REPRESENTING PHYSICAL AND DESIGN KNOWLEDGE IN INNOVATIVE DESIGN

BY

NIKITAS MARINOS SGOUROS

Diploma, National Technical University of Athens, 1988
M.S., University of Edinburgh, 1990

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate School of
Northwestern University, 1993

Evanston, Illinois

ABSTRACT

Representing Physical and Design Knowledge in Innovative Design

Nikitas Marinos Sgouros

The ability to design is one of the hallmarks of intelligent behavior, since it allows an agent to shape its environment according to its needs. Therefore developing computational models of design should be one of the primary goals for a discipline such as Artificial Intelligence that aims to create intelligent artifacts.

This thesis describes `DIAS`, a computational framework for innovative engineering design. This framework provides ways for representing the physical and design knowledge in a domain, along with methods for allowing these different types of knowledge to interact during the design process. This model has been instantiated in `OUZO`, a program that designs separation systems in chemical engineering.

To my family

ACKNOWLEDGEMENTS

I would like to thank:

- My advisor, Kenneth Forbus, for all the support and direction during this research.
- The members of my committee, Larry Birnbaum, Gregg Collins and Chris Riesbeck, for serving on my final committee and providing helpful comments.
- The NASA Johnson Space Center and the NASA Langley Research Center for supporting this research.
- All of the members of the Qualitative Reasoning Group at Northwestern and especially Dennis Decoste, John Everett, Keith Law, Chris Lopez and Yusuf Pisan for all the discussions and the technical support through all these years.
- All of my friends during this time and especially Joyce Montigny, Costas Sakellariou, Mike Sang and Chris Wisdo.
- My mother, Pinelopi, and my sister, Marina, for their love and support throughout the years.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction

1.1 The Design Problem

Design is the process by which a set of functional specifications and a set of available components are used to create a description of an artifact that satisfies these specifications. The design description has to be detailed enough to allow the manufacturing, fabrication or construction of the desired artifact [25].

Engineering is defined as [66]:

> "The application of science and mathematics by which the properties of matter and the sources of energy in nature are made useful to people in structures, machines, products, systems and processes."

Engineering design differs from other kinds of design in that it applies engineering methods to generate descriptions of useful devices. It consists of two major phases [47]. The *analysis* phase elaborates on the design specifications and determines the behavior of the current design. The *synthesis* phase decides on the individual components and indicates their possible interconnections.

Engineering design problems belong to three categories, depending on the type of search space they explore [58]:

- *Routine* design problems are those in which the components of design are known and there is a specific method for assembling these components. Examples of routine design problems are the design of simple electronic components (adders, inverters, etc), or the design of simple car parts like clutches.
- *Innovative* design problems are those in which the components of design are known, but there is no straightforward method for assembling these components in a way that satisfies the design specifications. Examples include the design of a new car model, or a new CD player.
- *Creative* design problems are those in which not even the components of design are completely known. A typical example is the invention of the wheel.

In the rest of the thesis the word design refers to engineering design.

1.2 What is this thesis all about?

This thesis describes `DIAS`[1], a computational framework that enables the transformation of informal accounts of engineering design into precise computational

[1] Design Integrating Analysis and Synthesis.

Engineering Costs to prepare alternatives (1977)			
Type of estimate	**Less than $1 mil Plant**	**$1-$5 mil Plant**	**$5-$50 mil Plant**
Study ($ thousand)	5-15	12-30	20-40
Preliminary ($ thousand)	15-35	30-60	50-90
Definitive ($ thousand)	25-60	60-120	100-230

Table 1.1: Engineering costs to prepare process designs.

models for performing the design task. Informal accounts of design typically include references to the physical principles for analyzing the behavior of the designed artifact, along with general descriptions of heuristics and the methods by which these are applied. `DIAS` includes a set of representations and techniques by which these different types of knowledge are integrated in a computational model of design.

This research concentrates on innovative design problems. The ideas we present have been implemented in a program called `OUZO`[2] that supports the synthesis of separation systems, an important innovative design problem in chemical engineering [57]. The program can generate designs that are similar in complexity to examples from the chemical engineering research literature.

There is more than one level of engineering design and `DIAS` focuses on the early part of the process, commonly refered to as *conceptual design*. Conceptual design applies a set of approximate but efficient evaluation criteria in minimizing the number of candidate designs. It creates rough drafts for a small set of promising designs that are analyzed in more detail in later stages of the process. For example, in chemical engineering a rough draft is a flowsheet that indicates the interconnections between the different process units and provides cost estimates for each one of them. These estimates are approximate screening calculations aimed at reducing the number of design choices rather than providing accurate cost data [13]. Table 1.1[3] gives an idea of the relative costs associated with producing estimates in chemical engineering design. Study estimates have a probable accuracy up to $\pm 25\%$ of the final cost. Preliminary estimates have a $\pm 12\%$ accu-

[2] `OUZO` is not an acronym for anything. It merely denotes the state of euphoria of the author while developing this program.

[3] The data are taken from [48].

racy. Finally, definitive estimates should lie within $\pm 6\%$ of the actual cost. OUZO is intended to support the creation of study estimates.

The importance of conceptual design is indicated by the fact that only 1% of the alternatives examined at this level constitute viable designs [13]. Therefore, inefficiencies at the conceptual level have serious consequences for the rest of design, since they result in erroneous decisions very early on in the process. These decisions propagate to more detailed levels of design and are very hard to recover from. While current engineering design programs concentrate mostly on the detailed levels of design, our research aims to automate the conceptual level of design.

Figure 1.1 describes a sample input and output for OUZO. The input specifications require the design of a separation system for recovering the components of a 6-component mixture in four specific sets of products. One of the possible designs is shown below the problem specification. It consists of five columns. For each column the program computes an estimate for its installation cost which is shown at the bottom of the figure. There are 1,344 possible designs for this problem. The program examines only four of them before it reaches a final solution. The results of the program are consistent with the ones presented in the chemical engineering research literature [43].

1.3 Background and Motivation

1.3.1 Design from Physical Principles

The role of computers in engineering or scientific problem-solving tasks has been mainly to provide numerical solutions to sets of equations. Thus, computational techniques typically are used only in the later stages of the problem-solving activity as model solvers, and play insignificant roles during the earlier and very important part of model formulation.

Recently, a new approach to design known as *design from physical principles* has emerged as a promising alternative to either semi-automatic approaches like computer-aided design systems or fully automatic but very domain-specific programs like expert systems. Physical principles systems can be loosely defined as [34]:

> "Programs that integrate representations of skills from mathematics, physics, and engineering in design."

Problem Specification

Feed			Desired Products		Conditions
Component	**Component Name**	**Mole Fraction**	**Product**	**Component**	Temperature =
1	Propane	.0147	1	1	= 53.89^{o}C
2	n-Butane	.5029	2	2	Pressure =
3	Butene-1	.1475	3	3, 4, 5	= 5.62 kg/cm^2
4	Trans-Butene-2	.1563	4	6	Total Flow Rate =
5	Cis-Butene-2	.1196			= 303.04 kg mol/h
6	n-Pentane	.0590			

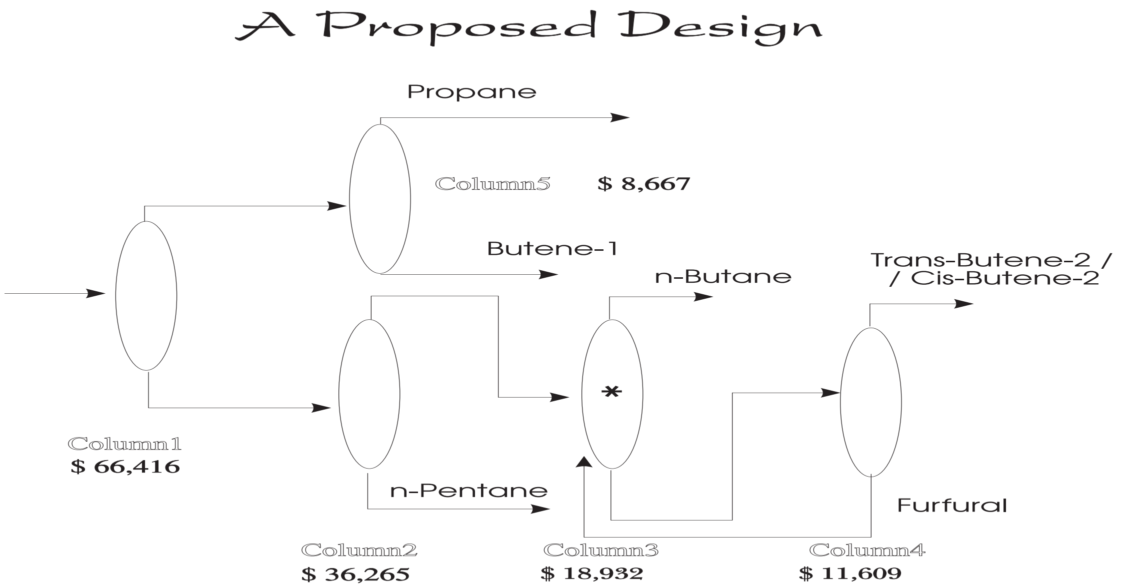


Figure 1.1: Problem specification and possible design for the n-Butylene purification problem. Columns with an asterisk (*) correspond to extractive distillation units. The rest are ordinary distillation columns.

The hope behind these programs is that the availability and use of all these different types of knowledge will provide them with the efficiency and the generality that are necessary for automating innovative or even creative design.

There are two main problems that need to be addressed for programs designing from physical principles:

- *The Representation Problem* consists of developing adequate representations for the knowledge used in design.
- *The Integration Problem* consists of developing ways of integrating all these different kinds of knowledge in design.

`DIAS` addresses both problems in the context of innovative design. In particular, as a solution to the representation problem we partition the knowledge used into two categories:

- *Physical knowledge* describes the principles that engineers use in modeling physical phenomena.
- *Design knowledge* describes the methods and rules that are specific to design. Most of the design knowledge is heuristic in nature.

We deal with the integration problem by developing a computational framework that integrates qualitative, numerical and heuristic knowledge (Figure 1.2). More specifically, `DIAS` accepts as inputs the specifications for the design problem, along with a set of components describing the physical and the design knowledge for the domain of interest. Physical knowledge generates and analyzes design alternatives. Design knowledge prunes the number of design choices. The output in this approach is a list of numerical and structural descriptions for each proposed design.

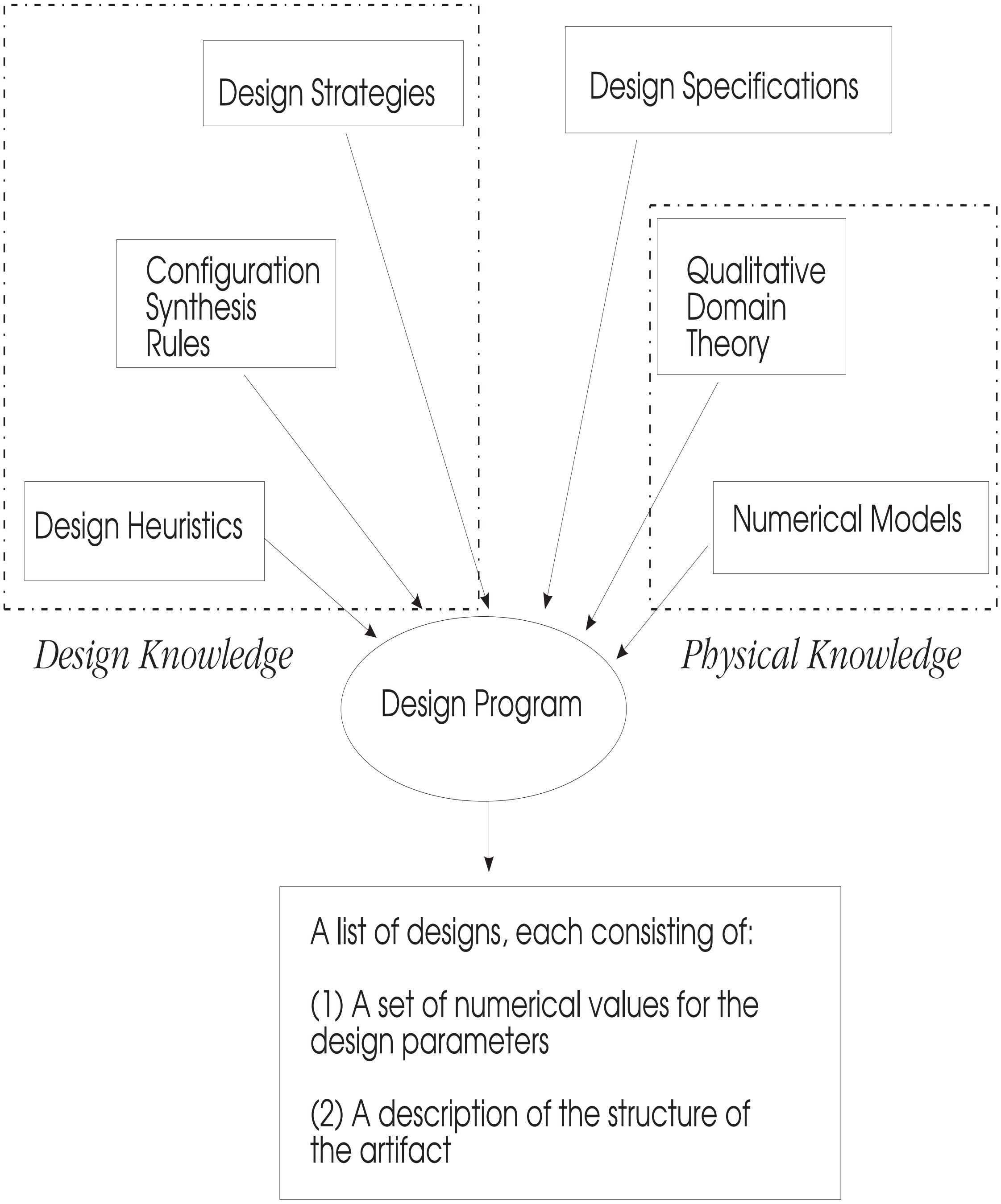
1.3.2 Qualitative Reasoning

The purpose of the qualitative reasoning enterprise is [71]:

> "... to develop computational theories of the core skills underlying engineers, scientists and just plain folk's ability to hypothesize, test, predict, create, optimize, diagnose and debug physical mechanisms."

Until recently, most of the applications of qualitative reasoning were centered around producing qualitative simulations of physical systems. This was a consequence of the fact that the field had developed early on quite sophisticated

Figure 1.2: The architecture of DIAS.



techniques for simulating physical systems. Design programs were not popular in these early days, due to the lack of detailed domain theories for sufficiently complex physical systems. The problem was exacerbated by:

- The absence of modeling strategies that could support the creation of qualitative descriptions for complex physical systems.

- The large computational complexity involved with qualitative simulations. This allowed large domain theories to be tested only under certain operating conditions (e.g. steady-state assumptions).

- The lack of integration between the qualitative models and the quantitative knowledge used by engineers in their domains.

The recent development of *self-explanatory simulator compilers* like SIMGEN [21] extends the ways in which qualitative methods can be used in the design process. The main idea behind these systems is the automatic creation of simulators that integrate qualitative and numerical models during simulation. These programs can then be used either as ordinary numerical simulators tracking the behavior of a given model over time, or as powerful explanation tools that generate qualitative explanations for the behavior of a system at any point in time. SIMGEN-like systems possess two features that are of potentially great interest to automated design systems:

- They are capable of automatically producing simulation code for the behavior of a system. Consequently, the efficiency of the analysis phase of design is improved.

- The kind of qualitative analysis they use is computationally more efficient than traditional qualitative simulation.

More specifically, the qualitative analysis in SIMGEN determines the conditions under which each qualitative model fragment [15] is active and guides the selection of appropriate numerical models. `DIAS` uses a subset of this analysis to support the construction of numerical models for possible designs.

This research demonstrates how qualitative models and reasoning techniques can be integrated with representations for numerical models and design knowledge. In particular, it describes how a qualitative modeling language like QP Theory [18] and compositional modeling techniques [15] can support the generation and analysis of alternatives during conceptual design.

1.4 Summary of Results

There are two main results for this research:

1. A computational model of innovative engineering design (DIAS) that allows informal ideas expressed in terms of heuristic design strategies and knowledge about the physical principles that describe physical systems to be transformed into precise computational models for performing the design task.

2. A computer program (OUZO) that applies this model to the design of separation systems in chemical engineering.

1.5 Reader's Guide

The thesis is organized around seven chapters. Here is what they contain:

1. **Introduction.** This chapter provides an overview of the design process and the types of design problems. It explains the background and motivation for this work and summarizes the major results of this research.

2. **A Computational Account of Design.** This chapter presents the main ideas of the design approach in this thesis. It shows how we model engineering design as a sequence of design cycles in DIAS. Furthermore, the decomposition between the physical and design knowledge, the design interpreter and the way in which the design process is controlled in our model are described and discussed. Finally, related work in AI and engineering is presented and compared with DIAS.

3. **The Design of Separation Systems in Chemical Engineering.** This chapter serves as an introduction to separation systems for readers who are unfamiliar with chemical engineering. It offers a brief overview of separation processes and in particular equilibrium separation processes. In addition, it provides a description of the state of the art in process synthesis with a particular emphasis on the design of separation systems.. Finally, it presents and discusses a set of filtering strategies we developed for describing the major heuristics in separation system design, along with the computational costs of the design methods implemented in OUZO.

4. **Representing Physical Knowledge in OUZO.** This chapter describes the domain theory that models separation systems. Both qualitative and numerical models are presented. The justifications behind our representation decisions are presented and analyzed.

5. **Representing Design Knowledge in OUZO.** This chapter presents and analyzes the representations for heuristics, design strategies and configuration synthesis rules in OUZO. In addition, it describes the implementation of an interpreter for the design knowledge. Finally, it explains the algorithm for controlling the design process in this system.

6. **Examples**. One example for binary distillation design and four examples for multicomponent separation sequences are presented in detail.

7. **Conclusions**. In this chapter we give a summary of the whole approach. We review the contributions of this research. Finally, we make suggestions for future work in this area.

There are four appendices:

- **Appendix A** reviews Qualitative Process Theory.
- **Appendix B** describes the numerical models that calculate the relative volatilities and the cost estimates for the design alternatives.
- **Appendix C** describes the configuration synthesis rules that create the actual design description.
- **Appendix D** gives an example of how we develop the representations for heuristic rules in OUZO.

Chapter 2

A Computational Account of Innovative Design

2.1 The Main Ideas

This thesis describes DIAS, a computational framework for innovative engineering design that enables the transformation of informal accounts of design into computational models for performing the design task. DIAS supports a common distinction drawn in these informal descriptions between experience-learned heuristics and science-based models [40]. In our approach *physical* knowledge consists of science-based models and represents the understanding that engineers have of physical systems. During the analysis phase, it generates descriptions of the behavior for possible designs. In addition, it computes design alternatives during the synthesis phase. *Design* knowledge refers to heuristic knowledge that describes how design is performed. It is used during the synthesis phase to prune the number of design choices. Furthermore, it suggests physical conditions or particular parameter values that increase the efficiency of the analysis phase.

DIAS uses qualitative and numerical models for capturing the physical knowledge in design. Heuristics, strategies and configuration synthesis rules are used for representing the design knowledge in a domain. We present an interpreter that transforms the design knowledge representations into appropriate rules and implements the actions suggested by the heuristics using a set of primitives.

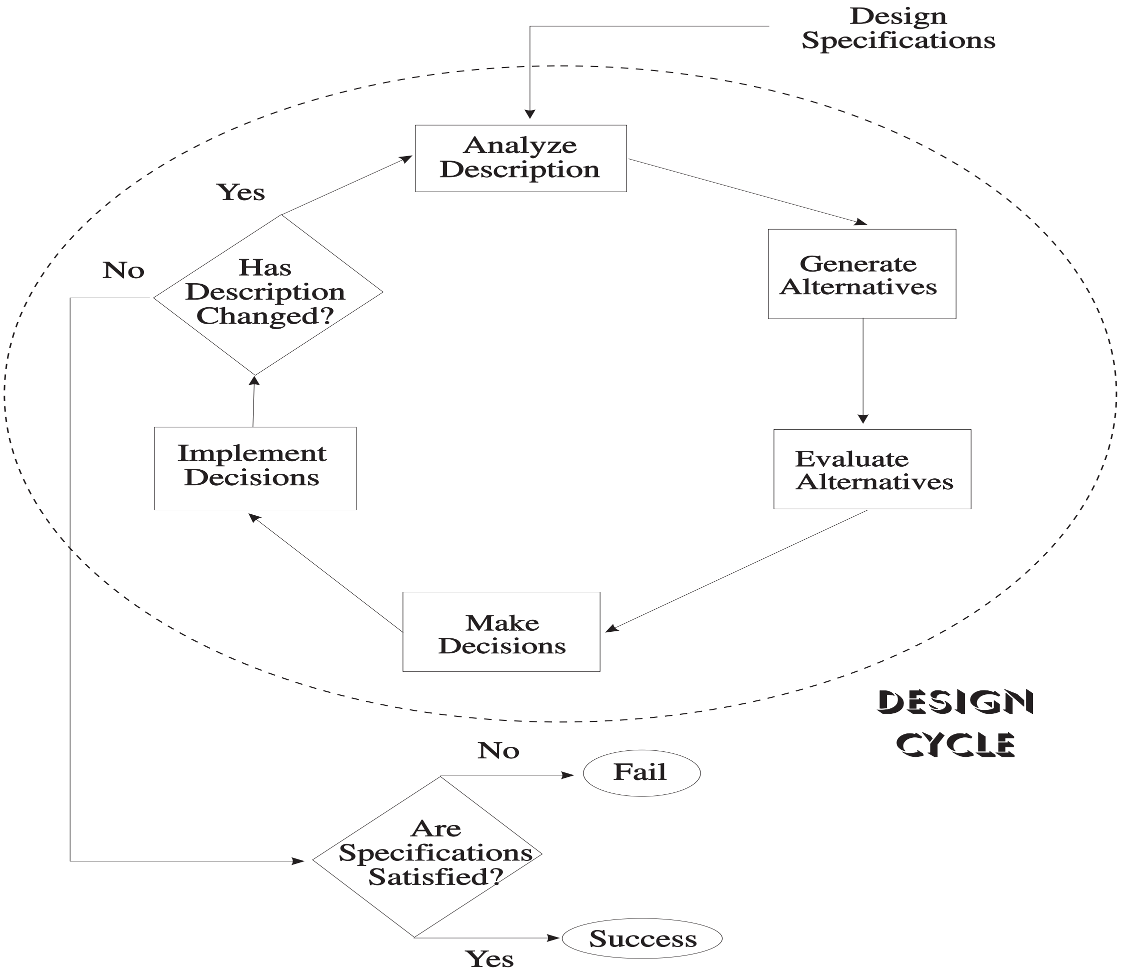
DIAS models engineering design as the generation, evaluation and implementation of design alternatives in a sequence of design cycles. This thesis describes a novel controller algorithm that orchestrates the use of the different types of knowledge in the design cycle and is independent of specific design methodologies (i.e. evolutionary or heuristic methods). The rest of this chapter outlines the elements of this approach in more detail.

Although the need for integrating qualitative and numerical models with heuristic representations has been stressed repeatedly in the engineering literature (e.g. [60], [40]), computational models that use this approach to fully support the generation and evaluation of designs have not previously existed. This work demonstrates that these systems are indeed possible.

2.2 Modeling Engineering Design

DIAS describes engineering design as the generation, evaluation and implementation of *design alternatives.* Design alternatives (or choices) are mutually exclusive decisions that trigger changes to the structure of the artifact. For example, some of the design alternatives for multicomponent mixtures in OUZO consist of mutually exclusive choices on the separation method and the pair of possible keys

Figure 2.1: The engineering design process.



for the current column. Decisions on these parameters determine the structure (e.g. diameter, height, number of stages, etc) for the current separation unit[1]. The structure of the artifact during each design cycle is captured by the *design description.*

Design alternatives are processed in a sequence of *design cycles* (Fig. 2.1). In the beginning a design cycle accepts as input and analyzes the design specifications using the physical knowledge. The purpose of this analysis is to compute relevant features of the design description such as the behavior it entails and/or to elaborate on the design specifications and determine sets of parameters that are important for the current design problem.

In the rest of the cycle, DIAS generates and evaluates design alternatives using the physical and the design knowledge, and proposes and implements changes to the current description using the design knowledge. At the end of the cycle if the design description has changed, a new cycle starts with the current description and the design specifications as inputs, otherwise the design cycle terminates. In the later case, if the description does not satisfy the design specifications the process exits with failure, otherwise it terminates succesfully.

A *design method* optimizes the generation and evaluation of design alternatives. For example, the case-based design method optimizes the generation and evaluation of alternatives through the re-use and adaptation of previous cases. Evolutionary design is another design method similar to the case-based methodology, that optimizes the generation of alternatives using a standard set of heuristics and their evaluation using a series of evolutionary rules.

A *design strategy* is a domain-specific instance of a design method. For example, one of the strategies in OUZO, the Nath & Motard strategy, is an instance of an evolutionary design method. As section 2.4.3 explains in more detail, a design strategy represents an optimal application of a design method on a specific type of problem.

2.3 Physical Knowledge

2.3.1 Representations

DIAS uses qualitative and numerical representations for capturing physical knowledge in design. Qualitative representations provide an ontological framework for

[1]An example of a decision that is not considered a design alternative in our approach is the choice of notation for describing components in design diagrams. This decision does not have an impact on the structure of the artifact.

describing physical phenomena and represent the causal dependencies between its parameters and the modeling assumptions used in the description of a physical system. Numerical representations consist of systems of numerical relations (equations or inequalities) between the parameters of a system. This approach combines the rich modeling language of qualitative formalisms with the accuracy offered by numerical models.

2.3.2 Reasoning Methods

DIAS uses physical knowledge to generate design alternatives and determine their behavior during the design cycle. It does this using a set of physical principles to generate a set of possible designs and construct models for their behavior based on numerical values for the system variables, descriptions for the structure of possible designs, sets of modeling assumptions and the results of the evaluation process performed by the design heuristics. Qualitative representations support this process using compositional modeling techniques [15] that are sensitive to changes in all of these parameters. Three reasoning methods support the generation and analysis of design alternatives in the physical knowledge component: qualitative analysis, numerical model construction and numerical equation solving.

2.3.2.1 Qualitative Analysis

Qualitative analysis generates qualitative models for the design description and the alternatives during each design cycle. It accepts as input parameters the design description, the qualitative domain theory and a set of modeling assumptions and computes the minimal sets of conditions under which a model fragment is active (Fig. 2.2). The results of this step are used to activate qualitative model fragments that are consistent with the input parameters.

This analysis corresponds to the first step of the qualitative analysis in SIMGEN [21]. OUZO demonstrates that this type of qualitative analysis is general enough to support typical conceptual design tasks like the design of separation systems that deal with either steady-state or macroscopic models for physical systems.

2.3.2.2 Numerical Model Construction

Numerical model construction creates numerical models for the design description and the alternatives during each design cycle. During this process the numerical relations in the physical knowledge are combined with the results of the qualitative

Figure 2.2: Qualitative analysis flowchart.

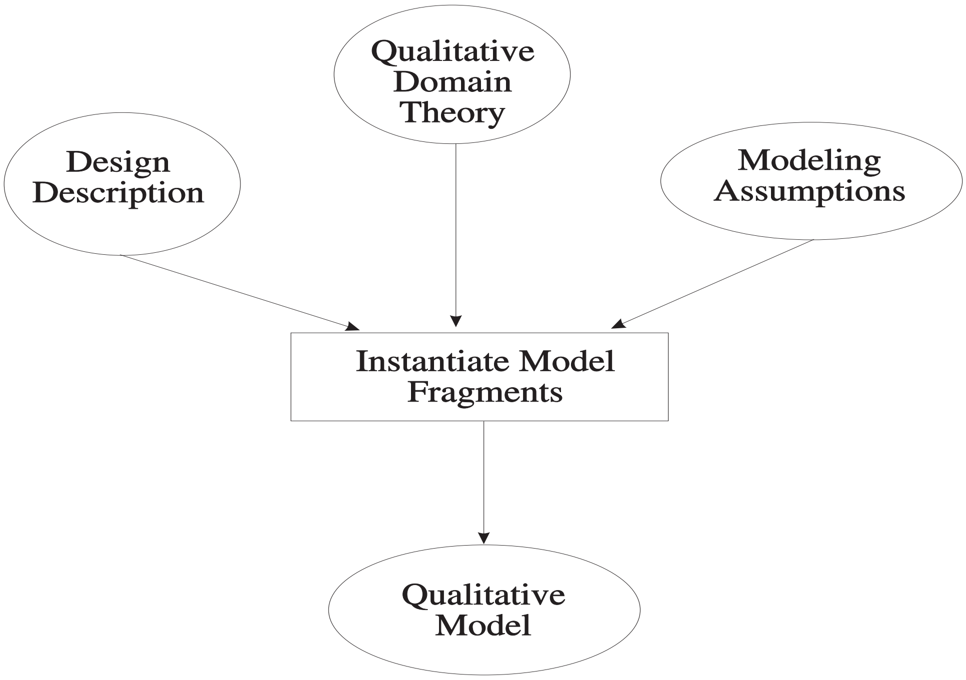
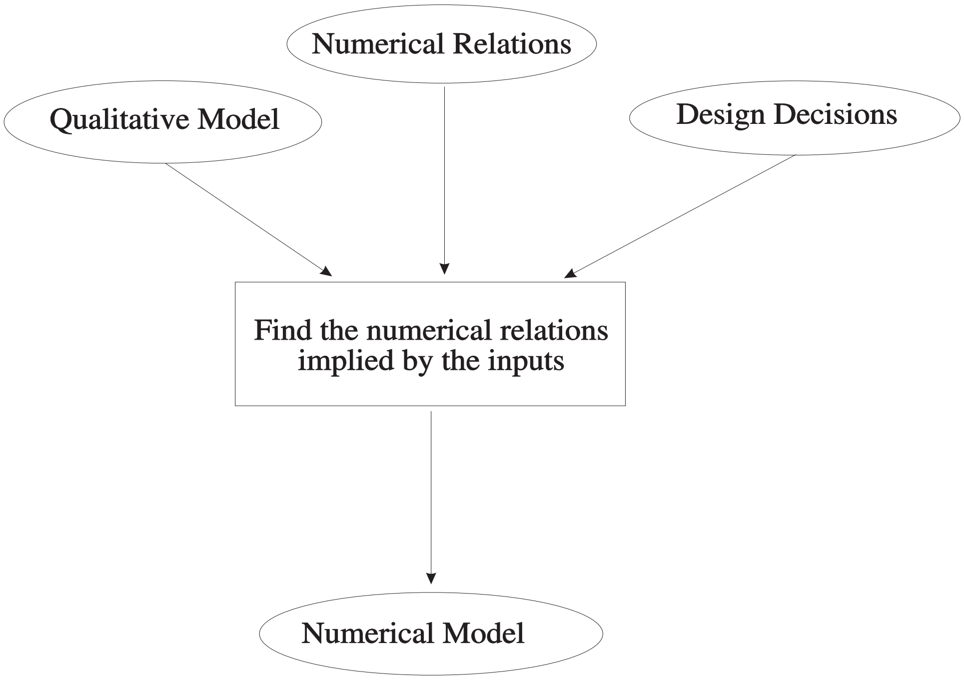


Figure 2.3: Numerical Model Construction flowchart.



analysis and the results of the evaluation of the current design alternatives by the heuristics[2] to activate a set of numerical models (Fig. 2.3). This method ensures that these numerical models are consistent with all the analysis parameters and the actions suggested by the heuristics. Furthermore, it allows the design knowledge to decide on the type of analysis for each alternative. Typically in conceptual design very approximate types of analysis are used in the beginning to screen the alternatives. As the design evolves, the heuristics pick a set of promising candidates which are combined with the results of the qualitative analysis to activate more detailed numerical models during the design cycle.

In OUZO, this task is accomplished using the results of the qualitative analysis and the current focus environment. The later is an ATMS focus environment [19] that consists of the major design decisions taken by the decision primitives in Table 2.3 (see section 2.4.2 for a discussion of these primitives). In OUZO these decisions consist of the separation schemes for each column, along with predicates that denote which columns are being examined by the design system at the current stage. For example, whenever the heuristic analysis decides on a particular separation for a column with the assert-in-design primitive, the current focus environment is updated to reflect this decision. As a result more detailed numerical models that are now implied by the updated focus environment are activated, resulting in a deeper level of analysis for the chosen separation.

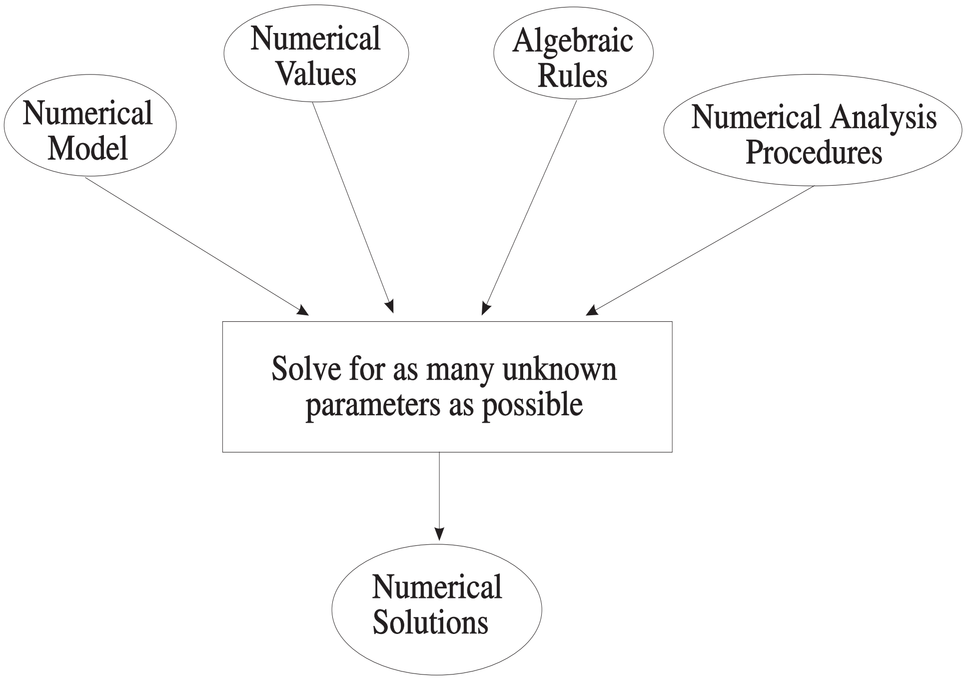
2.3.2.3 Numerical Equation Solving

Numerical equation solving applies algebraic techniques and numerical analysis procedures to solve numerical models. It accepts as inputs the results of the numerical model construction process, a set of numerical values for some of the design parameters, a set of rules for algebraic equation solving and numerical analysis procedures and tries to find numerical solutions for as many design parameters as possible (Fig. 2.4).

In OUZO, this task is accomplished by indexing the set of equations according to the quantities they involve and then replacing these quantities in every equation they occur with their numerical values as soon as the later are computed. Equation solving continues as long as there are equations with only one unknown quantity in them.

[2] Technically speaking the result of the heuristic analysis is the creation of an ATMS focus environment that contains the design decisions that were taken by the heuristics (see chapter 5 and [19]).

Figure 2.4: Numerical Equation Solving flowchart.



2.4 Design Knowledge

Three types of representations express the design knowledge in DIAS:

1. *Heuristics*, i.e., rules of thumb.
2. *Strategies*. Plans for optimizing the application of the heuristic rules.
3. *Configuration Synthesis Rules*. Rules for monitoring the design and producing the actual artifact descriptions.

A design interpreter transforms these representations into appropriate rules and implements the actions suggested by the heuristics via a set of primitives.

The rest of this section describes these representations and the interpreter commands in more detail.

2.4.1 Heuristics

Heuristics in conceptual design prune the search space. They apply criteria that typically capture the interaction between the physical behavior of artifacts and the economics associated with producing or operating an artifact. This means that heuristic knowledge is grounded in the physical knowledge for the domain. DIAS supports this grounding by providing qualitative and numerical models in which the terms referenced by the heuristics are described. As a result, the representation and use of heuristic knowledge are significantly facilitated.

For example, one of the heuristics for designing separation systems in chemical engineering suggests that the least tight separation should be more preferable than any other alternative for the current separation unit. One of the actual heuristic rules (Fig. 2.5) for representing this heuristic in OUZO translates the least tight criterion into differences between the relative volatilities of the design alternatives. Because all of the terms in this rule are defined in the physical knowledge component, the designer knows what their semantics are and at what level of detail the system is described by them. For example, the designer is able to correctly determine how the quantity *Alpha-LK-HK* (the relative volatility between two mixture components) is defined and what are the reference conditions *(:reference)* under which it is measured, since this information is part of the physical knowledge component. The designer is also able to trace the methods by which this quantity is calculated by accessing the equations that are active in the numerical model.

Tables 2.1 (taken from [45]) and 2.2 (taken from [43]) contain some of the heuristics used in OUZO for the design of separation systems. These tables are shown again and analyzed in detail in chapters 3 and 5.

#	**Heuristic Rule**
1	Remove components one by one as overhead products
2	Save the most difficult separation for last
3	Favor 50-50 splits
4	Sequence with the minumum total vapor flow
5	Make high recovery fractions last
6	Separate the most plentiful components first
7	Choose the cheapest as the next separator
8	Remove the thermally unstable and corrosive material early.
9	Disregard separations with very small relative volatility between the keys.
10	Perform least tight separation first
11	Favor the smallest production set
12	Avoid separations using a mass separating agent (MSA)
13	Remove a MSA from one of the products in another, subsequent separation process
14	A separation method using a MSA cannot be used to isolate another MSA
15	Favor distillation
16	Separate first the components which might undergo undesirable reactions
17	Set split fractions of the key components to prespecified values
18	Avoid extreme operating conditions
19	Favor ambient operating pressure

Table 2.1: Major heuristics for separation system design.

```
(defHeuristic Difficulty-of-Separation-1
 :Class Perform-Least-Tight-Separation-First
 :Conditions ((Possible (Separation ?method (?l-k-1 ?h-k-1) ?column)) :Var ?f1
              (Value-of (A (Alpha-LK-HK ?l-k-1 ?h-k-1 ?column :reference)) ?a-lk-hk-1 ?eq-1)
              (Possible (Separation ?method (?l-k-2 ?h-k-2) ?column)) :Var ?f2
              (Value-of (A (Alpha-LK-HK ?l-k-2 ?h-k-2 ?column :reference)) ?a-lk-hk-2 ?eq-2)
              :Test (and (> ?a-lk-hk-1 ?a-lk-hk-2)
                         (> (- ?a-lk-hk-1 ?a-lk-hk-2) *alpha-difference*)))
 :Action ((prefer ?f1 :Over ?f2)))

 IF there is a separation alternative (a) for the current column
    AND the relative volatility between the keys in (a) in reference conditions is known
    AND there is another separation alternative (b)
    AND the relative volatility between the keys in (b) in reference conditions is known
    AND there is a significant difference (> *alpha-difference*)
        between the relative volatilities of (a) and (b)
 THEN prefer the separation alternative with the largest value for the relative volatility.
```

Figure 2.5: Perform least tight separation first. The actual heuristic in OUZO and its interpretation. The terms preceded with a '?' correspond to variables.

#	**Evolutionary Rule**
1	Challenge Heuristic 11
2	Examine the neighboring structures if separations of similar difficulty take place in them.
3	Challenge Heuristic 15
4	Examine neighbors to decide if the MSA removal should be delayed
5	Challenge Heuristic 10 if in the current design an easy separation is followed by a very difficult one.

Table 2.2: Some evolutionary heuristics used in OUZO.

2.4.2 Representing Heuristics

There are three kinds of knowledge involved in the representation of the conditions of heuristic rules; modeling and design assumptions about the problem, structural features of the design and numerical values for parameters in the design. All these are grounded in the physical knowledge component. In addition to these kinds of knowledge, languages for heuristics and design knowledge in general must contain primitives that implement the actions suggested by these rules. In `DIAS` these primitives are commands to the interpreter describing how to update the set of design alternatives. We use thirteen primitives for capturing design actions based on the following classification of heuristic rules:

1. **Rejection** rules prune the number of design alternatives by eliminating solutions that do not meet certain criteria. Examples include heuristics 1[3], 8, 9, 12 and 14 in Table 2.1.

2. **Ordering rules** establish preferences between various design choices. The rule in Figure 2.5 provides an example of an ordering heuristic. Other examples include heuristics 2, 3, 4, 5, 6, 10, 11, 15 and 16 in Table 2.1.

3. **Decision rules** select a design alternative. Heuristics 7 and 13 in Table 2.1 provide an example.

4. **Analysis rules** propose numerical values for some of the parameters of the system in order to facilitate the analysis of proposed designs. Examples include heuristics 17, 18 and 19 in Table 2.1.

5. **Evolutionary rules** challenge the design decisions made by other heuristics. Table 2.2 provides an example for some of these rules. The number of the heuristics that are challenged in this table refers to the row in which they appear in Table 2.1.

In general, some heuristics can be described either as rejection or ordering rules. For example heuristic 1 can be modeled either as a rejection or an ordering rule by the builder of the design knowledge base. When represented as a rejection rule, it will eliminate from the analysis all the design choices that do not result

[3] It is not always clear from the natural language description of a heuristic that it corresponds to a rejection rule. For example, the description for heuristic 1 does not contain any verbs like 'avoid' or 'disregard' that make the rejection of certain design choices clear. However, by focusing the attention of the designer on separations that result in single overhead products, it essentially rejects all the other alternatives. A similar situation is true for heuristic 8.

Table 2.3: Primitives for design actions.

Heuristic Type	Primitive	Interpretation
Rejection	reject	Reject an alternative for the current design cycle
Ordering	prefer	Establishes an order of preference between two alternatives
	consult-user	Ask the user to select between two alternatives
Decision	assert-in-design	Assert that an alternative holds in the rest of design
	assume-in-cycle	Assume that an alternative holds for the current design cycle
	assume-in-design	Assume that an alternative holds in the rest of design
	cover-specifications	Indicate that all the design specifications are satisfied
Analysis	propose-value	Assign a value to a parameter
	exists	Check whether an item is part of the current design description
Evolutionary	invalidate-decision	Do not consider an alternative in the rest of design
	examined-alternative	Check to see if an alternative has been already examined
	store-design	Stores the current design description
	pop-design	Reinstates the most recent design description

in single overhead products. When represented as an ordering rule, it will not eliminate these choices, but it will indicate that these alternatives are less favorable. Analyzing the conditions under which each modeling choice makes more sense (e.g. number of alternatives discarded, overall evaluation of the alternatives that are eliminated and the ones that remain) can be a promising extension to this work. The classification for the examples presented above corresponds to the representations of these rules in `OUZO`.

Most of the heuristics in engineering design can be classified as belonging to one of these heuristic types. For example, all of the heuristics for mechanical engineering design in [73] are rejection rules, while the heuristics for civil engineering design in [29] are a mix of rejection, ordering and decision rules.

Table 2.3 contains the actual primitives that capture the actions of these five kinds of heuristic rules. Chapter 5 presents the actual algorithms for each one of these primitives in `OUZO`.

2.4.3 Strategies

Some of the actions proposed by the design heuristics may be in conflict with each other. For example, in Table 2.1 heuristics 1 and 10 can be contradictory in cases where the least tight separation is not the one that results in a single distillate product. Therefore it is necessary to create consistent subsets of heuristics or sequence their application in ways that resolve possible conflicts during design. For example, in separation system design we can create sets of heuristics in which rules 1 and 10 do not coexist, or we can sequence their application so that rule 1 is always applied before rule 10. The later means that we always pick the least tight among all the separations that result in single distillate products. Design strategies provide ways for organizing the application of heuristic knowledge along these lines.

More specifically, the design strategies in `DIAS` are plans for sequencing the execution of heuristic rules in ways that were found by the chemical engineering research community to be capable of producing optimal designs. This organization of design knowledge is general enough to support different design methods ranging from pure heuristic to evolutionary strategies in `OUZO`.

Figure 2.6 contains part of the description of a strategy for the design of separation systems in [43].

"Evolutionary rule 1 questions the validity of the heuristic rule and is applied before any other evolutionary rule to resolve the question of the product set definition. Evolutionary rules 2, 3, 4 and 5 are treated equally, but of course cannot be applied at the same time. Therefore, starting from the feed stream forward, evolutionary rule 2 is applied next. If any modification is suggested by this rule, it is adapted in the starting structure and a new structure is produced. The new structure or the starting one, whichever is superior is evolved further by applying rule 2 to the portion of the structure not checked by rule 2 in the earlier application. Evolutionary rule 3 is applied starting from the feed stream forward, after no further structural modifications are suggested by rule 2."

Figure 2.6: An example of a design strategy description.

```
IF the steady-state design features for the column are active
   AND the column has a partial condenser and a partial reboiler
   AND the particular stages for these units have been defined
   AND the value for the number of stages in the column is N
   AND the current design description does not correspond to a column with N stages
THEN create a new design description for a column with a partial
       condenser, a partial reboiler and N stages.
```

Figure 2.7: Example of a configuration synthesis rule in OUZO.

2.4.4 Configuration Synthesis Rules

A significant part of the design knowledge in a domain relates to procedures for creating the actual design descriptions. These descriptions range from flowsheets in chemical engineering, to CAD drawings in mechanical engineering, to circuit diagrams in electrical engineering and so on. There are two major characteristics for the knowledge that supports the creation of these descriptions:

1. It is very procedural in nature.
2. It is largely domain-specific, as each engineering domain has developed its own conventions for creating design documents.

In addition to these procedures, part of the design knowledge in a domain monitors the state of the design, determining, for example, the conditions under which the design process succeeds or fails.

