Flow
EnthFlow_W1_W           AS  Enthalpy_Flow
EnthFlow_B_Po2          AS  Enthalpy_Flow
EnthFlow_B1_W1_Po2      AS  Enthalpy_Flow
EnthFlow_W_Po2          AS  Enthalpy_Flow
EnthFlow_Po2            AS  Enthalpy_Flow
EnthFlow_B1_B           AS  Enthalpy_Flow
EnthFlow_B_Po1          AS  Enthalpy_Flow
EnthFlow_B1_W1_Po1      AS  Enthalpy_Flow
EnthFlow_W_Po1          AS  Enthalpy_Flow
EnthFlow_Po1            AS  Enthalpy_Flow
MassFrac_W              AS  Array(NoComp) of Fraction
MassFrac_B              AS  Array(NoComp) of Fraction
MassFrac_Po1            AS  Array(NoComp) of Fraction
MassFrac_B1_W1          AS  Array(NoComp) of Fraction
MassFrac_W1             AS  Array(NoComp) of Fraction
MassFrac_B1             AS  Array(NoComp) of Fraction
Ratio_Pi                AS  Fraction
Ratio_B1                AS  Fraction
Ratio_B                 AS  Fraction
Ratio_W                 AS  Fraction
Ratio_Po2               AS  Fraction
Ratio_Po1               AS  Fraction
Z_Po2                   AS  Positive
Level_B                 AS  Positive
Level_W                 AS  Positive
Bot_B                   AS  Positive
Bot_W                   AS  Positive
C_W1_W                  AS  Positive
PhaseType_Pi            AS  Positive
PhaseType_W             AS  Positive
PhaseType_Po2           AS  Positive
PhaseType_B             AS  Positive
Level_Po2               AS  Positive
Level_Po1               AS  Positive
PhaseType_Po1           AS  Positive
Level_B1_W1             AS  Positive
PhaseType_B1_W1         AS  Positive
PhaseType_W1            AS  Positive
PhaseType_B1            AS  Positive
Bot_B1_W1               AS  Positive
```

```
  area                                    AS  Positive
  Top_B                                   AS  Positive
  Top_W                                   AS  Positive
  Top_B1_W1                               AS  Positive

STREAM
  Pi  : Rate_Pi,  EnthFlow_Pi,  Ratio_Pi,  PhaseType_Pi   AS  MassStream
  Po1 : Rate_Po1, EnthFlow_Po1, Ratio_Po1, PhaseType_Po1  AS  MassStream
  Po2 : Rate_Po2, EnthFlow_Po2, Ratio_Po2, PhaseType_Po2  AS  MassStream

SET
  Z_Po1                                   := 0.000000 ;
  height                                  := 5.000000 ;
  liquid1_liquid2                         := 3 ;
  liquid1                                 := 1 ;
  liquid2                                 := 2 ;
  Z_Pi                                    := 1.500000 ;
  diameter                                := 1.000000 ;

EQUATION

  # case invariant mass balance
  $Mass_B = - Rate_B_Po1 - Rate_B_Po2 + Rate_B1_B ;

  $Mass_B1 = Rate_Pi * Ratio_Pi - Rate_B1_W1_Po1 * Ratio_Po1 -
             Rate_B1_W1_Po2 * Ratio_Po2 - Rate_B1_B ;

  $Mass_W1 = Rate_Pi * (1 - Ratio_Pi) - Rate_B1_W1_Po1 *
             (1 - Ratio_Po1) - Rate_B1_W1_Po2 * (1 - Ratio_Po2) - Rate_W1_W ;

  $Mass_W = - Rate_W_Po1 - Rate_W_Po2 + Rate_W1_W ;

  # case invariant energy balance
  $IntEnergy_B * SIGMA(Mass_B) + IntEnergy_B * SIGMA($Mass_B) =
    - EnthFlow_B_Po1 - EnthFlow_B_Po2 + EnthFlow_B1_B ;

  $IntEnergy_B1 * SIGMA(Mass_B1) + IntEnergy_B1 * SIGMA($Mass_B1) =
    EnthFlow_Pi * Ratio_Pi - EnthFlow_B1_W1_Po1 * Ratio_Po1 -
    EnthFlow_B1_W1_Po2 * Ratio_Po2 - EnthFlow_B1_B ;