DIAS uses a separate set of rules, the configuration synthesis rules, that capture the knowledge associated with producing design descriptions and monitoring the state of the design process. This organization allows the builder of the design knowledge base to separate more general forms of knowledge like heuristics that can apply to more than one classes of systems from more specific forms of knowledge like configuration synthesis rules that deal with specific classes of devices. Figure 2.7 contains an example of the english interpretation of such a rule in `OUZO`.

2.5 Controlling Design

`DIAS` controls the design process by an algorithm that provides a general way for orchestrating the use of various types of knowledge in design. It consists of three steps (Fig. 2.8):

2.5.1 Qualitatively analyze the current design description.

This step performs a subset of the qualitative analysis used in SIMGEN [21]. It uses the current design description to instantiate a set of qualitative model fragments that are consistent with it. Furthermore, it determines the set of conditions under which each model fragment becomes active. Because qualitative analysis is used in determining the behavior of the design description and in generating design alternatives, this step is part of the Analyze-Description and Generate-Alternatives steps of the design cycle in Figure 2.1.

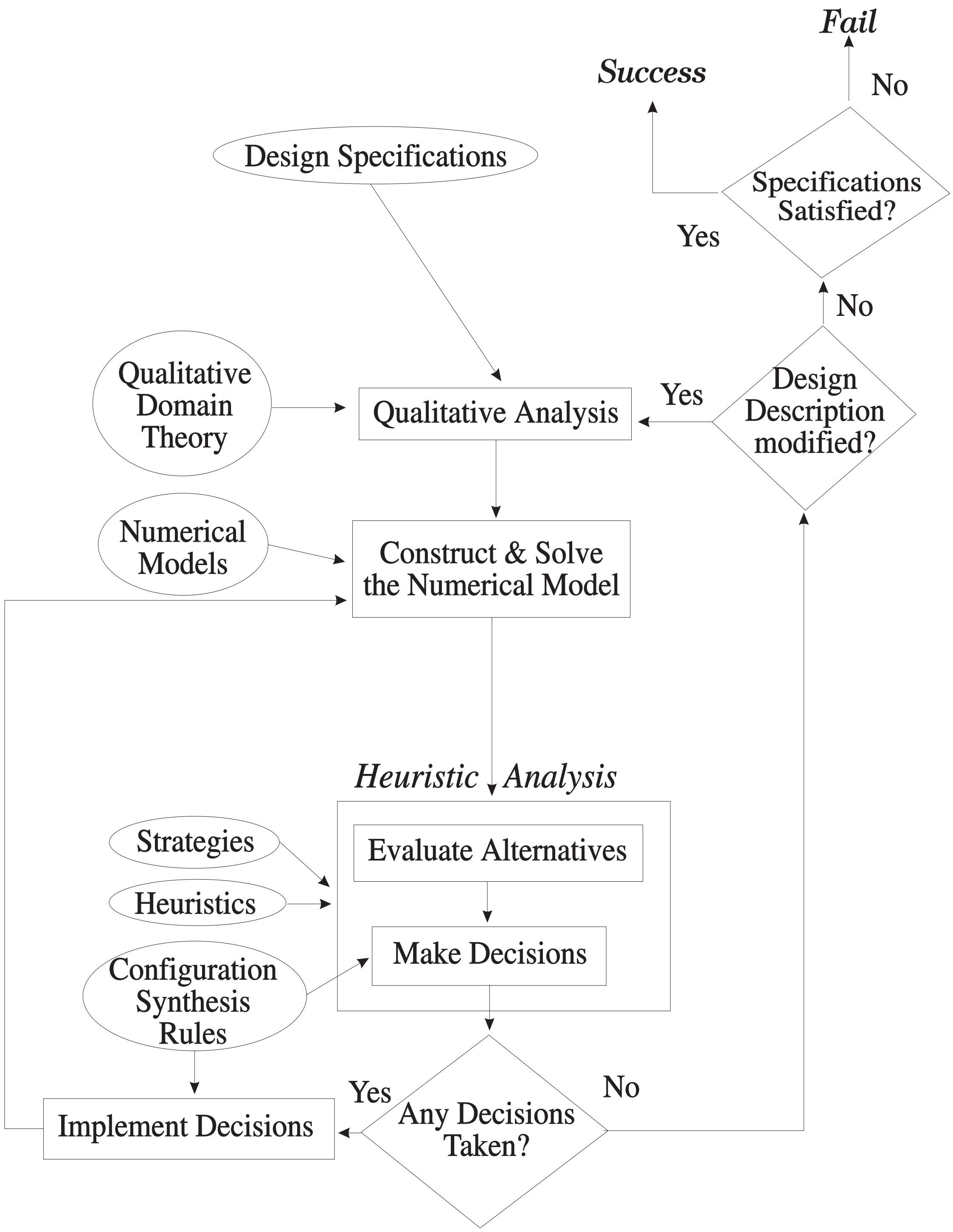
2.5.2 Construct and solve the numerical models.

The numerical model construction (section 2.3.2.2) and equation solving methods (section 2.3.2.3) are applied at this point. Since numerical models are used in `OUZO` to describe more accurately the behavior of the artifact and the design alternatives created by the qualitative models this step corresponds to the Analyze-Description and Generate-Alternatives steps of the design cycle as well.

2.5.3 Apply the design strategies and the configuration synthesis rules.

During this step the heuristics choose design alternatives and the current design description is updated accordingly. In particular, strategies and heuristics evaluate alternatives and make decisions, while configuration synthesis rules implement these decisions and monitor the state of design. This step corresponds to the

Figure 2.8: The controller algorithm in DIAS.



Evaluate-Alternatives, Make-Decisions and Implement-Decisions steps in Figure 2.1.

Steps 2 and 3 are executed in an inner loop that ends when there are no more design decisions taken. In this case, if the design description has been modified the system goes back to Step 1. The design process ends when the design description remains unchanged during a cycle. In this case, if the design specifications are satisfied the algorithm terminates with success, otherwise it exits with failure.

The loop between steps 2 and 3 does not correspond to a similar cycle in Figure 2.1. Its purpose is to make implementations such as `OUZO` more efficient. In particular, because qualitative analysis is computationally the most expensive stage in the design cycle, `OUZO` tries to do as much of the analysis as possible using the numerical models and the heuristics before resorting to qualitative analysis during the Analyze-Description step in Figure 2.1.

This controller is not tied to any particular heuristic or evolutionary design method. `OUZO` applies this controller on two types of problems. The first one deals with a heuristic method for binary distillation design, while the second one supports the evolutionary design of separation systems for multicomponent mixtures.

2.6 Where does `DIAS` apply?

Design is one of the major engineering activities. Consequently, much of engineering research is concerned with developing knowledge and methods that improve the efficiency of design. In the engineering research literature, design is usually described in terms of the physical knowledge involved in analyzing design alternatives, the list of heuristics for pruning the number of possible solutions and the strategies for applying the heuristic knowledge.

For example, papers that describe heuristic or evolutionary design of separation systems in chemical engineering typically consist of a list of heuristics for eliminating some of the design choices, along with strategies for sequencing the application of these heuristics [54], [43], [45]. The authors of these papers assume a shared corpus of physical knowledge for describing these systems, which they refer to by listing the modeling assumptions they use in analysis.

Another example is the synthesis of structural mechanisms in mechanical engineering. The design of mechanisms typically consists of four possible steps [73]:

1. Determination of the degrees of freedom, the number of links and joints and the complexity of the mechanism.

2. Enumeration of all possible graph topologies for a given combination of vertices and edges.

3. Application of various rejection criteria based on the design requirements.

4. Sketching of mechanisms corresponding to the generated graphs.

Papers that describe this kind of design ([46]) usually present the physical knowledge they use for performing the first two steps. A list of heuristics is given for the third step. Finally, it is assumed that there is a shared procedural knowledge for performing the sketching part. This procedural knowledge is captured in our approach by the configuration synthesis rules in a domain.

`DIAS` provides a computational account of engineering design that points out how informal descriptions for the design process can be transformed into computational models for performing the design task. This work does not propose either a cognitive model for design or an environment for supporting design education, although it seems possible that the `DIAS` framework is relevant to both problems.

Engineering design is not the only kind of design. There are other forms of design (e.g., fashion design, musical composition and other forms of artistic expression) that are performed and documented in different ways. For example, a fashion designer relies more on aesthetics and therefore on psychological and sociological knowledge about form and function, rather than on the qualitative and analytical knowledge about physics or chemistry that guide the understanding of an engineer. Consequently, descriptions of the design process in these domains appear to be less systematic compared to engineering design, since they take into account psychological and sociological processes that are highly idiosyncratic and variable. This thesis does not provide a computational model of design in these domains.

Furthermore, this model is inappropriate for routine or creative design. The number of possible configurations in routine design is relatively small and there are well-specified procedures for generating designs that satisfy particular sets of specifications. Therefore, specialized representations that either enumerate all the possible configurations or represent in advance all the possible design steps appear to be more efficient than our approach for routine design.

At the other end of the spectrum, part of the knowledge used in creative design problems is often transfered by analogy from other domains. Therefore, computational models of creative design will probably need to work with physical and design knowledge representations that support analogical reasoning (e.g. [16], [14]). This is not the case with our approach. However, we believe that general representations of design knowledge, similar to the ones proposed in this thesis (see section 3.5.6), can form the basis for developing efficient sets of heuristics in creative design.

2.7 Advantages of this approach

Four main features make DIAS attractive:

2.7.1 DIAS grounds design knowledge in physical knowledge

All of the terms in the heuristic rules are described as combinations of qualitative and numerical model fragments in the physical knowledge component. In addition, we describe a set of primitives for expressing design decisions made by the heuristic rules. Therefore, design knowledge representations become easier to write, understand and extend.

2.7.2 DIAS automates the interaction between physical and design knowledge.

Physical knowledge generates and analyzes design alternatives using reasoning methods that are sensitive to the design decisions taken during each design cycle. Design knowledge on the other hand evaluates all these alternatives based on the qualitative and numerical descriptions supplied by the physical knowledge component and makes design decisions using a well-defined set of primitives. A controller orchestrates the interaction between the different knowledge components. Consequently, this approach automates the interaction between the synthesis and the analysis phases in innovative design.

2.7.3 DIAS supports hierarchies of decisions in innovative design.

Systematic approaches to design reduce the design problem to a hierarchy of decisions that increase the efficiency of the process [13]. For example, the decomposition of process from equipment decisions in separation system design reduces significantly the number of possible designs[4]. DIAS supports this hierarchical structure through the use of compositional modeling techniques that organize physical knowledge based on these decisions and reasoning methods for the physical knowledge that are sensitive to design decisions during each design cycle.

2.7.4 DIAS supports different design methods.

Design methods are specified as part of the design knowledge through a combination of strategies and heuristic rules. Furthermore, the algorithm for controlling the design process is not tied to any particular method. As a result DIAS is flexible enough to support different heuristic or evolutionary design methods.

2.8 Relation to other work

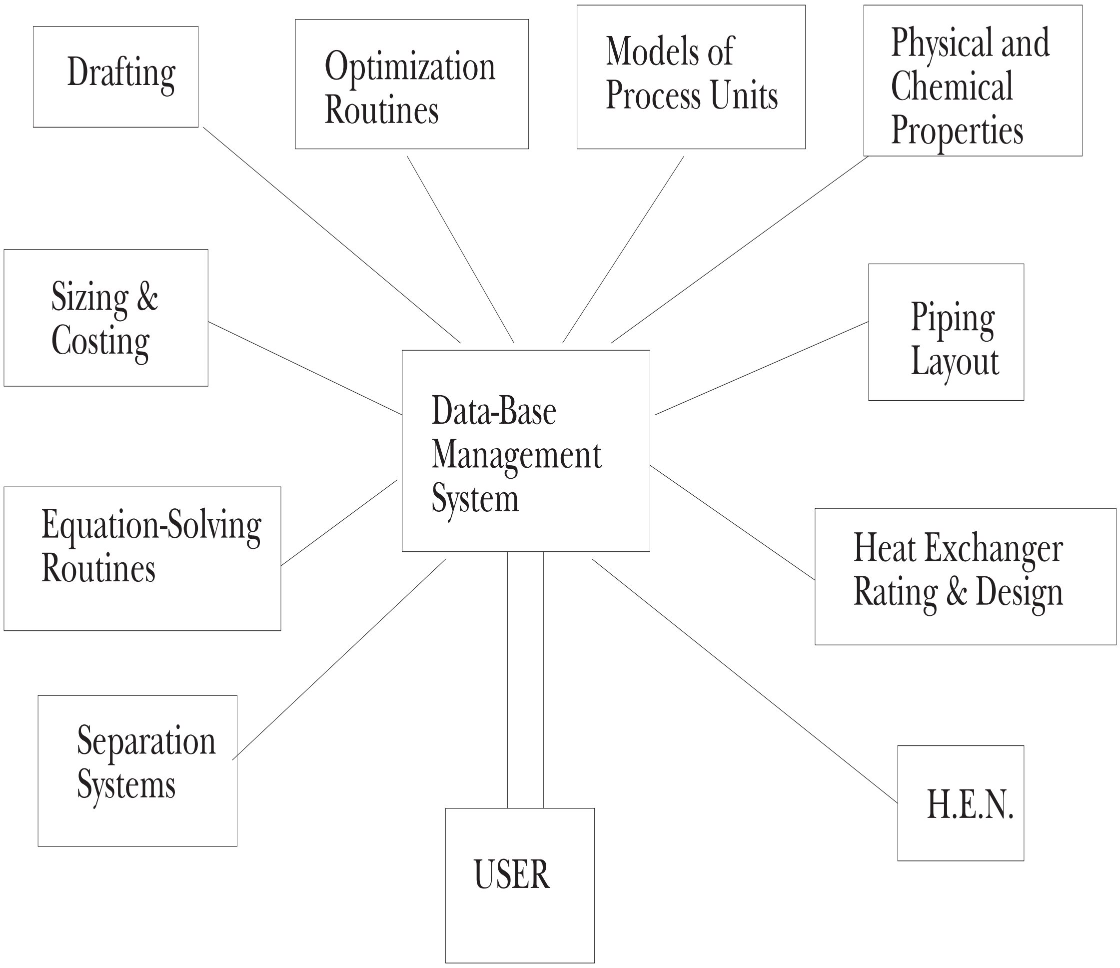
2.8.1 Computer-Aided Design (CAD) Systems

The major operating assumption behind the CAD methodology is that the computer provides a set of highly specialized support tools in order to help the user during design [60]. Computer-aided design (CAD) programs are extensively used in most engineering fields today. Figure 2.9 taken from [60] shows an example of an advanced CAD environment for chemical process design. In this program a database management system is used to coordinate the information flow between a set of specialized subroutines that deal with various design subtasks such as equipment sizing, physical and chemical properties, numerical optimization routines, etc.

Although CAD programs make it possible to derive numerical solutions for complex descriptions of artifacts, they rely exclusively on the human designer to formulate the alternatives and coordinate the use of all the programs during design. This happens because these systems do not possess either explicit models of the design knowledge (e.g. heuristics, design strategies) for formulating and evaluating design alternatives, or representations of physical knowledge that can automate the analysis phase of design. This is not the case in our approach, where the interaction between the synthesis and the analysis phases is fully automated.

[4] See section 3.5.7.

Figure 2.9: Typical computer-aided design environment.



2.8.2 Expert Systems for Design

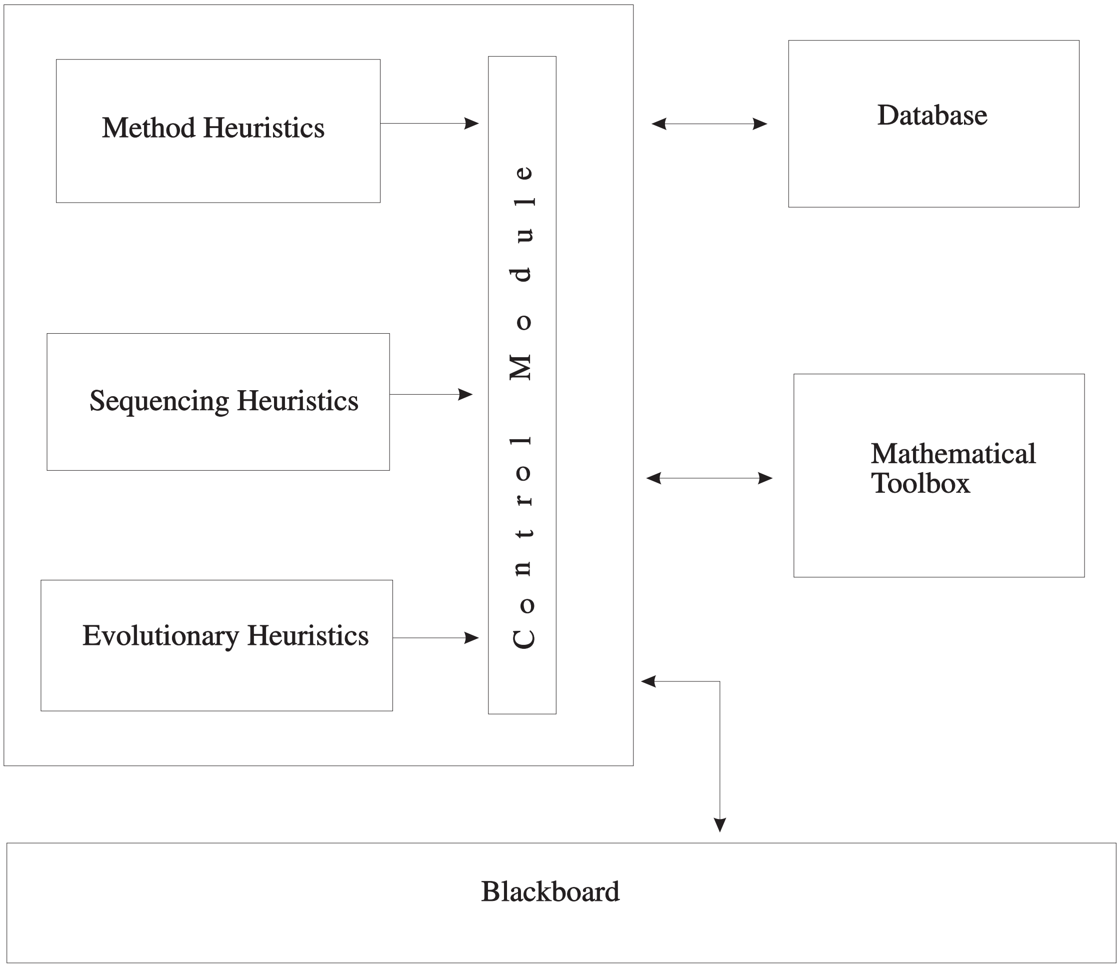
Expert systems are widely used in engineering design, mainly because they provide a simple way of representing design heuristics as sets of `if..then` rules [5], [59], [33], [40], [73]. Most of these systems use some form of reasoning about uncertainty to control the execution of these rules. In addition, they rely on specialized numerical procedures for supporting the analysis phase in design. Figure 2.10 adapted from [33] provides an example of a typical design expert system described in [33]. This particular example supports the design of separation sequences in chemical engineering. The system consists of a list of heuristic rules organized into three classes according to their effects on the current design. Method rules select a separation method, sequence rules arrange the separation units in a design and evolutionary rules propose changes to the current flowsheet. The execution of these rules is controlled by fuzzy logic methods. A list of mathematical routines (e.g. linear programming methods) in the mathematical toolbox and a database that contains physical properties for the various substances or data for separations that were calculated by the system in the past, provide the means by which heuristic rules compute relevant parameters for each design alternative. The current design description is kept in a blackboard data structure which is updated by the heuristics.

Although expert systems can be efficient enough for complex innovative design tasks (e.g. separation system design), they have certain drawbacks compared with our approach. In particular, these systems contain no explicit physical knowledge representations for grounding their heuristic rules. For example, an abstract description of one of the heuristics in [33] states that:

> 'IF adjacent separation properties do not vary widely AND relative quantities vary widely THEN separate at the component with the largest quantity.'

In order to use this rule in design, it has to be translated into a more specific representation in which the separation properties whose variance is measured are defined, the notion of 'wide variance' is quantified and the parameters that measure the relative quantities of the components are selected. In `DIAS` this process is facilitated by the physical knowledge component which contains qualitative and numerical descriptions for all the terms that will eventually be used in the rule. Consequently, the builder of the design knowledge base can create a representation for this rule based on these descriptions, instead of resorting to ad-hoc representations as is the case with current expert systems. Furthermore, physical

Figure 2.10: Typical expert system for design.



knowledge models provide a common reference to the rest of the community for understanding and possibly extending the design knowledge in our approach.

Finally, the lack of explicit physical knowledge representations in expert systems makes their heuristics hardwired to specialized numerical procedures for calculating relevant design parameters. For example, the mathematical toolbox in Figure 2.10 contains numerical methods that are called directly by the heuristic rules. Typically, the workings of these procedures are opaque to the user and to the rest of the system. Therefore, their interpretation depends largely on the intuitions of the engineer. This severely limits efforts to understand or extend these systems.

In contrast to the hardwired analysis methods in expert systems, the analysis phase in DIAS consists of the construction of numerical models based on sets of active qualitative model fragments. The activation of these models is controlled and justified by explicit sets of modeling assumptions, the current structure of the artifact and the results of the heuristic analysis. Consequently, the analysis phase becomes more transparent, because the way models are formed and solved can be described in terms of the contents of the physical knowledge component and the current design decisions. This results in a more intuitive account of design that captures how decisions taken during the synthesis phase have an impact on the analysis of the artifact. Although we do not have an explanation system that demonstrates this ability, we believe that such a capability would be easy to add by tracing the justifications for the model fragments used during the analysis phase.

2.8.3 Case-Based Design Systems

Case-based design systems ([32], [26], [44]) contain a library of design cases that are indexed based on features that are relevant to the design task (e.g. function, cost, etc). The system looks up in the library a set of cases that can potentially match with the design specifications. The rest of the design procedure consists of a series of adaptations performed by the system on the selected cases in order to achieve a closer match between these and the user specifications.

Case-based design systems take advantage of the large number of design cases that exist in many domains. An additional advantage is the similarity betweeen the way they operate and the way engineers solve design problems by updating already existing designs. There are two main challenges for case-based systems:

1. Selecting the features under which each design case should be indexed (the indexing problem [51]). Recent work in this area [4] suggests that engineering 'themes' containing abstract knowledge common to large numbers

of design cases could form the basis for the indexing vocabulary in this case. Other researchers have argued for functionally motivated vocabularies [28]. Despite some recent progress [23], both engineering themes and functional representations are still relatively unexplored areas in AI research

2. Providing principled ways of adapting the retrieved cases to satisfy the design specifications. Design adaptation is a complex process requiring detailed knowledge of the physical principles and the design heuristics of a domain. Therefore, an integration of this research with case-based methods is a viable method of addressing this problem.

There is no direct comparison between case-based design systems and our approach. Case-based systems support a particular design methodology, while `DIAS` is a computational framework for organizing the types of knowledge in design that is independent of heuristic or evolutionary methodologies. For example, `OUZO` contains a set of evolutionary strategies which are similar to case-based design methods (see section 3.5.5). Furthermore, `DIAS` uses compositional modeling techniques to address the problem of decomposing the design problem in hierarchies of decisions that increase the efficieny of the process. The ways by which this decomposition is achieved are independent of design method (see section 3.5.7) and have not been addressed in case-based design systems yet.

Most of the current implementations of case-based systems focus on design synthesis [44], [32]. We believe that this research will be valuable to CBR researchers, because it describes analysis methods that integrate qualitative and numerical models in design. These methods can enhance the limited analysis capabilities of existing CBR design systems.

2.8.4 Physical Principles Design Systems

Physical principles systems can be loosely defined as programs that concentrate on integrating representations from mathematics, physics, and engineering in the design process [34], [65]. This research can be classified as innovative design from physical principles.

Until recently, most of the research in this area has concentrated on using predominantly qualitative models for capturing the physical knowledge used in design. The development of design knowledge representations and the integration between physical and design models have not received considerable amount of attention yet. This research tries to fill this gap.

Furthermore, most physical principles systems focus on creative design problems [69], [1]. Current creative design approaches have two main problems:

1. Most creative design problems generate a huge number of possible configurations. Current design programs do not have effective ways of reducing the computational complexity associated with searching this space.

2. No good criteria for evaluating the quality of the proposed designs have been proposed yet.

While the search space for innovative design problems is comparably large, innovative design systems do not suffer from the same problems, since there are many heuristics that engineers use to choose between alternatives. Finding efficient patterns of interaction between the design knowledge that is predominantly heuristic and the physical knowledge that describes design artifacts is one of the major challenges facing any innovative design system.

There is considerable interest from the process engineering community in applying AI techniques and in particular qualitative reasoning in a variety of problems including process synthesis, monitoring and diagnosis [60], [61], [62], [8], [6], [39]. Most of the work in this area focuses on developing computational accounts that allow the automatic construction of models for chemical processes at multiple levels of detail. The integration of qualitative and numerical models with design knowledge representations has not been the main focus of that research yet. We believe that this thesis provides significant insights on how this integration is possible in process engineering.

Chapter 3

The Design of Separation Systems in Chemical Engineering

3.1 Introduction

Substances tend to mix together in intimate ways. Two examples are the solution of salt with water and the various hydrocarbons in oil. This phenomenon is an instance of the second law of thermodynamics, which requires all natural processes to occur in ways that increase the entropy, or randomness of the universe. As a result, the inverse process, that of the separation of mixtures of species into products of different composition, can only take place through the creation of a process or a device that uses energy to overcome this natural tendency.

Separation processes in chemical engineering are defined as operations which transform a mixture of substances into two or more products which differ from each other in composition [35].

Depending on the way by which the desired separation is achieved, there are two main categories of separation processes [35]:

1. *Rate-governed* processes achieve separation via differences in transport rates through some medium, under a driving force resulting from a gradient in pressure, temperature, composition, electric potential, etc. Examples of typical rate-governed separations include gaseous/sweep/thermal diffusion, mass spectometry, electrophoresis, electrodialysis, reverse osmosis, etc.

2. *Equilibrium* processes achieve separation through the equilibration of two immiscible phases which have different compositions at equilibrium. Two phases of a mixture are in *equilibrium* with each other if the temperature, pressure and chemical potentials for each component in the mixture are assumed to be uniform in both phases, i.e. these variables have no spatial or temporal gradients. Examples of typical equilibrium separations include distillation, stripping, absorption, extraction, leaching, etc.

The program (`OUZO`) that implements the ideas in this thesis deals with the design of equilibrium separations and in particular with the design of ordinary and extractive distillation processes. The rest of this chapter describes the physical principles underlying these operations. In addition, we present the various existing methods for designing separation systems and talk about their computational cost. Finally, we describe a set of general filtering strategies we developed for classifying the heuristics used in separation system design. Readers who are familiar with these phenomena may skip this chapter.

3.2 Distillation

Distillation is one of the most widely used separation processes in chemical engineering. It involves the separation of the components of a mixture based upon differences in their tendencies to evaporate at a given temperature. In a binary (two-component) mixture, the component with the highest tendency to evaporate is called the *volatile* component of the mixture. The other component is called the *non-volatile* component of the mixture.

Distillation usually involves a series of stages in which vapor and liquid phases come into contact and (ideally) achieve equilibrium. In general, the liquid and vapor compositions for each component in the mixture are not the same in equilibrium. In many cases the volatile component tends to concentrate in the vapor phase, leaving the liquid richer in the non-volatile component. This provides the basis for separation in distillation.

Figure 3.1 depicts a typical 12-stage column for a continuous distillation process. In this example, a binary mixture (*feed*) enters the column at stage 6. Furthermore, there is a total condenser[1] at the top of the column and a partial reboiler[2] at the bottom. In each stage liquid and vapor come into contact, causing some of the liquid to evaporate and some of the vapor to condense. In the column the liquid flows down due to the force of gravity, while the vapor flows upward under the force generated by a slight pressure drop from stage to stage. The vapor that reaches the top of the column is condensed in the total condenser and part of the resulting liquid (*reflux*) is returned back to the tower, while the rest of it is retrieved as the distillate product. An analogous situation occurs in the partial reboiler where part of the liquid at the bottom of the column is retrieved as the bottom product of the process and the rest of it is vaporized in the reboiler and returned back to the tower. The net effect of the process is an increase in the concentration of the volatile component in the vapor and of the non-volatile component in the liquid.

The analysis used to describe distillation columns that accept as feed multicomponent mixtures is similar to the one used for binary distillation. In these cases two of the components of the mixture with neighboring boiling points are selected as the *key components* of the separation. Usually, the one with the lower boiling point is called the *light key* while the other one is called the *heavy key*. All the other components are called *nonkeys*. Under this description, distillation causes most of the light key and all the more volatile nonkeys to appear in the

[1] A total condenser condenses all of its vapor input.

[2] A partial reboiler vaporizes part of its liquid input.

Figure 3.1: A typical distillation column

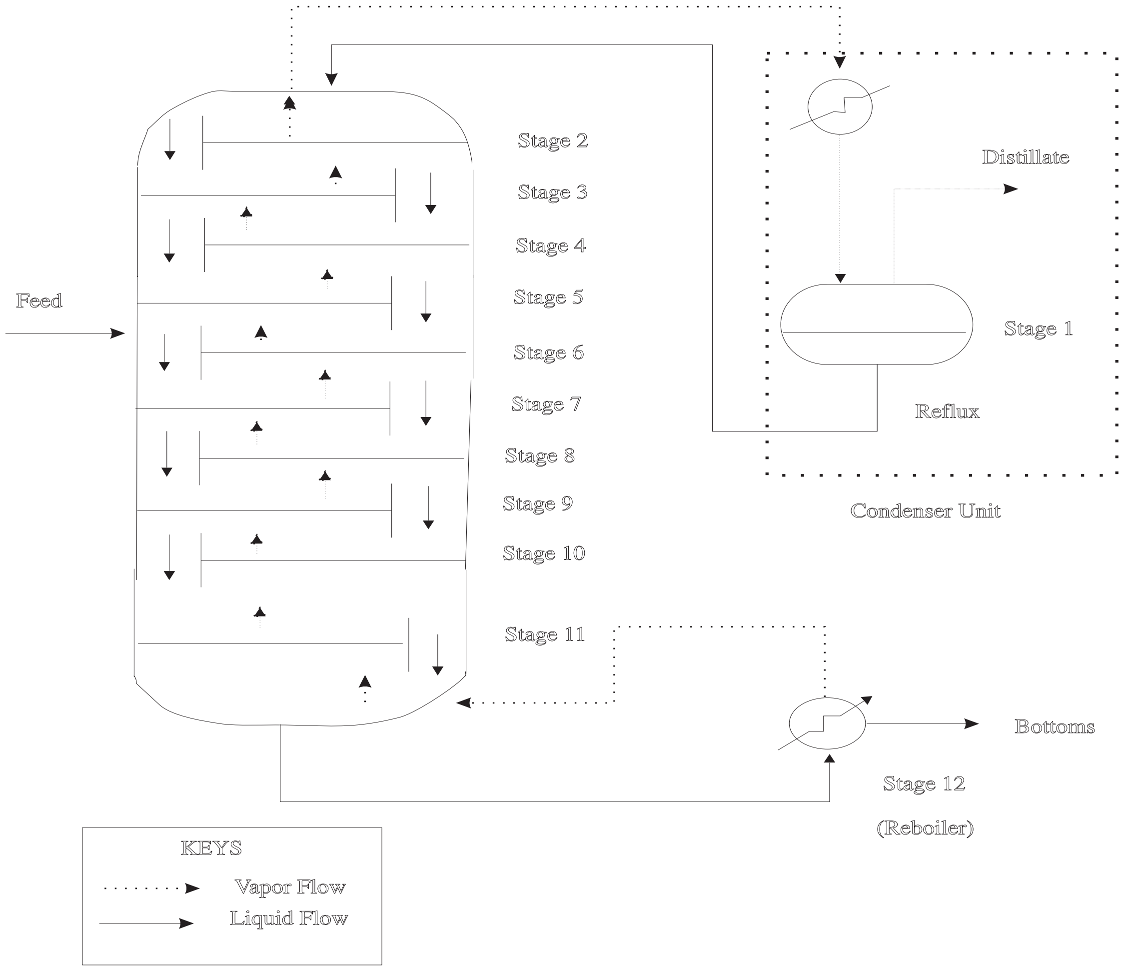


Table 3.1: Physical properties for multicomponent distillation input example.

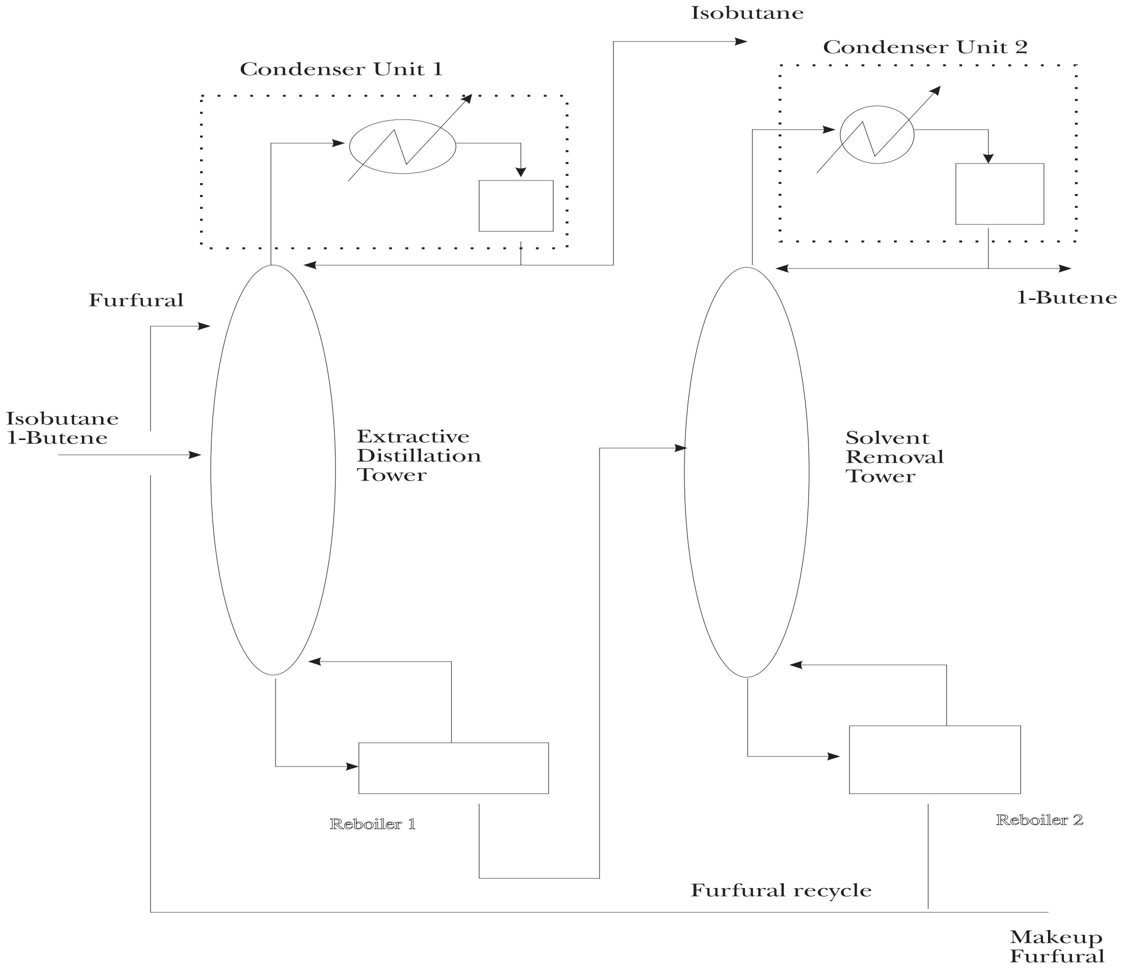
Species	Normal Boiling Point °C
Propane	-42.1
Isobutane	-11.7 ← *light key*
n-Butane	-0.5 ← *heavy key*
Isopentane	27.8
n-Pentane	36.1

distillate product, while most of the heavy key and all the less volatile nonkeys end up in the bottoms. For example, if a distillation column accepts as input the mixture shown in Table 3.1 adapted from [54] and we specify Isobutane and n-Butane as our light and heavy keys respectively, then we are going to end up with a distillate product containing most of the Isobutane, all of the Propane and some of the n-Butane. The bottom product will consist of most of the n-Butane, some of the Isobutane and all of the Isopentane and n-Pentane.

3.3 Extractive Distillation

Extractive distillation involves the addition of a new component to a mixture in order to facilitate the separation of the system by distillation [35]. The added component modifies the equilibrium relations between the vapor and the liquid phases in the mixture in a direction that favors the desired separation. Figure 3.2 taken from [35] depicts a typical extractive distillation unit. This usually involves two distillation columns. The first one is used to accomplish the desired separation while the second one is used to recycle the added component. The added component is usually refered to as the *mass separating agent (MSA)* or the *solvent*. The need for recycling the solvent at a later stage dictates the use of a solvent that is much less volatile than the original species. This facilitates the separation of the MSA from the rest of the mixture in the second column.

Figure 3.2: Extractive distillation unit for the separation of isobutane from 1-butene using furfural as the mass separating agent.



3.4 Separation Properties of Substances

At the level of analysis we are using in OUZO, there are three major parameters that describe the separation properties of substances in distillation: their *equilibrium ratios*, their *vapor pressures* and their *relative volatilities*.

The equilibrium ratio of a substance A is defined as the ratio between the mole fractions of A in two phases (gas and liquid in our case) at equilibrium conditions [35].

The vapor pressure for a substance at a specified temperature is the pressure in which the liquid phase of the substance can exist in equilibrium contact with its vapor [3].

Finally, the relative volatility between two substances A and B (α_{AB}) is defined as the ratio of their equilibrium ratios. In systems that obey Raoult's and Dalton's laws, equilibrium ratios can be linked to their vapor pressures. Consequently, in these systems the relative volatility is defined as the ratio of their vapor pressures. Intuitively, the relative volatility of two substances A and B is a measure of the difference in their tendencies to evaporate. A large value for α_{AB} means that A is much more volatile than B. The difficulty of separating two substances with distillation is inversely proportional to their relative volatility.

3.5 Design of Separation Sequences in Chemical Engineering

3.5.1 Overview of Process Synthesis

Process synthesis in Chemical Engineering is defined as [45]:

> "an act of determining the optimal interconnection of processing units as well as the optimal type and design of the units within a process system."

Process synthesis is the initial and most creative part of process design. It is also the part that is most crucial for the quality of the whole design procedure [33]. What makes this area particularly interesting for AI research is the need for integrating different types of knowledge in the process. These include knowledge of the physical principles that describe chemical processes, along with the design knowledge that is encapsulated in sets of heuristic rules and design strategies.

There are three important problems in process synthesis [42]:

1. The *Representation Problem* consists of developing a representation of the problem that is:

    - *Expressive.* All the alternatives can be represented and reasoned about.
    - *Effective.* The knowledge captured in the representation can guide the system in solving design problems.

2. The *Evaluation Problem* consists of finding ways of efficiently evaluating design alternatives.

3. The *Strategy Problem* is the development of strategies that optimize the design process itself.

The following chapters describe a program that is able to deal with all these problems in the context of separation system design.

Process synthesis research is usually classified according to the nature of the problem it addresses. A common approach is to decompose the synthesis problem into reaction path synthesis, heat exchanger network synthesis, separation system synthesis, reactor network synthesis, entire flowsheet synthesis and control system synthesis. `OUZO` focuses on the synthesis of separation systems and in particular on the binary distillation design problem and on the synthesis of separation sequences for multicomponent mixtures. Both problems are defined below.

3.5.2 The Binary Distillation Design Problem

The binary distillation design problem deals with the calculation of the number of stages for achieving a desired separation in a binary distillation column, given that the desired separation and the flow at some point in the column (usually the reflux) are specified [35].

`OUZO` solves this problem using qualitative and numerical models of a binary distillation column at steady-state, along with heuristics for locating the feed stage and for detecting possible inconsistencies between the design specifications and the results of the analysis phase.

3.5.3 Separation System Design for Multicomponent Mixtures

3.5.3.1 Problem Definition

The synthesis of separation sequences is defined as follows [43]:

> "Given a feed stream of known conditions (i.e., composition, flow rate, temperature, pressure), synthesize systematically a process that can isolate the desired (specified) products from the feed at minimum cost."

There is a large number of possible designs associated with this problem. For a mixture of N components to be separated into N pure component products using M separation methods, the number of possible flowsheets is given by [64]:

$$R = \frac{[2(N-1)]!}{N!(N-1)!} M^{N-1} \tag{3.1}$$

This equation is valid for *sharp* separators. A sharp separator splits a single feed stream into two product streams. Each entering component in the feed exits in only one product stream. The multicomponent mixture example in section 3.2 (Table 3.1) describes a sharp separation.

The version of the synthesis problem we concentrate on involves sharp separation units. It also involves ideal columns, i.e. columns in which the liquid and vapor phase in each stage are assumed to be in equilibrium. At its current stage, this research covers only ordinary and extractive distillation methods.

Separation system synthesis has many industrial applications because separation processes are used in almost every chemical plant. The design problem can be decomposed into two distinct subtasks [45]:

1. *The Synthesis Task:* Find the optimum sequence of separations and the nature of each separator.

2. *The Analysis Task:* Find the optimum values for the design variables (sizes, operating conditions) of each separator.

Searching for an optimal design usually involves the iterative execution of the synthesis and analysis steps at varying levels of detail.

3.5.4 Design Methods

There are three major design methods for the design of separation systems [45]:

1. *Heuristic* methods which use rules-of-thumb derived from engineering experience and the insights in the physics and chemistry of the separation methods.

2. *Evolutionary* methods which attempt to identify the best separation system through a sequence of evolutionary improvements.

3. *Algorithmic* techniques which employ various algorithms developed in the area of nonlinear mathematical programming. Algorithmic methods rely explicitly on well-developed numerical methods that produce optimal design solutions but are computationally inefficient and cumbersome to use in the majority of cases.

Instances of both heuristic and evolutionary methods have been implemented in OUZO. A heuristic strategy deals with the binary distillation design problem. Evolutionary strategies are used in the design of separation sequences for multicomponent mixtures.

3.5.5 Evolutionary Methods

An evolutionary method [45] consists of the following steps:

1. Generate an initial separation sequence. Many of the heuristic methods for developing separation sequences can be used at this stage in order to get a sequence that is reasonably close to the optimal one.

2. Apply a set of evolutionary rules that modify the initial structure in ways that are likely to lead to a better design.

Over the years a number of evolutionary strategies have been proposed [63], [37], [38]. All of these approaches use overlapping subsets of nineteen major design heuristics[3] shown in Table 2.1 taken from [45] for synthesizing the initial separation sequence together with a set of evolutionary rules that differ between approaches.

Evolutionary methods are similar to case-based design methods. The main difference between traditional case-based systems and these methods is the absence

[3] Chapter 5 describes these heuristics in detail.

of a case library in the later for coming up with an initial design. Instead they use a set of general heuristics for creating a reasonably good initial case. These heuristics have been abstracted from a large number of past designs, therefore they correspond to ossified cases [51]. The adaptation of the initial case proceeds in an analogous way with case-based systems via a set of evolutionary heuristics.

The design approach in this thesis can support evolutionary methods. As an example two recent approaches [43], [54] have been implemented in OUZO. We decided to focus on these strategies mainly because they are the most complex and general evolutionary strategies in terms of the number of heuristic and evolutionary rules and of the scope of the separation processes they support.

3.5.6 A Commonsense Interpretation of the General Separation Heuristics

This section presents a commonsense explanation of the heuristics in Table 2.1 so that readers without a chemical engineering background can understand more easily the implementation of these heuristics in OUZO (see chapter 5). Given that the purpose of a separation system is to isolate specific components from an input mixture, one can represent the heuristics involved as instantiations of general filtering strategies[4]. We developed the following eleven filtering strategies for describing these heuristics:

3.5.6.1 Preserve the purity of the product during the filtering process

Heuristic 1 is an instantiation of this rule. In distillation, for example, liquid flows downward and vapor flows upward through the column. Because of the effects of gravity, most of the contaminants tend to concentrate in the liquid phase, while the vapor phase remains relatively pure. Therefore, overhead (distillate) products, which result from the vapor phase at the top of a column, are more pure than the bottom products, which result from the liquid phase at the bottom of a column.

[4] Thanks to Larry Birnbaum for suggesting filtering as the unifying idea for developing these explanations.

3.5.6.2 Minimize the interference between the components during filtering

Interference introduces additional noise in the input, therefore it results in a more difficult filtering process. For example, in separation system design interference means either undesirable reactions that take place during the process or thermally unstable components. Consequently, heuristics 8 and 16 are attempts to prevent substances in a mixture to associate during the separation process.

3.5.6.3 Avoid damaging the filters

Corrosive materials destroy the trays and the piping that are necessary to run a separation unit. Therefore, according to heuristic 8 they should be removed from the process as early as possible in order to localize possible tear only at the first columns of a separation system.

3.5.6.4 Preserve the original specifications for the filtering process

For every process that violates the original design specifications (e.g. product recovery, desired products) there has to be some other process at a later stage that reinstates the initial specifications. For example, suppose that we want to separate a mixture of four components A, B, C and D into a 3-component mixture ABC and a single component D.. If instead of two mixtures, our final product consists of three mixtures, mixture (1) containing A and mixture (2) containing B and C and mixture (3) containing D, we have to design an extra process for blending mixtures (1) and (2) in order to recover the desired 3-component mixture. In short, violating the original specifications usually results in a more complicated and, potentially, more expensive design.

Heuristics 11 and 17 are examples of this strategy.

3.5.6.5 Avoid extreme operating conditions in the filter

There are two major disadvantages with extreme operating conditions: (i) They result in high energy requirements for the process and (ii) It is more expensive to build a filter that can withstand extreme operating conditions (temperature and pressure in the case of distillation). Examples for this strategy include heuristics 18 and 19.