  $IntEnergy_W1 * SIGMA(Mass_W1) + IntEnergy_W1 * SIGMA($Mass_W1) =
    EnthFlow_Pi * (1 - Ratio_Pi) - EnthFlow_B1_W1_Po1 * (1 - Ratio_Po1) -
    EnthFlow_B1_W1_Po2 * (1 - Ratio_Po2) - EnthFlow_W1_W ;

  $IntEnergy_W * SIGMA(Mass_W) + IntEnergy_W * SIGMA($Mass_W) =
    - EnthFlow_W_Po1 - EnthFlow_W_Po2 + EnthFlow_W1_W ;


  # ratio of dispersed phase
  Ratio_B = 1 ;
  SIGMA(Mass_B1) = Ratio_B1 * SIGMA(Mass_B1_W1) ;
  Ratio_W = 1 ;

  # HeavyPhaseTransfer :
  Rate_W1_W = C_W1_W * Mass_W1 ;
  EnthFlow_W1_W = SIGMA(Rate_W1_W) * Enth_W1 ;

  # BubbleRise :
  Rate_B1_B = Const_B1_B * Mass_B1 ;
  EnthFlow_B1_B = SIGMA(Rate_B1_B) * Enth_B1 ;

  # mass = mass fraction * total mass
  Mass_B1_W1 = MassFrac_B1_W1 * SIGMA(Mass_B1_W1) ;
  Mass_W = MassFrac_W * SIGMA(Mass_W) ;
  Mass_W1 = MassFrac_W1 * SIGMA(Mass_W1) ;
  Mass_B = MassFrac_B * SIGMA(Mass_B) ;
  Mass_B1 = MassFrac_B1 * SIGMA(Mass_B1) ;
```

```
# total mass = density * volume
SIGMA(Mass_W) = Den_W * Vol_W ;
SIGMA(Mass_W1) = Den_W1 * Vol_W1 ;
SIGMA(Mass_B) = Den_B * Vol_B ;
SIGMA(Mass_B1) = Den_B1 * Vol_B1 ;

# phase type
PhaseType_B1_W1 = liquid1_liquid2 ;
PhaseType_W = liquid2 ;
PhaseType_W1 = liquid2 ;
PhaseType_B = liquid1 ;
PhaseType_B1 = liquid1 ;

# aggregated mass
Mass_B1_W1 = Mass_B1 + Mass_W1 ;

# aggregated phase density
Den_B1_W1 = Ratio_B1 * Den_B1 + (1 - Ratio_B1) * Den_W1 ;

# aggregated phase enthalpy
Enth_B1_W1 = Ratio_B1 * Enth_B1 + (1 - Ratio_B1) * Enth_W1 ;

# aggregated phase internal energy
IntEnergy_B1_W1 = Ratio_B1 * IntEnergy_B1 + (1 - Ratio_B1) * IntEnergy_W1 ;

# aggregated phase viscosity
Viscosity_B1_W1 = Ratio_B1 * Viscosity_B1 + (1 - Ratio_B1) * Viscosity_W1 ;

# volume relationship
Vol_B1_W1 = Vol_B1 + Vol_W1 ;
Vol_decanter = Vol_B + Vol_B1_W1 + Vol_W + Vol_Empty ;

# uniform pressure within vessel
Press_decanter = Press_B1_W1 ;
Press_decanter = Press_W ;
Press_decanter = Press_W1 ;
Press_decanter = Press_B ;
Press_decanter = Press_B1 ;

# phase bound : upper/low bound of phase volume = level
Top_W = Level_W ;
Bot_W = 0 ;
Top_B1_W1 = Level_B1_W1 ;
Bot_B1_W1 = Level_W ;
Top_B = Level_B ;
Bot_B = Level_B1_W1 ;

# phase volume : volume = area * (top - bottom)
area = (3.14/4) * diameter^2 ;
Vol_Empty = area * (height - Top_B) ;
Vol_W = area * (Top_W - Bot_W) ;
Vol_B1_W1 = area * (Top_B1_W1 - Bot_B1_W1) ;
Vol_B = area * (Top_B - Bot_B) ;


# discontinuity on output port, "Po1"
IF Z_Po1 >= Bot_B AND Z_Po1 < Top_B  THEN
  Level_B = Level_Po1 ;
  Viscosity_B = Viscosity_Po1 ;
  Enth_B = Enth_Po1 ;
  MassFrac_B = MassFrac_Po1 ;
  Den_B = Den_Po1 ;
  Rate_B_Po1 = Rate_Po1 ;
  EnthFlow_B_Po1 = EnthFlow_Po1 ;
  Ratio_B = Ratio_Po1 ;
  PhaseType_B = PhaseType_Po1 ;
```

```
    Rate_W_Po1 = 0 ;
    EnthFlow_W_Po1 = 0 ;
    Rate_B1_W1_Po1 = 0 ;
    EnthFlow_B1_W1_Po1 = 0 ;
  ELSE
    IF Z_Po1 >= Bot_B1_W1 AND Z_Po1 < Top_B1_W1  THEN
      Level_B1_W1 = Level_Po1 ;
      Viscosity_B1_W1 = Viscosity_Po1 ;
      Enth_B1_W1 = Enth_Po1 ;
      MassFrac_B1_W1 = MassFrac_Po1 ;
      Den_B1_W1 = Den_Po1 ;
      Rate_B1_W1_Po1 = Rate_Po1 ;
      EnthFlow_B1_W1_Po1 = EnthFlow_Po1 ;
      Ratio_B1 = Ratio_Po1 ;
      PhaseType_B1_W1 = PhaseType_Po1 ;
      Rate_W_Po1 = 0 ;
      EnthFlow_W_Po1 = 0 ;
      Rate_B_Po1 = 0 ;
      EnthFlow_B_Po1 = 0 ;
    ELSE
      IF Z_Po1 >= Bot_W AND Z_Po1 < Top_W  THEN
        Level_W = Level_Po1 ;
        Viscosity_W = Viscosity_Po1 ;
        Enth_W = Enth_Po1 ;
        MassFrac_W = MassFrac_Po1 ;
        Den_W = Den_Po1 ;
        Rate_W_Po1 = Rate_Po1 ;
        EnthFlow_W_Po1 = EnthFlow_Po1 ;
        Ratio_W = Ratio_Po1 ;
        PhaseType_W = PhaseType_Po1 ;
        Rate_B1_W1_Po1 = 0 ;
        EnthFlow_B1_W1_Po1 = 0 ;
        Rate_B_Po1 = 0 ;
        EnthFlow_B_Po1 = 0 ;
      ELSE
        Level_Po1 = 0 ;
        Viscosity_Po1 = 0 ;
        Enth_Po1 = 0 ;
        MassFrac_Po1 = 0 ;
        Den_Po1 = 0 ;
        Rate_W_Po1 = 0 ;
        EnthFlow_W_Po1 = 0 ;
        Rate_B1_W1_Po1 = 0 ;
        EnthFlow_B1_W1_Po1 = 0 ;
        Rate_B_Po1 = 0 ;
        EnthFlow_B_Po1 = 0 ;
        Ratio_Po1 = 0 ;
        PhaseType_Po1 = 0 ;
      END
    END
  END
# end of discontinuity on output port, "Po1"


# discontinuity on output port, "Po2"
IF Z_Po2 >= Bot_B AND Z_Po2 < Top_B  THEN
  Den_B = Den_Po2 ;
  Level_B = Level_Po2 ;
  Enth_B = Enth_Po2 ;
  Rate_B_Po2 = Rate_Po2 ;
  EnthFlow_B_Po2 = EnthFlow_Po2 ;
  Ratio_B = Ratio_Po2 ;
  PhaseType_B = PhaseType_Po2 ;
  Rate_W_Po2 = 0 ;
  EnthFlow_W_Po2 = 0 ;
  Rate_B1_W1_Po2 = 0 ;
  EnthFlow_B1_W1_Po2 = 0 ;
ELSE
```