3.5.6.6 The fewer filters the better

Typically the cost of a filtering system increases with the number of components in it. Therefore, processes that introduce more filters in a design should be avoided. For example, extractive distillation processes in separation system design usually involve two columns; one for introducing the mass separating agent (MSA) that facilitates the desired separation and another one for removing the MSA from the products. Because ordinary distillation processes involve only one column, they usually result in cheaper flowsheets and therefore are prefered over extractive distillation processes. Heuristics 15 and 12 are examples of this strategy.

3.5.6.7 Prefer the filtering process with the smaller energy requirements

Filters use some form of energy to separate their input. The higher this energy requirement is, the more expensive it is to run the filter. Research in separation system design has found that columns in which the amounts of distillate and bottoms products are almost the same have minimum heat requirements [43]. Furthermore, columns that operate close to ambient conditions require less energy than the ones operating in extreme conditions. Heuristics 3, 18 and 19 capture this line of reasoning.

3.5.6.8 Base the filtering process on properties of the input for which there is the maximum variance between the components

For example, distillation is based on the different tendency to evaporate between the components of a mixture. The relative volatility provides a measure for this tendency. The higher the relative volatility between two components, the easier it is to separate them using distillation. Heuristics 9 and 10 instantiate this general filtering rule in separation system design.

3.5.6.9 Perform difficult filtering operations with the minimal amount of input and the minimal number of components in the input

The cost of the filter increases with the amount of its input and the number of components in it. More input results in greater operating and installation costs, because the structure that is required to process the input becomes more complicated. More components increase the noise in the input. Consequently, in separation system design the most difficult separation should be done last (heuristic 2), since the input flow rates will be minimal in this case. In addition, separating the more plentiful components first minimizes the amount of input to the rest of the separation units, typically resulting in a cheaper design (heuristic 6). In the case of extractive distillation processes, the presence of an extra mass separating agent increases the input flow rates downstream. Therefore, the MSA should be removed as early as possible (heuristic 13). Finally, the higher the number of different components in the input, the more difficult it is to separate any combination of them. Consequently, high-recovery separations should be performed last, because both the amounts and the number of components in the input is minimal (heuristic 5).

3.5.6.10 If the filtering process introduces extra agents in the input, they should be easily removed

Introducing extra agents in the input of a filter adds another level of complexity in the design. For example, in separation system design extractive distillation processes typically involve two distillation columns. The first one introduces the mass separating agent (MSA) in the mixture, while the second one isolates the MSA from it. In order to make the cost of this process appealing, the isolation of the MSA in the second column should be an easy separation. Therefore, it should not be the case that extractive distillation (a generally expensive process) has to be used to isolate the MSA in the second column (heuristic 14).

3.5.6.11 Use the least expensive filter

It is not clear that the previous rules will always lead to the cheapest design. This rule makes this design criterion explicit. Heuristics 7 is an instantiation of this rule. Furthermore, heuristic 4 seems to be a very domain-specific instantiation of this rule. Typically, the operating cost for a distillation column is directly proportional to the amount of the distillate product. The distillate product is in turn proportional to the vapor flow in the column. Therefore, columns with less vapor flow usually result in cheaper designs.

3.5.7 The Cost of Adaptation in Evolutionary Strategies

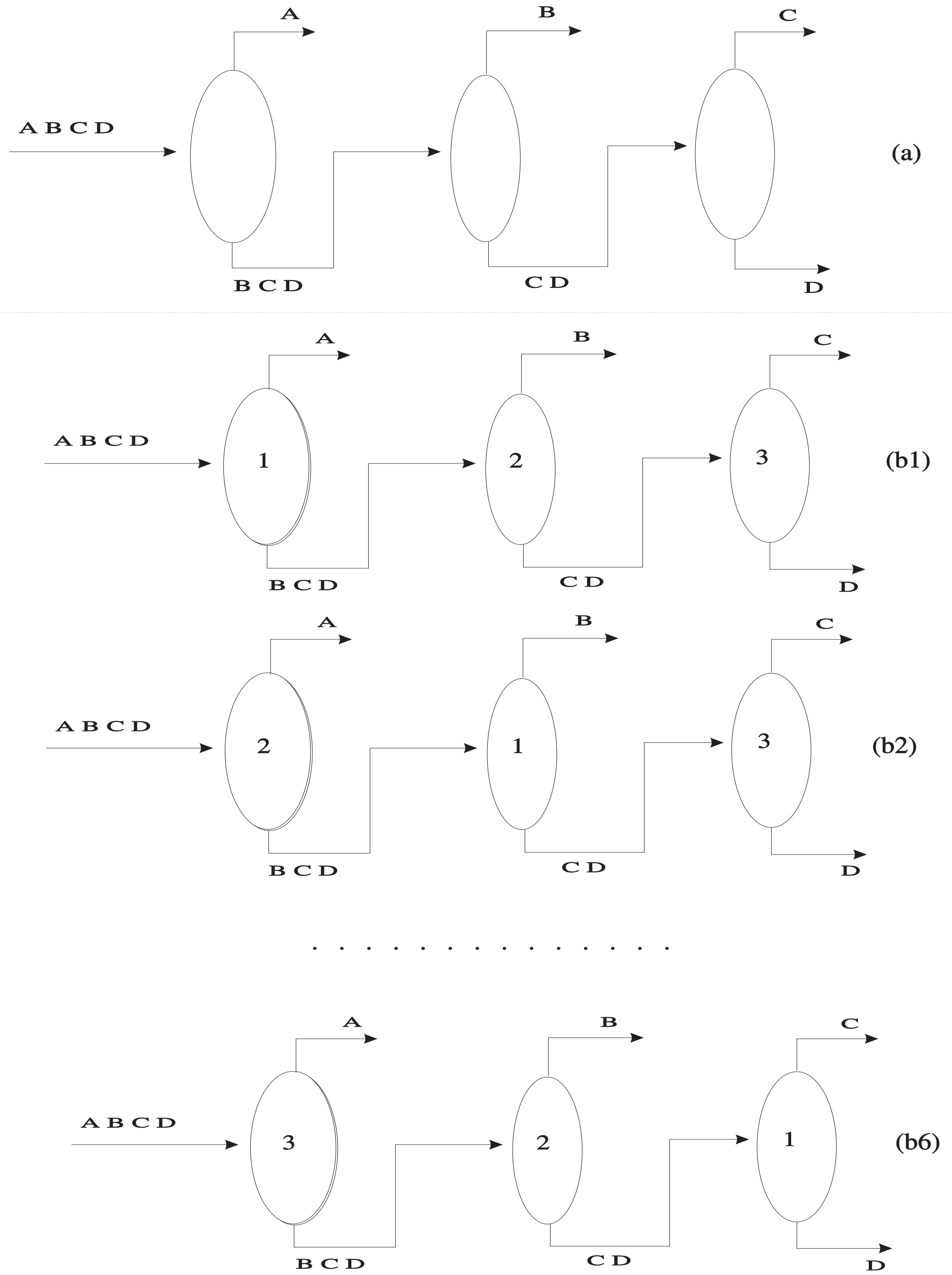
The design of separation systems in chemical engineering provides a good example of the computational costs associated with adaptation in evolutionary methods and in case-based design in general. In particular, the cost of adaptation in this domain increases with the number of levels at which design decisions have to be made concurrently. For example, evolutionary methods can be decomposed into two categories. The first one, known as *retrofit design* considers process decisions and equipment decisions in parallel [27]. In separation systems, for example, retrofit design considers both the actual process units (i.e. columns) and the separations that take place in them (e.g. a binary mixture AB is separated into its individual components A and B) during adaptation. This is not the case for the rest of the evolutionary strategies in which process decisions are made first, followed by equipment decisions. The later approach is known as *grassroots design* and in the case of separation systems it supports a process by which a sequence of separation tasks is constructed first, followed by the design of the equipment for each one of them.

Because of these different decompositions of the design task, the upper limit on the number of alternatives in separation system design is much higher for the retrofit than for the grassroots approach. In particular, if *S(N)*[5] is the number of possible sequences for separating a N-component mixture into its individual components in the grassroots case, the number of possible retrofit designs for the same case assuming that we already use N-1 columns is *(N-1)!S(N)*, while the number of possible sequences with one spare column in addition to the N-1 existing ones is $N \times N!S(N)$ [27]. Figure 3.3 explains why this is true for the separation of a 4-component mixture into its individual components. A, B, C and D are the individual mixture components. The oval shapes represent distillation columns. Part (a) of the figure indicates one possible sequence of separation tasks in grassroots design (the A/BCD, B/CD, C/D sequence). For each such sequence of 3 columns, there are 3! alternatives in the retrofit case corresponding to all the possible permutations of the three existing columns. Parts (b1), (b2) and (b3) of Figure 3.3 show three of the six alternatives that have to be examined in the retrofit case. Furthermore, current analysis methods are tailored for grassroots design, making the analysis in the retrofit case significantly harder.

This example indicates that although a particular design method (e.g. evolutionary or case-based design) can help in reducing the amount of search for finding an optimal design, an equally important parameter that improves performance is the structuring of the design task in ways that allow decisions at different levels to be taken independently. The later is independent of the choice of a design method and it argues for a compositional modeling [15] style of analysis that allows the construction of physical models that support the decision process at different levels of abstraction (e.g. at the process or the equipment level in separation system design). Case-based design systems have little to offer here, since they focus on the design method. Our approach on the other hand offers a broader account

[5] S(N) is given by equation 3.1.

Figure 3.3: The difference in the number of alternatives in grassroots and retrofit design.



for design that is able to support different design methods and a compositional modeling approach for the analysis phase in design.

The evolutionary strategies currently implemented in OUZO support the grass-roots design approach.

Chapter 4

Representing Physical Knowledge in OUZO

4.1 Physical Knowledge and Physical Models

Central to all the reasoning styles in science or engineering is the creation of representations that capture the understanding that engineers or scientists have about physical phenomena. These representations comprise the *physical knowledge* for a domain. Physical knowledge is organized around *models*, i.e. structured descriptions of the phenomena of interest. Each model contains:

- The constraint relations between the parameters of the system. We refer to these relations as the *physical principles* of a domain, since they provide the basis for understanding physical phenomena. Examples of physical principles include Newton's second law of motion, Ohm's Law, the relation between the phases of a substance at equilibrium, etc.

- Sets of *modeling assumptions* under which the physical principles in a domain are valid. These assumptions are used to construct simplified descriptions of the phenomena of interest. Examples include bodies that move with velocities much lower than the speed of light in the case of Newton's Law or ideal resistors in the case of Ohm's Law, etc.

- Sets of relevant features for the phenomena of interest. These include the introduction of quantities that measure important attributes of the system we are examining (e.g. quantities that measure cost in the case of design) along with methods for calculating them.

There are many ways in which models may vary from each other. Some of these include the *ontologies* they use, their *scope*, the *domain of applicability*, their *accuracy* and the *resolution* they offer [67].

The scope of a model refers to the range of phenomena it describes. Its domain of applicability specifies the constraints and/or the parameter values under which the model is valid. The accuracy of the model indicates how close are the model predictions with the observed behavior.

In terms of ontologies, there are two main modeling approaches: *process-centered* and *device-centered* ontologies. Process-centered ontologies [18] postulate a set of processes as the agents of causation in the world. For example, the boiling process is responsible for the conversion of water into steam in the kettle I forgot on the stove while writing this piece[1]. Device-centered ontologies [10], [70] view physical systems as networks of components, the behavior of which is specified

[1] Ooops!

by a set of internal laws. For example, a circuit can be modeled as a network of electrical components (e.g., transistors, resistors, etc) which can be analyzed in isolation using sets of equations that correspond to different operating regions or states.

In terms of resolution there are two kinds of models: *qualitative* and *numerical.* Numerical models contain systems of numerical relations (equations or inequalities) between the parameters of a system. Qualitative models include qualitative abstractions of the relations in a numerical model, the causal dependencies between its parameters and explicit representations of the modeling assumptions used to describe a physical system.

The following sections describe a set of qualitative and numerical models for separation processes. The scope of these models extends to ordinary distillation processes for binary mixtures along with ordinary and extractive distillation columns for multicomponent mixtures.

The domain of applicability for our models includes ideal columns for binary mixtures and sharp separation units for multicomponent mixtures.

In terms of accuracy, the models we are using provide approximate descriptions for separation processes. These descriptions are used in conceptual design to generate cost estimates for the design alternatives.

4.2 Qualitative Models

Qualitative models and reasoning techniques attempt to analyze the behavior of physical systems without resorting to numerical computations. This is typically done by partitioning the range of quantity values into a number of relevant subregions. For example, mapping the values of quantities into their signs is a common technique in this reasoning style.

OUZO investigates the use of process-centered qualitative models in capturing the physical knowledge used in the design of separation systems. We use Qualitative Process Theory (QPT) [18] as our modeling language. Appendix A provides a brief review of QPT for readers not familiar with it. QPT is especially suitable for modeling chemical processes, because its process-centered ontology is able to capture the physical principles on which unit operations in Chemical Engineering are based [6].

4.2.1 Qualitative Analysis

There are two major styles of qualitative analysis: (i) qualitative simulation and (ii) model construction. Qualitative simulation generates qualitative descriptions for the behavior of a physical system. The Qualitative Process Engine (QPE) [17] is an example of a system that is based on QPT and produces qualitative simulations of physical systems.

Model construction generates models that are consistent with a *domain model* and a *scenario*. The domain model contains qualitative descriptions of the domain's physics. The scenario contains a structural description of the system that is analyzed (i.e. what objects exist and how they are related) along with the modeling assumptions underlying the analysis. Model construction is usually a subset of what a qualitative simulator does. SIMGEN [21] is an example of a system that uses this kind of analysis. `OUZO` uses a subset of the qualitative analysis in SIMGEN.

4.3 Binary Distillation Columns

4.3.1 Overview

The following sections provide a detailed description of the qualitative model for a binary distillation column. The model we describe below serves two purposes. The first one is to provide a way of creating simulators in SIMGEN for the dynamic behavior of binary distillation columns. Section 4.3.2 describes the model components that support this task. The second one is to provide the physical knowledge that is necessary to solve the design problem for binary distillation columns [35]. The part of the model used in `OUZO` consists of the dynamic behavior component (Section 4.3.2) overlayed by the model fragments implementing the steady-state analysis (Section 4.3.3).

The qualitative domain theory consists of three main objects; mixtures, stages and column. In addition, there are two kinds of processes active; material flows and phase transition processes. At each object level there are a series of operations taking place during design. These are:

- Operations at the Mixture Level.

    1. Introduce quantities related to the pressure, temperature, bubble point and dew point of the mixture.

```
(defPredicate Non-Negative-Quantity  (defPredicate Positive-Quantity  (defPredicate Negative-Quantity
  (Quantity ?self)                     (Quantity ?self)                   (Quantity ?self)
  (not (less-than (A ?self) ZERO)))    (greater-than (A ?self) ZERO))     (less-than (A ?self) ZERO))
```

Figure 4.1: Predicates refering to quantities in the distillation model.

2. Introduce quantities for the total mass of the mixture and its components.

- Operations at the Stage Level.

  1. Activate the gas and liquid flows between stages.
  2. Enforce a set of modeling assumptions for each stage (e.g., equilibrium stages, negligible vapor holdup, constant molal overflow, etc).
  3. Relate the mass of the gas and vapor at each stage to the capacity of the stage.
  4. Establish the hudraulic properties of the stage.

- Operations at the Column Level.

  1. Establish operating conditions for the column (pressure, temperature).
  2. Enforce the constant relative volatility assumption.

In the rest of the thesis, the words in *emphasis* refer to predicate names or variables used in the figures.

4.3.2 The Dynamic Behavior of Binary Distillation Columns

Predicates. The *defPredicate* form in QPT specifies consequences of a single antecedent predicate. The first argument to defPredicate is the predicate whose consequences are being defined. The rest of the form includes a set of consequences which should be believed when the predicate is believed. Figure 4.1 defines some of the predicates that relate to the quantities in the distillation model. For example, the *Positive-Quantity* predicate indicates that a positive quantity is a quantity with a positive amount. The variable *?self* is a special variable in QPE that is bound to the object of the model fragment it belongs to (in this case the object of the defPredicate form).

```
(defentity (Physob ?obj))

(defentity (Contained-Binary-Mixture (2-C-S (?substance1 ?substance2) ?phase ?can))
  (Physob (2-C-S (?substance1 ?substance2) ?phase ?can))
  (Quantity (Temperature (2-C-S (?substance1 ?substance2) ?phase ?can)))
  (Non-Negative-Quantity (Amount-of (2-C-S (?substance1 ?substance2) ?phase ?can))))

(defentity (Contained-Binary-Liquid-Mixture (2-C-S (?substance1 ?substance2) liquid ?can))
  (Contained-Binary-Mixture (2-C-S (?substance1 ?substance2) liquid ?can))
  (Stage ?can)
  (Non-Negative-Quantity (Amount-of-in ?substance1 (2-C-S (?substance1 ?substance2) liquid ?can)))
  (Positive-Quantity
    (TBubble (2-C-S (?substance1 ?substance2) liquid ?can))))

(defentity (Contained-Binary-Gas-Mixture (2-C-S (?substance1 ?substance2) gas ?can))
  (Contained-Binary-Mixture (2-C-S (?substance1 ?substance2) gas ?can))
  (Stage ?can)
  (Non-Negative-Quantity (Amount-of-in ?substance1 (2-C-S (?substance1 ?substance2) gas ?can)))
  (Positive-Quantity (TDew (2-C-S (?substance1 ?substance2) gas ?can))))

(defentity (Container ?can)
  (Non-Negative-Quantity (Pressure ?can)))

(defentity (Stage ?can)
   (Container ?can)
   (Positive-Quantity (Beta ?can)))

(defentity (Distillation-Column ?column)
  (Container ?column))
```

Figure 4.2: The objects in the binary distllation model.

Objects. The model contains the following kinds of objects: *physobs, containers, stages, distillation-columns* and *contained-mixtures* (either in the vapor or liquid phase). Figure 4.2 provides the definitions for all the objects in the qualitative model. A *physob* is the simplest kind of object in the domain theory.

In order to describe all the fluid pieces of stuff that are present in a column we use the *contained stuff* ontology [11]. More specifically, a *Contained-Binary-Mixture* is defined by the substances it is composed of, by the phase it is in and by the container which holds it. This definition is similar to the one used for contained stuffs in [12]. The following function denotes a contained-binary-mixture:

$$\text{2-C-S: (component1, component2)} \times \text{phase} \times \text{container} \longrightarrow$$
$$\longrightarrow \text{Contained-Binary-Mixture}$$

Contained-Binary-Liquid-Mixtures and *Contained-Binary-Gas-Mixtures* are specializations of Contained-Binary-Mixtures. For each one of them we define the quantity *Amount-of-in* for a component to correspond to the number of moles of the component in the particular mixture and the *Amount-of* quantity to be the total number of moles for both components in the mixture.

The quantity *TBubble* for a Contained-Binary-Liquid-Mixture corresponds to the bubble point of the mixture. The bubble point is the temperature in which the mixture begins to vaporize as its temperature is increased. An analogous quantity for gas mixtures is the *TDew* quantity which corresponds to the dew point for the mixture. This is the temperature at which the mixture starts condensing as its temperature is decreased.

A *stage* object corresponds to a stage in the column and is a specialization of a *container*. Defining each stage in a column to be a container allows us to talk about the liquid or vapor in it using the contained stuff ontology. Associated with each stage is a constant *Beta* which is the hydraulic time constant associated with the liquid flow through this stage.

A *distillation-column* is also defined to be a type of container. Each stage object in the model is defined to be *Part-of* a given column.

Object Views. Figure 4.3 describes the views associated with binary mixtures in the model. When the Amount-of a given mixture is positive we say that the *Binary-Mixture* view exists. As a result, a set of views corresponding to the phase of the given mixture becomes active (*Binary-Gas(Liquid)-Mixture*). When a mixture exists we can talk about the mole fraction for each component in the mixture. This quantity corresponds to the ratio of the moles of the specific component over the total moles of the various components in the mixture. When a liquid mixture exists we can talk about its *Initial-Amount-of* at each stage. This

```
(defview (Binary-Mixture ?mixture)
  Individuals ((?mixture :Type Contained-Binary-Mixture
                         :Form (2-C-S (?substance1 ?substance2) ?phase ?can)))
  QuantityConditions ((greater-than (A (Amount-of ?mixture)) ZERO))
  Relations ((Positive-Quantity (Mole-Fraction ?substance1 ?mixture))
             (Positive-Quantity (Mole-Fraction ?substance2 ?mixture))
             (Qprop- (Mole-Fraction ?substance2 ?mixture) (Mole-Fraction ?substance1 ?mixture))))

(defView (Binary-Gas-Mixture (2-C-S (?substance1 ?substance2) gas ?stage))
  Individuals ((?stage :type Stage)
               (?mixture
                :Conditions
;; ?mixture is an instance of the Binary-Mixture view.
                ((View-Instance Binary-Mixture) ?mixture)
;; This specific instance of the Binary-Mixture view has its mixture variable bound to
;; the 2-C-S function described below.
                (?mixture MIXTURE (2-C-S (?substance1 ?substance2) gas ?stage))
                (More-Volatile ?substance1 :Than ?substance2)))
  QuantityConditions ((Active ?mixture)))

(defView (Binary-Liquid-Mixture (2-C-S (?substance1 ?substance2) liquid ?stage))
  Individuals ((?stage :type Stage)
               (?mixture
                :Conditions
                ((View-Instance Binary-Mixture) ?mixture)
                (?mixture MIXTURE (2-C-S (?substance1 ?substance2) liquid ?stage))
                (More-Volatile ?substance1 :Than ?substance2)))
  QuantityConditions ((Active ?mixture))
  Relations ((Positive-Quantity
              (Initial-Amount-of (2-C-S (?substance1 ?substance2) liquid ?stage)))))
```

Figure 4.3: Views associated with objects in the binary distillation model.

quantity allows us to specify initial conditions for the liquid mass at each stage in binary column simulations.

The predicate *Active* in the view definitions holds whenever the predicate it accepts as an argument holds. For example, in the case of the Binary-Gas-Mixture view it is active when the view Binary-Mixture denoting a gas mixture is active.

Modeling Assumptions. Four major modeling assumptions are used to simplify the qualitative description for the process (see Figs 4.4, 4.5, 4.6):

1. *Negligible Vapor Holdup.* We assume that the amount of vapor at each stage (the vapor holdup for this stage) is zero. Consequently, at each stage the holdup for the volatile component in the liquid is equal to the total holdup for this component in the vapor and the liquid. The *Negligible-Vapor-HoldUp* view (Fig. 4.4) describes this relation. The negligible vapor holdup assumption is enforced when the predicate *(Consider (Negligible-Vapor-HoldUp-in ?column))* is asserted in the scenario.

2. *Constant-Molal Overflow.* At each stage in the column the liquid and the vapor phase come into contact, causing some of the liquid to evaporate and some of the vapor to condense. If we assume that the molar heats of vaporization[2] of the two components are the same, then for every mole of vapor that condenses we have a mole of liquid that vaporizes. This assumption together with the negligible vapor holdup assumption cause the vapor flow rates to equalize throughout the column. The view *Conditions-for-Stage-Flows* (Fig. 4.5) asserts the consequences of the constant molal overflow assumption for each stage in the column. The Constant-Molal-Overflow assumption is enforced when the predicate *(Consider (Constant-Molal-Overflow-in ?column))* is asserted in the scenario.

3. *Constant Relative Volatility.* The relative volatility of a binary mixture is relatively insensitive to changes in pressure, temperature and composition. Therefore, it is usually a good assumption to consider a constant relative volatility throughout a binary distillation column. The *Constant-Relative-Volatility-Conditions* perspective (Fig. 4.6) describes the conditions under which this assumption is valid. The view *Constant-Relative-Volatility-Consequences* (Fig. 4.6) asserts the consequences of this assumption in the model by making the relative volatility between the mixture components

[2]The molar heat of vaporization for a component is the heat required to vaporize one mole of this component.

```
;; Describes qualitatively the relation of the total holdup for the volatile component at each stage
;; to the sum of the amounts of this component in the liquid and the vapor mixtures at each stage.
(defView (Stage-Operating-Features ?stage (2-C-S (?substance1 ?substance2) liquid ?stage))
  Individuals ((?stage :Type Stage
                       :Conditions (not (Reboiler-Stage ?stage))
                                   (not (Condenser-Stage ?stage)))
               (?liquid
                :Conditions ((View-Instance Binary-Mixture) ?liquid)
                            (?liquid MIXTURE (2-C-S (?substance1 ?substance2) liquid ?stage))
                            (More-Volatile ?substance1 :Than ?substance2))
               (?vapor
                :Conditions ((View-Instance Binary-Mixture) ?vapor)
                            (?vapor MIXTURE (2-C-S (?substance1 ?substance2) gas ?stage))))
  QuantityConditions ((Active ?liquid) (Active ?vapor))
  Relations ((Quantity (HoldUp-of-Material ?substance1 ?stage))
             (Q= (HoldUp-of-Material ?substance1 ?stage)
                 (+ (Amount-of-in ?substance1 (2-C-S (?substance1 ?substance2) liquid ?stage))
                    (Amount-of-in ?substance1 (2-C-S (?substance1 ?substance2) gas ?stage))))))

;; Describes the approximations that the Negligible Vapor HoldUp assumption introduces at each stage. In
;; particular under this assumption the total holdup of the volatile component in the liquid is equal to the
;; total holdup for this component in the vapor and liquid phase at each stage. If this assumption was not
;; valid the holdup of the volatile component in the liquid at each stage would have been equal to the Amount-of-in
;; for this component at this stage and we would not have to create two separate quantities for this parameter.
(defView (Negligible-Vapor-HoldUp ?stage)
  Individuals ((?column :Type Distillation-Column
                        :Conditions (Consider (Neglect Vapor-HoldUp-in ?column)))
               (?stage :Type Stage
                       :Conditions (not (Reboiler-Stage ?stage))
                                   (not (Condenser-Stage ?stage))
                                   (Part-of ?column ?stage))
               (?l-mix
                :Conditions ((View-Instance Binary-Mixture) ?l-mix)
                            (?l-mix MIXTURE (2-C-S (?substance1 ?substance2) liquid ?stage)))
               (?g-mix
                :Conditions ((View-Instance Binary-Mixture) ?g-mix)
                            (?g-mix MIXTURE (2-C-S (?substance1 ?substance2) gas ?stage))))
  QuantityConditions ((Active ?l-mix) (Active ?g-mix))
  Relations ((Q= (HoldUp-of ?substance1 (2-C-S (?substance1 ?substance2) liquid ?stage))
                 (HoldUp-of-Material ?substance1 ?stage))))
```

Figure 4.4: View and perspective associated with the negligible vapor holdup assumption.

```
;; It describes the consequences of assuming equilibrium between the vapor and liquid phase at each stage.
(defView (Binary-Vapor-Liquid-Equilibrium ?stage)
  Individuals ((?column :Type Distillation-Column)
               (?stage :Type Stage
                       :Conditions (Part-of ?column ?stage)
                                   (Consider (Equilibrium-Stage ?stage)))
               (?vapor
                :Conditions ((View-Instance Binary-Mixture) ?vapor)
                            (?vapor MIXTURE (2-C-S (?substance1 ?substance2) gas ?stage))
                            (More-Volatile ?substance1 :Than ?substance2))
               (?liquid
                :Conditions ((View-Instance Binary-Mixture) ?liquid)
                            (?liquid MIXTURE (2-C-S (?substance1 ?substance2) liquid ?stage))))
  QuantityConditions ((Active ?vapor) (Active ?liquid))
  Relations ((Quantity (Alpha (2-C-S (?substance1 ?substance2) liquid ?stage)
                              (2-C-S (?substance1 ?substance2) gas ?stage)))
             (Qprop (Mole-Fraction ?substance1 (2-C-S (?substance1 ?substance2) gas ?stage))
                   (Alpha (2-C-S (?substance1 ?substance2) liquid ?stage)
                          (2-C-S (?substance1 ?substance2) gas ?stage)))
             (Qprop (Mole-Fraction ?substance1 (2-C-S (?substance1 ?substance2) gas ?stage))
                    (Mole-Fraction ?substance1 (2-C-S (?substance1 ?substance2) liquid ?stage)))))

(defView (Conditions-for-Stage-Flows ?stage ?liquid)
  Individuals ((?stage :Type Stage
                       :Conditions (not (Reboiler-Stage ?stage))
                                   (not (Condenser-Stage ?stage)))
               (?column :Type Distillation-Column
                        :Conditions (Part-of ?column ?stage))
               (?liquid
                :Conditions ((View-Instance Binary-Mixture) ?liquid)
                            (?liquid MIXTURE (2-C-S (?substance1 ?substance2) liquid ?stage))
                            (More-Volatile ?substance1 :Than ?substance2))
               (?in-g-f :Conditions ((Process-Instance Gas-Flow) ?in-g-f)
                                    (?in-g-f DST ?stage))
               (?out-g-f :Conditions ((Process-Instance Gas-Flow) ?out-g-f)
                                     (?out-g-f SRC ?stage))
               (?out-l-f :Conditions ((Process-Instance Liquid-Flow) ?out-l-f)
                                     (?out-l-f SRC ?stage)))
  Preconditions ((Consider (Constant-Molal-Overflow-in ?column))
                 (Consider (Neglect Vapor-HoldUp-in ?column)))
  QuantityConditions ((Active ?in-g-f) (Active ?out-g-f) (Active ?liquid) (Active ?out-l-f))
  Relations ((Q= (Flow-Rate ?out-g-f) (Flow-Rate ?in-g-f))
             (Qprop (Flow-Rate ?out-l-f)
                    (Amount-of (2-C-S (?substance1 ?substance2) liquid ?stage)))))
```

Figure 4.5: Views associated with the equilibrium stages and constant molal overflow assumptions.

```
(defPerspective (Constant-Relative-Volatility-Conditions ?column)
  Individuals ((?column
               :Type Distillation-Column
               :Conditions (Consider (Negligible-Pressure-Drop-in ?column))
                           (Consider (Negligible-Temperature-Drop-in ?column))
                           (Consider (Ideal-System ?substance1 ?substance2 :in ?column))))
  Relations ((Constant-Relative-Volatility ?substance1 ?substance2 :in ?column)))

(defView (Constant-Relative-Volatility-Consequences
          ?column (2-C-S (?substance1 ?substance2) liquid ?stage)
          (2-C-S (?substance1 ?substance2) gas ?stage))
  Individuals ((?column :Type Distillation-Column)
               (?stage :Type Stage
                       :Conditions (Part-of ?column ?stage)
                                   (Consider (Equilibrium-Stage ?stage)))
               (?vapor
                :Conditions ((View-Instance Binary-Mixture) ?vapor)
                            (?vapor MIXTURE (2-C-S (?substance1 ?substance2) gas ?stage))
                            (Constant-Relative-Volatility ?substance1 ?substance2 :in ?column))
               (?liquid
                :Conditions ((View-Instance Binary-Mixture) ?liquid)
                            (?liquid MIXTURE (2-C-S (?substance1 ?substance2) liquid ?stage))))
  QuantityConditions ((Active ?vapor) (Active ?liquid))
  Relations ((Quantity (Constant-Alpha ?column))
             (Q= (Alpha (2-C-S (?substance1 ?substance2) liquid ?stage)
                        (2-C-S (?substance1 ?substance2) gas ?stage))
                 (Constant-Alpha ?column))))
```

Figure 4.6: View and perspective associated with the constant relative volatility assumption.

at each stage (the quantity *Alpha*) equal to some constant (the quantity *Constant-Alpha*).

4. *Equilibrium stages.* We assume that each stage in the column is an equilibrium stage. This means that the vapor and liquid streams that leave each stage are in equilibrium with each other. The view *Binary-Vapor-Liquid-Equilibrium* (Fig. 4.5) describes the relation between the mole fractions of the more volatile component in the vapor and the liquid phase at equilibrium. The equilibrium stages assumption is introduced when the predicate *(Consider (Equilibrium-Stages-in ?column))* is asserted in the scenario.

The conditions under which these assumptions are valid are described in more detail in [35], [22], [36], [24]. This set of idealizations is commonly used in describing the dynamic behavior of staged binary distillation columns for nearly ideal systems operating at 'medium' pressures. With these assumptions our column model captures the effects that changes in the feed composition or in the liquid and gas flow rates have in the behavior of the column. This model is not able to capture the effects of changes in the temperature or the pressure under which the column operates.

Constant molal overflow, constant relative volatility, equilibrium stages and steady-state assumptions are standard simplifications used in solving the design problem for binary columns [35]. These assumptions along with the negligible vapor holdup are also typical assumptions made in simple models for the dynamic behavior of binary columns [36]. All these assumptions are always in force for all the tasks the model is used in (design & simulation).

Fluid mass flows. Figures 4.7, 4.8, 4.9 and 4.10 contain the model fragments described in this section. There are two kinds of fluid mass flows in a distillation column, gas and liquid flows. We represent these flows in a simple general way with the *Gas-Flow* and *Liquid-Flow* processes respectively.

Both flows transfer stuff from a source to a target container. In order for the flow to occur, stuff has to exist in the source container and a gas or liquid path must connect the two containers. Both processes can become active in two ways:

1. When a certain set of quantity conditions holds (e.g. a pressure drop between the source and the target container in the case of gas flow). These conditions are described in the *Gas(Liquid)-Flow-Conditions* view (Figures 4.7 and 4.9).

2. When the occurence of these process is explicitly asserted with the *(Consider (Gas(Liquid)-Flow-Between ?source-mixture ?target-mixture))* predicate in the scenario.

```
;; Gas flow can be active either when there is a pressure drop between stages....
(defView (Gas-Flow-Conditions ?src ?dst ?src-gas)
  Individuals ((?src :Type Container)
               (?dst :Type Container
                     :Conditions (Gas-Path ?src ?dst))
               (?src-gas :Conditions ((View-Instance Binary-Mixture) ?src-gas)
                                     (?src-gas MIXTURE (2-C-S (?substance1 ?substance2) gas ?src))
                                     (More-Volatile ?substance1 :Than ?substance2))
               (?dst-gas :Bind (2-C-S (?substance1 ?substance2) gas ?dst)))
  QuantityConditions ((Active ?src-gas)
                      (greater-than (A (Pressure ?src)) (A (Pressure ?dst))))
  Relations
  ((Satisfied-Gas-Flow-Conditions-Between (2-C-S (?substance1 ?substance2) gas ?src) ?dst-gas)))

;; ... or when we explicilty assert that it is active in the scenario.
(defPerspective (Gas-Flow-Assumption ?src ?dst ?src-gas)
  Individuals ((?src :Type Container)
               (?dst :Type Container
                     :Conditions (Gas-Path ?src ?dst))
               (?src-gas
                :Conditions
                ((View-Instance Binary-Mixture) ?src-gas)
                (?src-gas MIXTURE (2-C-S (?substance1 ?substance2) gas ?src))
                (More-Volatile ?substance1 :Than ?substance2)
                (Consider (Gas-Flow-Between (2-C-S (?substance1 ?substance2) gas ?src)
                                            (2-C-S (?substance1 ?substance2) gas ?dst))))
               (?dst-gas :Bind (2-C-S (?substance1 ?substance2) gas ?dst)))
  Relations
  ((Satisfied-Gas-Flow-Conditions-Between (2-C-S (?substance1 ?substance2) gas ?src) ?dst-gas)))

(DefClosed-Predicate Satisfied-Gas-Flow-Conditions-Between)
```

Figure 4.7: View and perspective associated with gas flow.

```
(defProcess (Gas-Flow ?src ?dst ?src-gas ?dst-gas)
  Individuals ((?src :Type Container)
               (?dst :Type Container
                     :Conditions (Gas-Path ?src ?dst))
               (?src-gas
                :Conditions
                ((View-Instance Binary-Mixture) ?src-gas)
                (?src-gas MIXTURE (2-C-S (?substance1 ?substance2) gas ?src))
                (More-Volatile ?substance1 :Than ?substance2)
                (Satisfied-Gas-Flow-Conditions-Between
                 (2-C-S (?substance1 ?substance2) gas ?src)
                 (2-C-S (?substance1 ?substance2) gas ?dst)))
               (?dst-gas :Bind (2-C-S (?substance1 ?substance2) gas ?dst)))
  QuantityConditions ((Active ?src-gas))
  Relations ((Introduces ?dst-gas)
             (Contained-Binary-Gas-Mixture ?dst-gas)
             (Positive-Quantity (Flow-Rate ?self))
             (Quantity (Mass-Flow-of ?substance1
                                     (2-C-S (?substance1 ?substance2) gas ?src)
                                     ?self))
             (Q= (Mass-Flow-of ?substance1
                               (2-C-S (?substance1 ?substance2) gas ?src)
                               ?self)
                  (* (Flow-Rate ?self)
                     (Mole-Fraction ?substance1 (2-C-S (?substance1 ?substance2) gas ?src)))))
  Influences ((I+ (Amount-of ?dst-gas) (A (Flow-Rate ?self)))
              (I- (Amount-of (2-C-S (?substance1 ?substance2) gas ?src)) (A (Flow-Rate ?self)))
              (I- (Amount-of-in ?substance1 (2-C-S (?substance1 ?substance2) gas ?src))
                  (A (Mass-Flow-of ?substance1
                                   (2-C-S (?substance1 ?substance2) gas ?src)
                                   ?self)))
              (I+ (Amount-of-in ?substance1 ?dst-gas)
                  (A (Mass-Flow-of ?substance1
                                   (2-C-S (?substance1 ?substance2) gas ?src)
                                   ?self)))))
```

Figure 4.8: The gas flow process.

```
(defView (Liquid-Flow-Conditions ?src ?dst ?src-lqd)
  Individuals ((?src :Type Container)
               (?dst :Type Container
                     :Conditions (Liquid-Path ?src ?dst))
               (?src-lqd
                :Conditions ((View-Instance Binary-Mixture) ?src-lqd)
                           (?src-lqd MIXTURE (2-C-S (?substance1 ?substance2) liquid ?src))
                           (More-Volatile ?substance1 :Than ?substance2))
               (?dst-lqd :Bind (2-C-S (?substance1 ?substance2) liquid ?dst)))
  QuantityConditions ((Active ?src-lqd)
                      (greater-than (A (Pressure ?src)) (A (Pressure ?dst))))
  Relations
  ((Satisfied-Liquid-Flow-Conditions-Between
    (2-C-S (?substance1 ?substance2) liquid ?src) ?dst-lqd)))

(defPerspective (Liquid-Flow-Assumption ?src ?dst ?src-lqd)
  Individuals ((?src :Type Container)
               (?dst :Type Container
                     :Conditions (Liquid-Path ?src ?dst))
               (?src-lqd
                :Conditions
                ((View-Instance Binary-Mixture) ?src-lqd)
                (?src-lqd MIXTURE (2-C-S (?substance1 ?substance2) liquid ?src))
                (More-Volatile ?substance1 :Than ?substance2)
                (Consider (Liquid-Flow-Between
                          (2-C-S (?substance1 ?substance2) liquid ?src)
                          (2-C-S (?substance1 ?substance2) liquid ?dst))))
               (?dst-lqd :Bind (2-C-S (?substance1 ?substance2) liquid ?dst)))
  Relations
  ((Satisfied-Liquid-Flow-Conditions-Between
    (2-C-S (?substance1 ?substance2) liquid ?src) ?dst-lqd)))

(DefClosed-Predicate Satisfied-Liquid-Flow-Conditions-Between)
```

Figure 4.9: View and perspective associated with liquid flow.

```
(defProcess (Liquid-Flow ?src ?dst ?src-lqd ?dst-lqd)
  Individuals ((?src :Type Container)
               (?dst :Type Container
                     :Conditions (Liquid-Path ?src ?dst))
               (?src-lqd
                :Conditions
                ((View-Instance Binary-Mixture) ?src-lqd)
                (?src-lqd MIXTURE (2-C-S (?substance1 ?substance2) liquid ?src))
                (More-Volatile ?substance1 :Than ?substance2)
                (Satisfied-Liquid-Flow-Conditions-Between
                 (2-C-S (?substance1 ?substance2) liquid ?src)
                 (2-C-S (?substance1 ?substance2) liquid ?dst)))
               (?dst-lqd :Bind (2-C-S (?substance1 ?substance2) liquid ?dst)))
  QuantityConditions ((Active ?src-lqd))
  Relations ((Introduces ?dst-lqd)
             (Contained-Binary-Liquid-Mixture ?dst-lqd)
             (Positive-Quantity (Flow-Rate ?self))
             (Positive-Quantity (Initial-Liquid-Flow-Rate ?self))
             (Quantity (Mass-Flow-of ?substance1
                                     (2-C-S (?substance1 ?substance2) liquid ?src)
                                     ?self))
             (Q= (Mass-Flow-of ?substance1
                               (2-C-S (?substance1 ?substance2) liquid ?src)
                               ?self)
                 (* (Flow-Rate ?self)
                    (Mole-Fraction
                     ?substance1 (2-C-S (?substance1 ?substance2) liquid ?src)))))
  Influences ((I+ (Amount-of ?dst-lqd) (A (Flow-Rate ?self)))
              (I- (Amount-of (2-C-S (?substance1 ?substance2) liquid ?src)) (A (Flow-Rate ?self)))
              (I- (Amount-of-in ?substance1 (2-C-S (?substance1 ?substance2) liquid ?src))
                  (A (Mass-Flow-of ?substance1
                                   (2-C-S (?substance1 ?substance2) liquid ?src)
                                   ?self)))
              (I+ (Amount-of-in ?substance1 ?dst-lqd)
                  (A (Mass-Flow-of ?substance1
                                   (2-C-S (?substance1 ?substance2) liquid ?src)
                                   ?self)))))
```

Figure 4.10: The liquid flow process.

Currently, flows are activated using the second option. There are two reasons for this:

1. The liquid flows down the column due to gravity and not because of some pressure difference between the stages. Simple column models such as this one do not consider the effects of gravity in liquid flow.

2. As the definition of the *Constant-Relative-Volatility* view indicates (Fig. 4.6), the constant relative volatility assumption is valid if there are no significant pressure changes throughout the column. This is possible if the pressure drop for the vapor at each stage is negligible. Asserting the existence of a pressure drop between successive stages would activate the Gas-Flow process but it would contradict the constant relative volatility assumption.

The *DefClosed-Predicate* predicate (Figs 4.9 and 4.7) is a form of negation by failure, causing the relation specified as its argument not to be believed unless it is explicitly known to be true.

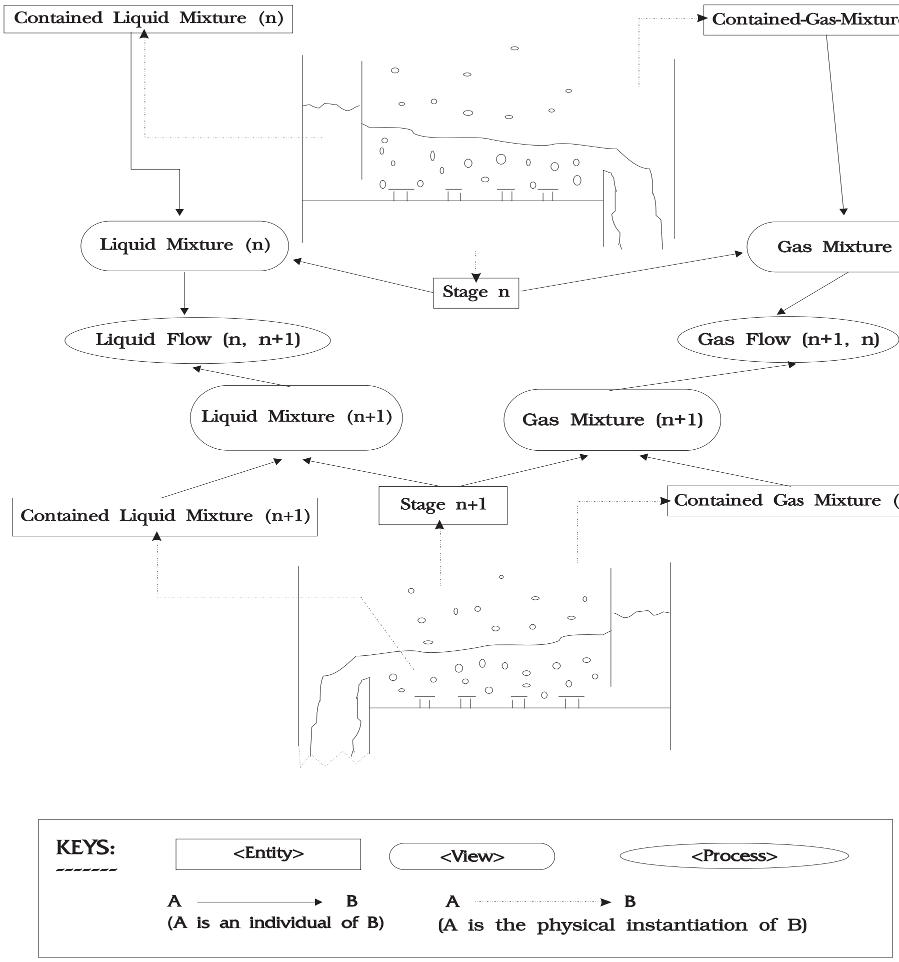
Each flow process introduces two quantities in its Relations field (Fig. 4.8). The *Flow-Rate* quantity corresponds to the flow rate of the total mass for each phase. The *Mass-Flow-of* quantity provides a way of refering to the flow rate for each of the substances at each phase. The *Introduces* predicate in the relations field for both processes declares its argument to be an individual which physically exists.

There are two direct influences per flow process. The *Amount-of* quantity for each phase is directly influenced by the flow rate. The *Amount-of-in* quantity for each component is directly influenced by the mass flow for this component.

Normally, there are two gas and two liquid flows active for each stage (Fig. 4.11). Vapor is rising from the stage immediately below the current stage, vapor is flowing from the current stage to the one immediately above it, liquid is flowing in the current stage from the one immediately above it and finally liquid is flowing from the current stage to the stage immediately below it.

Phase Transition Processes. Figures 4.12, 4.13, 4.14 and 4.15 contain the model fragments described in this section. The constant molal overflow assumption allows us to ignore the condensation and vaporization phenomena for each stage in the column. Under this assumption, the condensation rate for the vapor at each stage is equal to the vaporization rate for the liquid. Consequently, the overall vapor and liquid stuff at each stage (the total holdup for this stage) remains unchanged, since the effects of condensation and vaporization cancel each other. However, we still have to model the condensation process for the condenser at the

Figure 4.11: Model fragments that decribe the physical activity between two succesive stages in a column.



```
(defView (Vaporization-Conditions ?can ?hf ?l-mix)
  Individuals ((?can :Type Stage)
               (?l-mix
                :Conditions ((View-Instance Binary-Mixture) ?l-mix)
                            (?l-mix MIXTURE (2-C-S (?substance1 ?substance2) liquid ?can))
                            (More-Volatile ?substance1 :Than ?substance2))
               (?hf :Conditions ((Process-Instance Heat-Flow) ?hf)
                                (?hf DST ?can)))
  QuantityConditions
  ((Active ?hf)
   (Active ?l-mix)
   (not (greater-than (A (TBubble (2-C-S (?substance1 ?substance2) liquid ?can)))
                      (A (Temperature (2-C-S (?substance1 ?substance2) liquid ?can))))))
  Relations ((Satisfied-Vaporization-Conditions-For (2-C-S (?substance1 ?substance2) liquid ?can))))

(defView (Vaporization-Assumption ?can ?l-mix)
  Individuals ((?can :Type Container)
               (?l-mix
                :Conditions
                ((View-Instance Binary-Mixture) ?l-mix)
                (?l-mix MIXTURE (2-C-S (?substance1 ?substance2) liquid ?can))
                (More-Volatile ?substance1 :Than ?substance2)
                (Consider (Vaporize (2-C-S (?substance1 ?substance2) liquid ?can)))))
  Relations ((Satisfied-Vaporization-Conditions-For (2-C-S (?substance1 ?substance2) liquid ?can))))

(defClosed-Predicate Satisfied-Vaporization-Conditions-For)
```

Figure 4.12: Views associated with the vaporization process in the binary distillation model.

```
(defProcess (Vaporization ?l-mix ?g-mix ?can)
   Individuals ((?can :Type Container)
                (?l-mix
                 :Conditions ((View-Instance Binary-Mixture) ?l-mix)
                             (?l-mix MIXTURE (2-C-S (?substance1 ?substance2) liquid ?can))
                             (More-Volatile ?substance1 :Than ?substance2)
                             (Satisfied-Vaporization-Conditions-For
                              (2-C-S (?substance1 ?substance2) liquid ?can)))
                (?g-mix :Bind (2-C-S (?substance1 ?substance2) gas ?can)))
   QuantityConditions ((Active ?l-mix))
   Relations ((Positive-Quantity (Vaporization-Rate ?self))
              (Introduces ?g-mix)
              (Contained-Binary-Mixture ?g-mix)
              (Quantity (Vaporized-Mass-of
                          ?substance1 (2-C-S (?substance1 ?substance2) liquid ?can) ?self))
              (Q= (Vaporized-Mass-of
                   ?substance1 (2-C-S (?substance1 ?substance2) liquid ?can) ?self)
                  (* (Vaporization-Rate ?self)
                     (Mole-Fraction ?substance1 (2-C-S (?substance1 ?substance2) gas ?can)))))
   Influences ((I+ (Amount-of (2-C-S (?substance1 ?substance2) gas ?can))
                   (A (Vaporization-Rate ?self)))
               (I- (Amount-of (2-C-S (?substance1 ?substance2) liquid ?can))
                   (A (Vaporization-Rate ?self)))
               (I- (Amount-of-in ?substance1 (2-C-S (?substance1 ?substance2) liquid ?can))
                   (A (Vaporized-Mass-of
                       ?substance1 (2-C-S (?substance1 ?substance2) liquid ?can) ?self)))
               (I+ (Amount-of-in ?substance1 (2-C-S (?substance1 ?substance2) gas ?can))
                   (A (Vaporized-Mass-of
                       ?substance1 (2-C-S (?substance1 ?substance2) liquid ?can) ?self)))))
```

Figure 4.13: The vaporization process in the binary distillation model.

```
(defView (Condensation-Conditions ?can ?hf ?g-mix)
   Individuals ((?can :Type Stage)
                (?g-mix
                 :Conditions ((View-Instance Binary-Mixture) ?g-mix)
                             (?g-mix MIXTURE (2-C-S (?substance1 ?substance2) gas ?can))
                             (More-Volatile ?substance1 :than ?substance2))
                (?hf :Conditions ((Process-Instance Heat-Flow) ?hf)
                                 (?hf SRC ?can)))
   QuantityConditions
   ((Active ?g-mix)
    (Active ?hf)
    (not (less-than (A (Tdew (2-C-S (?substance1 ?substance2) gas ?can)))
                    (A (Temperature (2-C-S (?substance1 ?substance2) gas ?can))))))
   Relations ((Satisfied-Condensation-Conditions-For (2-C-S (?substance1 ?substance2) gas ?can))))

(defPerspective (Condensation-Assumption ?can ?g-mix)
  Individuals ((?can :Type Container)
               (?g-mix
                :Conditions ((View-Instance Binary-Mixture) ?g-mix)
                            (?g-mix MIXTURE (2-C-S (?substance1 ?substance2) gas ?can))
                            (More-Volatile ?substance1 :Than ?substance2)
                            (Consider (Condense (2-C-S (?substance1 ?substance2) gas ?can)))))
  Relations ((Satisfied-Condensation-Conditions-For (2-C-S (?substance1 ?substance2) gas ?can))))

(defClosed-Predicate Satisfied-Condensation-Conditions-For)
```

Figure 4.14: View and perspective associated with the condensation process in the binary distillation model.

```
(defProcess (Condensation ?g-mix ?l-mix ?can)
   Individuals ((?can :Type Container)
                (?g-mix
                 :Conditions ((View-Instance Binary-Mixture) ?g-mix)
                             (?g-mix MIXTURE (2-C-S (?substance1 ?substance2) gas ?can))
                             (More-Volatile ?substance1 :than ?substance2)
                             (Satisfied-Condensation-Conditions-For
                              (2-C-S (?substance1 ?substance2) gas ?can)))
                (?l-mix :Bind (2-C-S (?substance1 ?substance2) liquid ?can)))
   QuantityConditions ((Active ?g-mix))
   Relations ((Positive-Quantity (Condensation-Rate ?self))
              (Introduces ?l-mix)
              (Contained-Binary-Mixture ?l-mix)
              (Quantity
               (Condensed-Mass-of ?substance1 (2-C-S (?substance1 ?substance2) gas ?can) ?self))
              (Q= (Condensed-Mass-of ?substance1 (2-C-S (?substance1 ?substance2) gas ?can) ?self)
                  (* (Condensation-Rate ?self)
                     (Mole-Fraction ?substance1 (2-C-S (?substance1 ?substance2) liquid ?can)))))
   Influences ((I+ (Amount-of ?l-mix) (A (Condensation-Rate ?self)))
               (I- (Amount-of (2-C-S (?substance1 ?substance2) gas ?can))
                   (A (Condensation-Rate ?self)))
               (I+ (Amount-of-in ?substance1 (2-C-S (?substance1 ?substance2) liquid ?can))
                   (A (Condensed-Mass-of
                       ?substance1 (2-C-S (?substance1 ?substance2) gas ?can) ?self)))
               (I- (Amount-of-in ?substance1 (2-C-S (?substance1 ?substance2) gas ?can))
                   (A (Condensed-Mass-of
                       ?substance1 (2-C-S (?substance1 ?substance2) gas ?can) ?self)))))
```

Figure 4.15: The condensation process in the binary distillation model.

top of the column and the vaporization process for the reboiler at the bottom of the column.

We describe only the vaporization process since the representations for the two phase transition processes are symmetrical. Vaporization occurs when there exists a binary liquid mixture in a container for which the predicate *Satisfied-Vaporization-Conditions-For* holds[3]. In an analogous way with the flow processes, this predicate holds either when a set of quantity conditions are active[4] or when the occurence of vaporization is assumed in the scenario. Currently, both phase transition processes are activated with the second option, since the numerical model that we use ignores the heat exchange phenomena in the reboiler and the condenser

When vaporization is active, it introduces a binary gas mixture in the container (if one does not already exist) and it directly influences the amounts of the vapor and liquid mixtures and the amount of the volatile component in each one of these mixtures via a positive vaporization rate. We use a very simple general model for vaporization which is consistent with the conservation of mass but ignores the enthalpy changes in the vapor and the liquid associated with the process.

In an analogous way with the flow processes, the *Vaporization* process introduces two quantities in its Relations field. The *Vaporization-Rate* refers to the rate by which both components in the binary mixture evaporate. The *Vaporized-Mass-of* quantity refers to the rate of evaporation of each individual component in the mixture. In the Influences field for the process, the *Amount-of* quantity for each phase is directly influenced by the *Vaporization-Rate*, while the *Amount-of-in* quantity for each component in each phase is directly influenced by the *Vaporized-Mass-of* quantity. A similar situation holds for the *Condensation* process.

Condenser, Reboiler & Feed Stage. Figure 4.16 contains the processes described in this section. There are two choices for the condenser in a distillation column. It can either be a partial condenser condensing part of the vapor that ends up at the top of the column or a total condenser in which all of the incoming vapor is condensed into liquid. An analogous situation holds for the reboiler at the bottom of the column. Depending on whether it vaporizes part or all of the liquid that reaches the bottom of the column it can either be a partial or a total reboiler. The predicates *(Consider (Total(Partial)-Condenser(Reboiler) ?container))* in the scenario specify the type of condenser (reboiler) for the column.

---

[3]See the definitions for the *Vaporization-Conditions* and the *Vaporization-Assumption* views in Fig. 4.12.

[4]E.g., when the bubble point for the liquid mixture is not greater than its current temperature and there is a heat flow process supplying heat to the container.

```
(defProcess (Continuous-Binary-Feed-Flow ?feed ?stage)
  Individuals ((?feed
               :Conditions
               ((View-Instance Binary-Mixture) ?feed)
               (?feed MIXTURE (2-C-S (?substance1 ?substance2) ?phase ?stage))
               (Binary-Mixture-Feed-in-Stage (2-C-S (?substance1 ?substance2) ?phase ?stage) ?stage)
               (More-Volatile ?substance1 :Than ?substance2))
              (?stage :Type Stage))
  QuantityConditions ((Active ?feed))
  Relations
  ((Non-Negative-Quantity (Feed-Rate ?self))
   (Positive-Quantity
    (Feed-Composition ?substance1 (2-C-S (?substance1 ?substance2) ?phase ?stage)))
   (Quantity (Feed-Amount-of ?substance1 (2-C-S (?substance1 ?substance2) ?phase ?stage) ?self))
   (Q= (Feed-Amount-of ?substance1 (2-C-S (?substance1 ?substance2) ?phase ?stage) ?self)
       (* (Feed-Composition ?substance1 (2-C-S (?substance1 ?substance2) ?phase ?stage))
          (Feed-Rate ?self))))
  Influences ((I+ (Amount-of (2-C-S (?substance1 ?substance2) ?phase ?stage)) (A (Feed-Rate ?self)))
             (I+ (Amount-of-in ?substance1 (2-C-S (?substance1 ?substance2) ?phase ?stage))
                 (A (Feed-Amount-of
                     ?substance1 (2-C-S (?substance1 ?substance2) ?phase ?stage) ?self)))))

(defProcess (Continuous-Binary-Mixture-Product-Flow ?product ?stage)
  Individuals ((?product
               :Conditions
               ((View-Instance Binary-Mixture) ?product)
               (?product MIXTURE (2-C-S (?substance1 ?substance2) ?phase ?stage))
               (More-Volatile ?substance1 :Than ?substance2)
               (Recover-Binary-Mixture-Product
                (2-C-S (?substance1 ?substance2) ?phase ?stage) ?stage))
              (?stage :Type Stage))
  QuantityConditions ((Active ?product))
  Relations
  ((Non-Negative-Quantity (Product-Flow ?self))
   (Quantity (Product-Amount-of ?substance1 (2-C-S (?substance1 ?substance2) ?phase ?stage) ?self))
   (Qprop (Product-Amount-of ?substance1 (2-C-S (?substance1 ?substance2) ?phase ?stage) ?self)
          (Mole-Fraction ?substance1 (2-C-S (?substance1 ?substance2) ?phase ?stage)))
   (Qprop (Product-Amount-of ?substance1 (2-C-S (?substance1 ?substance2) ?phase ?stage) ?self)
          (Product-Flow ?self)))
  Influences ((I- (Amount-of (2-C-S (?substance1 ?substance2) ?phase ?stage))
                 (A (Product-Flow ?self)))
             (I- (Amount-of-in ?substance1 (2-C-S (?substance1 ?substance2) ?phase ?stage))
                 (A (Product-Amount-of
                     ?substance1 (2-C-S (?substance1 ?substance2) ?phase ?stage) ?self))))
```

Figure 4.16: Feed and product flow processes for the binary distillation model.

Usually there are two product streams coming out of a distillation tower. The stream that we recover from the top is called the *distillate.* The stream at the bottom of the tower is called the *bottom product.* A *Continuous-Binary-Mixture-Product-Flow* process is associated with each product stream. Each product flow is active when the mixture specified in the *(Recover-Binary-Mixture-Product ?mixture ?container)* predicate exists.

The process description is analogous to the one used for describing fluid mass flows with the difference that it does not take into account conservation of mass, since we do not know the destination of the product. This information would be available in more complicated process plant models containing sequences of columns. In this case, views or perspectives would be used to relate the various feed and product flows.

The predicates *(Feed-Stage ?stage)* and *(Binary-Mixture-Feed-in-Stage ?mixture ?stage)* in the scenario specify the location and the type of the feed streams entering the distillation process. For each feed stream we define a *Continuous-Binary-Feed-Flow* process. The description for this process is analogous to the product flow models. The quantity *Feed-Composition* inside the feed flow corresponds to the mole fraction of the volatile component in the feed stream. The Feed-Composition is usually different from the mole fraction of the volatile component in the feed stage.

4.3.3 Steady-State Model of an Ideal Binary Distillation Column

Most of the physical knowledge for binary distillation design describes the steady-state operation of columns. Steady-state operating assumptions are common in the early stages of chemical process design since they greatly simplify the numerical models. In the case of qualitative models this means more constrained and therefore more tractable models.

Steady-state analysis is enforced with the *(Consider (Steady-State-in ?column))* predicate in the scenario. As a result, the *Steady-State* perspective (Fig. 4.17) is activated pinning down the derivative of every quantity in the model to zero. The *Steady-State-Column-Design-Features* view (Fig. 4.18) introduces a set of relevant features for binary column design. These include the specifications of the input stream to the process (the *Feed-Flow* and *Feed-Fraction* quantities), the number of stages in the column (then *Num-of-Stages* quantity), the reflux (i.e. the flow rate of the distillate product that is recycled back to the column in order to increase the efficiency of the process) and the desired recoveries for the dis-

```
(defPerspective (Steady-State ?column ?stage ?quant)
  Individuals ((?column :Type Distillation-Column
                        :Conditions (Consider (Steady-State-in ?column)))
               (?stage :Type Stage
                       :Conditions (Part-of ?column ?stage))
               (?quant :Type Quantity
                       :Form (?quant-name . ?args)
                       :Test (contained-in? ?stage ?args)))
  Relations ((equal-to (D ?quant) ZERO)))
```

Figure 4.17: Enforce the steady-state assumption.

tillation products. The *Steady-State-Conditions-for-Feed-Stage* view (Fig. 4.19) establishes a set of relations between the mole fractions of the input (feed) to the process and the mole fractions of the distillate and bottom products. The *Steady-State-Conditions-for-Stage-Gas-Flows* and *Steady-State-Liquid-Flows* views (Fig. 4.20) relate the rates of the incoming and outgoing gas and liquid flows at each stage in the column. In addition, the model contains a set of views that relate the flow rates used in the reboiler and condenser parts of a column (see Figure 4.21).

4.4 Separation Systems for Multicomponent Mixtures

4.4.1 Overview

This section describes in detail the qualitative model for multicomponent distillation columns. The model is used to compute cost estimates for possible flowsheets during the conceptual design of separation systems. At this early stage of design, these cost estimates are rough approximations for pruning the number of design choices [13]. The model does not describe the dynamic behavior of columns for multicomponent mixtures.

The qualitative domain theory consists of four objects; separation systems (a sequence of separation units (e.g. distillation columns)), columns, mixtures and substances. The model computes the properties for each object in two sets of physical conditions (pressure and temperature); reference and operating conditions. At each object level there are a series of operations taking place during design. These are:

```
(defPerspective (Steady-State-Column-Design-Features
                 ?column ?c-stage ?c-phase ?r-stage ?r-phase ?distillate-flow
                 ?bproduct-flow ?substance1 ?substance2)
  Individuals ((?column :Type Distillation-Column
                        :Conditions (Consider (Steady-State-in ?column)))
               (?distillate
                :Type Contained-Binary-Mixture
                :Form (2-C-S (?substance1 ?substance2) ?c-phase ?c-stage)
                :Conditions (More-Volatile ?substance1 :Than ?substance2)
                            (Recover-Binary-Mixture-Product ?distillate ?c-stage)
                            (Condenser-Stage ?c-stage))
               (?bproduct
                :Type Contained-Binary-Mixture
                :Form (2-C-S (?substance1 ?substance2) ?r-phase ?r-stage)
                :Conditions (Recover-Binary-Mixture-Product ?bproduct ?r-stage)
                            (Reboiler-Stage ?r-stage))
               (?distillate-flow
                :Conditions
                ((Process-Instance Continuous-Binary-Mixture-Product-Flow) ?distillate-flow)
                (?distillate-flow STAGE ?c-stage)
                (Active ?distillate-flow))
               (?bproduct-flow
                :Conditions
                ((Process-Instance Continuous-Binary-Mixture-Product-Flow) ?bproduct-flow)
                (?bproduct-flow STAGE ?r-stage)
                (Active ?bproduct-flow)))
  Relations ((Non-Negative-Quantity (Feed-Flow ?column))
             (Non-Negative-Quantity (Feed-Fraction ?substance1 ?column))
             (Non-Negative-Quantity (Num-of-Stages ?column))
             (Non-Negative-Quantity (Reflux ?column))
             (Non-Negative-Quantity (Fraction-Recovery ?substance1 ?c-stage ?column))
             (Non-Negative-Quantity (Fraction-Recovery ?substance1 ?r-stage ?column))
             (Qprop (Mole-Fraction ?substance1
                                   (2-C-S (?substance1 ?substance2) ?c-phase ?c-stage))
                    (Num-of-Stages ?column))
             (Qprop- (Mole-Fraction ?substance1
                                    (2-C-S (?substance1 ?substance2) ?r-phase ?r-stage))
                     (Num-of-Stages ?column))
             (Qprop (Product-Flow ?distillate-flow) (Feed-Flow ?column))
             (Qprop (Product-Flow ?bproduct-flow) (Feed-Flow ?column))))
```

Figure 4.18: A set of relevant features for binary distillation design.

```
(defView (Steady-State-Conditions-for-Feed-Stage
         ?column ?stage ?in-l-f ?out-l-f ?distillate-flow ?bproduct-flow ?feed-flow ?substance1
         ?substance2 ?phase ?c-phase ?bproduct-phase ?c-stage ?r-stage)
  Individuals ((?column :Type Distillation-Column
                        :Conditions (Consider (Steady-State-in ?column)))
               (?stage :Type Stage
                       :Conditions (Part-of ?column ?stage)
                                   (Feed-Stage ?stage))
               (?feed-flow
                :Conditions
                ((Process-Instance Continuous-Binary-Feed-Flow) ?feed-flow)
                (?feed-flow STAGE ?stage)
                (Binary-Mixture-Feed-in-Stage
                 (2-C-S (?substance1 ?substance2) ?phase ?stage) ?stage))
               (?in-l-f :Conditions ((Process-Instance Liquid-Flow) ?in-l-f)
                                     (?in-l-f DST ?stage))
               (?out-l-f :Conditions ((Process-Instance Liquid-Flow) ?out-l-f)
                                      (?out-l-f SRC ?stage))
               (?distillate-flow
                :Conditions
                ((Process-Instance Continuous-Binary-Mixture-Product-Flow) ?distillate-flow)
                (?distillate-flow STAGE ?c-stage)
                (Condenser-Stage ?c-stage)
                (Recover-Binary-Mixture-Product
                 (2-C-S (?substance1 ?substance2) ?c-phase ?c-stage) ?c-stage))
               (?bproduct-flow
                :Conditions
                ((Process-Instance Continuous-Binary-Mixture-Product-Flow) ?bproduct-flow)
                (?bproduct-flow STAGE ?r-stage)
                (Reboiler-Stage ?r-stage)
                (Recover-Binary-Mixture-Product
                 (2-C-S (?substance1 ?substance2) ?bproduct-phase ?r-stage) ?r-stage)))
  QuantityConditions ((Active ?feed-flow) (Active ?in-l-f) (Active ?out-l-f)
                      (Active ?distillate-flow) (Active ?bproduct-flow))
  Relations
  ((Qprop (Flow-Rate ?out-l-f) (Feed-Rate ?feed-flow))
   (Qprop (Flow-Rate ?out-l-f) (Flow-Rate ?in-l-f))
   (Qprop (Mole-Fraction ?substance1 (2-C-S (?substance1 ?substance2) ?c-phase ?c-stage))
          (Feed-Composition ?substance1 (2-C-S (?substance1 ?substance2) ?phase ?stage)))
   (Qprop (Mole-Fraction ?substance1 (2-C-S (?substance1 ?substance2) ?bproduct-phase ?r-stage))
          (Feed-Composition ?substance1 (2-C-S (?substance1 ?substance2) ?phase ?stage)))
   (Equal-to (A (Feed-Flow ?column)) (A (Feed-Rate ?feed-flow)))
   (Equal-to (A (Feed-Fraction ?substance1 ?column))
             (A (Feed-Composition ?substance1 (2-C-S (?substance1 ?substance2) ?phase ?stage))))))
```

Figure 4.19: The relations between the feed and product compositions under a steady-state asssumption.

```
(defView (Steady-State-Conditions-for-Stage-Gas-Flows ?stage ?in-g-f ?out-g-f)
  Individuals ((?column
                :Type Distillation-Column
                :Conditions (Part-of ?column ?stage)
                            (Consider (Constant-Molal-Overflow-in ?column))
                            (Consider (Neglect Vapor-HoldUp-in ?column))
                            (Consider (Steady-State-in ?column)))
               (?stage :Type Stage
                       :Conditions (not (Feed-Stage ?stage)))
               (?in-g-f :Conditions ((Process-Instance Gas-Flow) ?in-g-f)
                                    (?in-g-f DST ?stage))
               (?out-g-f :Conditions ((Process-Instance Gas-Flow) ?out-g-f)
                                     (?out-g-f SRC ?stage)))
  QuantityConditions ((Active ?in-g-f) (Active ?out-g-f))
  Relations ((equal-to (A (Flow-Rate ?out-g-f)) (A (Flow-Rate ?in-g-f)))))

(defView (Steady-State-Liquid-Flows ?stage ?in-l-f ?out-l-f)
  Individuals ((?column
                :Type Distillation-Column
                :Conditions (Part-of ?column ?stage)
                            (Consider (Steady-State-in ?column))
                            (Consider (Constant-Molal-Overflow-in ?column))
                            (Consider (Neglect Vapor-HoldUp-in ?column)))
               (?stage :Type Stage
                       :Conditions (not (Feed-Stage ?stage)))
               (?in-l-f :Conditions ((Process-Instance Liquid-Flow) ?in-l-f)
                                    (?in-l-f DST ?stage))
               (?out-l-f :Conditions ((Process-Instance Liquid-Flow) ?out-l-f)
                                     (?out-l-f SRC ?stage)))
  QuantityConditions ((Active ?in-l-f) (Active ?out-l-f))
  Relations ((Equal-to (A (Flow-Rate ?out-l-f)) (A (Flow-Rate ?in-l-f)))))
```

Figure 4.20: The relation between incoming and outgoing gas and liquid flow rates between successive stages in a column under steady-state conditions.

```
(defView (Total-Condenser-Steady-State-Conditions
         ?column ?stage ?l-mix ?g-mix ?product-flow ?reflux-flow ?in-g-f
         (2-C-S (?substance1 ?substance2) liquid ?stage)
         (2-C-S (?substance1 ?substance2) gas ?stage))
  Individuals ((?stage :Type Stage
                       :Conditions (Condenser-Stage ?stage))
               (?column :Type Distillation-Column
                        :Conditions
                        (Consider (Total-Condenser ?column))
                        (Part-of ?column ?stage)
                        (Consider (Steady-State-in ?column)))
               (?l-mix
                :Conditions ((View-Instance Binary-Mixture) ?l-mix)
                            (?l-mix MIXTURE (2-C-S (?substance1 ?substance2) liquid ?stage)))
               (?g-mix
                :Conditions ((View-Instance Gas-Mixture) ?g-mix)
                            (?g-mix MIXTURE (2-C-S (?substance1 ?substance2) gas ?stage)))
               (?in-g-f :Conditions ((Process-Instance Gas-Flow) ?in-g-f)
                                    (?in-g-f DST ?stage))
               (?product-flow
                :Conditions
                 ((Process-Instance Continuous-Binary-Mixture-Product-Flow) ?product-flow)
                 (?product-flow STAGE ?stage))
               (?reflux-flow
                :Conditions ((Process-Instance Liquid-Flow) ?reflux-flow)
                            (?reflux-flow SRC ?stage)))
  QuantityConditions ((Active ?l-mix) (Active ?g-mix) (Active ?in-g-f)
                      (Active ?product-flow) (Active ?reflux-flow))
  Relations ((Equal-to (A (Mole-Fraction
                           ?substance1 (2-C-S (?substance1 ?substance2) liquid ?stage)))
                       (A (Mole-Fraction
                           ?substance1 (2-C-S (?substance1 ?substance2) gas ?stage))))
             (Equal-to (A (Reflux ?column)) (A (Flow-Rate ?reflux-flow)))))
```

Figure 4.21: The relation between incoming and outgoing gas and liquid flow rates under steady-state conditions in a total condenser. Similar views describe these relations in the case of a partial condenser and a partial or total reboiler.

```
(defentity (Substance ?substance)
  (Positive-Quantity (Critical-Pressure ?substance))
  (Non-Negative-Quantity (Alpha-1 ?substance))
  (Non-Negative-Quantity (Alpha-2 ?substance))
  (Non-Negative-Quantity (Alpha-3 ?substance))
  (Positive-Quantity (Liquid-Volume-Constant ?substance))
  (Quantity (K-Ideal-Reference-Temperature ?substance))
  (Non-Negative-Quantity (Critical-Temperature ?substance)))

(defentity (Multicomponent-Mixture (M-C-S ?components ?phase ?stage))
   (Quantity (Temperature (M-C-S ?components ?phase ?stage)))
   (Quantity (Pressure (M-C-S ?components ?phase ?stage))))

(defentity (Distillation-Column ?column)
  (Positive-Quantity (Column-Diameter ?column))
  (Positive-Quantity (Column-Height ?column))
  (Positive-Quantity (Reference-Temperature ?column))
  (Positive-Quantity (Reference-Pressure ?column))
  (Positive-Quantity (Operating-Pressure ?column))
  (Positive-Quantity (Operating-Temperature ?column))
  (Positive-Quantity (Installation-Cost ?column))
  (Positive-Quantity (Operating-Reflux-Estimate ?column)))

(defentity (Environment ?env)
  (Positive-Quantity (Atmospheric-Pressure ?env))
  (Positive-Quantity (Gas-Constant ?env)))

(defentity (Separation-System ?system)
  (Non-Negative-Quantity (Total-Cost ?system))
  (Positive-Quantity (Reference-Temperature ?system))
  (Positive-Quantity (Reference-Pressure ?system)))
```

Figure 4.22: Defining the objects in the multicomponent distillation model.

- Operations at the Substance Level.
  1. Establish physical properties at reference and operating conditions (e.g. equilibrium ratios, vapor pressures and specific parameters for computing them).
  2. Introduce design features (e.g. product recovery).
- Operations at the Mixture Level.
  1. Introduce relative volatilities at reference and operating conditions.
  2. Establish feed condition properties (e.g. temperature, pressure and flow rate).
- Operations at the Column Level.
  1. Generate the separation alternatives.
  2. Compute the distillate and bottom products for the design alternatives.
  3. Introduce operating and design features (e.g. diameter, height, installation cost, reflux ratios, reference and operating conditions, product flow rates, number of stages, etc).
  4. Equate the operating conditions for the column to the feed operating conditions.
- Operations at the Separation System Level.
  1. Establish reference conditions and product specifications for the system.
  2. Equate the reference conditions and the product specifications for the first column to the system with those for the separation system.

Some of the model fragments below, use the numerical values for specific parameters in a separation system. The predicate *Value-of* provides the representation for these numerical values. Value-of takes three arguments: (i) the amount or the derivative for some quantity (ii) the actual numerical value for it and (iii) the name of the equation that produced the solution. For example,

(Value-of (A (TBoil water system1 :reference)) 100.0 :Tboil-Calculation)

asserts that the boiling point for the water in system1 in reference conditions (atmospheric pressure) is equal to 100^0C. This value was calculated using the :TBoil-Calculation equation.

4.4.2 Multicomponent Distillation

Objects. The model contains the following kinds of objects: *substances, multicomponent mixtures, distillation-columns, separation-systems and environment.* Figure 4.22 provides the definitions for all the objects in the qualitative model.

The definition for a substance introduces a set of quantities that are independent of pressure and temperature. The *Critical-Pressure* and *Critical-Temperature* quantities indicate the conditions under which each substance reaches its critical point. The critical point is a set of conditions under which all three phases for a substance are present. The *Alpha-1*, *Alpha-2* and *Alpha-3* quantities correspond to the Antoine equation coefficients [72] for the substance. This equation provides a way of calculating the vapor pressure of a substance based on its critical pressure and its temperature. The *K-Ideal-Reference-Temperature* for the substance is a reference temperature used in the calculation of the equilibrium ratios for the substance. The *Liquid-Volume-Constant* is a constant specific to each substance that is used in the same calculation (see Figure B.1). The equilibrium ratios are used to compute relative volatilities.

We use the contained-stuff ontology to describe all the pieces of stuff in the system. In an analogous way with the binary case, a multicomponent mixture is defined by the substances it is composed of, by the phase it is in and by the container which holds it. The following function denotes a multicomponent mixture:

$$\text{M-C-S: components} \times \text{phase} \times \text{container} \longrightarrow \text{Multicomponent-Mixture}$$

There are two properties associated with a multicomponent mixture: its temperature and its pressure.

A *Separation-System* object is an abstraction describing a number of interconnected separation units (columns) that achieve a certain separation. The *Total-Cost* quantity stands for a cost estimate for the separation system.

The definition for a *Distillation-Column* introduces a set of features that describe the structure of a column. These include its diameter, its height and the minimum number of stages needed to achieve the separation that takes place in the device. In addition, the *Installation-Cost* quantity represents the installation cost for the column. Finally, the *Operating-Reflux-Estimate* quantity provides us with an estimate for the ratio between the operating reflux ratio and the minimum reflux ratio for a column. This is an important operating feature since it influences the product recoveries and the diameter of the column for a given separation.

The analysis of separation system and column behavior is based on two sets of conditions: the *:reference* and the *:operating* conditions. Each one of them corresponds to a set of values for pressure and temperature. The :reference conditions

```
(defPerspective (Substance-Properties-Dependent-on-Pressure ?substance ?system)
  Individuals ((?system :Type Separation-System
                        :Conditions
                         (Separation-System-Feed (M-C-S ?components ?phase ?stage) ?system))
               (?substance :Type Substance
                           :Test (member ?substance ?components)))
  Relations ((Quantity (TBoil ?substance ?system :reference))))

(defPerspective (Substance-Properties-Dependent-on-Temperature ?substance ?column)
  Individuals
  ((?substance :Type Substance)
   (?column :Conditions (Examine ?column)))
  Relations ((Positive-Quantity (Molar-Volume ?substance ?column :reference))
             (Positive-Quantity (NonPolar-Solubility ?substance ?column :reference))
             (Positive-Quantity (Molar-Volume ?substance ?column :operating))
             (Positive-Quantity (NonPolar-Solubility ?substance ?column :operating))))

(defPerspective (Polar-Solvent-Properties-Dependent-on-Temperature
                 ?substance ?column)
  Individuals
  ((?substance :Type Substance
               :Conditions (Polar-Solvent ?substance))
   (?column :Conditions (Examine ?column)))
  Relations
  ((Positive-Quantity (Molar-Volume ?substance ?column :reference))
   (Positive-Quantity (Molar-Volume ?substance ?column :operating))
   (Positive-Quantity (NonPolar-Solubility ?substance ?column :reference))
   (Positive-Quantity (NonPolar-Solubility ?substance ?column :operating))
   (Positive-Quantity (Polar-Solubility ?substance ?column :reference))
   (Positive-Quantity (Polar-Solubility ?substance ?column :operating))
   (Non-Negative-Quantity (Vapor-Pressure ?substance ?column :operating))
   (Non-Negative-Quantity (Vapor-Pressure ?substance ?column :reference))))
```

Figure 4.23: Introducing substance properties that depend on pressure or temperature.

are temperature and pressure values for which estimates for the physical properties we are interested in are available. The :operating conditions correspond to the actual operating conditions in a column. The values for the quantities *Reference-Pressure* and *Reference-Temperature* define the reference conditions for a system. Each column in a system will have the same reference conditions. The operating conditions for each column are defined by the *Operating-Pressure* and the *Operating-Temperature* quantities.

The *Environment* object serves as a way of introducing into the model physical constants such as the atmospheric pressure or the gas constant in the ideal gas law.

Substance Properties. Figure 4.23 provides the definitions for the model fragments described in this section. The *Substance-Properties-Dependent-on-Pressure*

```
(defPerspective (K-Ideal-Substance-Properties ?substance ?column)
  Individuals
  ((?substance :Type Substance)
   (?column :Type Distillation-Column
            :Conditions (Examine ?column))
   (?feed :Type Multicomponent-Mixture
          :Form (M-C-S ?components ?phase ?f-stage)
          :Test (member ?substance ?components)))
  Relations
  ((Positive-Quantity (Liquid-Molal-Volume ?substance ?column :physical))
   (Positive-Quantity (Liquid-Molal-Volume ?substance ?column :reference))
   (Positive-Quantity (Liquid-Molal-Volume ?substance ?column :operating))))
```

Figure 4.24: Introducing the quantities for computing equilibrium ratios.

perspective introduces properties that depend on the pressure in which we study a substance, like the boiling point of a substance under atmospheric conditions.

The *Substance-Properties-Dependent-on-Temperature* perspective introduces substance properties that depend on temperature. These properties along with the ones introduced by the *Polar-Solvent-Properties-Dependent-in-Temperature* perspective are used to calculate the relative volatilites between the substances in an extractive distillation unit. In particular, extractive distillation is based on the addition of a polar solvent in a mixture that can change the relative volatilities between the substances in the feed mixture. All the properties introduced by these two perspectives are used in the Weimer and Prausnitz method [72] for calculating the relative volatilities between the substances after the introduction of the polar solvent in the mixture.

The *K-Ideal-Substance-Properties* perspective in Figure 4.24 introduces a set of quantities that are used to calculate equilibrium ratios (see Figure B.1) at reference and operating conditions. The *:physical* operating condition is specific to this perspective and it corresponds to the value of the K-Ideal-Reference-Temperature.

Separation Properties. Figure 4.25 provides the definitions for the model fragments describing general separation properties. The *Separation-Properties-in-Reference-Conditions* and *Separation-Properties-in-Feed-Conditions* perspectives introduce separation properties at reference and actual operating conditions (temperature and pressure) for all the components of a mixture. The *K-Value* quantity stands for the equilibrium ratio of a substance in a column under reference or operating conditions. The quantity *Alpha-LK-HK* stands for the relative volatility between two components of the mixture. It takes four arguments: (i, ii) the names of the substances it refers to (iii) the column in which it is measured and (iv) the conditions for which it is valid. Finally, the *Vapor-Pressure* quantity stands for

```
(defPerspective (Separation-Properties-in-Reference-Conditions ?substance ?reference ?column ?feed)
  Individuals
  ((?column :Type Distillation-Column
            :Conditions (Examine ?column))
   (?feed :Type Multicomponent-Mixture
          :Form (M-C-S ?components ?phase ?feed-stage)
          :Conditions (Column-Feed ?feed ?column))
   (?reference :Type Substance
               :Test (and (member ?reference ?components)
                          (highest-boiling-point? ?reference ?components)))
   (?substance :Type Substance
               :Test (member ?substance ?components)))
  Relations
  ((Positive-Quantity (K-Value ?substance ?column :reference))
   (Consider (Reference-Component ?reference ?feed))
   (Non-Negative-Quantity (Vapor-Pressure ?substance ?column :reference))
   (Positive-Quantity (Alpha-LK-HK ?substance ?reference ?column :reference))))

(defPerspective (Separation-Properties-in-Feed-Conditions ?feed ?column ?substance)
  Individuals
  ((?column :Type Distillation-Column
            :Conditions (Examine ?column))
   (?feed :Type Multicomponent-Mixture
          :Form (M-C-S ?components ?phase ?feed-stage)
          :Conditions (Column-Feed ?feed ?column))
   (?substance :Type Substance
               :Test (member ?substance ?components)))
  Relations
  ((Positive-Quantity (K-Value ?substance ?column :operating))
   (Non-Negative-Quantity (Vapor-Pressure ?substance ?column :operating))))
```

Figure 4.25: Introducing separation properties for each substance.

the vapor pressure of a substance in a given column under a set of conditions. The physical significance of these three parameters is described in section 3.4 above.

The *highest-boiling-point?* procedure determines whether the substance in its first argument has the highest boiling point among the list of substances in its second argument under reference conditions. It is used in the Separation-Properties-in-Reference-Conditions perspective to establish the reference component on which all of the calculations of the relative volatilities between the substances of a mixture are based. According to this perspective the reference component is selected to be the heaviest (i.e. least volatile) of all the components in the mixture.

Separation System Properties. Figures 4.26 and 4.27 show the model fragments that introduce relevant properties for separation systems. The *Separation-System-&-Column-Feed-Properties* and *Separation-System-Feed-Properties* perspectives introduce the properties of interest for the input mixture to a separation system and its first column. These include its pressure, its temperature and the total feed flow rate for the mixture. Furthermore, it defines a set of analogous properties for the first column in the separation system and equates their amounts with the ones defined for the separation system. Finally, the operating conditions for the column are defined to be equal to the feed conditions. This is an assumption on which the method for calculating the cost estimates is based. The first column of a separation system is specified using the *First-Column* predicate in the current scenario. The *Examine* predicate in the Individuals field of these perspectives provides a way for the design system to focus its qualitative analysis to the columns specified using it.

For each component in the feed mixture of a separation system the *Separation-System-Input* perspective defines its feed flow rate. In addition, the total feed flow rate is defined to be monotonically increasing in the feed flow rate for each component.

The *Separation-System-Output* perspective defines the product recoveries for all the desired products of the system (i.e. the *Product-Specification* quantity).

Finally, the *Column-Feed* perspective (Fig. 4.27) defines the feed flow rates for the input components to the first column of a separation system and equates their amounts to those of the separation system.

Distillate and Bottom Product Features. Figures 4.28, 4.29, 4.30, 4.31, 4.32, 4.33, 4.34 and 4.35 provide the definitions for the model fragments that compute the products for possible separations. In the case of ordinary distillation processes, the *Column-Distillate-Output* perspective (Fig. 4.28) selects a set of possible pairs of light and heavy keys for the column that is currently examined and determines which of the desired products end up at the distillate product under this

```
(defPerspective (Separation-System-&-Column-Feed-Properties ?system ?column ?feed)
  Individuals ((?system :Type Separation-System)
               (?column :Type Distillation-Column
                        :Conditions (Examine ?column)
                                    (First-Column ?system ?column))
               (?feed :Type Multicomponent-Mixture
                      :Form (M-C-S ?components ?phase ?container)
                      :Conditions (Separation-System-Feed ?feed ?system)
                                  (Column-Feed ?feed ?column)))
  Relations
  ((Positive-Quantity (Feed-Temperature ?feed ?column))
   (Non-Negative-Quantity (Feed-Pressure ?feed ?column))
   (Non-Negative-Quantity (Total-Feed-Flow ?feed ?column))
   (Equal-to (A (Operating-Pressure ?column)) (A (Feed-Pressure ?feed ?column)))
   (Equal-to (A (Operating-Temperature ?column)) (A (Feed-Temperature ?feed ?column)))
   (Equal-to (A (Feed-Temperature ?feed ?column)) (A (Feed-Temperature ?feed ?system)))
   (Equal-to (A (Feed-Pressure ?feed ?column)) (A (Feed-Pressure ?feed ?system)))
   (Equal-to (A (Total-Feed-Flow ?feed ?column)) (A (Total-Feed-Flow ?feed ?system)))))

(defPerspective (Separation-System-Feed-Properties ?system ?feed)
  Individuals
  ((?system :Type Separation-System)
   (?feed :Type Multicomponent-Mixture
          :Form (M-C-S ?components ?phase ?container)
          :Conditions (Separation-System-Feed ?feed ?system)))
  Relations ((Positive-Quantity (Feed-Temperature ?feed ?system))
             (Non-Negative-Quantity (Feed-Pressure ?feed ?system))
             (Non-Negative-Quantity (Total-Feed-Flow ?feed ?system))))

(defPerspective (Separation-System-Input ?system ?feed ?component)
  Individuals ((?system :Type Separation-System)
               (?feed :Type Multicomponent-Mixture
                      :Form (M-C-S ?components ?phase ?container)
                      :Conditions (Separation-System-Feed ?feed ?system))
               (?component :Type Substance
                           :Test (member ?component ?components)))
  Relations ((Non-Negative-Quantity (Feed-Flow ?component ?feed ?system))
             (Qprop (Total-Feed-Flow ?feed ?system) (Feed-Flow ?component ?feed ?system))))

(defPerspective (Separation-System-Output ?system ?products)
  Individuals ((?system :Type Separation-System)
               (?products :Conditions (Desired-Products ?products ?system)))
  Relations ((Non-Negative-Quantity (Product-Specification ?products ?system))))
```

Figure 4.26: Introducing general separation system properties.

```
(defPerspective (Column-Feed ?system ?column ?feed ?component)
  Individuals ((?column :Type Distillation-Column
                        :Conditions (First-Column ?system ?column))
               (?feed :Type Multicomponent-Mixture
                      :Form (M-C-S ?components ?phase ?container)
                      :Conditions (Examine ?column)
                                  (Column-Feed ?feed ?column))
               (?component :Type Substance
                           :Test (member ?component ?components)))
  Relations ((Non-Negative-Quantity (Feed-Flow ?component ?feed ?column))
             (Equal-to (A (Feed-Flow ?component ?feed ?column))
                       (A (Feed-Flow ?component ?feed ?system)))))
```

Figure 4.27: Introducing the properties for the input mixture to the first column of a separation system.

set of keys. The possible pairs of keys are any substances with neighboring boiling points The distillate products are all the components of the mixture that are more volatile than the light key. The relations field for this perspective defines the flow rate for each component in the distillate product and a quantity describing the desired recovery for this component under the current separation scheme (*Product-Recovery*). In addition, it asserts that the flow rate for any of the components in the distillate product is proportional to the feed flow rate for this component. Finally, the perspective asserts that the total distillate product flow rate for the column is proportional to the flow rate for each of the components in the distillate product.

Given a set of substances and a pressure the procedure *neighboring-boiling-points?* in the *Column-Distillate-Output* perspective determines whether the substances specified in its first and second arguments have neighboring boiling points. The procedure *less-volatile?* determines whether its first argument has a lower boiling point than its second argument at the pressure conditions specified in its third argument.

A set of analogous properties are defined in the *Column-Bottom-Output* perspective (Fig. 4.29). This perspective determines which of the desired products end up at the bottom product of a column under different selections of keys. The bottom product consists of all the components of the mixture that are less volatile than the heavy key. The product recoveries and the flow rates for the bottom product components along with their relationships with the feed flows of the components and the total bottoms flow rate for the column are defined in an analogous way with the Column-Distillate-Output perspective.

```
(defPerspective (Column-Distillate-Output ?column ?product ?feed ?light-key ?heavy-key ?products)
  Individuals
  ((?column :Type Distillation-Column
            :Conditions (Examine ?column)
                        (Consider (Sharp-Separation-For ?column))
                        (Desired-Products ?products ?column)
                        (Value-of (A (Reference-Pressure ?column ?pressure ?eqn))
   (?feed :Type Multicomponent-Mixture
          :Form (M-C-S ?components ?phase ?container)
          :Test (subsetp ?products ?components)
          :Conditions (Column-Feed ?feed ?column))
   (?light-key :Type Substance
               :Test (and (member ?light-key ?components)
                          (non-polar? ?light-key)
                          (check-scenario ?column ?light-key 'distillation)))
   (?heavy-key :Type Substance
               :Test (and (member ?heavy-key ?components)
                          (not (eql ?heavy-key ?light-key))
                          (neighboring-boiling-points?
                           ?heavy-key ?light-key ?pressure ?components)))
   (?product :Type Substance
             :Test (and (member ?product ?products)
                        (or (eql ?product ?heavy-key)
                            (less-volatile? ?light-key ?product ?pressure)))))
  Relations
  ((Non-Negative-Quantity
     (Distillate-Component-Flow-Rate ?product ?column (?light-key ?heavy-key)))
   (Non-Negative-Quantity (Product-Recovery ?product ?column (?light-key ?heavy-key)))
   (Qprop (Total-Distillate-Flow-Rate ?column (?light-key ?heavy-key))
          (Distillate-Component-Flow-Rate ?product ?column (?light-key ?heavy-key)))
   (Qprop (Distillate-Component-Flow-Rate ?product ?column (?light-key ?heavy-key))
          (Feed-Flow ?product ?feed ?column)))))
```

Figure 4.28: Determining the distillate product components for an ordinary distillation column.

```
(defPerspective (Column-Bottom-Output ?column ?product ?feed ?light-key ?heavy-key ?products)
  Individuals
  ((?column :Type Distillation-Column
            :Conditions (Examine ?column)
                        (Consider (Sharp-Separation-For ?column))
                        (Desired-Products ?products ?column)
                        (Value-of (A (Reference-Pressure ?column)) ?pressure ?eqn))
   (?feed :Type Multicomponent-Mixture
          :Form (M-C-S ?components ?phase ?container)
          :Test (subsetp ?products ?components)
          :Conditions (Column-Feed ?feed ?column))
   (?light-key :Type Substance
               :Test (and (member ?light-key ?components)
                          (non-polar? ?light-key)
                          (check-scenario ?column ?light-key 'distillation)))
   (?heavy-key :Type Substance
               :Test (and (member ?heavy-key ?components)
                          (not (eql ?heavy-key ?light-key))
                          (neighboring-boiling-points?
                           ?heavy-key ?light-key ?pressure ?components)))
   (?product :Type Substance
             :Test (and (member ?product ?products)
                        (or (eql ?product ?light-key)
                            (less-volatile? ?product ?heavy-key ?pressure)))))
  Relations
  ((Non-Negative-Quantity (Bottom-Component-Flow-Rate ?product ?column (?light-key ?heavy-key)))
   (Non-Negative-Quantity (Product-Recovery ?product ?column (?light-key ?heavy-key)))
   (Qprop (Total-Bottom-Flow-Rate ?column (?light-key ?heavy-key))
          (Bottom-Component-Flow-Rate ?product ?column (?light-key ?heavy-key)))
   (Qprop (Bottom-Component-Flow-Rate ?product ?column (?light-key ?heavy-key))
          (Feed-Flow ?product ?feed ?column))))
```

Figure 4.29: Determining the bottom product components for an ordinary distillation column.