```
IF Z_Po2 >= Bot_B1_W1 AND Z_Po2 < Top_B1_W1  THEN
  Den_B1_W1 = Den_Po2 ;
  Level_B1_W1 = Level_Po2 ;
  Enth_B1_W1 = Enth_Po2 ;
  Rate_B1_W1_Po2 = Rate_Po2 ;
  EnthFlow_B1_W1_Po2 = EnthFlow_Po2 ;
  Ratio_B1 = Ratio_Po2 ;
  PhaseType_B1_W1 = PhaseType_Po2 ;
  Rate_W_Po2 = 0 ;
  EnthFlow_W_Po2 = 0 ;
  Rate_B_Po2 = 0 ;
  EnthFlow_B_Po2 = 0 ;
ELSE
  IF Z_Po2 >= Bot_W AND Z_Po2 < Top_W  THEN
    Den_W = Den_Po2 ;
    Level_W = Level_Po2 ;
    Enth_W = Enth_Po2 ;
    Rate_W_Po2 = Rate_Po2 ;
    EnthFlow_W_Po2 = EnthFlow_Po2 ;
    Ratio_W = Ratio_Po2 ;
    PhaseType_W = PhaseType_Po2 ;
    Rate_B1_W1_Po2 = 0 ;
    EnthFlow_B1_W1_Po2 = 0 ;
    Rate_B_Po2 = 0 ;
    EnthFlow_B_Po2 = 0 ;
  ELSE
    Den_Po2 = 0 ;
    Level_Po2 = 0 ;
    Enth_Po2 = 0 ;
    Rate_W_Po2 = 0 ;
    EnthFlow_W_Po2 = 0 ;
    Rate_B1_W1_Po2 = 0 ;
    EnthFlow_B1_W1_Po2 = 0 ;
    Rate_B_Po2 = 0 ;
    EnthFlow_B_Po2 = 0 ;
    Ratio_Po2 = 0 ;
    PhaseType_Po2 = 0 ;
  END
 END
 END
 # end of discontinuity on output port, "Po2"

 # port position
 Z_Po2 = height ;

 # ============================================== #
 # specification of C_W1_W in HeavyPhaseTransfer #
 # ============================================== #
 C_W1_W =  0.5E-1;

END  # end of MODEL m_decanter


MODEL  Flowsheet

PARAMETER
  NoComp                    AS  INTEGER
  Const2_C1                 AS  REAL
  Const3_C1                 AS  REAL
  Const_C1                  AS  REAL
  Const_C2                  AS  REAL
  Const_C3                  AS  REAL

VARIABLE
  ReynoldsConst1_C1         AS  Positive
  ReynoldsConst2_C1         AS  Positive
  ReynoldsNo_C1             AS  Positive
  DrivingForce_C1           AS  NoType
```

```
UNIT
  R1                                    AS  m_R1
  R2                                    AS  m_R2
  R3                                    AS  m_R3
  decanter                              AS  m_decanter

SELECTOR
  FlowType_C1    AS  (Turbulent, Laminar)

EQUATION

                #==================================#
                # stream connections through ports #
                #==================================#
  R1.P                  IS      decanter.Pi ;
  decanter.Po1          IS      R2.P ;
  decanter.Po2          IS      R3.P ;


                #==================================#
                # transfer law of each connection #
                #==================================#
  # "IrreversiblePressureDrivenFlow" in connection, "C1"
  DrivingForce_C1 = R1.Press_P - decanter.Press_decanter ;
  (4/3.14) * SIGMA(R1.Rate_P) = ReynoldsNo_C1 * Const_C1 * R1.Viscosity_P ;
  ReynoldsConst1_C1 = 2100 ;
  ReynoldsConst2_C1 = 4000 ;
  R1.EnthFlow_P = SIGMA(R1.Rate_P) * R1.Enth_P ;
  IF DrivingForce_C1 > 0  THEN
    CASE FlowType_C1 OF
      WHEN Turbulent :
        R1.Rate_P = Const2_C1 * R1.Den_P * R1.MassFrac_P *
                    SQRT(DrivingForce_C1) ;
        SWITCH TO Laminar IF ReynoldsNo_C1 < ReynoldsConst1_C1 ;
      WHEN Laminar :
        R1.Rate_P = Const3_C1 * R1.Den_P * R1.MassFrac_P *
                    DrivingForce_C1 ;
        SWITCH TO Turbulent IF ReynoldsNo_C1 > ReynoldsConst2_C1 ;
    END
  ELSE
    R1.Rate_P = 0 ;
  END