Finally, the *Distillate-Bottom-Key-Rates* perspective (Fig. 4.30) determines the distillate and bottoms flow rates for all the possible keys. For every possible selection of keys the separation between them is not perfect. Small amounts of the light key end up in the bottom product while small amounts of the heavy key end up as distillate. In the numerical model the flow rates for these components are determined by the desired product recoveries for the keys. This is why this perspective tries to find in which set of desired products each one of the keys belongs. This information is used by the numerical equations to determine the desired product recoveries and the distillate and the bottoms flow rates for the keys.

The *non-polar?* procedure in the perspective definitions returns true if the substance it takes as an argument is not a polar solvent. The procedure *check-scenario* takes three arguments: (i) a particular column (ii) the light key for the current separation (iii) the type of the current separation. It returns true if the design heuristics have already decided for a separation with the same light key and the same type as the current one. This is a way of pruning the number of possible model fragments that get instantiated in cases where the evolutionary heuristics have decided on a particular separation in advance.

In the case of extractive distillation processes, the *Extractive-Column-Distillate-Output* perspective (Fig. 4.31) determines the distillate product components for the process. The *Extractive-Column-Bottom-Output* perspective (Fig. 4.32) determines the bottoms product components for the process and finally the *Extractive-Distillate-Bottom-Key-Rates* perspective (Fig. 4.33) defines the flow rates for the keys in the distillate and bottom products. All of these model fragments are defined in an analogous way with the ones described for the ordinary distillation case. The only difference is that they require an additional object in their Individuals field for the polar solvent in the extractive distillation process. This solvent is specified as the argument to the *Polar-Solvent* predicate. The predicate *Attracts* denotes the substance that associates with the polar solvent in extractive distillation. This substance leaves the column in the bottom product of an extractive distillation tower.

Once the heuristic analysis has proposed a separation for the current column a set of perspectives is activated that compares the products for this separation with the desired ones. The *Unobtained-Column-Products* (Fig. 4.34) and *Extractive-Distillation-Unobtained-Products* (Fig. 4.35) perspectives determine which of the desired products were not obtained using the current separation scheme. The *Missing-Products* predicate indicates the desired products that were not recovered in the separation. It takes four arguments: (i) the missing products (ii) the current

```
(defPerspective (Distillate-Bottom-Key-Rates ?column ?feed ?h-k ?l-k ?products-1 ?products-2)
  Individuals
  ((?column :Type Distillation-Column
            :Conditions (Examine ?column)
                        (Consider (Sharp-Separation-For ?column))
                        (Value-of (A (Reference-Pressure ?column)) ?pressure ?eqn))
   (?products-1 :Conditions (Desired-Products ?products-1 ?column))
   (?products-2 :Conditions (Desired-Products ?products-2 ?column))
   (?feed :Type Multicomponent-Mixture
          :Form (M-C-S ?components ?phase ?container)
          :Test (and (subsetp ?products-1 ?components)
                     (subsetp ?products-2 ?components)
                     (not (intersection ?products-1 ?products-2)))
          :Conditions (Column-Feed ?feed ?column))
   (?l-k :Type Substance
         :Test (and (member ?l-k ?products-1)
                    (non-polar? ?l-k)
                    (check-scenario ?column ?l-k 'distillation)))
   (?h-k :Type Substance
         :Test (and (member ?h-k ?products-2)
                    (not (eql ?h-k ?l-k))
                    (neighboring-boiling-points? ?h-k ?l-k ?pressure ?components))))
  Relations
  ((Non-Negative-Quantity (Distillate-Component-Flow-Rate ?h-k ?column (?l-k ?h-k)))
   (Non-Negative-Quantity (Bottom-Component-Flow-Rate ?l-k ?column (?l-k ?h-k)))
   (Non-Negative-Quantity (Bottom-Component-Flow-Rate ?h-k ?column (?l-k ?h-k)))
   (Non-Negative-Quantity (Distillate-Component-Flow-Rate ?l-k ?column (?l-k ?h-k)))
   (Qprop (Distillate-Component-Flow-Rate ?l-k ?column (?l-k ?h-k)) (Feed-Flow ?l-k ?feed ?column))
   (Qprop (Bottom-Component-Flow-Rate ?h-k ?column (?l-k ?h-k)) (Feed-Flow ?h-k ?feed ?column))
   (Qprop (Distillate-Component-Flow-Rate ?h-k ?column (?l-k ?h-k)) (Feed-Flow ?h-k ?feed ?column))
   (Qprop (Bottom-Component-Flow-Rate ?l-k ?column (?l-k ?h-k)) (Feed-Flow ?l-k ?feed ?column))))
```

Figure 4.30: Describing the distillate and bottom flow rates for the keys.

```
(defPerspective (Extractive-Column-Distillate-Output
                 ?column ?product ?feed ?light-key ?heavy-key ?products ?p-s)
  Individuals
  ((?column :Type Distillation-Column
            :Conditions (Examine ?column)
                        (Consider (Sharp-Separation-For ?column))
                        (Desired-Products ?products ?column)
                        (Value-of (A (Reference-Pressure ?column)) ?pressure ?eqn))
   (?feed :Type Multicomponent-Mixture
          :Form (M-C-S ?components ?phase ?container)
          :Test (subsetp ?products ?components)
          :Conditions (Column-Feed ?feed ?column))
   (?light-key :Type Substance
               :Test (and (member ?light-key ?components)
                          (non-polar? ?light-key)
                          (check-scenario ?column ?light-key 'extractive-distillation)))
   (?heavy-key
    :Type Substance
    :Test (and (member ?heavy-key ?components)
               (not (eql ?heavy-key ?light-key))
               (neighboring-boiling-points? ?light-key ?heavy-key ?pressure ?components)))
   (?p-s :Type Substance
         :Conditions (Polar-Solvent ?p-s)
                     (Attracts ?p-s ?heavy-key))
   (?product :Type Substance
             :Test (and (member ?product ?products)
                        (or (eql ?product ?light-key)
                            (less-volatile? ?heavy-key ?product ?pressure)))))
  Relations
  ((Non-Negative-Quantity (Product-Recovery ?product ?column (?light-key ?heavy-key)))
   (Non-Negative-Quantity
    (Extractive-Distillate-Component-Flow-Rate ?product ?column (?light-key ?heavy-key) ?p-s))))
```

Figure 4.31: Determining the distillate product components in the extractive distillation case. The *?p-s* variable stands for the polar solvent used in extractive distillation.

```
(defPerspective (Extractive-Column-Bottom-Output
                 ?column ?product ?feed ?light-key ?heavy-key ?products ?p-s)
  Individuals
  ((?column :Type Distillation-Column
            :Conditions (Examine ?column)
                        (Consider (Sharp-Separation-For ?column))
                        (Desired-Products ?products ?column)
                        (Value-of (A (Reference-Pressure ?column)) ?pressure ?eqn))
   (?feed :Type Multicomponent-Mixture
          :Form (M-C-S ?components ?phase ?container)
          :Test (subsetp ?products ?components)
          :Conditions (Column-Feed ?feed ?column))
   (?light-key :Type Substance
               :Test (and (member ?light-key ?components)
                          (non-polar? ?light-key)
                          (check-scenario ?column ?light-key 'extractive-distillation)))
   (?heavy-key
    :Type Substance
    :Test (and (member ?heavy-key ?components)
               (not (eql ?heavy-key ?light-key))
               (neighboring-boiling-points? ?light-key ?heavy-key ?pressure ?components)))
   (?p-s :Type Substance
         :Conditions (Polar-Solvent ?p-s)
                     (Attracts ?p-s ?heavy-key))
   (?product :Type Substance
             :Test (and (member ?product ?products)
                        (not (eql ?product ?light-key))
                        (or (eql ?product ?heavy-key)
                            (less-volatile? ?product ?light-key ?pressure)))))
  Relations
  ((Non-Negative-Quantity (Product-Recovery ?product ?column (?light-key ?heavy-key)))
   (Non-Negative-Quantity
    (Extractive-Bottom-Component-Flow-Rate ?product ?column (?light-key ?heavy-key) ?p-s))))
```

Figure 4.32: Determining the bottom product components in the extractive distillation case.

```
(defPerspective (Extractive-Distillate-Bottom-Key-Rates
                 ?column ?feed ?h-k ?l-k ?products-1 ?products-2 ?p-s)
  Individuals ((?column :Type Distillation-Column
                        :Conditions (Examine ?column)
                                    (Consider (Sharp-Separation-For ?column))
                                    (Value-of (A (Reference-Pressure ?column)) ?pressure ?eqn))
               (?products-1 :Conditions (Desired-Products ?products-1 ?column))
               (?products-2 :Conditions (Desired-Products ?products-2 ?column))
               (?feed :Type Multicomponent-Mixture
                      :Form (M-C-S ?components ?phase ?container)
                      :Test (and (subsetp ?products-1 ?components)
                                 (subsetp ?products-2 ?components)
                                 (not (equal ?products-1 ?products-2)))
                      :Conditions (Column-Feed ?feed ?column))
               (?l-k :Type Substance
                     :Test (and (member ?l-k ?products-1)
                                (check-scenario ?column ?l-k 'extractive-distillation)
                                (heavier? ?l-k ?products-1 ?pressure)))
               (?h-k :Type Substance
                     :Test (and (member ?h-k ?products-2)
                                (not (eql ?h-k ?l-k))
                                (neighboring-boiling-points? ?l-k ?h-k ?pressure ?components)))
               (?p-s :Type Substance
                     :Conditions (Polar-Solvent ?p-s)
                                 (Attracts ?p-s ?h-k)))
  Relations ((Non-Negative-Quantity
              (Extractive-Distillate-Component-Flow-Rate ?h-k ?column (?l-k ?h-k) ?p-s))
             (Non-Negative-Quantity
              (Extractive-Bottom-Component-Flow-Rate ?l-k ?column (?l-k ?h-k) ?p-s))))
```

Figure 4.33: Determining the key flow rates in the extractive distillation case.

```
(defView (Unobtained-Column-Products ?desired-products ?column)
  Individuals
  ((?column :Type Distillation-Column
            :Conditions (Examine ?column)
                        (Column-Feed (M-C-S ?components ?phase ?stage) ?column)
                        (Value-of (A (Reference-Pressure ?column)) ?ref-pres ?a-eqn))
   (?products :Conditions (Column-Products ?products (?l-k ?h-k) ?column))
   (?l-k :Type Substance
         :Test (and (member ?l-k ?components)
                    (non-polar? ?l-k)
                    (check-scenario ?column ?l-k 'distillation)))
   (?h-k :Type Substance
         :Test (and (member ?h-k ?components)
                    (not (eql ?h-k ?l-k))
                    (neighboring-boiling-points? ?h-k ?l-k ?ref-pres ?components)))
   (?desired-products :Test (and (subsetp ?desired-products ?products)
                                 (not (subsetp ?products ?desired-products)))
                      :Conditions (Desired-Products ?desired-products ?column)))
  Preconditions ((Consider (Possible (Separation distillation (?l-k ?h-k) ?column))))
  Relations ((Missing-Products ?desired-products ?products (?l-k ?h-k) ?column)))
```

Figure 4.34: Computing the unobtained column products for distillation.

```
(defView (Extractive-Distillation-Unobtained-Products ?products ?desired-products ?p-s ?column)
  Individuals
  ((?column :Type Distillation-Column
            :Conditions (Examine ?column))
   (?products :Conditions (Column-Products ?products (?l-k ?h-k) ?column))
   (?desired-products :Test (and (subsetp ?desired-products ?products)
                                 (not (subsetp ?products ?desired-products)))
                      :Conditions (Desired-Products ?desired-products ?column))
   (?p-s :Type Substance
         :Conditions (Polar-Solvent ?p-s)
                     (Attracts ?p-s ?h-k)))
  Preconditions
  ((Consider (Possible (Separation extractive-distillation (?l-k ?h-k) ?column))))
  Relations ((Missing-Products ?desired-products ?products (?l-k ?h-k) ?column)))
```

Figure 4.35: Computing the unobtained products for extractive distillation.

```
(defPerspective (Distillation-Features-in-Reference-Conditions
                  distillation ?column ?heavy-key ?light-key ?ref-pres ?feed ?op-pres ?ref-temp)
  Individuals
  ((?column
    :Type Distillation-Column
    :Conditions (Examine ?column)
                (Consider (Sharp-Separation-for ?column))
                (Value-of (A (Reference-Temperature ?column)) ?ref-temp ?ref-t-eqn)
                (Value-of (A (Reference-Pressure ?column)) ?ref-pres ?eqn))
   (?feed :Type MultiComponent-Mixture
          :Form (M-C-S ?components ?phase ?container)
          :Conditions (Column-Feed ?feed ?column))
   (?op-pres :Conditions (Value-of (A (Operating-Pressure ?column)) ?op-pres ?op-eqn))
   (?light-key :Type Substance
               :Test (and (member ?light-key ?components)
                          (non-polar? ?light-key)
                          (check-scenario ?column ?light-key 'distillation)))
   (?heavy-key
    :Type Substance
    :Test (and (member ?heavy-key ?components)
               (not (eql ?heavy-key ?light-key))
               (not (associates-with? ?light-key ?heavy-key))
               (neighboring-boiling-points? ?heavy-key ?light-key ?ref-pres ?components))))
  Relations
  ((Positive-Quantity (Alpha-LK-HK ?light-key ?heavy-key ?column :reference))
   (Quantity (CDS distillation ?column (?light-key ?heavy-key) :reference))
   (Non-Negative-Quantity
    (Total-Distillate-Flow-Rate ?column (?light-key ?heavy-key)))
   (Non-Negative-Quantity
    (Total-Bottom-Flow-Rate ?column (?light-key ?heavy-key)))
   (Reference-Conditions ?column (Pressure ?ref-pres) (Temp ?ref-temp))
   (Propose
    (Possible (Separation distillation (?light-key ?heavy-key) ?column)))))
```

Figure 4.36: Introducing ordinary distillation features at reference conditions.

```
(defPerspective (Distillation-Features-in-Reference-Conditions
                 extractive-distillation ?column ?h-k ?l-k ?p-s ?temp ?feed ?ref-pres)
  Individuals
  ((?column :Type Distillation-Column
            :Conditions (Examine ?column)
                        (Value-of (A (Reference-Temperature ?system)) ?temp ?temp-eqn)
                        (Value-of (A (Reference-Pressure ?system)) ?ref-pres ?eq0))
                        (Consider (Sharp-Separation-for ?column)))
   (?feed :Type MultiComponent-Mixture
          :Form (M-C-S ?components ?phase ?container)
          :Conditions (Column-Feed ?feed ?column))
   (?l-k :Type Substance
         :Test (member ?l-k ?components))
   (?h-k :Type Substance
         :Test (and (member ?h-k ?components)
                    (not (eql ?h-k ?l-k))
                    (neighboring-boiling-points? ?l-k ?h-k ?ref-pres ?components)))
   (?p-s :Type Substance
         :Test (and (not (eql ?l-k ?p-s))
                    (or (equal ?p-s ?h-k)
                        (associates-with? ?p-s ?l-k)
                        (associates-with? ?p-s ?h-k)))
         :Conditions (Polar-Solvent ?p-s)))
  Relations
  ((Positive-Quantity (Extractive-Alpha-LK-HK ?l-k ?h-k ?p-s ?column :reference))
   (Positive-Quantity (Energy-of-Interaction ?h-k ?p-s ?column :reference))
   (Positive-Quantity (Energy-of-Interaction ?l-k ?p-s ?column :reference))
   (Positive-Quantity (Selectivity ?l-k ?h-k ?p-s ?column :reference))
   (Positive-Quantity (Infinite-Activity-Coefficient ?l-k ?p-s ?column :reference))
   (Positive-Quantity (Infinite-Activity-Coefficient ?h-k ?p-s ?column :reference))
   (Quantity (CDS extractive-distillation ?column (?l-k ?h-k) :reference))
   (Non-Negative-Quantity (Total-Extractive-Distillate-Flow-Rate ?column (?l-k ?h-k ?p-s)))
   (Non-Negative-Quantity (Total-Extractive-Bottom-Flow-Rate ?column (?l-k ?h-k ?p-s)))
;;; It is assumed to be .9 because of Infinite-Activity-Coefficients
   (Consider (Polar-Solvent-Concentration ?p-s ?column .9))
   (Positive-Quantity (Extractive-Component-Feed-Flow ?p-s ?feed ?column))
   (Positive-Quantity (Total-Extractive-Feed-Flow ?p-s ?feed ?column))
   (Reference-Conditions ?column (Pressure ?ref-pres) (Temp ?ref-temp))
   (Propose (Possible (Separation extractive-distillation (?l-k ?h-k) ?column)))))
```

Figure 4.37: Introducing extractive distillation features at reference conditions.

column products of which the missing products are a subset (iii, iv) the separation scheme (the keys and the column) under which (i) and (ii) hold.

Distillation Features in Reference Conditions. Figures 4.36 and 4.37 provide the definitions for the model fragments in this section. There are two *Distillation-Features-in-Reference-Conditions* perspectives that cover ordinary (Fig. 4.36) and extractive (Fig. 4.37) distillation processes. The model fragment for the ordinary distillation determines the possible keys for the separations. It then instantiates a set of quantities including the *Alpha-LK-HK* parameter that measures the relative volatility between the heavy and the light key under reference conditions, the coefficient of the difficulty of a separation (*CDS*) that gives a numerical estimate for the difficulty of a separation and the total distillate and bottom product flow rates for the separation. Finally, it introduces a predicate establishing the reference conditions for the column (*Reference-Conditions*) and a predicate *(Propose (Possible (Separation distillation (?light-key ?heavy-key) ?column)))* that represents the ordinary distillation alternatives for the current column.

The procedure *associates-with?* in the perspective definition looks for possible attraction between the molecules of the two substances specified in its arguments. If there is an attraction then we are dealing with an extractive distillation process.

The perspective for the extractive distillation process determines the possible keys for the separation. It then instantiates a set of analogous parameters with the ones described for the ordinary distillation case (*Extractive-Alpha-LK-HK*, *CDS*, *Total-Extractive-Distillate-Flow-Rate* and *Total-Extra-Bottom-Flow-Rate*). In addition, it introduces a set of quantities specific to the extractive distillation case. These include the energies of interaction and the infinite activity coefficients for the two keys along with the selectivity between them. All these are quantities that are used in the Weimer-Prausnitz method [72] for calculating the relative volatilities between substances, in the presence of a polar solvent in the mixture for the reference temperature specified for the column. Finally, the quantities that describe the feed flow rate of the polar solvent in the process (*Extractive-Component-Feed-Flow*), the total feed flow rate for the process that takes into account the presence of the polar solvent (*Total-Extractive-Feed-Flow*) and an assumption concerning the proposed concentration for the polar solvent in the feed mixture (*Polar-Solvent-Concentration*) are asserted. In an analogous way with the ordinary distillation case, this perspective introduces a predicate *(Propose (Possible (Separation extractive-distillation (?l-k ?h-k) ?column)))* that represents the extractive distillation alternatives for the current column.

Distillation Features in Actual Conditions. Figures 4.38 and 4.39 provide the definitions for the model fragments in this section. There are two *Distillation-Features-in-Actual-Conditions* perspectives that cover ordinary and extractive dis-

```
(defView (Distillation-Features-in-Actual-Conditions
          distillation ?column ?heavy-key ?light-key ?pressure ?feed ?op-temp)
  Individuals
  ((?column :Type Distillation-Column
            :Conditions (Examine ?column)
                        (First-Column ?system ?column)
                        (Value-of (A (Reference-Pressure ?system)) ?ref-pres ?eqn-0)
                        (Consider (Sharp-Separation-for ?column)))
   (?feed :Type Multicomponent-Mixture
          :Form (M-C-S ?components ?phase ?stage)
          :Conditions (Column-Feed ?feed ?column))
   (?pressure :Conditions (Value-of (A (Operating-Pressure ?column)) ?pressure ?pres-eqn))
   (?op-temp :Conditions (Value-of (A (Operating-Temperature ?column)) ?op-temp ?temp-eqn))
   (?light-key :Type Substance
               :Test (and (member ?light-key ?components)
                          (non-polar? ?light-key)
                          (check-scenario ?column ?light-key 'distillation)))
   (?heavy-key
    :Type Substance
    :Test (and (not (eql ?light-key ?heavy-key))
               (member ?heavy-key ?components)
               (neighboring-boiling-points? ?heavy-key ?light-key ?ref-pres ?components))))
  Preconditions ((Consider (Possible (Separation distillation (?light-key ?heavy-key) ?column))))
  Relations ((Non-Negative-Quantity (Min-Reflux-Ratio (?light-key ?heavy-key) ?column))
             (Positive-Quantity (Alpha-LK-HK ?light-key ?heavy-key ?column :operating))
             (Positive-Quantity (Min-#-of-Stages ?column (?light-key ?heavy-key))
             (Positive-Quantity (Average-Vapor-Velocity ?column (?light-key ?heavy-key)))
             (Positive-Quantity (Reflux-Ratio (?light-key ?heavy-key) ?column))
             (Positive-Quantity (Phi ?components ?column))
             (Actual-Conditions
              ?column distillation (Pressure ?pressure) (Temp ?op-temp) (?light-key ?heavy-key)))))
```

Figure 4.38: Introducing ordinary distillation features in actual conditions.

```
(defView (Distillation-Features-in-Actual-Conditions
         extractive-distillation?column ?l-k ?h-k ?p-s ?pres ?temp ?feed)
  Individuals ((?column :Type Distillation-Column
                        :Conditions (Examine ?column))
               (?feed :Type Multicomponent-Mixture
                      :Form (M-C-S ?components ?phase ?f-stage)
                      :Conditions (Column-Feed ?feed ?column))
               (?pres :Conditions (Value-of (A (Operating-Pressure ?column)) ?pres ?pres-eqn))
               (?temp :Conditions (Value-of (A (Operating-Temperature ?column)) ?temp ?temp-eqn))
               (?l-k :Type Substance
               :Test (and (member ?l-k ?components)
                          (non-polar? ?l-k)
                          (check-scenario ?column ?l-k 'extractive-distillation)))
               (?h-k :Type Substance
                     :Test (and (not (eql ?l-k ?h-k)) (member ?h-k ?components)))
               (?p-s :Type Substance
                     :Test (and (not (eql ?l-k ?p-s))
                                (or (associates-with? ?p-s ?l-k)
                                    (associates-with? ?p-s ?h-k)))
                     :Conditions (Polar-Solvent ?p-s)))
  Preconditions ((Consider (Possible (Separation extractive-distillation (?l-k ?h-k) ?column))))
  Relations ((Positive-Quantity (Extractive-Alpha-LK-HK ?l-k ?h-k ?p-s ?column :operating))
             (Positive-Quantity (Energy-of-Interaction ?h-k ?p-s ?column :operating))
             (Positive-Quantity (Energy-of-Interaction ?l-k ?p-s ?column :operating))
             (Positive-Quantity (Selectivity ?l-k ?h-k ?p-s ?column :operating))
             (Positive-Quantity (Infinite-Activity-Coefficient ?l-k ?p-s ?column :operating))
             (Positive-Quantity (Infinite-Activity-Coefficient ?h-k ?p-s ?column :operating))
             (Positive-Quantity (Min-Reflux-Ratio (?l-k ?h-k) ?p-s ?column))
             (Positive-Quantity (Extractive-Min-#-of-Stages ?column (?l-k ?h-k) ?p-s))
             (Positive-Quantity (Average-Vapor-Velocity ?column (?l-k ?h-k)))
             (Positive-Quantity (Reflux-Ratio (?l-k ?h-k) ?column))
             (Positive-Quantity (Extractive-Phi ?p-s ?components ?column))
             (Actual-Conditions
              ?column extractive-distillation (Pressure ?pressure) (Temp ?op-temp) (?l-k ?h-k))
             (Mass-Separating-Agent ?p-s ?column)))
```

Figure 4.39: Introducing extractive distillation features in actual conditions.

tillation. The Individuals for these perspectives calculate the possible keys for the separation. As chapter 5 explains in detail, the heuristic analysis determines when the preconditions for these perspectives hold. Consequently, the features in their relations fields hold only for separations that are considered promising by the design heuristics.

The relations field for the ordinary distillation perspective introduces a number of quantities that describe the separation in more detail. These include the minimum and the actual reflux ratios for the column under each separation (*Min-Reflux-Ratio* and *Reflux-Ratio* respectively), the relative volatilities for the possible keys at operating conditions, the *Phi* quantity which is a special-purpose quantity used in the Underwood method [54] for calculating minimum reflux ratios and the minimum number of stages for each alternative. These quantities are used in the numerical model to compute structural features for the column such as its diameter and height. These features along with the average vapor velocity and the reflux ratio are used to calculate cost estimates for the column. Furthermore, the *Actual-Conditions* predicate establishes the actual operating conditions for the particular separation in the column.

The description for the extractive distillation perspective is analogous to the one presented above. In addition, there is a set of quanitities that describe properties special to the extractive distillation process. These are the same with the ones described for the Distillation-Features-in-Reference-Conditions perspective, except that they correspond to actual operating conditions. The *Mass-Separating-Agent* predicate establishes the particular solvent for the extractive distillation in the column.

Applying the multicomponent separation model. The model serves two main functions:

- It supplies the design knowledge component with appropriate qualitative and numerical descriptions for the design alternatives.
- It guides the creation of appropriate numerical models that describe the separation alternatives.

With regards to the first purpose, the model computes the possible combinations of keys that comprise the separation alternatives[5] under the sharp separation assumption. For each one of these the model determines its distillate and bottom products. In addition, a set of separation properties for the diffficulty of each separation and for computing cost estimates are instantiated. These include a set

[5]The selection of key combinations is explained in section 3.2 above.

of physical properties for the substances that take part in the separation (like the vapor pressures and the equilibrium ratios for calculating the relative volatilities between the keys for each separation), the input and output flow rates for each alternative, some parameters that are specific to the extractive distillation process (e.g. parameters that describe the energy of interaction and the solubility of various substances), certain design parameters (such as the coefficient of the difficulty of separation) and finally structural and operational features for each possible separation (e.g. reflux ratios, column height and diameter, number of stages, installation cost for the column, etc.).

The qualitative and the numerical models in the physical knowledge along with the heuristic rules in the design knowledge are translated into rules that are fed into an assumption-based truth maintenance system (ATMS) [9]. A rule engine (ATMoSphere) [17] provides the interface to the ATMS. The heuristic analysis uses the design knowledge to construct focus environments [19] that determine the set of 'promising' separation alternatives at each design step. These environments are combined with the instantiated qualitative model fragments to determine the actual numerical models at each point in the design process.

4.5 Numerical models

Representation. In the case of binary mixtures, the numerical models consist of a set of equations that describe the dynamic and steady-state behavior of binary columns. In the case of multicomponent mixtures, the numerical models are based on the Fenske-Underwood-Gilliland short-cut method for designing distillation and extractive distillation units [13].

Figure 4.40 provides an example of the representation for each one of the equations in the system. Each *defEquation* form consists of four parts:

1. The *Name* part provides a name for the equation. This name is stored along with any solutions obtained using this equation in order to support debugging and explanation tasks in the future.

2. The *Conditions* part refers to a set of model fragments in the qualitative domain theory that have to be implied by the current focus environment for the equation to hold. For example the *Distillation-Features-in-Reference-Conditions* predicate in Figure 4.40 refers to the name of the perspective defined in Figure 4.36.

```
( defEquation
 Relative-Volatility-Calculation-with-Vapor-Pressures  ;; The Name part
;; The Conditions part
((Distillation-Features-in-Reference-Conditions
  distillation ?column ?h-k ?l-k ?pressure (M-C-S ?components ?phase ?stage) ?op-pres ?ref-temp)
  :Test (null (dsn::polar-solvent-in? ?components)) )
;; The Quantities part
((Alpha-LK-HK ?l-k ?h-k ?pressure ?components)
    (Vapor-Pressure ?l-k ?column :reference)
    (Vapor-Pressure ?h-k ?column :reference) )
;; The Numerical-Form part
(= (A (Alpha-LK-HK ?l-k ?h-k ?column :reference))
   (/ (A (Vapor-Pressure ?l-k ?column :reference)) (A (Vapor-Pressure ?h-k ?column :reference)))) )
```

Figure 4.40: Calculating the relative volatility for two possible keys in the reference conditions specified by the design system.

3. The *Quantities* field contains al the quantities that are mentioned in the actual numerical form for the equation. The system tries to solve an equation when it is active and when there are solutions for all but one of the quantities in this field.

4. The *Numerical-Form* field contains the actual numerical form for the equation.

Dynamic Behavior of Binary Columns. The conservation of mass principle is used to describe the mass flow between the stages of a binary column. Consequently, there are two differential equations for the total holdup of material (THM) (liquid and vapor) and the total holdup for the volatile component (THV) at each stage[6]. If V is the vapor flow throughout the column, L_n is the flow rate of the liquid leaving stage n, L_{n+1} is the flow rate of the liquid coming from stage n+1, x_n and x_{n+1} are the mole fractions for the volatile component in the liquid mixture at stages n and n+1 respectively and finally y_n and y_{n-1} are the mole fractions for the volatile component in the vapor at stages n and n-1 respectively then:

$$\frac{dTHM(n))}{dt} = L_{n+1} - L_n \tag{4.1}$$

$$\frac{dTHV(n))}{dt} = L_{n+1}x_{n+1} - L_n x_n + V y_{n-1} - V y_n \tag{4.2}$$

[6]We make the assumption that if stage n is the current stage then stage n+1 is directly above it and stage n-1 is directly below it.

In the qualitative model we do not have to write an explicit equation for the conservation of mass principle at each stage in the column. The fact that each flow process model takes into consideration the conservation of mass allows the qualitative analysis to construct the equations (4.1) and (4.2) for each stage by summing up the direct influences for the Amount-of and the Amount-of-in quantities.

Let THML(n) be the total holdup for the liquid and THMLV(n) be the total holdup for the volatile component in the liquid at stage n. Because we assumed that we have negligible vapor holdup at each stage the following relations will also hold:

$$\text{THML(n)} \approx \text{THM(n)} \text{ and } \text{THMLV(n)} \approx \text{THV(n)}$$

The qualitative equivalents for the last two equations are included in the Negligible-Vapor-HoldUp view (Fig. 4.4) for each stage.

For each stage in the column there is a vapor-liquid equilibrium relation between the mole fractions of the volatile component in the two phases. If α is the relative volatility between the two mixture components then:

$$y_n = \frac{\alpha x_n}{1 + (\alpha - 1)x_n} \tag{4.3}$$

The qualitative equivalent of this relationship is described in the Binary-Vapor-Liquid-Equilibrium view (Fig. 4.5) for each stage.

Finally, there is an equation that represents the time delay that occurs when liquid is flowing through a stage. The equation has the following form:

$$L_n = \bar{L}_n + \frac{THML(n) - \overline{THML(n)}}{\beta} \tag{4.4}$$

In the qualitative model the quantity Beta represents the hydraulic time constant β, the Initial-Amount-of quantity is the initial liquid holdup for each stage and corresponds to the $\overline{THML(n)}$ constant while the Initial-Liquid-Flow-Rate quantity is the initial flow rate of the liquid leaving each stage and corresponds to the $\bar{L}_n$ constant.

Stedy-State Behavior of Binary Columns. An analytical version of the McCabe-Thiele method [35] is used to capture the steady-state behavior of a binary column. This model is based on the one described above for the dynamic behavior of a column. The main differences are that the derivatives in (4.2) and (4.1) are zero and (4.4) is not used since the liquid flow rate does not change with time.

In addition to the equations based on the McCabe-Thiele method, the system uses equations that provide estimates for some of the design parameters. Figure 4.41 gives an example of an approximation equation (the *def-First-Guess-Eqn*

```
(def-First-Guess-Eqn Gilland-Fenske-Equation      ;; The name of the equation
 (Num-of-Stages ?column)                          ;; The quantity it estimates
 ;; The activation conditions
 ((Steady-State-Column-Design-Features ?column ?c-stage ?c-phase ?r-stage ?r-phase ?distillate-flow
                                       ?bproduct-flow ?substance1 ?substance2)
 ;; The quantities that take part in the equation
 ((Mole-Fraction ?substance1 (2-C-S (?substance1 ?substance2) ?c-phase ?c-stage))
  (Mole-Fraction ?substance1 (2-C-S (?substance1 ?substance2) ?r-phase ?r-stage))
  (Constant-Alpha ?column))
 ;; The numerical form for the equation
 (= (A (Num-of-Stages ?column))
    (* 2. (/ (log (*  (/ (A (Mole-Fraction ?substance1
                                           (2-C-S (?substance1 ?substance2) ?c-phase ?c-stage)))
                         (- 1. (A (Mole-Fraction
                                   ?substance1
                                   (2-C-S (?substance1 ?substance2) ?c-phase ?c-stage)))))
                      (/ (- 1. (A (Mole-Fraction
                                   ?substance1
                                   (2-C-S (?substance1 ?substance2) ?r-phase ?r-stage))))
                         (A (Mole-Fraction
                             ?substance1
                             (2-C-S (?substance1 ?substance2) ?r-phase ?r-stage))))))
             (log (A (Constant-Alpha ?column)))))))

(def-First-Guess-Parameter (Num-of-Stages ?column)  ;; The parameter predicate
 ;; The conditions under which the parameter representation is valid.
 ((Steady-State-Column-Design-Features ?column ?c-stage ?c-phase ?r-stage ?r-phase ?distillate-flow
                                       ?bproduct-flow ?substance1 ?substance2)
 ;; The unit step for changing the parameter if the estimate is not accurate. The particular entry in this form instructs the system
 ;; to change by one the number of stages in a column when updating an estimate. The direction of change (increase or
 ;; decrease) is determined by the design program.
 1)
```

Figure 4.41: Estimating the number of stages in a binary column.

form). It consists of the same fields as the *defEquation* form with the addition of a slot indicating the quantity it is estimating. The approximation equations are indexed under these quantities. They are used by the system in underconstrained problems when the program is not able to solve for all of the unknown parameters using just the design specifications.

The *def-First-Guess-Parameter* form (Fig. 4.41) is defined for each parameter estimated in an approximation equation. It contains the predicate for the parameter, the conditions under which it holds and a number or a procedure indicating a method for updating the value for the parameter in case the initial estimate is not accurate.

Multicomponent Columns. The analysis of multicomponent columns is based on a simplified version of the Fenske-Underwood-Gilliland method [54] for multicomponent distillation. The major steps are:

1. Specify the splits for all the components. Since we assume that we have sharp separation units, all the nonkey components end up in one of the products. The splits for the keys are assumed to be equal to their desired recoveries in the separation products.

2. Determine the column pressure, the condenser type and the phase of the feed. For simplicity, we assume that all the columns operate at the same pressure, we use partial condensers and the feed for every column is in the liquid phase.

3. Calculate the minimum theoretical stages for the desired separation. This is done using a version of the Gilliland-Fenske equation for multicomponent mixtures [13].

4. Calculate the minimum reflux ratio. The Underwood equation [54] provides a good approximation for this calculation.

Most of these calculations require the relative volatilities between the separation keys and the distillate and product flow rates for the separation scheme. The relative volatilities are computed using a set of equations that calculate physical properties and separation characteristics for various pressure and temperature conditions. Appendix B.1 describes the numerical models for this case.

Finally, the cost estimates for each of the multicomponent columns are generated based on part of the method presented in [49]. Appendix B.2 describes the numerical models for this case.

Although the results we get from this simplified method are not very accurate, they allow us to create flowsheets similar to the ones generated by more accurate versions of the method.

4.6 Model Testing

The model for the dynamic behavior of binary columns was tested in SIMGEN. The particular example was a 20-stage binary distillation column. The machine used was an IBM RS/6000, Model 320, with 64MB of RAM running Lucid Common Lisp. Table 4.1 summarizes some of the statistics for the compilation of the simulator for the distillation model in SIMGEN.

Table 4.1: Statistics for the performance of SIMGEN in the distillation model.

Total Run Time for Qualitative Analysis	1 hr 21 mins
Total Run Time for Code Generation	1 min 48.57 sec
Number of Quantities in the Model	599
Number of active Processes	46
Number of instantiated Views	258

The simulator produced by SIMGEN was tested against a handwritten numerical simulator similar to the one described in [36] for the simulation of an ideal binary distillation column[7]. Both simulators produced the same results.

All the models for the binary columns were used in a system for solving the design problem for binary distillation [35]. The multicomponent model was used in a program for the evolutionary design of multicomponent separation systems. Results for both applications are presented in chapter 6.

4.7 Discussion

There are two criteria for evaluating whether a set of representations for the physical knowledge in a domain can be succesfully used in design:

1. *Expressiveness.* The representations should be able to generate and represent the design alternatives.

2. *Effectiveness.* The knowledge captured in the representations should be able to guide the system in solving design problems.

Physical knowledge representations that are not expressive do not allow the design knowledge to explore the design space systematically. As a result, the system misses interesting designs. Physical knowledge representations that are not effective are not able to meet certain time and space constraints imposed by the

[7] The only difference between our handwritten numerical simulator and the one used in [36] is that the later assumes that the vapor flow rate and the reflux rate in the column are controlled with two PI controllers.

hardware or the designer for generating possible designs. Although design is usually not performed in real-time (as opposed to monitoring for example), the need for effectiveness is justified in terms of the large number of possible solutions for each problem. Creating succinct representations that contain information relevant to the design knowledge is the major task of the physical knowledge component.

The models we presented above satisfy both criteria. In terms of expressiveness, the physical knowledge automatically generates qualitative and numerical models for alternatives that are consistent with the modeling assumptions specified in design. In addition, these models introduce properties (e.g., cost estimates, relative volatility measures) that are relevant to the design knowledge.

In terms of effectiveness, there is significant interaction between the physical and design knowledge in determining the type of analysis for each alternative. For example, in the case of multicomponent mixtures every alternative is initially analyzed at an abstract level by activating model fragments like the perspectives in Figures 4.36 and 4.37. The results of this analysis are used by the design knowledge to identify a set of 'promising' designs. When these are determined, the design knowledge constructs focus environments that activate more detailed models for the remaining alternatives (e.g. the views in Figure 4.39 and their associated numerical models).

Furthermore, the physical knowledge contains information that guides the program through the selection of different design alternatives. For example, in the binary distillation case the relation between the number of stages in the column and the recovery of the volatile component in the distillate (Figure 4.18) guides the system in increasing the number of stages whenever the recovery of the volatile component is below the design specifications for the current column.

In both cases the effectiveness of the analysis phase in design is improved through the explicit representation of the causal relations between the design parameters in the binary model and the ability to dynamically construct models at varying levels of detail in the multicomponent case.

Chapter 5

Representing Design Knowledge in OUZO

5.1 Introduction

Design is the process of creating a description of an artifact that satisfies a set of functional specifications. The term design knowledge refers to a corpus of knowledge that describes the ways design is performed in a domain. This research focuses on computational models of design knowledge. OUZO contains three kinds of design knowledge:

1. *Heuristics*, i.e., rules of thumb.

2. *Strategies.* Plans for optimizing the application of the heuristic rules.

3. *Configuration Synthesis Rules.* Rules for producing the actual artifact descriptions and monitoring the state of design.

In the following sections we describe the interpreter in OUZO that transforms these knowledge types into appropriate rules and implements the actions suggested by the heuristic rules via a set of primitives. In addition, we describe and discuss the design knowledge models for separation system design that are implemented in OUZO.

5.1.1 Heuristics

Figure 5.1 provides an example of the form used to represent the heuristic rules in the system. Each *defHeuristic* form consists of four fields:

- The *Name* field provides a way for assigning a name to a heuristic rule.

```
(defHeuristic Ordinary-vs-Extractive-Distillation  ;; The Name field
   :Class Favor-Distillation
;; The predicates in the Conditions slot are the antecedents of an ATMoSphere rule.
   :Conditions ((Possible (Separation extractive-distillation ?keys1 ?column)) :Var ?f1
                (Possible (Separation distillation ?keys2 ?column)) :Var ?f2)
;; The action slot is essentially the body of the ATMoSphere rule.
   :Action ((prefer ?f2 :Over ?f1)))
```

Figure 5.1: Typical heuristic form. This particular heuristic favors ordinary over extractive distillation for any column.

- The *:Class* field contains the name of the heuristic class in which the rule belongs. Classes provide a convenient way of grouping heuristics according to the physical principles they are based on, or by the general design heuristics they instantiate. They are used by the strategies to retrieve the heuristic rules they contain.
- The *:Conditions* field contains a set of conditions that must hold for the rule to fire.
- The *:Action* field contains the body of the heuristic rule.

Heuristic rules along with the rest of the design knowledge are implemented as rules that are fed into an assumption-based truth maintenance system (ATMS) [9]. A rule engine (ATMoSphere) [17] provides the interface to the ATMS. All of the heuristics are translated into ATMoSphere rules. The conventions used in ATMoSphere are explained in [20]. When necessary we use LISP to create new assertions.

5.1.2 The Interpreter Primitives

The design interpreter in `OUZO` contains thirteen primitives for implementing the actions suggested by the heuristic rules. These primitives are commands to the interpreter describing how to update the set of design choices. This section describes their syntax and implementation.

1. **Prefer.** Establish an order of preference between two design alternatives.
   - `Syntax`: (Prefer ?x :Over ?y :Rank ?z), where ?x and ?y are predicates and ?z is a number.
   - `Implementation`: Justifies the predicate (Prefer ?x :From ?y ?z) in terms of ?x and ?y in the ATMS. The rank typically represents the importance of the heuristic that establishes the preference. The interpretation of this number depends on the design strategy.
2. **Propose-Value.** Assigns a numerical value to a parameter.
   - `Syntax`: (Propose-Value ?quant ?val), where ?quant is a parameter and ?val is a numerical value.
   - `Implementation`: Asserts the predicate (Value-of (A ?quant) ?val :Propose) in the ATMS.

3. **Assert-in-Design.** Asserts that an alternative holds for the rest of design.
    - `Syntax`: (Assert-in-Design ?a), where ?a is a predicate.
    - `Implementation`: Inserts the predicate (assertq ?a) in the *scenario* list which contains the current design description and updates the focus environment.

4. **Assume-in-Design.** Assumes that an alternative holds for the rest of design.
    - `Syntax`: (Assume-in-Design ?a), where ?a is a predicate.
    - `Implementation`: Inserts the predicate (assume ?a) in the *scenario* list and updates the focus environment.

5. **Invalidate-Decision.** Do not consider the design alternative specified in its argument for the rest of the design process. It is used to retract design decisions made using the assert-in-design primitive.
    - `Syntax`: (Invalidate-Decision ?x), where ?x is a predicate.
    - `Implementation`: Asserts in *scenario* the predicate (No-Good ?x) which precludes the design heuristics from considering ?x for the rest of the design process and removes ?x from the design description.

6. **Assume-in-Cycle.** Assume that a design decision holds for the current design cycle.
    - `Syntax`: (Assume-in-Cycle ?x), where ?x is a predicate.
    - `Implementation`: Assumes ?x in the ATMS and includes it in the current focus environment.

7. **Cover-Specifications.** Indicate that all the design specifications are satisfied.
    - `Syntax`: (Cover-Specifications).
    - `Implementation`: Set the *design-success* variable to true.

8. **Reject.** Rejects a design alternative or a parameter estimate from further consideration during the current design cycle. It is used to reject assumptions made using the assume-in-cycle primitive.
    - `Syntax`: (Reject ?x), where ?x is a predicate.

- `Implementation`: Make ?x a nogood node in the ATMS and removes it from the current focus environment.

9. **Consult-User.** Asks the user to select between alternatives.

   - `Syntax`: (Consult-User ?a1 ?a2), where ?a1 and ?a2 are predicates.
   - `Implementation`: It asks the user to select between ?a1 and ?a2 and then it follows the same procedure with the assert-in-design primitive for the chosen alternative.

10. **Examined-Alternative.** Returns true if an alternative has been already examined by the program.

    - `Syntax`: (Examined-Alternative ?x), where ?x is a predicate.
    - `Implementation`: Checks whether ?x is a member in the *alternatives-checked* list which contains all the designs examined by the system so far. If it is not, it inserts ?x in this list.

11. **Exists.** Returns true if its argument is part of the current design description.

    - `Syntax`: (Exists ?x), where ?x is a predicate.
    - `Implementation`: Checks whether ?x is a member of the *scenario* list.

12. **Store-Design.** Stores the current design description.

    - `Syntax`: (Store-Design)
    - `Implementation`: Pushes the contents of the *scenario* variable to the *previous-designs* list.

13. **Pop-Design.** Reinstates the most recently stored design description.

    - `Syntax`: (Pop-Design)
    - `Implementation`: Pops the *previous-designs* list.

These primitives are used in conjunction with two standard ATMS primitives; *assume* which assumes a node in the ATMS and *assert* which asserts a node in the ATMS.

```
(defHeuristic Heuristic-0
  :Class Separation-Alternatives-Representation.
  :Conditions ((Propose (Possible (Separation distillation (?l-k ?h-k) ?column))))
  :Action ((assume '(Possible (Separation distillation (,?l-k ,?h-k) ,?column)))))

 IF an ordinary distillation process has been proposed by the qualitative domain theory
 THEN assume this separation as a possible separation during the heuristic analysis.
```

Figure 5.2: Feed the ordinary distillation alternatives in the heuristic analysis.

5.1.3 The Heuristic Library

The heuristic library in OUZO contains heuristics for the design of separation systems for multicomponent mixtures along with heuristics for binary distillation design.

Designers for multicomponent separation systems use a set of nineteen general heuristics to construct an initial separation flowsheet (see Table 5.1 taken from [45]). In addition to these heuristics, evolutionary design methods use their own sets of heuristics for refining the initial structure. The heuristic library in OUZO contains fourteen general heuristics along with nine evolutionary heuristic rules for multicomponent separation systems. Furthermore, this library contains ten heuristics for the binary distillation design problem.

The rest of the section describes the representations for these rules. Each one of the figures below, contains the actual encoding for a heuristic along with a piece of text summarizing its content.

#	**Heuristic Rule**
1	Remove components one by one as overhead products $\leftarrow$
2	Save the most difficult separation for last $\leftarrow$
3	Favor 50-50 splits $\leftarrow$
4	Sequence with the minumum total vapor flow
5	Make high recovery fractions last
6	Separate the most plentiful components first $\leftarrow$
7	Choose the cheapest as the next separator
8	Remove the thermally unstable and corrosive material early.
9	Disregard separations with very small relative volatility between the keys. $\leftarrow$
10	Perform least-tight separation first $\leftarrow$
11	Favor the smallest production set $\leftarrow$
12	Avoid separations using a mass separating agent (MSA). $\leftarrow$
13	Remove a MSA from one of the products in another, subsequent separation process $\leftarrow$
14	A separation method using a MSA cannot be used to isolate another MSA $\leftarrow$
15	Favor distillation $\leftarrow$
16	Separate first the components which might undergo undesirable reactions
17	Set split fractions of the key components to prespecified values $\leftarrow$
18	Avoid extreme operating conditions $\leftarrow$
19	Favor ambient operating pressure $\leftarrow$

Table 5.1: The major heuristic rules for establishing the initial separation structure. The arrows indicate the rules supported by the Nath & Motard and the Seader & Westerberg evolutionary strategies.

```
(defHeuristic Order-Choices
  :Class Order-Choices
  :Conditions ((Possible (Separation ?method ?keys ?column)) :Var ?f1
               :Test (most-preferable? ?method ?keys ?column))
  :Action ((assert-in-design
            '(Consider (Possible (Separation ,?method ,?keys ,?column))))))

 IF there exists a viable separation alternative (a) for the current column
    AND there is no other alternative designated as more preferable than (a)
 THEN assert (a) as the separation taking place in the column
```

Figure 5.3: Determine the separation for the current column.

5.1.3.1 Separation Systems for Multicomponent Mixtures

Represent the separation alternatives

Represent the ordinary distillation alternatives. This rule provides the interface between the qualitative domain theory and the design heuristics (Fig. 5.2). In particular, the predicates in the Conditions part of the rule represent all the possible ordinary distillation schemes that were computed using the qualitative domain theory. They are introduced by the Approximate-Distillation-Features perspectives in Figures 4.36 and 4.37. The Action part of the rule constructs the representations for the alternative separation schemes that are going to be evaluated by the rest of the heuristics. In both strategies extractive distillation alternatives are introduced as needed by the rest of the heuristic rules.

Decide on the separation task

Determine the separation for the current column. This rule (Figure 5.3) is used to inform the rest of the design system that the heuristic analysis has decided on the separation scheme that is going to take place in the current column. The *most-preferable?* procedure examines the partial order established by the prefer primitive to find whether the current alternative is the best.

```
(defHeuristic Product-Set-Selection
  :Class Favor-Smallest-Production-Set
  :Conditions ((Column-Feed (M-C-S ?components ?phase ?feed-stage) ?column)
               (Reference-Conditions ?column (Pressure ?pressure) (Temp ?t))
               (Possible (Separation ?method (?l-k ?h-k) ?column)) :Var ?f1
               :Test (null (desired-products? ?l-k ?h-k ?components ?column ?pressure))
               (Possible (Separation ?method (?l-k-1 ?h-k-1) ?column)) :Var ?f2
               :Test (desired-products? ?l-k-1 ?h-k-1 ?components ?column ?pressure))
  :Action ((reject ?f1)))
```

IF the components of the feed for the current column are known
  AND the value of the reference pressure for the current column is given
  AND there is one separation alternative (a)
  AND neither the distillate nor the bottoms products for (a) are equal to any of the desired products for the column in reference conditions
  AND there is another separation alternative (b)
  AND either the distillate or the bottoms products for (b) are equal to some of the desired products for the column in reference conditions
THEN reject (a) as a possible separation alternative.

Figure 5.4: The heuristic rule that favors the smallest production set.

General Design Heuristics (Table 5.1)

In the list below, each heuristic class is indexed using the name and the number under which it is presented in Table 5.1.

Favor the smallest production set (11). The rule is presented in Figure 5.4. It rejects any separation which does not generate any of the desired products, provided that there is at least one possible separation scheme that does. The procedure *desired-products?* in Figure 5.4 takes three arguments. The first two specify the light and heavy keys for a separation. The third one is a list of the substances that are fed to the column while the fourth is the reference pressure on which the product calculations are based. The procedure returns true, if any of the products of the separation scheme it examines, belongs to the set of desired products in the design specifications.

Avoid extreme operating conditions (18). Figure 5.5 describes the representation for this rule for both ordinary and extractive distillation cases. The Operating-Reflux-Estimate quantity provides us with an estimate for the ratio between the operating reflux flow rate and the minimum reflux flow rate for a column. This is an important operating feature since it influences the product recoveries and the diameter of the column for a given separation. This quantity is introduced by the Distillation-Column entity in the qualitative model (Fig. 4.22).

```
(defHeuristic Operating-Reflux-Estimate
 :Class Avoid-Extreme-Operating-Conditions
 :Conditions ((Quantity (Operating-Reflux-Estimate ?column)))
 :Action ((propose-value '(A (Operating-Reflux-Estimate ,?column)) 1.3)))

 IF the ratio between the operating reflux rate and the minimum reflux rate for a column
    has been introduced
 THEN define its value to be 1.3.
```

Figure 5.5: Avoid extreme operating conditions in the column.

The particular numerical value for the estimate (1.3) is part of the Nath & Motard method [43].

Disregard separations with very small relative volatilities between the keys (9). A small relative volatility between the key components in distillation means that there is no significant difference between the evaporation rates for these components at various temperature and pressure conditions. Consequently, distillation is an expensive process for separating these components due to the large number of stages that are necessary. Figure 5.6 presents the implementation of this rule for the reference conditions in a column. The Alpha-LK-HK quantity in both figures stands for the relative volatility between the key components in a separation scheme. The heuristics in Figure 5.6 reject any possible separation scheme with relative volatility between the keys less than the minimum value defined in heuristic Estimate-Min-Relative-Volatility.

Favor distillation (15). The form in Figure 5.8 is used to capture this heuristic rule. For every column it suggests ordinary to extractive distillation as a more promising alternative. Extractive distillation processes increase the number of columns in a separation system since the mass separating agent they use has to be recycled in separate columns. This is why ordinary distillation is prefered in general.

Avoid separations using a mass separating agent (MSA) (12). In addition to favoring distillation processes the heuristic rule in Figure 5.8 gives a low priority to extractive distillation operations.

Perform least-tight separation first (10). Different methods provide different interpretations for this heuristic rule, since what is a tight separation depends on the criteria used. Figure 5.9 presents the interpretation of this rule in the Nath & Motard strategy. In this case a numerical formula that combines a

```
;; This heuristic is used in the Seader & Westerberg method too.
(defHeuristic Estimate-Min-Relative-Volatility
 :Class Disregard-Small-Relative-Volatilities
 :Conditions ((Distillation-Column ?column))
 :Action ((assume `(Consider (Min-Relative-Volatility ,?column ,*min-volatility*)))))

 IF there exists a distillation column
 THEN assume that the minimum relative volatility for the column is equal to *min-volatility*.

(defHeuristic Separation-Feasibility-in-Atm-Conditions-1
 :Class Disregard-Small-Relative-Volatilities
 :Conditions ((Possible (Separation distillation (?light-key ?heavy-key) ?column)) :Var ?f1
             (Column-Feed (M-C-S ?components ?phase ?stage) ?column)
             (Value-of (A (Alpha-LK-HK ?light-key ?heavy-key ?column :reference)) ?alpha ?eq-1)
             (Consider (Min-Relative-Volatility ?column ?alpha-min))
             :Test (and (> ?alpha-min ?alpha) (associates-with? ?p-s ?light-key)))
 :Action
 ((reject ?f1)
  (when (exists `(Value-of (A (Extractive-Alpha-LK-HK ,?heavy-key ,?light-key . ?r1)) . ?r2))
     (assume `(Possible (Separation extractive-distillation (,?heavy-key ,?light-key) ,?column))))))

(defHeuristic Separation-Feasibility-in-Atm-Conditions-2
 :Class Disregard-Small-Relative-Volatilities
 :Conditions ((Possible (Separation distillation (?light-key ?heavy-key) ?column)) :Var ?f1
             (Column-Feed (M-C-S ?components ?phase ?stage) ?column)
             (Value-of (A (Alpha-LK-HK ?light-key ?heavy-key ?column :reference)) ?alpha ?eq-1)
             (Consider (Min-Relative-Volatility ?column ?alpha-min))
             :Test (and (> ?alpha-min ?alpha) (associates-with? ?p-s ?heavy-key)))
 :Action
 ((reject ?f1)
  (when (exists `(Value-of (A (Extractive-Alpha-LK-HK ,?light-key ,?heavy-key . ?r1)) . ?r2)))
     (assume `(Possible (Separation extractive-distillation (,?light-key ,?heavy-key) ,?column)))))

 IF there is a possible separation alternative for the current column
    AND the value for the relative volatility between the key components under reference conditions
          is less than the minimum relative volatility assumed for the column
    AND a polar solvent can associate either with the light or the heavy keys
 THEN reject the current separation alternative and pursue extractive distillation alternatives

(defHeuristic Extractive-Separation-Feasibility-in-Reference-Conditions
 :Class Disregard-Small-Relative-Volatilities
 :Conditions ((Possible (Separation extractive-distillation (?l-k ?h-k) ?column)) :Var ?f1
             (First-Column ?system ?column)
             (Value-of (A (Extractive-Alpha-LK-HK ?l-k ?h-k ?polar-solvent ?column :reference))
                       ?alpha ?eq-1)
             (Consider (Min-Relative-Volatility ?column ?alpha-min))
             :Test (> ?alpha-min ?alpha))
 :Action ((reject ?f1)))

 IF there is an extractive distillation alternative for the current column
    AND and the relative volatility between the keys in reference conditions is less than
         the minimum relative volatility for the current column
 THEN reject the extractive distillation alternative.
```

Figure 5.6: Disregard separations with very small relative volatility between the keys. The rules presented correspond to reference conditions for the column. There is an analogous set of rules for the actual operating conditions for the column.

```
(defHeuristic Difficulty-of-Separation-1
 :Class Perform-Least-Tight-Separation-First
 :Conditions ((Possible (Separation ?method (?l-k-1 ?h-k-1) ?column)) :Var ?f1
             (Value-of (A (Alpha-LK-HK ?l-k-1 ?h-k-1 ?column :reference)) ?a-lk-hk-1 ?eq-1)
             (Possible (Separation ?method (?l-k-2 ?h-k-2) ?column)) :Var ?f2
             (Value-of (A (Alpha-LK-HK ?l-k-2 ?h-k-2 ?column :reference)) ?a-lk-hk-2 ?eq-2)
             :Test (and (> ?a-lk-hk-1 ?a-lk-hk-2)
                        (> (- ?a-lk-hk-1 ?a-lk-hk-2) *alpha-difference*)))
 :Action ((prefer ?f1 :Over ?f2)))

 IF there is a separation alternative (a) for the current column
    AND the relative volatility between the keys in (a) in reference conditions is known
    AND there is another separation alternative (b)
    AND the relative volatility between the keys in (b) in reference conditions is known
    AND there is a significant difference (> *alpha-difference*)
          between the relative volatilities of (a) and (b)
 THEN prefer the separation alternative with the largest value for the relative volatility.

(defHeuristic Difficulty-of-Separation-2
 :Class Perform-Least-Tight-Separation-First
 :Conditions ((Possible (Separation ?method (?l-k-1 ?h-k-1) ?column)) :Var ?f1
             (Value-of (A (Alpha-LK-HK ?l-k-1 ?h-k-1 ?column :reference)) ?a-lk-hk-1 ?eq-1)
             (Possible (Separation ?method (?l-k-2 ?h-k-2) ?column)) :Var ?f2
             (Value-of (A (Alpha-LK-HK ?l-k-2 ?h-k-2 ?column :reference)) ?a-lk-hk-2 ?eq-2)
             :Test (and (> ?a-lk-hk-1 ?a-lk-hk-2)
                        (<= (- ?a-lk-hk-1 ?a-lk-hk-2) *alpha-difference*)))
 :Action ((consult-user ?f1 ?f2)))

 IF there is a separation alternative (a) for the current column
    AND the relative volatility between the keys in (a) in reference conditions is known
    AND there is another separation alternative (b)
    AND the relative volatility between the keys in (b) in reference conditions is known
    AND there is not a significant difference (< *alpha-difference*) between the
          relative volatilities of (a) and (b)
 THEN let the user decide which alternative is more preferable.

(defHeuristic Ordinary-vs-Extractive-Distillation
 :Class Perform-Least-Tight-Separation-First
 :Conditions ((Consider (Evolutionary-Strategy-For ?s Seader-&-Westerberg))
             (Column-Feed (M-C-S ?components ?phase ?stage) ?column)
             (Possible (Separation distillation (?lk-1 ?hk-1) ?column)) :Var ?f1
             (Value-of (A (Alpha-LK-HK ?lk-1 ?hk-1 ?column :reference)) ?a-1 ?eq-1)
             :Test (> ?a-1 2.0)
             (Possible (Separation extractive-distillation (?lk-2 ?hk-2) ?column)) :Var ?f2)
  ((prefer ?f1 :Over ?f2)))

 IF the Seader & Westerberg strategy is used
    AND the feed to the current column ?column is known
    AND there is an ordinary distillation alternative (a) with relative volatility betwwen the keys
          greater than 2 under reference conditions
    AND there is an extractive distillation alternative (b)
 THEN mark (a) as better than (b).
```

Figure 5.7: The forms in the figure combine representations for three heuristics: (i) Disregard separations with small relative volatility between the keys (ii) Perform least-tight separation first and (iii) Save the most difficult separation for last.

```
(defHeuristic Ordinary-vs-Extractive-Distillation
  :Class Favor-Distillation
  :Conditions ((Possible (Separation extractive-distillation ?keys-1 ?column)) :var ?f1
              (Possible (Separation distillation ?keys-2 ?column)) :var ?f2)
  :Action ((prefer ?f2 :Over ?f1)))

IF there is an extractive distillation alternative for the current column
   AND there is an ordinary distillation alternative for the current column
THEN prefer the ordinary distillation alternative.
```

Figure 5.8: Two heuristic rules are represented by the form in the figure: (i) Favor distillation (ii) Avoid separations using a mass separating agent (MSA).

set of criteria[1] for the tightness of a separation scheme is used [43]. The formula calculates what is called the coefficient of the difficulty of a separation (CDS) for a separation scheme. The lower the CDS value for an alternative the more promising this appears to be.

The Seader & Westerberg method uses the relative volatility between the key components as a measure of the tightness of a separation scheme. The heuristics in Figure 5.7 order the separation alternatives according to the values of the relative volatility between the key components. Separation schemes with higher relative volatilities are considered more promising.

Save the most difficult separation for last (2). Difficult means tight, therefore the heuristics based on the CDS function (Fig. 5.9) along with the ones using the relative volatility between the keys (Fig. 5.7), schedule the more difficult separations towards the end of the separation system.

A separation method using a MSA cannot be used to isolate another MSA (14). In other words the solvent selected for the extractive distillation should be easily separated from the products of the process. Otherwise the number of columns in the separation system will increase with a subsequent increase in the cost of the system. Figure 5.10 contains the representation for this heuristic rule. The form *(No-Good (Possible (Separation ....)))* in the figure represents possible separation alternatives that were already examined in some previous design cycle.

[1]The set of criteria include [43]: (i) Favor separations with large relative volatilities between the keys (ii) Favor balanced columns (i.e. columns in which the flow rates for the distillate and the bottoms products are approximately the same) (iii) Favor sloppy splits for the keys (i.e. separations that do not require high recoveries for the keys) (iv) Favor separations with low distillate product flow rate.

```
(defHeuristic Compare-CDS
 :Class Perform-Least-Tight-Separation-First
 :Conditions ((Consider (Evolutionary-Strategy-For ?system Nath-&-Motard))
              (Possible (Separation ?method (?l-k-1 ?h-k-1) ?column)) :var ?f1
              (Value-of (A (CDS ?method ?column (?l-k-1 ?h-k-1) :reference)) ?cds-1 ?eqn-1)
              (Possible (Separation ?method (?l-k-2 ?h-k-2) ?column)) :var ?f2
              (Value-of (A (CDS ?method ?column (?l-k-2 ?h-k-2) :reference)) ?cds-2 ?eqn-2)
              :Test (and (< ?cds-1 ?cds-2) (> ?cds-1 0.0)))
 :Action ((prefer ?f1 :Over ?f2)))

 IF the Nath & Motard evolutionary strategy is used
    AND there exists a separation alternative (a)
    AND the CDS value for (a) is known
    AND there is another separation alternative (b) with the same separation method as (a)
    AND the CDS value for (b) is known
    AND both CDS functions are positive
    AND the CDS value for (a) is less than the CDS value for (b)
 THEN prefer separation alternative (a)

(defHeuristic Reject-Negative-CDS
 :Class Perform-Least-Tight-Separation-First
 :Conditions ((Consider (Evolutionary-Strategy-For ?system Nath-&-Motard))
              (Possible (Separation ?method (?l-k-1 ?h-k-1) ?column)) :var ?f1
              (Value-of (A (CDS ?method ?column (?l-k-1 ?h-k-1) :reference)) ?cds-1 ?eqn-1)
              :Test (< ?cds-1 0.0)
              (Column-Feed (M-C-S ?components ?phase ?stage) ?column)
              (Value-of (A (Alpha-LK-HK ?l-k-1 ?h-k-1 ?column :reference)) ?alpha . ?y)
              :Test (< ?alpha *max-alpha*))
 :Action ((reject ?f1)))

 IF the Nath & Motard evolutionary strategy is used
    AND there is a possible separation alternative
    AND the coefficient of the difficulty of the separation for this alternative is negative
    AND the relative volatility between the keys in this separation scheme is less than *max-alpha*
 THEN unless there is a good reason to reject a simple distillation process by which the polar
        solvent is isolated reject the extractive distillation scheme as an alternative.
```

Figure 5.9: Perform least tight separation first. Usually a good reason for rejecting a simple distillation process in the second heuristic rule is the activation of evolutionary heuristics that challenge the separations proposed by this heuristic.

```
(defHeuristic MSA-Agent-Isolation
 :Class MSA-Agent-Isolation
 :Conditions ((Polar-Solvent ?p-s)
             (Column-Feed (M-C-S ?components ?phase ?stage) ?column)
             :Test (member ?p-s ?components)
             (Possible (Separation extractive-distillation ?keys ?column)) :Var ?f1)
 :Action ((when (not (exists '(No-Good (Possible (Separation distillation (?x ,?p-s) ,?column)))))
            (reject ?f1))))

IF there is a polar solvent which is part of the feed for the current column
   AND an extractive distillation process is an alternative for the current column
THEN unless there is a good reason to reject a simple distillation process by which the polar
       solvent is isolated reject the extractive distillation scheme as an alternative.
```

Figure 5.10: A separation method using a MSA cannot be used to isolate another MSA.

It is asserted by the evolutionary heuristics whenever they conclude that a design choice should not be duplicated for a particular column in a new design.

Remove a MSA from one of the products in another, subsequent separation process (13). This heuristic ensures that one of the alternatives explored after an extractive distillation column is a column in which the mass separating agent is removed and recycled. Figure 5.11 contains the representation for this heuristic. The two heuristics implement different design policies. The Reject-non-MSA-Removal-Splits rule eliminates from the rest of the analysis separations that do not isolate the MSA. The Favor-MSA-Removal rule gives a low priority to these splits.

Favor 50-50 splits (3). When the amounts of the distillate and bottom products can be made approximately the same we have a 50-50 split. This heuristic is taken into consideration during the construction of the CDS function (Figure 5.9).

Set split fractions of the key components to prespecified values (17). This rule provides specifications for the operating conditions for a given separation scheme. Figure 5.12 describes the representation for this heuristic. The Product-Specification quantity represents the desired recovery for a product of the separation system. The Product-Recovery quantity is the actual recovery used in the column that belongs to this separation system. This rule equates the two.

Remove components one by one as overhead products (1). The separation system that results from a repeated application of this heuristic is called the

```
(defHeuristic Reject-non-MSA-Removal-Splits
  :Class Remove-MSA-From-Products-in-Subsequent-Operation
  :Conditions ((Mass-Separating-Agent ?p-s ?prev-column)
               (Connects-to ?prev-column ?system)
               (First-Column ?system ?column)
               (Column-Feed (M-C-S ?components ?phase ?stage) ?column)
               :Test (member ?p-s ?components)
               (Possible (Separation ?method (?l-k ?h-k) ?column)) :Var ?f1
               :Test (not (polar-solvent ?h-k)))
  :Action ((reject ?f1)))
```

IF a mass separating agent was used in some previous column
   AND the previous column is connected to a separation system
   AND the separation system is connected to the current column
   AND the current column takes a multicomponent mixture as feed
   AND the mass separating agent is one of the components of the feed
   AND a possible separation scheme is proposed for the current column
   AND the heavy key is not a polar solvent in this scheme
THEN reject this separation scheme.

```
(defHeuristic Favor-MSA-Removal
  :Class Remove-MSA-From-Products-in-Subsequent-Operation
  :Conditions ((Mass-Separating-Agent ?p-s ?prev-column)
               (Connects-to ?prev-column ?system)
               (First-Column ?system ?column)
               (Column-Feed (M-C-S ?components ?phase ?f-stage) ?column)
               :Test (member ?p-s ?components)
               (Possible (Separation distillation (?substance ?p-s) ?column)) :Var ?f1
               (Possible (Separation distillation (?lk ?hk) ?column)) :Var ?f2
               :Test (not (eql ?hk ?p-s)))
  :Action
  (unless (exists (No-Good (Possible (Separation distillation (?substance ?p-s) ?column))))
     (prefer ?f1 :Over ?f2)))
```

IF a mass separating agent was used in some previous column
   AND the previous column is connected to a separation system
   AND the separation system is connected to the current column
   AND the current column takes a multicomponent mixture as feed
   AND the mass separating agent is one of the components of the feed
   AND there is a separation alternative (a) in which the mass separating agent is the heavy key
   AND there is a separation alternative (b) in which the mass separating agent
         is not the heavy key
THEN unless there is some good reason for rejecting (a), prefer (a) over (b).

Figure 5.11: Remove a MSA from one of the products in another, subsequent separation process. A good reason for rejecting the separation proposed by the second rule is the activation of evolutionary heuristics that challenge this rule.

```
(defHeuristic Key-Split-Fractions
 :Class Set-Split-Fractions-to-Specified-Values
 :Conditions ((First-Column ?system ?column)
              (Quantity (Product-Recovery ?product ?column (?l-k ?h-k)))
              :Test (member ?product '(,?l-k ,?h-k))
              (Value-of (A (Product-Specification ?product ?system)) ?spec . ?x))
 :Action ((propose-value '(A (Product-Recovery ,?product ,?column (,?l-k ,?h-k)))) ?spec))

 IF a column is part of a separation system
    AND there exists a quantity that describes the recovery of a product in this column
    AND the above product is one of the keys
    AND the value for the desired product specification for this key in the separation
        system is known
 THEN set the value of the product recovery for the key in the column to the value of the
       product specification for this key in the separation system.
```

Figure 5.12: Set split fractions of the key components to prespecified values.

```
(defHeuristic Favor-Directed-Sequences
 :Class Remove-Components-as-Overhead-Products
 :Conditions ((Column-Feed (M-C-S ?components ?feed ?stage) ?column)
              (Reference-Conditions ?column (Pressure ?ref-pres) (Temp ?t))
              (Possible (Separation ?method (?l-k ?h-k) ?column)) :Var ?f1
              :Test (only-overhead-product? ?l-k ?l-k ?h-k ?components ?ref-pres)
              (Possible (Separation ?method-1 (?l-k-1 ?h-k-1) ?column)) :Var ?f2
              :Test (null (only-overhead-product? ?l-k-1 ?l-k-1 ?h-k-1 ?components ?ref-pres)))
 :Action ((reject ?f2)))

 IF we know the feed to the current column
    AND we know the value for its reference pressure
    AND there is a separation alternative (a)
    AND the distillate product for (a) consists on only the light key
    AND there is another separation alternative (b)
    AND (b) has a mixture of components as its distillate product
 THEN reject alternative (b)
```

Figure 5.13: Remove components one by one as overhead products.

```
(defHeuristic Set-Operating-Pressure
 :Class Favor-Ambient-Operating-Pressure
 :Conditions ((Consider (Possible (Separation ?method (?l-k ?h-k) ?column)))
              (Value-of (A (Atmospheric-Pressure ?env)) ?atm-pres . ?eq)
              (Quantity (Operating-Pressure ?column)))
 :Action ((propose-value '(A (Operating-Pressure ,?column)) ?atm-pres)))

 IF the heuristic analysis has decided on a particular separation for the current column
    AND the value of the atmospheric pressure for it is known
    AND the quantity describing the operating pressure for the column has been introduced
 THEN set the value for the operating pressure for the current column to the atmospheric pressure
```

Figure 5.14: Favor ambient operating pressure.

direct sequence. This sequence is widely applied in practice since in the absense of more efficient heuristics it usually results in a relatively good design. Figure 5.13 contains the representation for this rule.

Favor ambient operating pressure (19). When developing an initial separation flowsheet a good approximation to use is to consider the operating pressure for each separation unit to be close to the atmospheric pressure. Figure 5.14 contains the representation for this heuristic.

Separate the most plentiful components first (6). In cases where the relative volatilities between the components are almost the same but their molar percentage in the feed varies widely it is a good strategy to remove the most plentiful components first. Figure 5.15 contains the description for this rule.

Evolutionary Heuristics in the Nath & Motard method

Table 5.2 taken from [43] describes the evolutionary heuristics used in the Nath & Motard design strategy. In the following we describe the representations for each one of these rules.

Challenge Heuristic 11[2]. Sometimes in order to favor the smallest product set the designer is forced to introduce extractive distillation columns in the sequence. This heuristic (Fig. 5.16) explores other possiblities by splitting the product set that made the use of extractive distillation necessary in the initial separation sequence.

Examine the neighboring structures if they involve separations of similar difficulty. The problem with the criteria for describing the difficulty of

[2] This is the row in which the heuristic rule that is being challenged appears in Table 5.1

```
(defHeuristic Separate-Most-Plentiful-Components-First
  :Class Separate-Most-Plentiful-Components-First
  :Conditions ((Value-of
                (A (Feed-Flow ?elem1 (M-C-S ?components ?phase ?f-stage) ?column)) ?f-flow-1 ?eq-1)
               (Value-of
                (A (Feed-Flow ?elem2 (M-C-S ?components ?phase ?f-stage) ?column)) ?f-flow-2 ?eq-2)
               :Test (and (not (eql ?elem1 ?elem2)) (> ?f-flow-1 ?f-flow-2))
               (Value-of (A (Alpha-LK-HK ?elem1 ?h-k ?column :reference)) ?alpha-1 ?eq-3)
               (Value-of (A (Alpha-LK-HK ?elem2 ?h-k ?column :reference)) ?alpha-2 ?eq-4)
               :Test (or (< (/ ?alpha-1 ?alpha-2) *min-volatility-ratio*)
                         (< (/ ?alpha-2 ?alpha-1) *min-volatility-ratio*))
               (Possible (Separation ?method ?keys-1 ?column)) :Var ?f1
               :Test (member ?elem1 ?keys-1)
               (Possible (Separation ?method ?keys-2 ?column)) :Var ?f2
               :Test (member ?elem2 ?keys-2))
  :Action ((prefer ?f1 :Over ?f2)))
```

IF the feed flow rate for *?elem1* is greater than the flow rate for *?elem2* in the same column
AND and their relative volatilities are almost the same
AND there is a separation method (a) with *?elem1* as one of its keys
AND there is a separation method (b) with *?elem2* as one of its keys
THEN prefer (a) over (b).

Figure 5.15: Separate the most plentiful component first.

#	**Evolutionary Rule**
1	Challenge Heuristic 11
2	Examine the neighboring structures if separations of similar difficulty take place in them.
3	Challenge Heuristic 15
4	Examine neighbors to decide if the MSA removal should be delayed
5	Challenge Heuristic 10 if in the current design an easy separation is followed by a very difficult one.

Table 5.2: The evolutionary rules for the Nath & Motard design method.

```
(defHeuristic Split-Product-Set
  :Class Challenge-Heuristic-11
  :Conditions ((Consider (Design-Complete ?system ?total-cost)) :Var ?f0
               :Test (original-product-set? ?system)
               (First-Column ?system ?first-column)
               (Consider (Possible (Separation extractive-distillation (?l-k ?h-k) ?column)))
               (Desired-Products ?products ?column)
               :Test (and (> (length ?products) 1)
                          (not (product-split-for ?products)))
               (Column-Products ?products ?keys ?next-column)
               (Connects-to ?column ?extr-system)
               (First-Column ?extr-system ?next-column)
               (Value-of (A (Product-Specification ?products ?system)) ?recovery ?eqn-name)
               :Test (setq alternative-product-set (find-alternative-products ?products)))
  :Action ((assert-in-design
            '(Establish-New-Product-Set ,alternative-product-set ,?products ,?recovery ,?system))
           (assert '(Evolve-Design-From ,?first-column))))
```

IF a design for the separation problem has been found with the original product set
AND the first column in this design is *?first-column*
AND there is an extractive distillation process taking place in some column *?column* in the design
AND the desired product sets for ?column is *?products*
AND *?products* has more than one components
AND *?products* is not the result of some product split from the original design problem
AND *?products* was actually one of the products for the extractive distillation group of columns
AND the desired product recovery for *?products* is known
THEN (i) Find a separation scheme with key components belonging to the *?products* list and with the highest relative volatility of all the alternatives
(ii) Reformulate the design problem to include the product set for the new separation scheme instead of the old one
(iii) Start the design process from scratch.

Figure 5.16: Challenge Heuristic 11.

```
(defHeuristic Challenge-CDS-Accuracy
  :Class Examine-Close-Estimates
  :Conditions ((Consider (Design-Complete ?system ?total-cost)) :Var ?f0
               (Consider (Possible (Separation ?method (?l-k-1 ?h-k-1) ?column))) :Var ?f1
               (Connects-to ?column ?next-system)
               (First-Column ?next-system ?next-column)
               (Consider (Possible (Separation ?method (?l-k-2 ?h-k-2) ?next-column)))
               (Value-of (A (CDS ?method ?column (?l-k-1 ?h-k-1) :reference)) ?cds-1 ?eqn-1)
               (Value-of (A (CDS ?method ?next-column (?l-k-2 ?h-k-2) :reference)) ?cds-2 ?eqn-2)
               :Test (and (< (abs (- ?cds-1 ?cds-2)) *min-cds-difference*)
                          (downstream-from-evolution ?column 2)))
  :Action
  ((unless (examined-alternative
            '((,?method (,?l-k-1 ,?h-k-1)) (,?method (,?l-k-2 ,?h-k-2))))
     (invalidate-decision '(Possible (Separation ,?method (,?l-k-1 ,?h-k-1) ,?column)))
     (assume-in-cycle '(Evolve-Design-From ,?column ,?f0 ,?f1))
     (assert-in-design '(Evolution-Point ,?column 2))
     (assert-in-design
      '(Consider (Possible (Separation ,?method (,?l-k-2 ,?h-k-2) ,?column)))))))
```

IF a design for the separation problem has been found
   AND a separation method (a) is used in column *?column*
   AND *?column* is connected to *?next-column*
   AND a separation method (b) is used in column *?next-column*
   AND the difference between the CDS values between the two methods
      is less than *min-cds-difference*
   AND *?column* is downstream from the latest evolution point
THEN retain the old design description up to the *?column* intact and try to evolve the rest of it
      assuming that separation method (b) is used in *?column*.

Figure 5.17: Examine the neighboring structures if the values of the approximations used to describe their difficulty are close to each other.

a separation is that they are not able to provide accurate estimates for design alternatives that are almost equivalent. This heuristic tries to account for the reduced accuracy in those cases. One of these criteria is the coefficient of the difficulty of separation (CDS) that we mentioned above. The rule in Figure 5.17 looks for separation schemes in neighboring columns with very close CDS values. It then tries to evolve an alternative design in which the second separation scheme takes place in the first column.

The procedure *downstream-from-evolution* returns true when the current column is downstream from every point in the design description in which adaptation was perfromed by a heuristic with the same or greater scheduling order than the current one. The predicate *Evolution-Point* in Figure 5.17 indicates these points in the design description using the name of the column from which the adaptation started and the scheduling order of the heuristic rule that suggested the adaptation.

Challenge heuristic 15. Distillation is not always the cheapest alternative, especially when there exists an extractive distillation process with relative volatilities between the keys that are much higher than the ones in the ordinary case. This heuristic (Fig. 5.18) tries to explore alternatives based on this criterion.

Examine neighbors to decide if the MSA removal should be delayed. Sometimes it is not cost-effective to remove the mass separating agent as soon as possible. A typical example of that is the case of two extractive distillation process units[4] in sequence that use the same MSA. Removing the MSA before the second unit is obviously something the designer should avoid since he/she will have to use it again in the next unit. Figure 5.19 describes the representation for this heuristic.

Challenge Heuristic 10. Performing the least tight separation first does not guarantee that we are going to get an optimal design. Sometimes the least tight separation is followed by a very difficult one. In this case it might be cheaper to perform two successive separations of similar difficulty. The heuristic in Figure 5.20 looks for such cases and if it finds one it schedules the difficult separation first. Difficulty is measured using the relative volatility between the key components in a separation scheme.

Evolutionary Heuristics in the Seader & Westerberg method

[4] Each one of these units consists of two columns, one in which the extractive distillation process takes place and the next one in which the MSA is removed.

```
(defHeuristic Ordinary-vs-Extractive-Splits
  :Class Challenge-Heuristic-15
  :Conditions ((Consider (Design-Complete ?system ?total-cost)) :Var ?f0
               (Evolve ?system)
               (First-Column ?system ?column)
               (Consider (Possible (Separation distillation (?l-k ?h-k) ?column))) :Var ?f1
               (Column-Feed (M-C-S ?components ?phase ?stage) ?column)
               (Value-of
                (A (Extractive-Alpha-LK-HK ?l-k ?h-k ?p-s ?column ?conditions)) ?extract-alpha ?eq-1)
               (Value-of (A (Alpha-LK-HK ?l-k ?h-k ?column ?conditions)) ?alpha ?eq-2)
               :Test (and (>= ?extract-alpha (expt ?alpha 1.95))
                          (downstream-from-evolution ?column 3)))
  :Action ((assume-in-cycle '(Evolve-Design-From ,?column ,?f0 ,?f1))
           (assert-in-design '(Evolution-Point ,?column 3))
           (assert-in-design
            '(Consider (Possible (Separation extractive-distillation (,?l-k ,?h-k) ,?column))))))
```

IF a design for the separation problem has been found
  AND *?column* belongs to a separation system we are trying to evolve
  AND the feed components for *column* are known
  AND the separation scheme used in *?column* is known
  AND the feed components, the relative volatility between the keys in the extractive distillation case and the relative volatility between the same keys in the ordinary distillation case are known under the same conditions
  AND the value for the relative volatility in the extractive case is greater or equal of some power of the relative volatility in the ordinary case[3]
  AND *?column* is downstream from the latest evolution point
THEN retain the old design description up to the *?column* intact and try to evolve the rest of it assuming that an extractive distillation between the same keys takes place in *?column*.

```
(defHeuristic Search-for-Competitive-Extractive-vs-Ordinary-Splits
  :Class Challenge-Heuristic-15
  :Conditions ((Consider (Design-Complete ?initial-system ?total-cost))
               (Evolve ?system)
               (First-Column ?system ?column)
               (Consider (Possible (Separation distillation (?l-k ?h-k) ?column)))
               (Column-Feed (M-C-S ?components ?phase ?stage) ?column)
               (Value-of
                (A (Extractive-Alpha-LK-HK ?l-k ?h-k ?p-s ?column ?conditions)) ?extract-alpha ?eq-1)
               (Value-of (A (Alpha-LK-HK ?l-k ?h-k ?column ?conditions)) ?alpha ?eq-2)
               :Test (< ?extract-alpha (expt ?alpha 1.95))
               (Connects-to ?column ?new-sep-system))
  :Action (unless (exists '(Destroy (Connections-to ,?column)))
             (assert '(Evolve ,?new-sep-system))))
```

IF a design for the separation problem has been found
  AND *?column* belongs to a separation system we are trying to evolve
  AND the feed components for *column* are known
  AND the separation scheme used in *?column* is known
  AND the feed components, the relative volatility between the keys in the extractive distillation case and the relative volatility between the same keys in the ordinary distillation case are known under the same conditions
  AND the value for the relative volatility in the extractive case is less than some power of the relative volatility in the ordinary case
  AND *?column* connects to a separation system *?new-sep-system*
THEN unless you have been instructed to remove the part of the design containing *?column* try to evolve the part of the design that contains *?new-sep-system*

Figure 5.18: Challenge Heuristic 15.

```
(defHeuristic Delay-MSA-Removal
  :Class Delay-MSA-Removal
  :Conditions ((Consider (Design-Complete ?system ?total-cost)) :Var ?f0
               (Mass-Separating-Agent ?p-s ?column-1)
               (Column-Products (?p-s) ?keys ?column-2)
               (Consider (Possible (Separation distillation (?l-k ?p-s) ?column-2)))
               :Var ?f1
               (Mass-Separating-Agent ?p-s ?column-3)
               :Test (and (not (equal ?column-1 ?column-2))
                          (not (equal ?column-1 ?column-3))
                          (not (equal ?column-2 ?column-3))
                          (in-order? '(,?column-1 ,?column-2 ,?column-3))
                          (downstream-from-evolution ?column-2 4)))
  :Action ((invalidate-decision '(Possible (Separation distillation (,?l-k ,?p-s) ,?column-2)))
           (assume-in-design '(Focus ,?column-2))
           (assert-in-design '(Evolution-Point ,?column-2 4))
           (assume-in-cycle '(Evolve-Design-From ,?column-2 ,?f0 ,?f1))))
```

IF a design for the separation problem has been found
  AND a mass separating agent *?p-s* has been used in some column *?column-1*
  AND *?p-s* belongs to the products of some column *?column-2*
  AND *?column-2* was used to separate *?p-s* from the rest of the substances
  AND the same mass separating agent *?p-s* has been used in some column *?column-3*
  AND *?column-1, ?column-2* and *?column-3* are connected in that order
  AND *?column-2* is downstream from the latest evolution point
THEN do not remove *?p-s* from *?column-2* and try to evolve the part of the design
  that contains *?column-2*.

Figure 5.19: Examine neighbors to decide if the MSA removal should be delayed.

```
(defHeuristic Move-Difficult-Split-One-Step-Earlier
  :Class Challenge-Heuristic-10
  :Conditions ((Consider (Design-Complete ?system ?total-cost)) :Var ?f0
               (Consider (Possible (Separation distillation (?lk1 ?hk1) ?column1)))
               :Var ?f1
               (Connects-to ?column1 ?system2)
               (First-Column ?system2 ?column2)
               (Consider (Possible (Separation distillation (?lk2 ?hk2) ?column2)))
               (Value-of (A (Alpha-LK-HK ?lk1 ?hk1 ?column1 ?conditions)) ?alpha1 . ?r2)
               (Value-of (A (Alpha-LK-HK ?lk2 ?hk2 ?column2 ?conditions)) ?alpha2 . ?r4)
               :Test (and (> (abs (- ?alpha1 ?alpha2)) *lower-alpha-difference*)
                          (< (abs (- ?alpha1 ?alpha2)) *upper-alpha-difference*))
               (Value-of (A (Installation-Cost ?column1 (?lk1 ?hk1))) ?cost-1 ?c-eq-1)
               (Value-of (A (Installation-Cost ?column2 (?lk2 ?hk2))) ?cost-2 ?c-eq-2)
               :Test (< ?cost-1 ?cost-2))

  :Action
  ((unless (examined-alternative
           '((distillation (,?lk1 ,?hk1)) (distillation (,?lk2 ,?hk2))))
      (invalidate-decision '(Possible (Separation distillation (,?l-k ,?p-s) ,?column1)))
      (assert-in-design '(Evolution-Point ,?column1 5))
      (assume-in-cycle '(Evolve-Design-From ,?column1 ,?f0 ,?f1))
      (assert-in-design
       '(Consider (Possible (Separation distillation (,?lk2 ,?hk2) ,?column1))))))
```

IF a design for the separation problem has been found
AND an ordinary distillation process takes place in *?column1*
AND *?column1* is connected to *?column2*
AND an ordinary distillation process takes place in *?column2*
AND and the difference between the relative volatilities between the keys in the two cases is between *lower-alpha-difference* and *upper-alpha-difference*
AND the separation in *?column1* is cheaper than the one in *?column2*
THEN if you have not already considered that as an alternative apply the separation scheme for *?column2* to *?column1*.

Figure 5.20: Challenge Heuristic 10.

```
(defHeuristic Create-All-Possible-Interchanges
  :Class Create-All-Possible-Interchanges
  :Conditions
  ((Consider (Possible (Separation ?method1 (?l-k-1 ?h-k-1) ?column1))) :Var ?f1
   (Connects-to ?column1 ?system1)
   (First-Column ?system1 ?column2)
   (Consider (Possible (Separation ?method2 (?l-k-2 ?h-k-2) ?column2))) :Var ?f2)
  :Action ((assume-in-cycle '(Exchange ,(car ?f1) :With ,(car ?f2)))))
```

For every pair of successive columns in the flowsheet consider as a possible design alternative exchanging the separation methods that take place in each one of them.

Figure 5.21: Create all the possible interchanges.

The Seader & Westerberg method contains only one evolutionary rule along with a set of criteria for pruning the alternatives proposed by this rule. The general rule is [54]:

> 'Interchange the relative positions of two adjacent subproblems, and for a given separation subproblem using separation method α, substitute separation method β.'

The set of criteria under which the interchange is possible is determined by trying to apply in a given order the heuristics that were not used during the creation of the initial flowsheet. For example, if the separation methods for two successive columns in the initial design were chosen according to the heuristics that try to disregard separations with very small relative volatility between the key components, the criteria used in the evolutionary phase will include all the heuristic rules belonging to the design strategy except those that disregard separations with very small relative volatilites.

All the possible interchanges for the Seader & Westerberg method are introduced by the heuristic rule described in Figure 5.21 and examined in a breadth-first fashion until a cheaper design is found. The possible interchanges are pruned by a set of rules that identify difficult separations in the interchanges proposed. For example, Figure 5.22 presents a rule that rejects an interchange that affects an extractive distillation unit. The particular interchange schedules the column used to recover the polar solvent in the unit before the extractive distillation column. Another example is Figure 5.23 which establishes an order of preference between exchanges that are almost equivalent under the application of a certain separation heuristic and those that are not. Finally, Figure 5.24 shows how the interchanges are scheduled in this strategy.

```
(defHeuristic Protect-Extractive-Distillation-Units
  :Class Filter-Possible-Interchanges
  :Conditions ((Exchange
               (Consider (Possible (Separation extractive-distillation (?lk-1 ?hk-1) ?col1)))
               :With (Consider (Possible (Separation distillation (?lk-2 ?p-s) ?col2))))
              :Var ?f1
              (Mass-Separating-Agent ?p-s ?col1))
  :Action ((examined-alternative ?f1)
           (reject ?f1)))
```

IF there has been suggested a possible interchange between columns *?col1* and *?col2*
AND *?col1* is an extractive distillation column
AND *?col2* is used to recover the polar solvent used in *?col1*
THEN reject this interchange and note that *?f1* is an examined alternative.