  # "StaticPressureDrivenFlow" in connection "C2"
  IF (decanter.Z_Po1 >= decanter.Bot_B AND decanter.Z_Po1 <
     decanter.Top_B) OR (decanter.Z_Po1 >= decanter.Bot_B1_W1 AND
     decanter.Z_Po1 < decanter.Top_B1_W1) OR (decanter.Z_Po1 >=
     decanter.Bot_W AND decanter.Z_Po1 < decanter.Top_W)  THEN
    decanter.Rate_Po1 = Const_C2 * decanter.Den_Po1 *
                        SQRT(2 * 9.8 * ABS(decanter.Level_Po1)) ;
  ELSE
    decanter.Rate_Po1 = 0 ;
  END
  decanter.EnthFlow_Po1 = SIGMA(decanter.Rate_Po1) * decanter.Enth_Po1;

  # "WeirOverFlow" in connection, "C3"
  IF (decanter.Z_Po2 >= decanter.Bot_B AND decanter.Z_Po2 <
     decanter.Top_B) OR (decanter.Z_Po2 >= decanter.Bot_B1_W1 AND
     decanter.Z_Po2 < decanter.Top_B1_W1) OR (decanter.Z_Po2 >=
     decanter.Bot_W AND decanter.Z_Po2 < decanter.Top_W)  THEN
    decanter.Rate_Po2 = Const_C3 * decanter.Den_Po2 *
                        (ABS(decanter.Level_Po2 - decanter.Z_Po2))^1.5 ;
  ELSE
    decanter.Rate_Po2 = 0 ;
  END
  decanter.EnthFlow_Po2 = SIGMA(decanter.Rate_Po2) * decanter.Enth_Po2 ;
```

```
END  # end of MODEL Flowsheet

############################
# END of generated models #
############################



PROCESS test

UNIT
  Plant AS Flowsheet

SET
  WITHIN Plant Do
    NoComp          := 1 ;
    Const_C1        := 0.05 ;

    # ============================== #
    # cross-sectional area of pipe, C2 #
    # ============================== #
    Const_C2        := (3.14 / 4) * 0.05^2 ;

    # ========================================== #
    # modified Francis formula in C3, "WeirOverFlow" #
    # weir length = 3.14*diameter,                   #
    # wall correction factor = 1.04                  #
    # ========================================== #
    Const_C3        := (2*3.14*0.5) / ((0.68175*1.04)^1.5) ;

    Const2_C1       := 9E-4 ;
    Const3_C1       := 5E-5 ;
    WITHIN R1 DO
      NoComp        := 1 ;
    END
    WITHIN R2 DO
      NoComp        := 1 ;
    END
    WITHIN R3 DO
      NoComp        := 1 ;
    END
    WITHIN decanter DO
      NoComp        := 1 ;
      Const_B1_B    := 0.3E-1 ;
    END
  END


ASSIGN
  WITHIN Plant DO
    WITHIN R1 DO
      Press_P       := 2*1.013E2 ;  # [kPa]
      Den_P         := 902.5 ;      # [kg/m3]
      Enth_P        := 0 ;
      MassFrac_P    := 1.0 ;
      Ratio_P       := 0.3 ;
      PhaseType_P   := 3 ;
      Viscosity_P   := 0.00178 ;
  END
  WITHIN decanter DO
    Enth_B          := 0 ;
    Enth_B1         := 0 ;
    Enth_W1         := 0 ;
    Enth_W          := 0 ;
    Temp_B          := 298 ;
    Temp_B1         := 298 ;
    Temp_W1         := 298 ;
    Temp_W          := 298 ;
                    ,
```

```
    Den_B           := 805 ;
    Den_B1          := 805 ;
    Den_W1          := 1000 ;
    Den_W           := 1000 ;
    Viscosity_B     := 2.65E-3 ;
    Viscosity_B1    := 2.65E-3 ;
    Viscosity_W1    := 9.0E-4 ;
    Viscosity_W     := 9.0E-4 ;
    Press_decanter  := 1.013E2 ;
  END
END
  END

PRESET
    PLANT.R2.RATIO_P := 0.00000E+00 : -1.000E-01 : 1.000E+01 ;
    PLANT.R3.RATIO_P := 0.00000E+00 : -1.000E-01 : 1.000E+01 ;
    PLANT.DECANTER.ENTHFLOW_W_PO1 := 0.00000E+00 : -1.000E+09 : 1.000E+07 ;
    PLANT.DECANTER.VOL_B1 := 0.00000E+00 : -1.000E+01 : 1.000E+01 ;
    PLANT.DECANTER.ENTHFLOW_W_PO2 := 0.00000E+00 : -1.000E+09 : 1.000E+07 ;
    PLANT.DECANTER.PHASETYPE_B1 := 1.00000E+00 : -1.000E-03 : 1.000E+09 ;
    PLANT.DECANTER.VOL_B := 1.24224E-02 : -1.000E+01 : 1.000E+01 ;
    PLANT.DECANTER.MASSFRAC_B1(1) := 5.00000E-01 : -1.000E-01 : 1.000E+01 ;
    PLANT.DECANTER.PHASETYPE_B := 1.00000E+00 : -1.000E-03 : 1.000E+09 ;
    PLANT.DECANTER.RATE_PO1(1) := 0.00000E+00 : -1.000E-01 : 1.000E+04 ;
    PLANT.DECANTER.INTENERGY_B1 := 0.00000E+00 : -1.000E+09 : 1.000E+04 ;
    PLANT.DECANTER.MASSFRAC_B(1) := 1.00000E+00 : -1.000E-01 : 1.000E+01 ;
    PLANT.DECANTER.RATE_PI(1) := 4.57116E+03 : -1.000E-01 : 1.000E+04 ;
    PLANT.DECANTER.ENTHFLOW_PI := 0.00000E+00 : -1.000E+09 : 1.000E+07 ;
    PLANT.DECANTER.RATIO_PI := 5.00000E-01 : -1.000E-01 : 1.000E+01 ;
    PLANT.DECANTER.PHASETYPE_PI := 3.00000E+00 : -1.000E-03 : 1.000E+09 ;
    PLANT.DECANTER.RATE_PO2(1) := 0.00000E+00 : -1.000E-01 : 1.000E+04 ;
    PLANT.DECANTER.INTENERGY_B := 0.00000E+00 : -1.000E+09 : 1.000E+04 ;
    PLANT.DECANTER.TOP_B := 4.13024E-02 : -1.000E-03 : 1.000E+09 ;
    PLANT.DECANTER.DEN_B1_W1 := 1.00000E+03 : -1.000E-01 : 1.000E+05 ;
    PLANT.DECANTER.ENTH_PO1 := 0.00000E+00 : -1.000E+07 : 1.000E+04 ;
    PLANT.DECANTER.PRESS_B1 := 1.01300E+02 : -1.000E-01 : 1.000E+04 ;
    PLANT.DECANTER.ENTH_PO2 := 0.00000E+00 : -1.000E+07 : 1.000E+04 ;
    PLANT.DECANTER.VISCOSITY_B1_W1 := 9.00000E-04 : -1.000E-01 : 1.000E+02 ;
    PLANT.DECANTER.PRESS_B := 1.01300E+02 : -1.000E-01 : 1.000E+04 ;
    PLANT.DECANTER.LEVEL_W := 1.27389E-02 : -1.000E-03 : 1.000E+09 ;
    PLANT.DECANTER.ENTHFLOW_PO1 := 0.00000E+00 : -1.000E+09 : 1.000E+07 ;
    PLANT.DECANTER.RATIO_PO1 := 0.00000E+00 : -1.000E-01 : 1.000E+01 ;