Figure 5.22: Reject the interchanges that rearrange extractive distillation units.

```
(defHeuristic Check-Relative-Volatilities
  :Class Filter-Possible-Interchanges
  :Conditions
  ((Exchange
    (Consider (Possible (Separation distillation (?lk-1 ?hk-1) ?col1)))
    :With
    (Consider (Possible (Separation distillation (?lk-2 ?hk-2) ?col2))))
   :Var ?f1
   (Value-of (A (Alpha-LK-HK ?lk-1 ?hk-1 ?col1 ?conditions)) ?a-1 ?eq-1)
   (Value-of (A (Alpha-LK-HK ?lk-2 ?hk-2 ?col2 ?conditions)) ?a-2 ?eq-2)
   :Test (<= (abs (- ?a-1 ?a-2)) *alpha-difference*)
   (Exchange
    (Consider (Possible (Separation distillation (?lk-3 ?hk-3) ?col3)))
    :With
    (Consider (Possible (Separation distillation (?lk-4 ?hk-4) ?col4))))
   :Var ?f2
   (Value-of (A (Alpha-LK-HK ?lk-3 ?hk-3 ?col3 ?conditions)) ?a-3 ?eq-3)
   (Value-of (A (Alpha-LK-HK ?lk-4 ?hk-4 ?col4 ?conditions)) ?a-4 ?eq-4)
   :Test (> (abs (- ?a-3 ?a-4)) *alpha-difference*))
  :Action
  ((unless (examined-alternative ?f2)
     (prefer ?f2 :Over ?f1 :Rank 1))))
```

IF *?f1* is a possible interchange between columns *?col1* and *?col2*
AND the relative volatilities between the two successive separations in *?f1* are almost the same
AND *?f2* is a possible interchange between columns *?col3* and *?col4*
AND the relative volatilities between the two successive separations in *?f2* vary widely
THEN unless *?f2* has already been examined, prefer *?f2* over *?f1* and include in the preference description the number of the heuristic that was responsible for this choice.

Figure 5.23: Prefer interchanges that do not constitute almost equivalent choices using the Perform-Least-Tight-Separation-First heuristic.

```
(defHeuristic Sequence-Heuristics
  :Class Schedule-Alternatives
  :Conditions
  ((Exchange ?f1 :With ?f2) :Var ?f3 :Test (most-preferable-for-evolution ?f3))
  :Action ((assume-in-cycle '(Schedule-Alternative ,?f3))))

IF ?f3 is  a possible interchange between columns ?col1 and ?col2
   AND the heuristic responsible for the retention of ?f3 has the lowest order
        between the heuristics in the initial part of the strategy
THEN choose ?f3 as the next interchange
```

Figure 5.24: Schedule alternatives in the Seader & Westerberg method.

5.1.3.2 Binary distillation

The heuristic rules for the binary distillation design problem serve four main functions; check whether the numerical results are consistent with the constraints in the qualitative model, determine the location for the introduction of the feed in the binary column, look for discrepancies between parameter values computed during the analysis phase and the design specifications and, finally, determine how to update inaccurate estimates for some design parameters. We present these rules in more detail below.

Check for Consistency. Inconsistencies in the numerical results arise as a result of inaccurate numerical estimates for some of the design parameters or as a result of incomplete design descriptions. For example, if the number of stages in a column is overestimated, then the analysis may compute negative values for the mole fractions of substances in some column stages. The same thing can happen if the location of the feed in a column has not been determined. There are two heuristics in OUZO that deal with these kinds of inconsistencies and propose changes to the values of the affected parameters (Figure 5.25).

Feed Location. Currently, OUZO contains only one heuristic for deciding on the stage in the column in which the feed is introduced. Figure 5.26 describes these heuristic in detail.

Compare with Specifications. Inaccurate estimates for some of the design parameters may lead to discrepancies between parameter values computed during the analysis phase and the design specifications. The heuristics in Figure 5.27 detect such discrepancies and propose changes in the values of the parameters that will eliminate the differences. The procedure *compare-equations* solves all the equations that calculate the parameter in its argument and detects any discrepancies between the computed values.

```
(defHeuristic Compare-with-Zero-1
  :Class Check-Consistency
  :Conditions ((not (Less-than (A ?param) ZERO)) :var ?f1
               (Value-of (A ?param) ?value ?eqn) :var ?f2 :Test (< ?value 0))
  :Action ((reject ?f2)
           (assert '(Increase (A ,?param)))))

 IF the amount of parameter ?param should not be negative
    AND the value computed for the amount of ?param is negative
 THEN the numerical value for ?param is inconsistent and the program
        should find ways of increasing this value.

(defHeuristic Compare-with-Zero-3
  :Class Check-Consistency
  :Conditions ((Greater-Than (A ?param) ZERO) :var ?f1
               (Value-of (A ?param) ?value ?eqn) :var ?f2 :Test (< ?value 0))
  :Action ((reject ?f2)
           (assert '(Increase (A ,?param)))))

 IF the amount of parameter ?param should be positive
    AND the value computed for the amount of ?param is not positive
 THEN the numerical value for ?param is inconsistent and the program
        should find ways of increasing this value.
```

Figure 5.25: Check for consistent numerical values in binary distillation design.

Update Estimates. These rules update the values for the parameters that were estimated using approximate equations. The heuristics in Figure 5.28 suggest an increase or descrease in the values for these parameters based on the suggestions made for the variables that were found to be inconsistent with either the qualitative constraints or the design specifications. The procedure *exists-path* in these rules searches for a causal path between two parameters.

Figure 5.29 describes the rules that modify the values for the parameters that were estimated using approximate equations.

5.1.4 Strategies

Design strategies are plans for sequencing the execution of the various classes of heuristic rules in ways that were found capable of producing optimal designs. Strategies are explicitly represented in our approach. This scheme enables the design system to reason about their appropriateness for a given task. In addition, it allows the system to be used as a testbed for experimenting with different strategies that can solve a specific problem. Figure 5.30 provides an example of the

```
(defHeuristic Feed-Location-Heuristic
 :Class Feed-Location
 :Conditions
 ((Gas-Path ?stage1 ?stage2)
  (Contained-Binary-Liquid-Mixture (2-C-S (?substance1 ?substance2) liquid ?stage1))
  (Contained-Binary-Liquid-Mixture (2-C-S (?substance1 ?substance2) liquid ?stage2))
  (Steady-State-Column-Design-Features
   ?column ?c-stage ?c-phase ?r-stage ?r-phase ?distillate-flow
   ?bproduct-flow ?substance1 ?substance2)
  (Value-of (A (Feed-Fraction ?substance1 ?column)) ?feed-fraction ?eqn1)
  (Value-of
   (A (Mole-Fraction ?substance1 (2-C-S (?substance1 ?substance2) liquid ?stage1))) ?mf-1 ?eqn2)
  (Value-of
   (A (Mole-Fraction ?substance1 (2-C-S (?substance1 ?substance2) liquid ?stage2))) ?mf-2 ?eqn3)
  :Test (and (< ?mf-1 ?feed-fraction) (> ?mf-2 ?feed-fraction))
  (not (Feed-Stage ?stage1)))
 :Action ((assert '(Install-Feed-Stage ,?stage1 ,?substance1 ,?substance2))))
```

IF *?stage1* and *?stage2* are successive stages in a column
AND each stage contains a binary liquid mixture
AND we assume steady-state behavior for the column
AND the mole fractions of the most volatile component in the feed *(?feed-fraction)* and in stages *?stage1 (mf-1)* and *?stage2 (?mf-2)* are known
AND *?feed-fraction* lies between *?mf-1* and *?mf-2*
AND the feed is not entering the column through the lower stage *(?stage1)* already
THEN introduce the feed at stage *?stage1*.

Figure 5.26: Determine the location of the feed stage in binary distillation.

```
(defHeuristic Increase-Analysis-Parameter
 :Class Compare-with-Specifications
 :Conditions ((Value-of (A ?param) . ?rest) :Var ?f1
              (Design-Spec (A ?param) ?val) :Test (> (compare-equations ?param) 0))
 :Action ((reject ?f1)
          (assert '(Increase (A ,?param)))))
```

IF there exists an estimated numerical value for *?param*
AND a value for *?param* is also part of the design specifications
AND by solving all the equations for *?param* the estimated value for it is found less than the design specifications
THEN reject the estimated value for *?param* and suggest an increase in this estimate.

```
(defHeuristic Decrease-Analysis-Parameter
 :Class Compare-with-Specifications
 :Conditions ((Value-of (A ?param) . ?rest) :Var ?f1
              (Design-Spec (A ?param) ?val) :Test (< (compare-equations ?param) 0))
 :Action ((reject ?f1)
          (assert '(Decrease (A ,?param)))))
```

IF there exists an estimated numerical value for *?param*
AND a value for *?param* is also part of the design specifications
AND by solving all the equations for *?param* the estimated value for it is found greater than the design specifications
THEN reject the estimated value for *?param* and suggest a decrease in this estimate.

Figure 5.27: Compare the design specifications with the results of the analysis phase.

```
(defHeuristic Change-Guess-Parameter-1
 :Class Update-Estimates
 :Conditions ((Guess-Parameter ?g-param)
              (Increase (A ?param)) :Test (> (exists-path ?g-param ?param ?g-param '(,?g-param)) 0))
 :Action ((assume (Increase (A ?g-param)))))
```

IF *?g-param* was estimated using an approximate equation
  AND an increase in the value for *?param* has been suggested
  AND there is a causal path from *?g-param* to *?param* that indicates
    that *?param* increases whenever *?g-param* increases
THEN suggest an increase in the estimate for *?g-param*.

```
(defHeuristic Change-Guess-Parameter-2
 :Class Update-Estimates
 :Conditions ((Guess-Parameter ?g-param)
              (Decrease (A ?param)) :Test (> (exists-path ?g-param ?param ?g-param '(,?g-param)) 0))
 :Action ((assume (Decrease (A ?g-param)))))
```

IF *?g-param* was estimated using an approximate equation
  AND a decrease in the value for *?param* has been suggested
  AND there is a causal path from *?g-param* to *?param* that indicates
    that *?param* decreases whenever *?g-param* decreases
THEN suggest a decrease in the estimate for *?g-param*.

```
(defHeuristic Change-Guess-Parameter-3
 :Class Update-Estimates
 :Conditions ((Guess-Parameter ?g-param)
              (Increase (A ?param)) :Test (< (exists-path ?g-param ?param ?g-param '(,?g-param)) 0))
 :Action ((assume (Decrease (A ?g-param)))))
```

IF *?g-param* was estimated using an approximate equation
  AND an increase in the value for *?param* has been suggested
  AND there is a causal path from *?g-param* to *?param* that indicates
    that *?param* increases whenever *?g-param* decreases
THEN suggest a decrease in the estimate for *?g-param*.

```
(defHeuristic Change-Guess-Parameter-4
 :Class Update-Estimates
 :Conditions ((Guess-Parameter ?g-param)
              (Decrease (A ?param))
              :Test (< (exists-path ?g-param ?param ?g-param '(,?g-param)) 0))
 :Action ((assume (Increase (A ?g-param)))))
```

IF *?g-param* was estimated using an approximate equation
  AND a decrease in the value for *?param* has been suggested
  AND there is a causal path from *?g-param* to *?param* that indicates
    that *?param* decreases whenever *?g-param* increases
THEN suggest an increase in the estimate for *?g-param*.

Figure 5.28: Update inaccurate estimates.

```
(defHeuristic Increase-Guess-Param
  :Class Update-Estimates
  :Conditions ((Increase ?param) :var ?f1
               (Change-Step ?param ?num)
               (Value-of ?param ?old-value ?eqn) :var ?f0)
  :Action ((reject ?f0)
           (reject ?f1)
           (propose-value ?param (+ ?num ?old-value))))
```

IF an increase in the value for *?param* has been suggested
  AND the unit change for *?param* is *?num*
  AND the old value for *?param* is *?old-value*
THEN increase *?param* by *?num* and invalidate the previous value
  and suggestion for it.

```
(defHeuristic Decrease-Guess-Param
  :Class Update-Estimates
  :Conditions ((Decrease ?param) :var ?f1
               (Change-Step ?param ?num)
               (Value-of ?param ?old-value ?eqn) :var ?f0)
  :Action ((reject ?f0)
           (reject ?f1)
           (propose-value ?param (- ?old-value ?num))))
```

IF a decrease in the value for *?param* has been suggested
  AND the unit change for *?param* is *?num*
  AND the old value for *?param* is *?old-value*
THEN decrease *?param* by *?num* and invalidate the previous value
  and suggestion for it.

Figure 5.29: Modify numerical estimates.

```
(defStrategy Nath-&-Motard-Evolutionary-Strategy
 :Heuristic-Classes
 (Challenge-Heuristic-11 Examine-Neighboring-Approximations Challenge-Heuristic-15
  Delay-MSA-Removal Challenge-Heuristic-10)
 ;; The bindings for the variables in the Conditions slot are the ones used in the rest
;; of the form. The predicates in the Conditions slot become the antecedents of an ATMoSphere rule.
 :Conditions
 ((Separation-System ?system)
  (Consider (Evolutionary-Strategy-for ?system Nath-&-Motard))
  (Consider (Design-Complete ?system ?cost)))
 :ATMS-Context :Implied-By
 ;; The instantiated predicates in this slot will be appended to the beginning of every
 ;; heuristic rule that belongs to this strategy.
 :Focus-Predicates ((Apply-Strategy-to ?system))
 ;; The body of the ATMoSphere rule in which the defStrategy form is translated consists of the contents
 ;; of the Action slot along with a set of functions in which the Execution-Order slot is translated.
 :Action ((cond ((< ?cost (find-previous-cost))
                 (store-design)
                 (assume-in-cycle '(Apply-Strategy-to ,?system)))
                (t (pop-design))))
 :Execution-Order
 (:SERIAL Challenge-Heuristic-11 Examine-Neighboring-Approximations Challenge-Heuristic-15
          Delay-MSA-Removal Challenge-Heuristic-10))

 IF we have completed the design of a separation system
    AND we use the Nath-&-Motard strategy to evolve the current design
 THEN if the current separation system costs less than the previous design
        mark the separation system we are going to evolve (the :Action part)
        and then apply the heuristics that are stored under the class Challenge-Heuristics-11
        followed by the heuristics stored under the class Examine-Neighboring-Approximations
        followed by the rest of the heuristic classes in the :Execution-Order slot of the form.
        If the separation system costs more then pop the previous design description
        and apply the evolutionary heuristics in the order described above.
```

Figure 5.30: Typical strategy form and its interpretation. This particular form represents the evolutionary rules for the Nath & Motard design strategy.

representation for a design strategy in the system. Each strategy representation includes the heuristic classes it uses and the preconditions under which it should be applied. In addition, it contains control information that optimizes the application of heuristic knowledge. This information is stored in the *ATMS-Context*, *Focus-Predicates* and *Execution-Order* slots.

The *ATMS-Context* slot determines the problem-solving context under which the heuristic rules are applied. In the current implementation, the problem-solving context is determined by an ATMS focus environment [19]. This consists of a set of assumptions representing the major design decisions considered by the system at the current stage. There are two possible problem-solving contexts. Under the *:Implied-By* context specified in Figure 5.30, all the application conditions for each heuristic rule must be implied by the current focus environment in the ATMS in order for the heuristic rule to trigger [19]. In the *:In* context all the applications conditions for each heuristic rule must hold for the rule to fire. A generic focus environment is used in this case.

The *Focus-Predicates* slot contains a set of predicates that will focus the application of heuristic rules to certain columns or systems in the separation sequence.

Finally, the *Execution-Order* slot determines the mode (serial, parallel or any combination of them) under which the heuristic classes are applied.

Currently, `OUZO` supports two of the most significant evolutionary strategies for the synthesis of separation sequences [43], [54]. For the Nath & Motard strategy the representation for the method that creates the initial design flowsheet is given in Figure 5.31. Figure 5.30 contains the representation for the evolutionary part of the strategy.

In the case of the Seader & Westerberg strategy, Figure 5.32 contains the form that represents the method used to create the initial design. Figure 5.33 contains the evolutionary part of the strategy.

Finally, `OUZO` supports a pure heuristic strategy for the binary distillation design problem that is based on the McGabe-Thiele method [35]. Figure 5.34 describes the implementation for this strategy.

5.1.5 Configuration Synthesis Rules

This is a set of rules that provide a set of procedures for changing the current design description and monitoring the state of design. For example, one such rule will fire whenever the products for a column in the sequence have been determined. It will then create descriptions for the columns that are going to be connected to the product streams (distillate and/or bottom products) of the current column that

```
(defStrategy Nath-&-Motard-Initial-Design-Heuristics
 :Heuristic-Classes
 (Separation-Alternatives-Representation Favor-Smallest-Production-Set Favor-Distillation
  Perform-Least-Tight-Separation-First Disregard-Small-Relative-Volatilities
  Avoid-Extreme-Operating-Conditions MSA-Agent-Isolation Order-Choices)
 :Conditions ((Separation-System ?system)
              (Consider (Evolutionary-Strategy-for ?system Nath-&-Motard)))
 :Focus-Predicates ((Focus ?column))
 :ATMS-Context :In
 :Action ((set-nm-strategy-parameters))
 :Execution-Order
 (:SERIAL Separation-Alternatives-Representation
          (:PARALLEL Favor-Smallest-Production-Set Favor-Distillation
                     Perform-Least-Tight-Separation-First Disregard-Small-Relative-Volatilities
                     Avoid-Extreme-Operating-Conditions MSA-Agent-Isolation)
          Order-Choices))
```

Figure 5.31: The part of the Nath & Motard strategy that develops the initial design flowsheet.

```
(defStrategy Seader-&-Westerberg-Initial-Strategy
 :Heuristic-Classes
 (Separation-Alternatives-Representation Disregard-Small-Relative-Volatilities
  Perform-Least-Tight-Separation-First Separate-Most-Plentiful-Components-First
  Remove-MSA-From-Products-in-Subsequent-Operation Favor-Smallest-Production-Set Order-Choices)
 :Conditions ((Separation-System ?system)
              (Consider (Evolutionary-Strategy-for ?system Seader-&-Westerberg)))
 :ATMS-Context :In
 :Focus-Predicates ((Focus ?column))
 :Action ((set-sw-strategy-parameters))
 :Execution-Order
 (:SERIAL Separation-Alternatives-Representation Disregard-Small-Relative-Volatilities
          Perform-Least-Tight-Separation-First Separate-Most-Plentiful-Components-First
          Remove-MSA-From-Products-in-Subsequent-Operation Favor-Smallest-Production-Set
          Order-Choices))
```

Figure 5.32: The part of the Seader & Westerberg strategy that develops the initial design flowsheet.

```
(defStrategy Seader-&-Westerberg-Evolutionary-Strategy
 :Heuristic-Classes (Create-All-Possible-Interchanges Filter-Possible-Interchanges
Schedule-Alternatives)
 :Conditions ((Separation-System ?system)
              (Consider (Evolutionary-Strategy-for ?system Seader-&-Westerberg))
              (Consider (Design-Complete ?system ?cost)))
 :ATMS-Context :Implied-By
 :Focus-Predicates ((Apply-Strategy-to ?system))
 :Action ((cond ((< ?cost (find-previous-cost))
                 (store-design)
                 (assume-in-cycle '(Apply-Strategy-to ,?system)))
                (t (when (all-interchanges-examined) (pop-design)))))
 :Execution-Order (:SERIAL Create-All-Possible-Interchanges
                           Filter-Possible-Interchanges
                           Schedule-Alternatives))
```

Figure 5.33: The part of the Seader & Westerberg strategy that controls the application of the evolutionary heuristics.

```
(defStrategy McGabe-Thiele-Strategy
 :Heuristic-Classes
 (Check-Consistency Feed-Location Compare-with-Specifications Update-Estimates)
 :Conditions ((Distillation-Column ?column)
      (Consider (Design-Strategy-for ?column McGabe-Thiele)))
 :ATMS-Context :In
 :Focus-Predicates ((Apply-Strategy-to ?column))
 :Action ((assume-in-cycle '(Apply-Strategy-to ,?column))
          (set-mcgabe-strategy-parameters))
 :Execution-Order
 (:SERIAL Check-Consistency Feed-Location Compare-with-Specifications Update-Estimates))
```

Figure 5.34: The design strategy for the binary distillation design problem.

do not correspond to any of the desired products in the problem specification. Appendix C describes these rules in detail.

5.2 Controlling OUZO

The controller algorithm in OUZO is a design cycle consisting of the following phases (Fig. 5.35):

(1) Qualitatively analyze the current design description. The qualitative analysis performed by the system is the same as the one used in SIMGEN [21]. It is used to determine the operating conditions under which each model fragment in the domain theory is active.

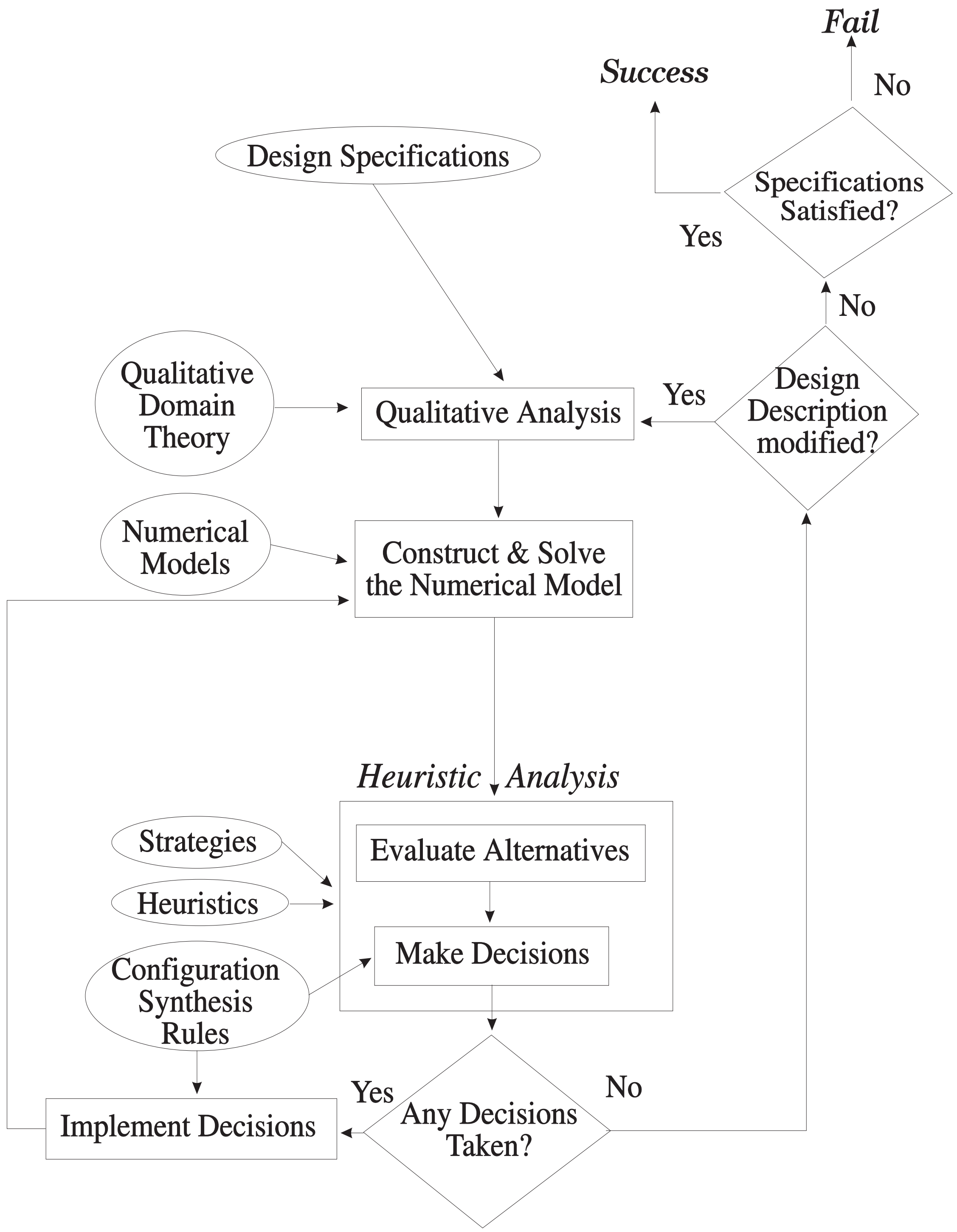
(2) Construct and solve the numerical model. The activation of the model fragments in the numerical model depends on the results of the qualitative analysis and the current focus environment. The later is an ATMS focus environment that consists of the major design decisions taken using the assume-in-cycle, assert-in-design or assume-in-design primitives. These design decisions include the separation schemes for each column, along with predicates that denote which columns are being examined by the design system at the current stage. For example, whenever the heuristic analysis decides on a particular separation for a column with the assert-in-design primitive, the current focus environment is updated to reflect this decision.

(3) Apply the design strategies and the configuration synthesis rules that are implied by the current focus. This step results in the generation, selection and implementation of design alternatives along with the updating of the focus environment.

Steps 2 and 3 constitute an inner loop which is applied until the heuristics suggest no more changes to the current focus environment or equivalently when there are no more design decisions taken. In this case, if the design description has been modified the system goes back to Step 1.

This scheme provides a way of controlling the level of detail in which each design alternative is analyzed. For example, when designing multicomponent separation systems, for every column in a flowsheet, OUZO decides which of several possible separations should be performed. This decision is based on an estimation of the properties for each design in a set of reference conditions. When this analysis is done, the current focus environment is updated to include only the promising separations. Consequently, a more detailed set of numerical models is activated that generates cost estimates only for the promising separations in a set of actual

Figure 5.35: The controller algorithm in OUZO.



conditions. Because qualitative analysis is computationally the most expensive stage in the design cycle, the system tries to do as much of the analysis as possible using the numerical models and the heuristics and that is the purpose for the loop between steps 2 and 3.

The design process ends when the design description remains unchanged during a design cycle. In this case, if the design specifications are satisfied OUZO terminates with success, otherwise it exits with failure.

5.3 Discussion

As we mentioned in chapter 3 there are three important problems in process synthesis [42]:

1. The *Representation Problem* consists of developing a representation of the problem that is both expressive and effective.

2. The *Evaluation Problem* consists of finding ways of efficiently evaluating the design alternatives.

3. The *Strategy Problem* is the development of strategies that optimize the screening of alternatives.

In chapter 4.1 we explained how the physical knowledge in the program deals with the representation problem. The design knowledge component addresses the remaining problems.

More specifically, OUZO contains an extensive set of heuristic rules that tackle the evaluation problem. These rules increase the efficiency of design, since they provide empirical methods for pruning the number of alternatives. Furthermore, in contrast to CAD systems, the program coordinates the use of physical and design knowledge without any user intervention. The qualitative analysis is able to automatically relate physical knowledge representations to the current design scenario. Finally, the use of focus environments allows the design knowledge to control the activation of qualitative and numerical models on an as needed basis. Both these features increase the efficiency of the analysis phase and therefore provide an attractive solution to the evaluation problem.

The development of explicit strategies that optimize the application of heuristic knowledge deals with the strategy problem. These strategies have a direct correspondence to empirical methods used by engineers in design. Furthermore, strategy representations make it possible to use the program as a testbed for different design methods. The fact that OUZO supports both pure heuristic and evolutionary methods which constitute general design methodologies in fields other than separation systems, allows us to conjecture that this framework can cover a wide set of design problems.

Chapter 6

Examples

6.1 Overview

An instance of a binary distillation design problem was used to test the physical and design knowledge components that deal with binary separations in OUZO. In addition, four well-known separation cases from the chemical engineering literature were used to test the physical and design knowledge components for the multicomponent mixtures.

6.2 The Binary Distillation Design Problem

6.2.1 The Design Specifications

In the binary distillation design problem [35] the desired separation is specified along with the flow at some point (usually the reflux). What has to be determined is the number of stages that are necessary to achieve the given separation.

In the particular instance of the problem we describe below (taken from [35]), we considered a benzene toluene distillation process with the pressure set at 1 atm and with a constant relative volatility between the two substances set at 2.25 throughout the column. The rest of the variables were set as follows:

1. The feed flow rate was 1 mol per unit time.
2. The mole fraction of benzene in the feed was 0.4.
3. The desired separation was for 90% of the benzene to be recovered in 95% purity.
4. The reflux was set to 1 mole per mole of feed.

Finally, we assumed that the feed to the column was a saturated liquid.

What we needed to find were the number of stages for achieving this separation and the location of the feed to the column (i.e. at which stage in the column should we introduce the feed). Figures 6.1 and 6.2 contain the problem specifications that were fed to the system.

In particular, Figure 6.1 contains the qualitative description for the problem. This includes a minimal structural description for the column consisting of the column name along with the names of the two stages we always assume to be present in a distillation column (a stage for the reboiler and a stage for the condenser). In addition, we make a set of design and operating assumptions for the separation process. These include the name of the design strategy we want to use, a steady-state assumption for the column, the types of condenser and reboiler we are using (a partial condenser and a partial reboiler[1]), the assumptions that both the gas and liquid phases of the mixture are present in the reboiler and the condenser, and finally the assumption that condensing and vaporization are always active in these stages.

Furthermore, the problem description contains the set of modeling assumptions the user is willing to make in design. The implications for all these assumptions were described in Figures 4.4, 4.5 and 4.6.

Finally, a statement relating the physical properties of the two substances is introduced in the problem specification. The fact that benzene is more volatile than toluene allows the qualitative model to compute the distillate and bottom products for the column.

Figure 6.2 describes the numerical data for the specific problem. The values for particular parameters in the problem are set using the *assign* predicate. The *solve-for* predicate indicates the parameters we are interested in finding a value for.

6.2.2 The Program Trace

`OUZO` accepts the qualitative and numerical descriptions in Figures 6.1 and 6.2 and instantiates the qualitative model that is consistent with this description (a distillation column with only two stages; a reboiler and a condenser). In the next step the equations that are consistent with this qualitative description are instantiated and the system tries to solve for all the parameters it is interested in (in our case the *Num-of-Stages* quantity). It turns out that there is not enough information to compute a numerical value for the number of stages in the column at this point. As a result, the system looks for equations that provide estimates for any of the design parameters. The Gilliland-Fenske equation [13] provides an estimate for the number of stages in the column that corresponds to a 13-stage column.

The results of the qualitative and numerical analysis analysis along with the contents of the design knowledge component are fed into the heuristic analysis stage of the design algorithm (Figure 5.35). The heuristics that belong to the

[1] A partial reboiler is one in which part of the input liquid is vaporized. This stands in contrast with a total reboiler in which all of the input liquid is vaporized. An analogous definition is used for a partial condenser.

```
;; Structural features for the column.

(assertq (Distillation-Column column))
(assertq (Distillation-Column-Stages column (reboiler condenser)))
(assertq (Stage reboiler))
(assertq (Stage condenser))
(assertq (Condenser-Stage condenser))
(assertq (Reboiler-Stage reboiler))

;; Design assumptions for the column

(assertq (Consider (Design-Strategy-for column McGabe-Thiele)))

;; Operational assumptions about the column.

(assertq (Consider (Steady-State-in column)))

(assertq (Consider (Partial-Condenser column)))
(assertq (Consider (Partial-Reboiler column)))

(assertq (Contained-Binary-Liquid-Mixture (2-C-S (benzene toluene) liquid reboiler)))
(assertq (greater-than (A (Amount-of (2-C-S (benzene toluene) liquid reboiler))) ZERO))
(assertq (Contained-Binary-Liquid-Mixture (2-C-S (benzene toluene) liquid condenser)))
(assertq (greater-than (A (Amount-of (2-C-S (benzene toluene) liquid condenser))) ZERO))
(assertq (Contained-Binary-Gas-Mixture (2-C-S (benzene toluene) gas reboiler)))
(assertq (greater-than (A (Amount-of (2-C-S (benzene toluene) gas reboiler))) ZERO))
(assertq (Contained-Binary-Gas-Mixture (2-C-S (benzene toluene) gas condenser)))
(assertq (greater-than (A (Amount-of (2-C-S (benzene toluene) gas condenser))) ZERO))

(assertq (Consider (Condense (2-C-S (benzene toluene) gas condenser))))
(assertq (Consider (Vaporize (2-C-S (benzene toluene) liquid reboiler))))

;; Modeling assumptions about the column.

(assertq (Consider (Equilibrium-Stages-in column)))
(assertq (Consider (Negligible-Pressure-Drop-in column)))
(assertq (Consider (Negligible-Temperature-Drop-in column)))
(assertq (Consider (Ideal-System benzene toluene :in column)))
(assertq (Consider (Constant-Molal-Overflow-in column)))
(assertq (Consider (Neglect Vapor-HoldUp-in column)))

;; Physical properties information concerning the substances of the binary mixture.

(assertq (More-Volatile benzene :than toluene))
```

Figure 6.1: The qualitative description for the binary distillation problem.

```
(assign (A (Feed-Flow column)) 1.)
(assign (A (Feed-Fraction benzene column)) .4)
(assign (A (Mole-Fraction benzene (2-C-S (benzene toluene) gas condenser))) .95)
(assign (A (Fraction-Recovery benzene condenser column)) .9)
(assign (A (Reflux column)) 1.)
(assign (A (Constant-Alpha column)) 2.25)
(assign (A (Pressure column)) 1.)

(solve-for (A (Num-of-Stages column)))
```

Figure 6.2: Numerical specifications for the binary distillation design problem.

McGabe-Thiele design strategy (Figure 5.34) are applied next. Since the problem statement does not include any specific column description at this point (Figure 6.1), no heuristic rules fire. The onfiguration synthesis rules create the description for a 13-stage column. This description is then fed to the qualitative analysis stage which computes the qualitative model for the column and then to the numerical analysis stage which instantiates and solves the numerical models that describe the steady-state behavior for the column.

In the next step the McGabe-Thiele design strategy is activated and the heuristics in the Check-Consistency class detect an inconsistency in the numerical value computed for the mole fraction of benzene in stage11[2]. In particular, the numerical model calculates a negative value for the mole fraction of benzene in stage11 (-0.825), while the qualitative theory specifies that it should always be positive. Furthermore, the heuristics in the Feed-Location class suggest that the feed should enter the column at stage 6[3]. This design step ends with the configuration synthesis rules changing the current design description to reflect the location of the feed stage.

Qualitative and numerical analysis are applied to the new column description. The heuristics in the Check-Consistency class detect a new contradiction. The value for the mole fraction of benzene in the reboiler is found to be negative (-0.0446), in contrast to the positive value specified in the qualitative domain theory. These heuristics suggest an increase in the value for this parameter. The rules in the Compare-with-Specifications class explain the inconsistency in terms of the estimates for the design parameters. Two values have been computed for the mole

[2] The numbering of the stages begins at the top with the condenser being stage1 and the reboiler being stage13.

[3] The inconsistency in the mole fraction of benzene in the reboiler was generated because the description for the column did not include a stage for introducing the feed.

fraction of benzene in the reboiler. According to the equations relating directly this mole fraction to the design specifications, the mole fraction of benzene in the reboiler is 0.0644. The value that is causing the contradiction (-0.0446) is based on the estimate for the number of stages from the Gilliland-Fenske equation. The heuristics in the Update-Estimates class try to eliminate the contradiction by looking for a causal path that relates the quantity for which there are conflicting values to the design parameters for which the approximation equations were applied (the Num-of-Stages quantity in our case). The relation

```
(Qprop- (Mole-Fraction ?substance1 (2-C-S (?substance1 ?substance2) ?r-phase ?r-stage))
        (Num-of-Stages ?column))
```

from Figure 4.18 provides a causal path between these two quantities and suggests that in order to increase the mole fraction of benzene in the reboiler we will have to decrease the number of stages in the column. OUZO decides on the specific number by which the number of stages is reduced (1 in this run) using the *Def-First-Guess-Parameter* form in Figure 4.41.

Qualitative and numerical models for a 12-stage distillation column are constructed and analyzed by the system. The heuristics in the Check-Consistency class detect a negative value for the mole fraction of benzene in stage 11 (-0.808). The rules in the Feed-Location class are activated, suggesting that the feed should enter the column at stage 7. The configuration synthesis rules change the current design description to reflect the new location for the feed.

After the equations for a 12-stage column are solved, the Compare-with-Specifications rules find a discrepancy between the values for the mole fraction of benzene in the reboiler. More specifically, the Gilliland-Fenske method estimates this mole fraction to be equal to 0.124, while the design specifications insist it is 0.064. There is a difference of 0.0597 between these two values and the system asks the user whether he/she is happy with it. The user informs the system that he/she is satisfied with the current result and the design process terminates. Figure 6.3 contains the proposed design.

6.2.3 Statistics

This example ran on an IBM RS/6000, Model 320, with 64MB of RAM running Lucid Common Lisp. The results from OUZO (a column with 12 equilibrium stages) were consistent with the ones found in [35]. These results correspond to the program trace we described above. As Table 6.1 indicates most of the running

Elapsed Real Time for the Design system	$\approx$ 1 hr 37 mins
Elapsed Real Time for Qualitative Analysis	$\approx$ 1 hr 27 mins
Number of scenario models tested in producing the final design proposal	4
Number of Quantities in the proposed design	314
Number of active Processes in the proposed design	26
Number of instantiated Views in the proposed design	139
Number of rules run	91,780

Table 6.1: Statistics for the distillation design example. The times shown correspond to the time spent in finding the first acceptable design in the problem.

time was spent in the qualitative analysis of the various column models that were proposed during the design process.

6.3 The C_6 Separation Synthesis Problem

The C_6 separation synthesis problem is described in [53]. The version we are going to describe below is the same as [43]. Table 6.2 contains the problem definition that we used. In this case we have a mixture of three components (n-Hexane, Benzene and Cyclohexane) which we want to separate into pure component products. The composition, the flow rate, the temperature and the pressure of the input mixture are given.

The qualitative description for this problem is given in Figure 6.4. First, the feed to the column is described. It is a multicomponent mixture consisting of three substances (n-Hexane, Benzene and Cyclohexane). Subsequently, the separation system we want to construct is designated (system1) and the relation between the multicomponent mixture and the separation system is established (the *Separation-System-Feed* predicate). At the end of this part we supply a name for the environment surrounding the column in order to associate physical constants (such as the atmospheric pressure or the gas constant) with this environment.

The next part of the problem description contains the list of desired products for separation system system1. In addition, the modeling assumptions we are willing to use in the analysis of the design alternatives for each column are specified.

Feed		
Component	**Component Name**	**Mole Fraction**
1	n-Hexane	.3333
2	Benzene	.3333
3	Cyclohexane	.3334
Desired Products		**Conditions**
Product	**Component**	T = 37.8°C
1	1	P = 1.033 kg/cm^2
2	2	Total Flow Rate =
3	3	= 170.1 kg mol/h

Table 6.2: Problem definition for the C_6 separation synthesis problem.

The first one is the sharp separation approximation for each separation unit (the *Sharp-Separation-For* predicate). The second one refers to the phase of the feed to the separation system. We assume that the feed is always liquid (the *Liquid-Phase-Feed* predicate). This simplifies the Underwood equations for calculating the minimum reflux ratio (see Appendix B.2).

The next part contains the set of design assumptions for this problem. The first statement specifies the design strategy we want the system to follow (the Nath & Motard strategy in the figure). In addition, we supply the system with values for some of the design parameters such as the efficiency for each stage in the column, the correlation between the minimum and the actual reflux rate for each column (the *Operating-Reflux-Estimate* predicate) and the projected life for the separation system.

Furthermore, the description contains some of the physical properties for the mass separating agent (furfural) used in extractive distillation, such as the substances in the original mixture with which it can associate (benzene and cyclohexane). The physical and design knowledge are not dependent on this particular agent. We chose to use furfural because it was easier to obtain physical properties data for this substance.

Finally, the value for the atmospheric pressure is given. We have to specify this value here rather than in the numerical description for the problem because some of the model fragments in the qualitative domain theory use this value in their argument list.

Figure 6.5 contains the numerical description for the problem. All the numerical data from Table 6.2 are included along with the reference conditions for the

Alternative	Method	Light Key	Heavy Key
alt-1	Distillation	n-Hexane	Cyclohexane
alt-2	Distillation	Benzene	Cyclohexane
alt-3	Distillation	Benzene	n-Hexane
alt-4	Distillation	Cyclohexane	Benzene
alt-5	Extractive Distillation	n-Hexane	Benzene
alt-6	Extractive Distillation	Cyclohexane	Benzene
alt-7	Extractive Distillation	Benzene	n-Hexane
alt-8	Extractive Distillation	Benzene	Cyclohexane

Table 6.3: The separation alternatives for COLUMN1.

column (130°F and 1 atm), the reference temperatures that are used to calculate equilibrium ratios and finally the value for the gas constant in the ideal gas law.

We implemented two evolutionary design strategies in OUZO for this problem, the Seader & Westerberg [54] and the Nath & Motard [43] strategies. The traces for both methods are described below.

6.3.1 The Nath & Motard Strategy Trace

The design process begins with a qualitative and numerical analysis of the problem description. As a result, a set of qualitative and numerical models introducing relevant physical properties for the multicomponent mixture and its substances (e.g. boiling points, vapor pressures, etc) are instantiated and solved. The results of this analysis along with the contents of the design knowledge component are fed into the heuristic analysis stage of the design algorithm (Figure 5.35). The heuristics that belong to the initial part of the Nath & Motard strategy (Figure 5.31) are applied next. Since the problem statement does not include any specific column description at this point (Figure 6.4), no heuristic rules fire. The current design scenario activates a set of configuration synthesis rules that create the description for the first column (COLUMN1) in the separation system.

The new design description is analyzed using the physical knowledge component. As a result a set of qualitative model fragments for each separation alternative in COLUMN1 are created. In particular, the qualitative domain theory computes four ordinary distillation alternatives and four extractive distillation alternatives that use furfural as the mass separating agent[4] (see Table 6.3).

For all these design choices the system instantiates and solves qualitative and quantitative models that correspond to the separation properties for the reference conditions[5] (Figures 4.36 and 4.37). Furthermore, for every alternative the system activates the model fragments that describe the distillate and bottoms products (Figures 4.28, 4.29, 4.30, 4.31, 4.32, 4.33 and 4.34) and their corresponding numerical models are solved. Finally, a set of qualitative model fragments that describe relevant features for each design choice in actual operating conditions (Figures 4.38, 4.39) are instantiated.

In the next step the initial part of the Nath & Motard strategy is activated in an effort to prune the design space. The heuristics in the Separation-Alternatives-Representation class (Figure 5.2) feed all the ordinary distillation alternatives in the heuristic analysis. The rules in the Disregard-Small-Relative-Volatilities class (Figure 5.6) reject the alternatives alt-2, alt-3 and alt-4[6]. The same heuristics introduce the alt-5, alt-6 and alt-8 extractive distillation alternatives in the analysis. Alt-8 is rejected by the rules in the Disregard-Small-Relative-Volatilities class. The heuristics in the Perform-Least-Tight-Separation-First class (Figure 5.9) indicate that alt-5 is better than alt-6 since its coefficient for the difficulty of separation is found to be the lowest of the two. Finally, the heuristics in the Favor-Distillation class (Figure 5.8) conclude that alt-1 is better than alt-5 and alt-6. As a result, the heuristics in class Order-Choices (Figure 5.3) indicate that alt-1 is the most promising design for COLUMN1.

The properties for alt-1 in actual operating conditions are now implied by the results of the heuristic analysis. The corresponding numerical models for alt-1 are constructed and solved. Using the equations in appendix B.2 the installation cost for alt-1 is estimated at $ 69,958. The configuration synthesis rules update the design description for COLUMN1 by generating the product streams that corre-

---

[4] In the solution developed in [43] the mass separating agent used was phenol. We had difficulty obtaining physical properties data for phenol, therefore we decided to use furfural instead. This decision did not have an effect on the flowsheets developed by OUZO for the C_6 separation which were analogous with the ones presented in [43].

[5] Presure and temperature.

[6] The relative volatility between cyclohexane and benzene has different values in [43] and in this research because we use different sources for physical properties data. In particular, the relative volatility between benzene and cyclohexane in reference conditions is found by [43] to be equal to 1.18 while we compute it equal to 0.99. As a result, alt-2 in [43] is rejected when compared to alt-1 because of a lower value for its coefficient of the difficulty of separation (CDS). In our case it is rejected because its relative volatility is found to be less than the minimum value specified by the design method (1.1).

spond to alt-1. In addition, the configuration synthesis rules check whether all the desired products have been recovered. There are two more products that we still need to separate from the input mixture (Benzene and Cyclohexane). Consequently, the configuration synthesis rules instantiate a new column (COLUMN2) that feeds from the bottom products of COLUMN1.

The same processing cycle is repeated for COLUMN2 and after two such cycles that correspond to columns COLUMN2 and COLUMN3 we end up with the flowsheet shown in part (a) of Figure 6.6.

At this point all the desired products have been recovered. The configuration synthesis rules inform the rest of the design system that a complete flowsheet has been found for the problem. As a result the evolutionary part of the Nath & Motard strategy (Figure 5.30) is activated. The heuristics belonging to the Challenge-Heuristic-15 class (Figure 5.18) notice that the relative volatility between n-Hexane and Benzene in alt-5 is larger than almost the square of the relative volatility between these two substances in alt-1. Consequently, COLUMN2 and COLUMN3 are removed from the flowsheet and the system starts to investigate a new design that does not involve an ordinary distillation process between n-Hexane and Bezene in COLUMN1. Another sequence of design cycles is initiated resulting in the flowsheet shown in part (b) of Figure 6.6. Flowsheet (b) is a better design alternative, since it costs less than the first one.

Once more the evolutionary part of the Nath & Motard strategy is activated for the most recent separation system. The heuristics in the Delay-MSA-Removal class (Figure 5.19) suggest that the removal of the mass separating agent in COLUMN2 should be delayed since there is an extractive distillation process that uses the same agent immediately following COLUMN2. The part of the flowsheet that feeds from the products of COLUMN2 is removed and the system investigates design alternatives for COLUMN2 that are different from an ordinary distillation split between Cyclohexane and Furfural (the previous design decision).

Another sequence of design cycles creates the flowsheet that is described in part (c) of Figure 6.6. This design is found to be cheaper than all the previous alternatives. No more evolutionary heuristics are applicable at this point and the design program exits with the most recent flowsheet as the proposed design.

6.3.2 The Seader & Westerberg Strategy Trace

The design process begins by creating the description for the first column (COLUMN1) in the separation system in a way similar to what is the case for the Nath & Motard strategy in section 6.3.1.

The new design description is analyzed using the physical knowledge component. As a result a set of qualitative model fragments for each separation alternative in COLUMN1 are created. In particular, the qualitative domain theory computes the same eight design alternatives with the ones included in Table 6.3 above. In addition, for each one of these design choices the same qualitative and numerical model fragments with the ones described in section 6.3.1 become active.