    PLANT.DECANTER.PHASETYPE_PO1 := 0.00000E+00 : -1.000E-03 : 1.000E+09 ;
    PLANT.DECANTER.ENTHFLOW_B1_W1_PO1 := 0.00000E+00 : -1.000E+09 : 1.000E+07 ;
    PLANT.DECANTER.ENTHFLOW_PO2 := 0.00000E+00 : -1.000E+09 : 1.000E+07 ;
    PLANT.DECANTER.RATIO_PO2 := 0.00000E+00 : -1.000E-01 : 1.000E+01 ;
    PLANT.DECANTER.PHASETYPE_PO2 := 0.00000E+00 : -1.000E-03 : 1.000E+09 ;
    PLANT.DECANTER.ENTHFLOW_B1_W1_PO2 := 0.00000E+00 : -1.000E+09 : 1.000E+07 ;
    PLANT.DECANTER.BOT_W := 0.00000E+00 : -1.000E-03 : 1.000E+09 ;
    PLANT.DECANTER.RATE_W1_W(1) := 0.00000E+00 : -1.000E-01 : 1.000E+04 ;
    PLANT.DECANTER.RATE_B_PO1(1) := 0.00000E+00 : -1.000E-01 : 1.000E+04 ;
    PLANT.DECANTER.RATE_B_PO2(1) := 0.00000E+00 : -1.000E-01 : 1.000E+04 ;
    PLANT.DECANTER.RATIO_W := 1.00000E+00 : -1.000E-01 : 1.000E+01 ;
    PLANT.DECANTER.LEVEL_B1_W1 := 2.54777E-02 : -1.000E-03 : 1.000E+09 ;
    PLANT.DECANTER.MASS_B1(1) := 0.00000E+00 : -1.000E-01 : 1.000E+09 ;
    PLANT.DECANTER.DEN_PO1 := 0.00000E+00 : -1.000E-01 : 1.000E+05 ;
    PLANT.DECANTER.VOL_DECANTER := 3.92500E+00 : -1.000E+01 : 1.000E+01 ;
    PLANT.DECANTER.MASS_B(1) := 1.00000E+01 : -1.000E-01 : 1.000E+09 ;
    PLANT.DECANTER.VISCOSITY_PO1 := 0.00000E+00 : -1.000E-01 : 1.000E+02 ;
    PLANT.DECANTER.BOT_B1_W1 := 1.27389E-02 : -1.000E-03 : 1.000E+09 ;
    PLANT.DECANTER.RATE_B1_B(1) := 0.00000E+00 : -1.000E-01 : 1.000E+04 ;
    PLANT.DECANTER.VOL_W1 := 1.00000E-02 : -1.000E+01 : 1.000E+01 ;
    PLANT.DECANTER.PHASETYPE_W1 := 2.00000E+00 : -1.000E-03 : 1.000E+09 ;
    PLANT.DECANTER.VOL_W := 1.00000E-02 : -1.000E+01 : 1.000E+01 ;
    PLANT.DECANTER.MASSFRAC_W1(1) := 1.00000E+00 : -1.000E-01 : 1.000E+01 ;
    PLANT.DECANTER.PHASETYPE_W := 2.00000E+00 : -1.000E-03 : 1.000E+09 ;
    PLANT.DECANTER.INTENERGY_W1 := 0.00000E+00 : -1.000E+09 : 1.000E+04 ;
    PLANT.DECANTER.MASSFRAC_W(1) := 1.00000E+00 : -1.000E-01 : 1.000E+01 ;
    PLANT.DECANTER.INTENERGY_W := 0.00000E+00 : -1.000E+09 : 1.000E+04 ;
```

```
PLANT.DECANTER.TOP_W := 1.27389E-02 : -1.000E-03 : 1.000E+09 ;
PLANT.DECANTER.PRESS_W1 := 1.01300E+02 : -1.000E-01 : 1.000E+04 ;
PLANT.DECANTER.ENTHFLOW_W1_W := 0.00000E+00 : -1.000E+09 : 1.000E+07 ;
PLANT.DECANTER.PRESS_W := 1.01300E+02 : -1.000E-01 : 1.000E+04 ;
PLANT.DECANTER.ENTHFLOW_B_PO1 := 0.00000E+00 : -1.000E+09 : 1.000E+07 ;
PLANT.DECANTER.ENTHFLOW_B_PO2 := 0.00000E+00 : -1.000E+09 : 1.000E+07 ;
PLANT.DECANTER.C_W1_W := 0.00000E+00 : -1.000E-03 : 1.000E+09 ;
PLANT.DECANTER.LEVEL_PO2 := 0.00000E+00 : -1.000E-03 : 1.000E+09 ;
PLANT.DECANTER.AREA := 7.85000E-01 : -1.000E-03 : 1.000E+09 ;
PLANT.DECANTER.VOL_B1_W1 := 1.00000E-02 : -1.000E+01 : 1.000E+01 ;
PLANT.DECANTER.PHASETYPE_B1_W1 := 3.00000E+00 : -1.000E-03 : 1.000E+09 ;
PLANT.DECANTER.MASSFRAC_B1_W1(1) := 1.00000E+00 : -1.000E-01 : 1.000E+01 ;
PLANT.DECANTER.INTENERGY_B1_W1 := 0.00000E+00 : -1.000E+09 : 1.000E+04 ;
PLANT.DECANTER.TOP_B1_W1 := 2.54777E-02 : -1.000E-03 : 1.000E+09 ;
PLANT.DECANTER.RATE_W_PO1(1) := 0.00000E+00 : -1.000E-01 : 1.000E+04 ;
PLANT.DECANTER.VOL_EMPTY := 3.89258E+00 : -1.000E+01 : 1.000E+01 ;
PLANT.DECANTER.RATE_W_PO2(1) := 0.00000E+00 : -1.000E-01 : 1.000E+04 ;
PLANT.DECANTER.ENTHFLOW_B1_B := 0.00000E+00 : -1.000E+09 : 1.000E+07 ;
PLANT.DECANTER.LEVEL_B := 4.13024E-02 : -1.000E-03 : 1.000E+09 ;
PLANT.DECANTER.PRESS_B1_W1 := 1.01300E+02 : -1.000E-01 : 1.000E+04 ;
PLANT.DECANTER.MASS_W1(1) := 1.00000E+01 : -1.000E-01 : 1.000E+09 ;
PLANT.DECANTER.MASS_W(1) := 1.00000E+01 : -1.000E-01 : 1.000E+09 ;
PLANT.DECANTER.BOT_B := 2.54777E-02 : -1.000E-03 : 1.000E+09 ;
PLANT.DECANTER.RATIO_B1 := 0.00000E+00 : -1.000E-01 : 1.000E+01 ;
PLANT.DECANTER.RATE_B1_W1_PO1(1) := 0.00000E+00 : -1.000E-01 : 1.000E+04 ;
PLANT.DECANTER.ENTH_B1_W1 := 0.00000E+00 : -1.000E+07 : 1.000E+04 ;
PLANT.DECANTER.RATIO_B := 1.00000E+00 : -1.000E-01 : 1.000E+01 ;
PLANT.DECANTER.Z_PO2 := 5.00000E+00 : -1.000E-03 : 1.000E+09 ;
PLANT.DECANTER.RATE_B1_W1_PO2(1) := 0.00000E+00 : -1.000E-01 : 1.000E+04 ;
PLANT.DECANTER.MASSFRAC_PO1(1) := 0.00000E+00 : -1.000E-01 : 1.000E+01 ;
PLANT.DECANTER.MASS_B1_W1(1) := 1.00000E+01 : -1.000E-01 : 1.000E+09 ;
PLANT.R3.PHASETYPE_P := 0.00000E+00 : -1.000E-03 : 1.000E+09 ;
  PLANT.R3.RATE_P(1) := 0.00000E+00 : -1.000E-01 : 1.000E+04 ;
  PLANT.R3.ENTHFLOW_P := 0.00000E+00 : -1.000E+09 : 1.000E+07 ;
  PLANT.R2.PHASETYPE_P := 0.00000E+00 : -1.000E-03 : 1.000E+09 ;
  PLANT.R2.RATE_P(1) := 0.00000E+00 : -1.000E-01 : 1.000E+04 ;
  PLANT.R2.ENTHFLOW_P := 0.00000E+00 : -1.000E+09 : 1.000E+07 ;
  PLANT.R1.RATE_P(1) := 4.57116E+03 : -1.000E-01 : 1.000E+04 ;
  PLANT.R1.ENTHFLOW_P := 0.00000E+00 : -1.000E+09 : 1.000E+07 ;
  PLANT.DRIVINGFORCE_C1 := 1.01300E+02 : -1.000E+09 : 1.000E+09 ;
  PLANT.REYNOLDSCONST2_C1 := 4.00000E+03 : -1.000E-03 : 1.000E+09 ;
  PLANT.REYNOLDSNO_C1 := 6.54285E+05 : -1.000E-03 : 1.000E+09 ;


SELECTOR
  WITHIN Plant DO
    FlowType_C1 := Turbulent ;
  END


INITIAL
  WITHIN Plant DO
    WITHIN decanter DO
      Mass_B       = 0 ;
      Mass_W       = 0 ;
      Mass_B1_W1   = 0 ;
      Mass_W1      = 0 ;
      IntEnergy_B  = 0 ;
      IntEnergy_B1 = 0 ;
      IntEnergy_W1 = 0 ;
      IntEnergy_W  = 0 ;
    END
  END

SOLUTIONPARAMETERS
  BLOCKDECOMPOSITION := ON ;
  OUTPUTLEVEL        := 1   ;
```

```
SCHEDULE
  SEQUENCE
    CONTINUE UNTIL TIME > 1500
  END

END # Process test
```