In the next step the initial part of the Seader & Westerberg strategy is activated in an effort to prune the design space. The heuristics in the Separation-Alternatives-Representation class (Figure 5.2) feed all the ordinary distillation alternatives in the heuristic analysis. The rules in the Disregard-Small-Relative-Volatilities class (Figure 5.6) reject alternatives alt-2, alt-3 and alt-4. Furthermore, these heuristics introduce alternatives alt-5, alt-6 and alt-8 in the heuristic analysis. Alt-8 is rejected by the heuristics in the Disregard-Small-Relative-Volatilities class. The rules in the Perform-Least-Tight-Separation-First class (Figure 5.7) conclude that alt-5 is more preferable than alt-6. There are two remaining alternatives; alt-1 and alt-5. The heuristics in the Order-Choices class (Figure 5.3) pick alt-1 as the most promising design for COLUMN1. This is a random choice since there is no heuristic rule in the Seader & Westerberg strategy capable of comparing these two alternatives.

The detailed distillation features for alt-1 are now implied by the results of the heuristic analysis and the corresponding numerical models for alt-1 are constructed and solved. At the end of this design cycle the system creates the description for a new column (COLUMN2) in a way similar to the Nath & Motard strategy in section 6.3.1.

A set of analogous design steps are applied for COLUMN2 and COLUMN3 resulting in the flowsheet shown in part (a) of Figure 6.7.

At this point all the desired products have been recovered. The configuration synthesis rules inform the rest of the design system that a complete flowsheet has been found for the problem. As a result the evolutionary part of the Seader & Westerberg strategy (Figure 5.33) is activated. The heuristics in the Create-All-Possible-Interchanges class (Figure 5.21) compute two possible interchanges for the current flowsheet:

1. Exchange the splits between columns COLUMN1 and COLUMN2 (inter-1).

2. Exchange the splits between columns COLUMN2 and COLUMN3 (inter-2).

Inter-2 is rejected by the rules in the Filter-Possible-Interchanges class (Figure 5.22) since COLUMN2 and COLUMN3 form an extractive distillation unit. As a

Problem	Strategy	Run Time	Designs	Num of Rules
C_6 separation	Nath & Motard	28 mins 30 secs	3	56,762
C_6 separation	Seader & Westerberg	17 mins 54 secs	2	35,673

Table 6.4: Performance results for the C_6 separation propblem in OUZO.

result, the part of the flowsheet that feeds from the products of COLUMN1 is destroyed and the system starts to investigate an alternative flowsheet that involves an extractive distillation between Cyclohexane and Benzene in COLUMN1. Another sequence of design cycles is initiated resulting in the flowsheet shown in part (b) of Figure 6.7. The second flowsheet is cheaper than the original one, therefore it is a better design alternative. No more evolutionary heuristics are applicable at this point and the design system exits with the most recent flowsheet as the proposed design.

6.3.3 Results

Table 6.4 presents the performance results for the C_6 separation synthesis problem in OUZO. The Designs column in the table refers to the number of designs examined by the system until an optimal solution was found. The Num of Rules refer to the total number of rules executed in ATMoSphere. The example ran on an IBM RS/6000, Model 340, with 64 MB of RAM running Lucid Common Lisp.

There are eight possible flowsheets for this problem. Both methods relied on the use of heuristic strategies to prune the number of design alternatives. The designs proposed by the Nath & Motard method were consistent with the ones reported in [43]. We know of no published results for the Seader & Westerberg method on the C_6 separation problem.

In terms of performance the two methods were not equivalent. The Seader & Westerberg strategy generated only two flowsheets but, assuming that our numerical calculations were accurate, its final design was not optimal. The Nath & Motard strategy generated three flowsheets but came up with an optimal design. In addition, the heuristics in the Nath & Motard strategy were more effective in evaluating the design choices. Our implementation of the Seader & Westerberg method had to resort to a random decision when choosing between alternatives alt-5 and alt-1 in section 6.3.2.

Feed			Desired Products		Conditions
Component	Component Name	Mole Fraction	Product	Component	T = 53.89°C
1	Propane	.0147	1	1	P = 5.62 kg/cm^2
2	n-Butane	.5029	2	2	Total Flow Rate =
3	Butene-1	.1475	3	3, 4, 5	= 303.04 kg mol/h
4	Trans-Butene-2	.1563	4	6	
5	Cis-Butene-2	.1196			
6	n-Pentane	.0590			

Table 6.5: Problem specification for the n-Butylene purification problem.

6.4 The n-Butylene Purification Problem

The n-Butylene purification problem is described in [30]. The version we are going to describe below is the same as the one presented in [43]. Table 6.5 contains the problem definition that we used.

The qualitative description for this problem is given in Table 6.8. First, the feed to the column is described. It is a multicomponent mixture consisting of six substances (Propane, n-Butane, Butene-1, trans-Butene-2, cis-Butene-2, n-Pentane). Subsequently, the separation system we want to construct is designated (system1) and the relation between the multicomponent mixture and the separation system is established (Separation-System-Feed). At the end of this part we supply a name for the environment surrounding the column in order to associate physical constants (such as the atmospheric pressure or the gas constant) with this environment.

The next part of the problem description contains the list of desired products for separation system system1. The rest of the problem description is analogous to the one described for the C_6 separation synthesis problem (see section 6.3).

Figure 6.9 contains the numerical description for the problem. All the numerical data from Table 6.5 are included along with the reference conditions for the column (130°F and 1 atm), the reference temperatures that are used to calculate equilibrium ratios and the value for the gas constant in the ideal gas law.

6.4.1 The Seader & Westerberg Trace

As in the C_6 separation problems we described above, the design process begins with a qualitative and numerical analysis of the problem description. As a result a set of qualitative and numerical models introducing relevant physical properties for the multicomponent mixture and its substances (e.g. boiling points, vapor pressures, etc) are instantiated and solved. The results of this analysis along with the design knowledge component are fed to the heuristic analysis stage of the design algorithm (Figure 5.35). At this point the heuristic rules that belong to the initial part of the Seader & Westerberg strategy (Figure 5.32) are activated. Since the problem statement does not include any specific column description at this point (Figure 6.8), no heuristic rules fire. The current design scenario activates a set of configuration synthesis rules that create the description for the first column (COLUMN1) in the separation system.

The new design description is analyzed using the physical knowledge component. As a result a set of qualitative model fragments for each separation alternative in COLUMN1 are created. In particular, the qualitative domain theory computes twenty ordinary and extractive distillation alternatives for COLUMN1 (see Table 6.6).

For all these alternatives the system instantiates and solves the appropriate qualitative and quantitative models. These are similar with the ones in section 6.3.1 above.

In the next step, the initial part of the Seader & Westerberg strategy is activated in an effort to reduce the number of design choices. The heuristics in the Separation-Alternatives-Representation class (Figure 5.2) feed all the ordinary distillation alternatives in the heuristic analysis. The rules in the Disregard-Small-Relative-Volatilities class (Figure 5.6) reject the alternatives alt-3, alt-4, alt-6, alt-7, alt-8, alt-9 and alt-10 since their relative volatilities are found to be less than the minimum relative volatility that is acceptable by the method. The same heuristics introduce alt-14, alt-15 and alt-19 as promising extractive distillation alternatives in the analysis. In addition, the heuristics in class Perform-Least-Tight-Separation-First (Figure 5.7) conclude that alternatives alt-5 and alt-1 are better than alt-2 since the difference in their relative volatilities is higher than *alpha-difference* (.01). Also, alt-5 and alt-1 are found better than all the extractive distillation alternatives by the Ordinary-vs-Extractive-Distillation heuristic in the Perform-Least-Tight-Separation-First class (Figure 5.7). Finally, the small difference in the relative volatilities between alt-5 and alt-1 (< *alpha-difference*) triggers the Difficulty-of-Separation-2 heuristic (Figure 5.7). Similar to what happens in [54] the user is prompted to select the most promising split. We choose alt-1 in order to be consistent with [54] and the heuristics in class Order-Choices (Figure 5.3) update the focus environment to reflect this development.

It should be noted that the relative volatilities between the keys in alt-1 and alt-5 are almost the same and the feed flows for Butane and n-Pentane are small compared to the rest of the input. Therefore, the ordering between alt-1 and alt-5 is insignificant, since the final flowsheets in both cases would end up costing almost the same, if the rest of the designs were identical.

The detailed distillation features for alt-1 are now implied by the focus environment and the corresponding numerical models for alt-1 are constructed and solved. Using the equations in appendix B.2 the installation cost for alt-1 is estimated at \$ 27,936. The configuration synthesis rules update the design description for COLUMN1 by generating the product streams that correspond to alt-1. In addition, the configuration synthesis rules check whether all the desired products have been recovered. There are three more products that we still need to separate from the input mixture (n-Butane, n-Pentane and the group of butenes). Consequently, the configuration synthesis rules create the description for a new column (COLUMN2) that feeds from the bottom products of COLUMN1.

The same processing cycle is repeated for COLUMN2 and after four such cycles that correspond to the next four columns we end up with the flowsheet shown in part (a) of Figure 6.10.

At this point all the desired products have been recovered. The configuration synthesis rules inform the rest of the design system that a complete flowsheet (F-1) has been found for the problem. The evolutionary part of the Seader & Westerberg strategy (Figure 5.30) is activated. The heuristics in the Create-All-Possible-Interchanges class (Figure 5.21) compute four possible interchanges for the current flowsheet:

1. Exchange the splits between columns COLUMN1 and COLUMN2 (inter-1).

2. Exchange the splits between columns COLUMN2 and COLUMN3 (inter-2).

3. Exchange the splits between columns COLUMN3 and COLUMN4 (inter-3).

4. Exchange the splits between columns COLUMN4 and COLUMN5 (inter-4).

The heuristics in the Filter-Possible-Interchanges class (Figure 5.22) are activated in order to prune the number of possible exchanges in the flowsheet. Inter-4 is rejected since COLUMN4 and COLUMN5 form an extractive distillation unit. Inter-1 is found less preferable because the difference in the relative volatilities between the two splits is negligible ($\approx$ 0,002 which is less than the *alpha-difference* (.01) set by the method). Finally, inter-3 is found less preferable since the relative volatility for the extractive split ' in COLUMN4 (1.7) is less than *min-extractive-alpha* (2.0). The most preferable alternative is inter-2 and the flowsheet is altered to reflect the interchange. The part of the flowsheet that feeds from the products of COLUMN2 is removed and alt-2 is chosen as the separation that takes place in

COLUMN2. Two design cycles later a new, cheaper flowsheet (F-2) emerges (part (b) of Figure 6.10).

Once again the evolutionary part of the Seader & Westerberg strategy (Figure 5.30) is activated. The heuristics in the Create-All-Possible-Interchanges class compute two possible interchanges for the current flowsheet:

1. Exchange the splits between columns COLUMN1 and COLUMN2 (inter-5).

2. Exchange the splits between columns COLUMN3 and COLUMN4 (inter-6).

Inter-6 is rejected by the heuristics in the Filter-Possible-Interchanges class, since the relative volatility for the extractive split in COLUMN4 (1.7) is less than *min-extractive-alpha* (2.0). Inter-5 is used to evolve the current design. In the end, the flowsheet in part (c) of Figure 6.10 (F-3) is constructed. The cost for F-3 is found to be less than F-2.

No more evolutionary heuristics are applicable at this point and the design system exits with F-3 as the proposed design. The design trace is consistent with the one reported in [54].

6.4.2 The Nath & Motard Strategy Trace

The design process begins by creating the description for the first column (COLUMN1) in the separation system in a way similar to what is the case for the Seader & Wsterberg strategy in section 6.4.1.

The new design description is analyzed using the physical knowledge component. As a result a set of qualitative model fragments for each separation alternative in COLUMN1 are created. In particular, the qualitative domain theory computes the same twenty design alternatives with the ones presented in Table 6.6.

In the next step the initial part of the Nath & Motard strategy is activated in an effort to prune the design space. The heuristics in the Separation-Alternatives-Representation class (Figure 5.2) feed all the ordinary distillation alternatives in the heuristic analysis. The Disregard-Small-Relative-Volatilities class rejects alternatives alt-6, alt-7, alt-4, alt-8, alt-9, alt-3 and alt-10. The same rules introduce extractive distillation alternatives alt-16 and alt-17 in the analysis. The heuristics in the Perform-Least-Tight-Separation-First class use the values of the coefficients of difficulty of separation to indicate that alt-5 is the worst of the remaining ordinary distillation choices and that alt-1 is better than alt-2. The same heuristic class concludes that among the extractive distillation alternatives alt-16 is the best choice. The heuristics in the Favor-Distillation class indicates that all the remaining ordinary distillation alternatives should be considered better than the extractive distillation ones. The Favor-Smallest-Production-Set class rejects all the extractive distillation alternatives because they do not produce any of the specified products directly. Finally, the heuristics in the Order-Choices class indicate that alt-1 is the best alternative for COLUMN1.

The detailed distillation features for alt-1 are now implied by the results of the heuristic analysis and the corresponding numerical models for alt-1 are constructed and solved. At the end of this design cycle the system creates the description for a new column (COLUMN2). A set of analogous design steps are applied for COLUMN2, COLUMN3 and COLUMN4 resulting in the flowsheet shown in part (a) of Figure 6.11.

At this point all the desired products have been recovered. The configuration synthesis rules inform the rest of the design system that a complete flowsheet has been found. As a result the evolutionary part of the Nath & Motard strategy (Figure 5.30) is activated. The heuristics belonging to the Challenge-Heuristic-11 class (Figure 5.18) notice that in order to isolate Butene-1, trans-Butene-2 and cis-Butene-2 as a product set we had to use an extractive distillation process in COLUMN3. These heuristics decide to split the product set in the hope of avoiding an extractive distillation process in the new flowsheet. There are two members in the new product set: (i) Butene-1 and (ii) trans-Butene-2 and cis-Butene-2. The system starts a new design from scratch with the new product set. The resulting flowsheet is shown in part (b) of Figure 6.11.

Once again the evolutionary part of the Nath & Motard strategy is activated. The heuristics in the Challenge-Heuristic-10 class notice that the separation in COLUMN1 is much easier than the one in COLUMN2. It has been observed that choosing a very easy separation at some stage may not always lead to an optimum design [43], since the next separation may be a very difficult one. In order to explore this possibility the heuristics in class Challenge-Heuristic-10 evolve a new design, excluding the use of alt-1 as a possible separation for COLUMN1. The flowsheet shown in part (a) of Figure 6.12 is created.

Finally, the evolutionary heuristics in the Challenge-Heuristic-10 class notice that the separation in COLUMN2 is much easier than the one in COLUMN3. Alt-5 is excluded as a possible separation for COLUMN2 and a new flowsheet is constructed for the bottom products of COLUMN1. The final flowsheet is shown in part (b) of Figure 6.12.

No more evolutionary heuristics are applicable at this point and `OUZO` exits with the flowsheet in part (a) of Figure 6.12 as the proposed design. The design trace is consistent with the one reported in [43].

6.4.3 Results

Table 6.7 presents the performance results for the n-Butylene purification problem. The Designs column in the table refers to the number of designs examined by the system until an optimal solution was found. The Num of Rules refer to the total number of rules executed in ATMoSphere. The system was run on an IBM RS/6000, Model 340, with 64 MB of RAM running Lucid Common Lisp.

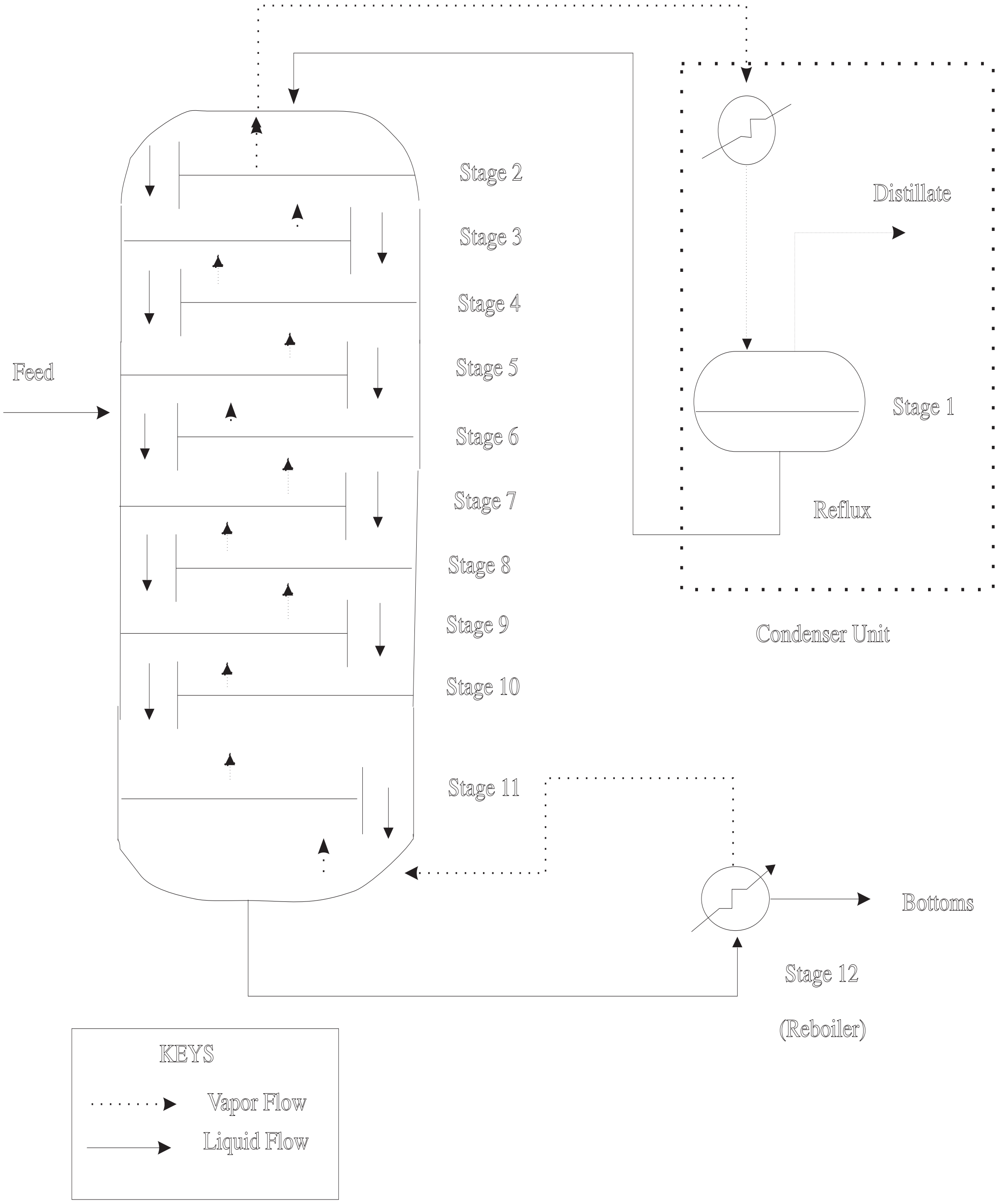
There are 1,344 possible flowsheets for this problem. Both methods relied on the use of heuristic strategies to prune the number of design alternatives. The Seader & Westerberg strategy was more effective in proposing a solution for this problem, since it proposed a solution after examining only 3 flowsheets. It took four flowsheets for the Nath & Motard strategy to come up with a final design. All of the results were consistent with the ones reported in [54], [43].

6.5 Discussion

Despite the fact that OUZO is able to deal with fairly complex problems the run times for most of the examples are not optimal. In particular, most of the running time for the program ($\approx$ 80%) is spent in the qualitative analysis stage. This is mainly a consequence of the fact that qualitative analysis is responsible for generating representations for all the design alternatives at each stage in the process. As a result, qualitative models have to support what seems to be the computationally most expensive stage of the design process. However, we believe that adapting the current analysis routines (which were taken from SIMGEN [21]) to fit more closely the needs of design, will have a positive effect on the run-time behavior of OUZO. For example, we believe that using a justification-based truth maintenance system (JTMS) in the examples we described above will improve significantly the performance. At least in the examples that we tested our system in, we have no more than two changes of focus environments per design cycle. In these cases, it is much better to compute all the nodes that are implied by the current focus at run-time using a JTMS, rather than compute and use the precompiled labels of an ATMS.

The examples described above clearly indicate the potential of using OUZO as a testbed for various design strategies. Providing explicit representations for the design knowledge in this domain allows the designer to use the system either for testing his/her ideas or (potentially) for generating better design strategies for different classes of problems automatically.

Figure 6.3: A 12-stage binary distillation column with a partial condenser. The feed enters the column at stage 6.



```
;; The separation system and its inputs.

(assertq (Substance n-Hexane))
(assertq (Substance Benzene))
(assertq (Substance Cyclohexane))
(assertq (MultiComponent-Mixture (M-C-S (n-Hexane Benzene Cyclohexane) liquid feed-stage)))

(assertq (Separation-System system1))
(assertq (Separation-System-Feed (M-C-S (n-Hexane Benzene Cyclohexane) liquid feed-stage) system1))
(assertq (Environment environment))

;; The desired products for the system.

(assertq (Desired-Products (n-Hexane) system1))
(assertq (Desired-Products (Benzene) system1))
(assertq (Desired-Products (Cyclohexane) system1))

;; The modeling assumptions for the problem.

(assertq (Consider (Sharp-Separation-For system1)))
(assertq (Consider (Liquid-Phase-Feed
                   (M-C-S (n-Hexane Benzene Cyclohexane) liquid feed-stage) system1)))

;; The design assumptions for the system.

(assertq (Consider (Evolutionary-Strategy-For system1 Nath-&-Motard)))
(assertq (Consider Stage-Efficiency .85))
(assertq (Consider (Operating-Reflux-Estimate 1.2)))
(assertq (Consider (Projected-Life 10))) ;; years

;; Physical Properties data.

(assertq (Value-of (A (Atmospheric-Pressure environment)) 101000.0 :user))  ;; 101000 Pa is 1 atm

;; Properties of the mass separating agent we are going to use.

(assertq (Substance Furfural))
(assertq (Polar-Solvent Furfural))
(assertq (Attracts Furfural Benzene))
(assertq (Attracts Furfural Cyclohexane))
(assertq (Consider (Polar-Solvent-Recovery Furfural .99)))
```

Figure 6.4: The qualitative description for the C_6 separation synthesis problem.

```
(assign
 (A (Feed-Flow n-Hexane    ;; kg mol/h
              (M-C-S (n-Hexane Benzene Cyclohexane) liquid feed-stage) system1)) 56.69433)
(assign
 (A (Feed-Flow Benzene
              (M-C-S (n-Hexane Benzene Cyclohexane) liquid feed-stage) system1)) 56.69433)
(assign
 (A (Feed-Flow Cyclohexane
              (M-C-S (n-Hexane Benzene Cyclohexane) liquid feed-stage) system1)) 56.71134)

(assign
 (A (Feed-Temperature (M-C-S (n-Hexane Benzene Cyclohexane) liquid feed-stage) system1))
 100.0)   ;; Fahreneit  (~ 37.8 oC)

(assign
 (A (Feed-Pressure
     (M-C-S (n-Hexane Benzene Cyclohexane) liquid feed-stage) system1)) 101306.31) ;; Pa

(assign (A (Product-Specification (n-Hexane) system1)) .99)
(assign (A (Product-Specification (Benzene) system1)) .98)
(assign (A (Product-Specification (Cyclohexane) system1)) .98)

(assign (A (Reference-Temperature system1)) 130) ;; Fahreneit (~54.4 oC)
(assign (A (Reference-Pressure system1)) 101000.0) ;; 1 atm

(assign (A (K-Ideal-Reference-Temperature n-Hexane)) 25) ;; Celsius
(assign (A (K-Ideal-Reference-Temperature Benzene)) 25)
(assign (A (K-Ideal-Reference-Temperature Cyclohexane)) 25)

(assign (A (Gas-Constant environment)) 1.987) ;;  cal/mol.K
```

Figure 6.5: The numerical data for the C_6 separation synthesis problem.

Figure 6.6: The flowsheets generated by the Nath & Motard method for the C_6 separation problem. The columns with an asterisk (*) indicate extractive distillation units.

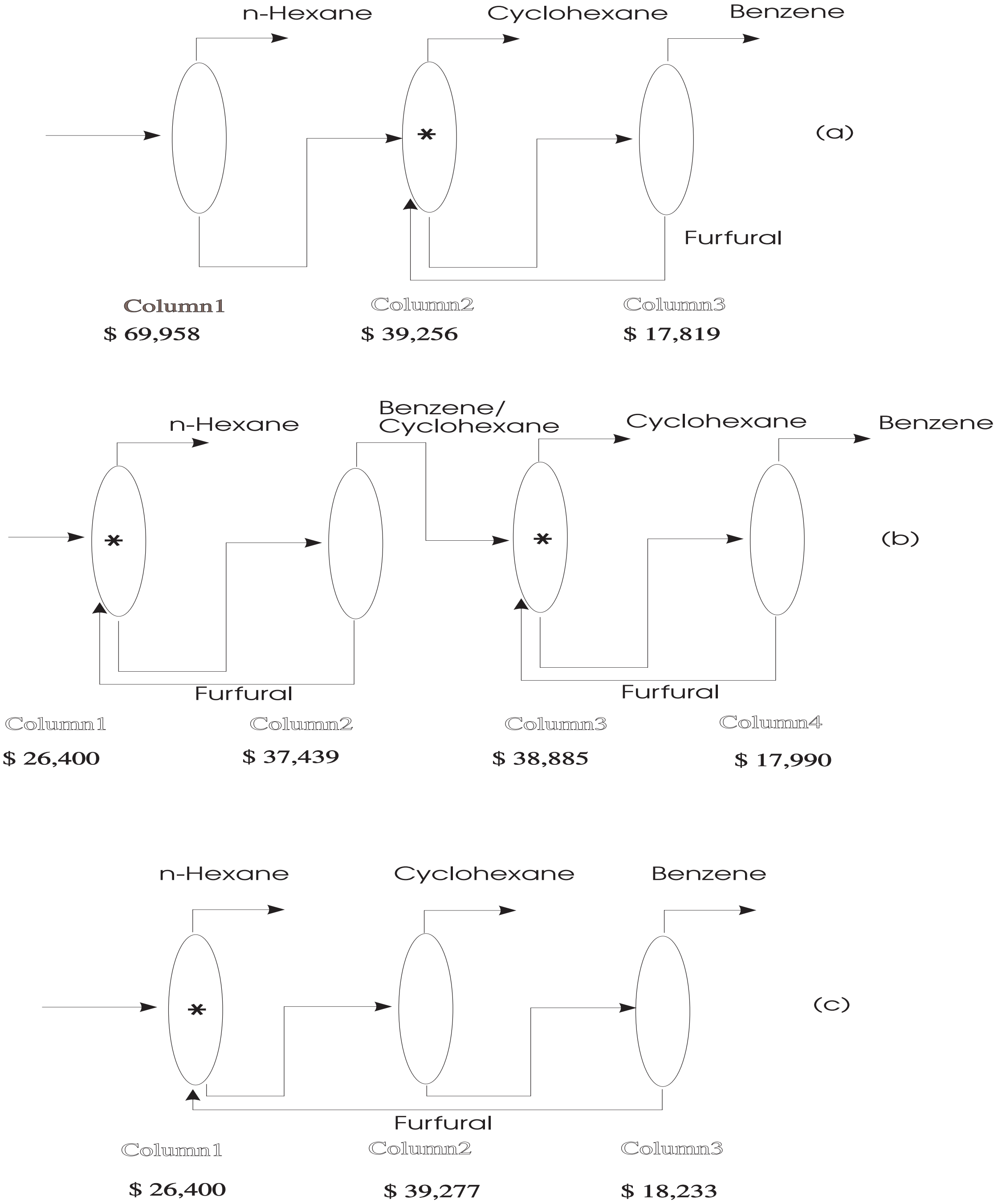
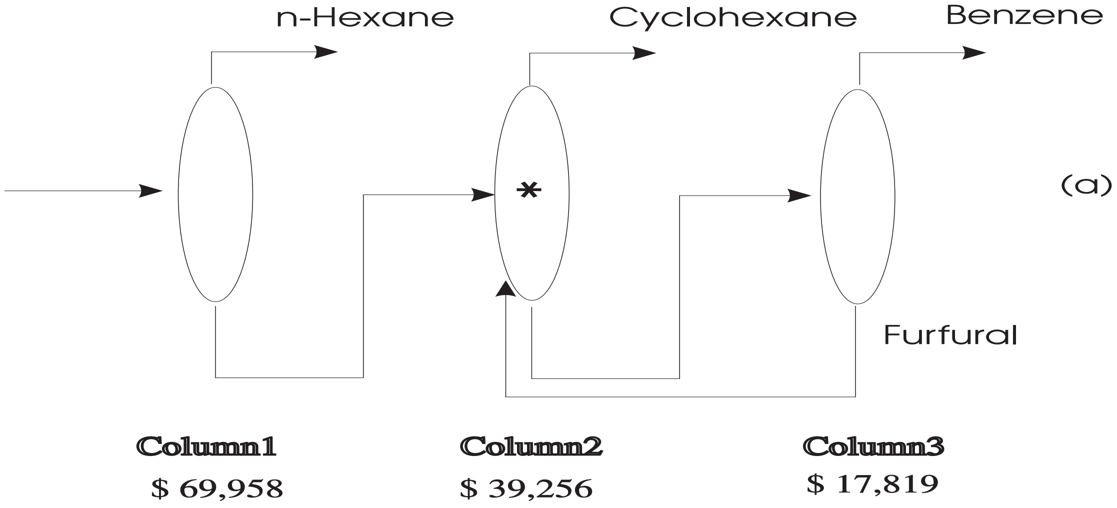
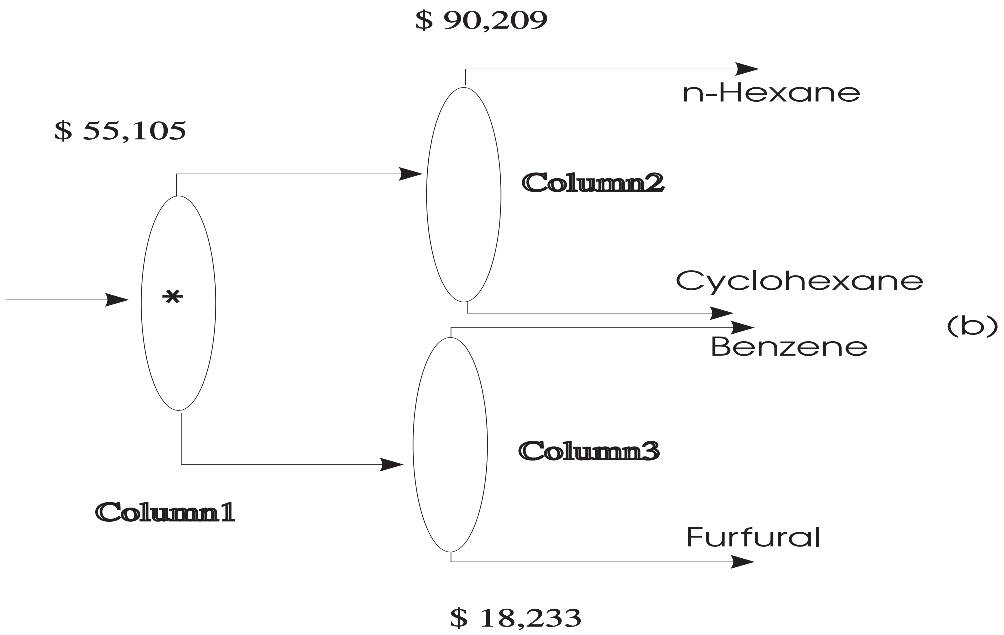


Figure 6.7: The flowsheets generated by the Seader & Westerberg method for the C_6 separation problem. The columns with an asterisk (*) indicate extractive distillation units.





```
(assertq (Substance Propane))
(assertq (Substance n-Butane))
(assertq (Substance Butene-1))
(assertq (Substance trans-Butene-2))
(assertq (Substance cis-Butene-2))
(assertq (Substance n-Pentane))
(assertq
 (MultiComponent-Mixture
  (M-C-S (Propane n-Butane Butene-1 trans-Butene-2 cis-Butene-2 n-Pentane) liquid feed-stage)))

(assertq (Separation-System system1))
(assertq (Separation-System-Feed
          (M-C-S (Propane n-Butane Butene-1 trans-Butene-2 cis-Butene-2 n-Pentane) liquid feed-stage)
          system1))
(assertq (Environment environment))

(assertq (Desired-Products (Propane) system1))
(assertq (Desired-Products (n-Butane) system1))
(assertq (Desired-Products (Butene-1 trans-Butene-2 cis-Butene-2) system1))
(assertq (Desired-Products (n-Pentane) system1))

(assertq (Consider (Sharp-Separation-For system1)))
(assertq (Consider
          (Liquid-Phase-Feed
           (M-C-S (Propane n-Butane Butene-1 trans-Butene-2 cis-Butene-2 n-Pentane) liquid
feed-stage)
           system1))

(assertq (Consider (Evolutionary-Strategy-For system1 Nath-&-Motard)))
(assertq (Consider Stage-Efficiency .85))
(assertq (Consider (Operating-Reflux-Estimate 1.2)))
(assertq (Consider (Projected-Life 10))) ;; years

(assertq (Value-of (A (Atmospheric-Pressure environment)) 101000.0 :user))

(assertq (Substance Furfural))
(assertq (Polar-Solvent Furfural))
(assertq (Attracts Furfural Trans-Butene-2))
(assertq (Attracts Furfural Butene-1))
(assertq (Attracts Furfural n-Pentane))
(assertq (Consider (Polar-Solvent-Recovery Furfural .99)))
```

Figure 6.8: The qualitative description for the n-Butylene purification problem.

```
(assign
 (A (Feed-Flow
     propane
     (M-C-S (Propane n-Butane Butene-1 trans-Butene-2 cis-Butene-2 n-Pentane) liquid feed-stage)
     system1)) 4.5) ;; kg mol/h
(assign
 (A (Feed-Flow
     n-butane
     (M-C-S (Propane n-Butane Butene-1 trans-Butene-2 cis-Butene-2 n-Pentane) liquid feed-stage)
     system1)) 154.7)
(assign
 (A (Feed-Flow
     butene-1
     (M-C-S (Propane n-Butane Butene-1 trans-Butene-2 cis-Butene-2 n-Pentane) liquid feed-stage)
     system1)) 45.4)
(assign
 (A (Feed-Flow
     trans-butene-2
     (M-C-S (Propane n-Butane Butene-1 trans-Butene-2 cis-Butene-2 n-Pentane) liquid feed-stage)
     system1)) 48.1)
(assign
 (A (Feed-Flow
     cis-butene-2
     (M-C-S (Propane n-Butane Butene-1 trans-Butene-2 cis-Butene-2 n-Pentane) liquid feed-stage)
     system1)) 36.7)
(assign
 (A (Feed-Flow
     n-pentane
     (M-C-S (Propane n-Butane Butene-1 trans-Butene-2 cis-Butene-2 n-Pentane) liquid feed-stage)
     system1)) 18.1)
(assign
 (A (Feed-Temperature
     (M-C-S (Propane n-Butane Butene-1 trans-Butene-2 cis-Butene-2 n-Pentane) liquid feed-stage)
     system1)) 205) ;; Fahreneit
(assign
 (A (Feed-Pressure
     (M-C-S (Propane n-Butane Butene-1 trans-Butene-2 cis-Butene-2 n-Pentane) liquid feed-stage)
     system1)) 2171862.0) ;; Pa

(assign (A (Product-Specification (Propane) system1)) .99)
(assign (A (Product-Specification (n-Butane) system1)) .96)
(assign (A (Product-Specification (Butene-1 trans-Butene-2 cis-Butene-2) system1)) .95)
(assign (A (Product-Specification (n-Pentane) system1)) .98)
(assign (A (Reference-Temperature system1)) 130) ;; Fahreneit (~ 54.4 oC)
(assign (A (Reference-Pressure system1)) 101000.0) ;; 1 atm
(assign (A (K-Ideal-Reference-Temperature Propane)) 25) ;; Celsius
(assign (A (K-Ideal-Reference-Temperature n-Butane)) 25)
(assign (A (K-Ideal-Reference-Temperature Butene-1)) 25)
(assign (A (K-Ideal-Reference-Temperature trans-Butene-2)) 25)
(assign (A (K-Ideal-Reference-Temperature cis-Butene-2)) 25)
(assign (A (K-Ideal-Reference-Temperature n-Pentane)) 25)
(assign (A (Gas-Constant environment)) 1.987)  ;;  cal/mol.K
```

Figure 6.9: The numerical data for the n-Butylene purification problem.

Alternative	Method	Light Key	Heavy Key
alt-1	Distillation	Propane	Butene-1
alt-2	Distillation	Butene-1	n-Butane
alt-3	Distillation	n-Butane	trans-Butene-2
alt-4	Distillation	trans-Butene-2	cis-Butene-2
alt-5	Distillation	cis-Butene-2	n-Pentane
alt-6	Distillation	n-Pentane	cis-Butene-2
alt-7	Distillation	cis-Butene-2	trans-Butene-2
alt-8	Distillation	trans-Butene-2	n-Butane
alt-9	Distillation	Butene-1	Propane
alt-10	Distillation	n-Butane	Butene-1
alt-11	Extractive Distillation	cis-Butene-2	trans-Butene-2
alt-12	Extractive Distillation	trans-Butene-2	cis-Butene-2
alt-13	Extractive Distillation	Butene-1	Propane
alt-14	Extractive Distillation	trans-Butene-2	n-Butane
alt-15	Extractive Distillation	Butene-1	n-Butane
alt-16	Extractive Distillation	n-Butane	trans-Butene-2
alt-17	Extractive Distillation	n-Butane	Butene-1
alt-18	Extractive Distillation	Propane	Butene-1
alt-19	Extractive Distillation	cis-Butene-2	n-Pentane
alt-20	Extractive Distillation	n-Pentane	cis-Butene-2

Table 6.6: The separation alternatives for COLUMN1.

Problem	Strategy	Run Time	Designs	Num of Rules
n-Butylene purification	N & M	3 hrs 36' 50"	4	210,746
n-Butylene purification	S & W	2 hrs 27' 25"	3	147,347

Table 6.7: Performance results for the n-Butylene purification problem.

Figure 6.10: The flowsheets generated by the Seader & Westerberg method for the n-Butylene purification problem. The columns with an asterisk (*) indicate extractive distillation units.

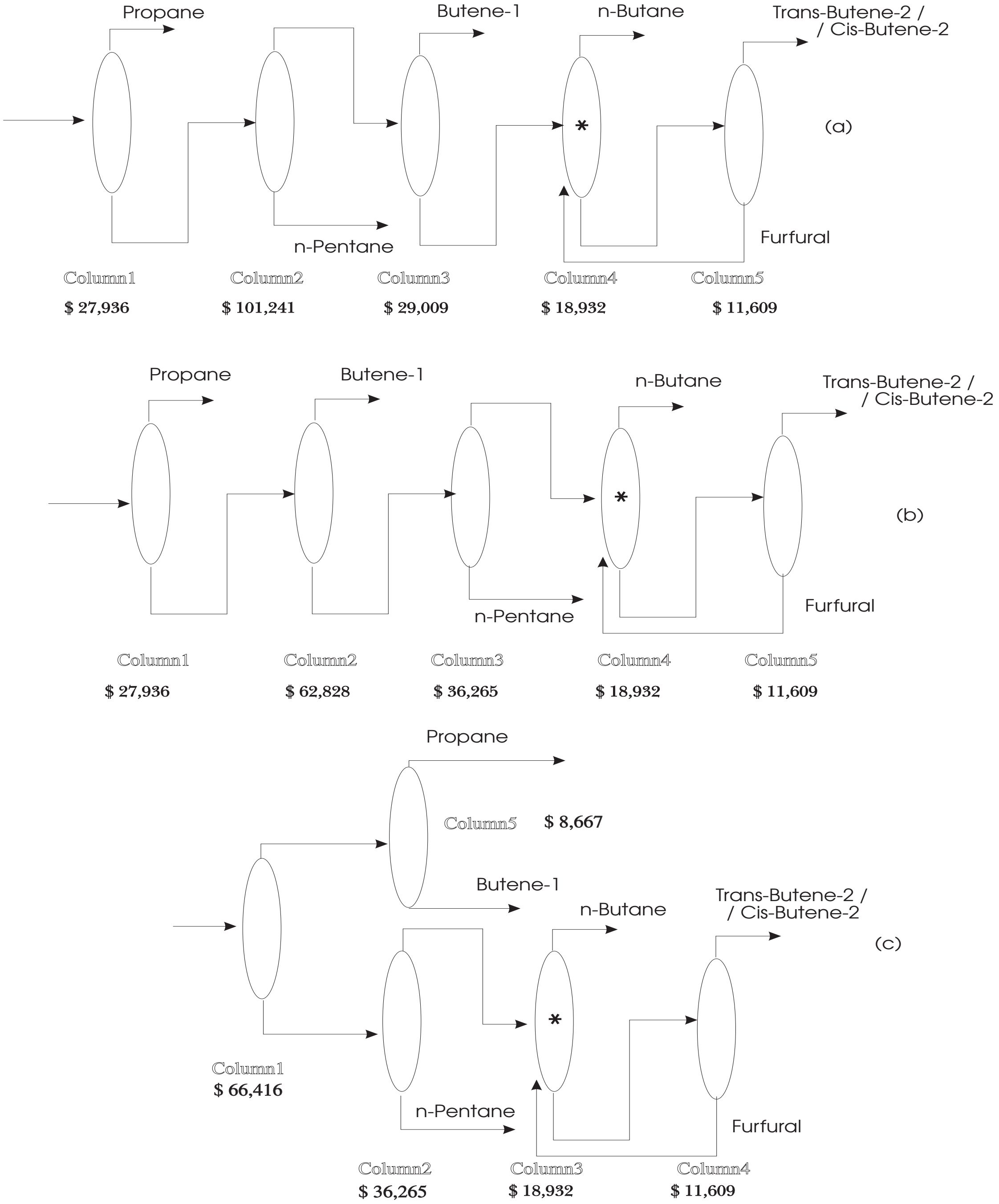
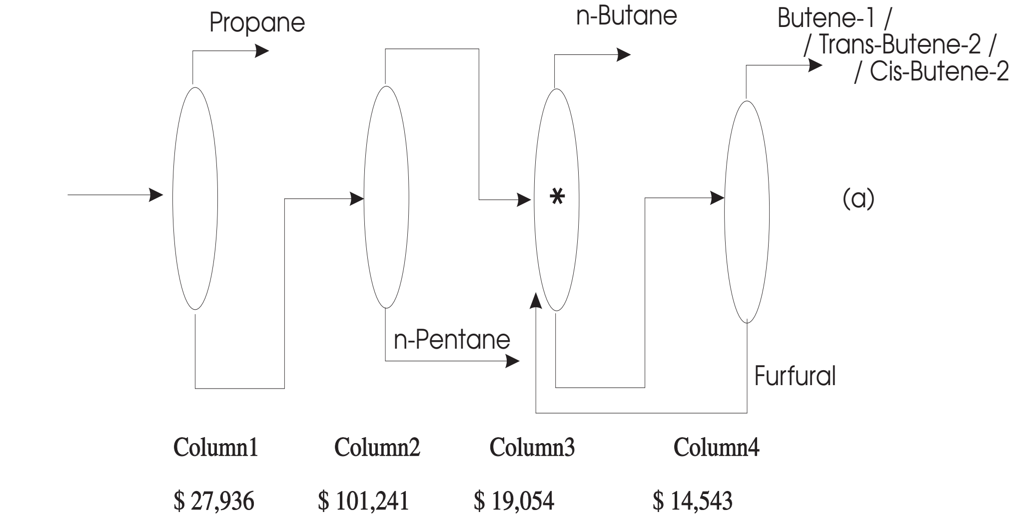


Figure 6.11: The first two flowsheets generated by the Nath & Notard method for the n-Butylene purification problem. The columns with an asterisk (*) indicate extractive distillation units.



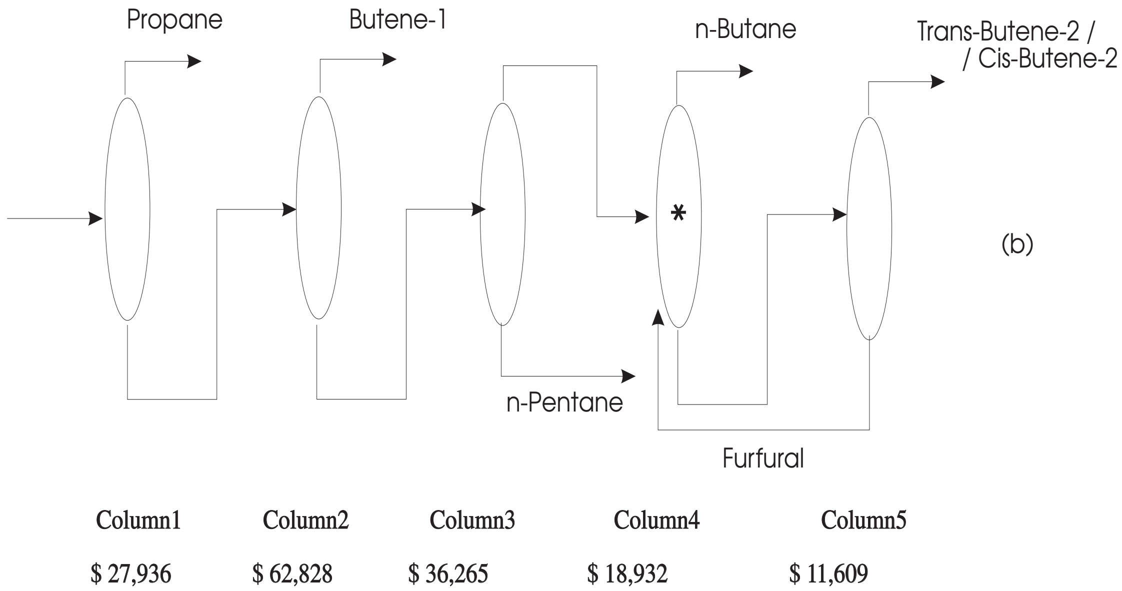
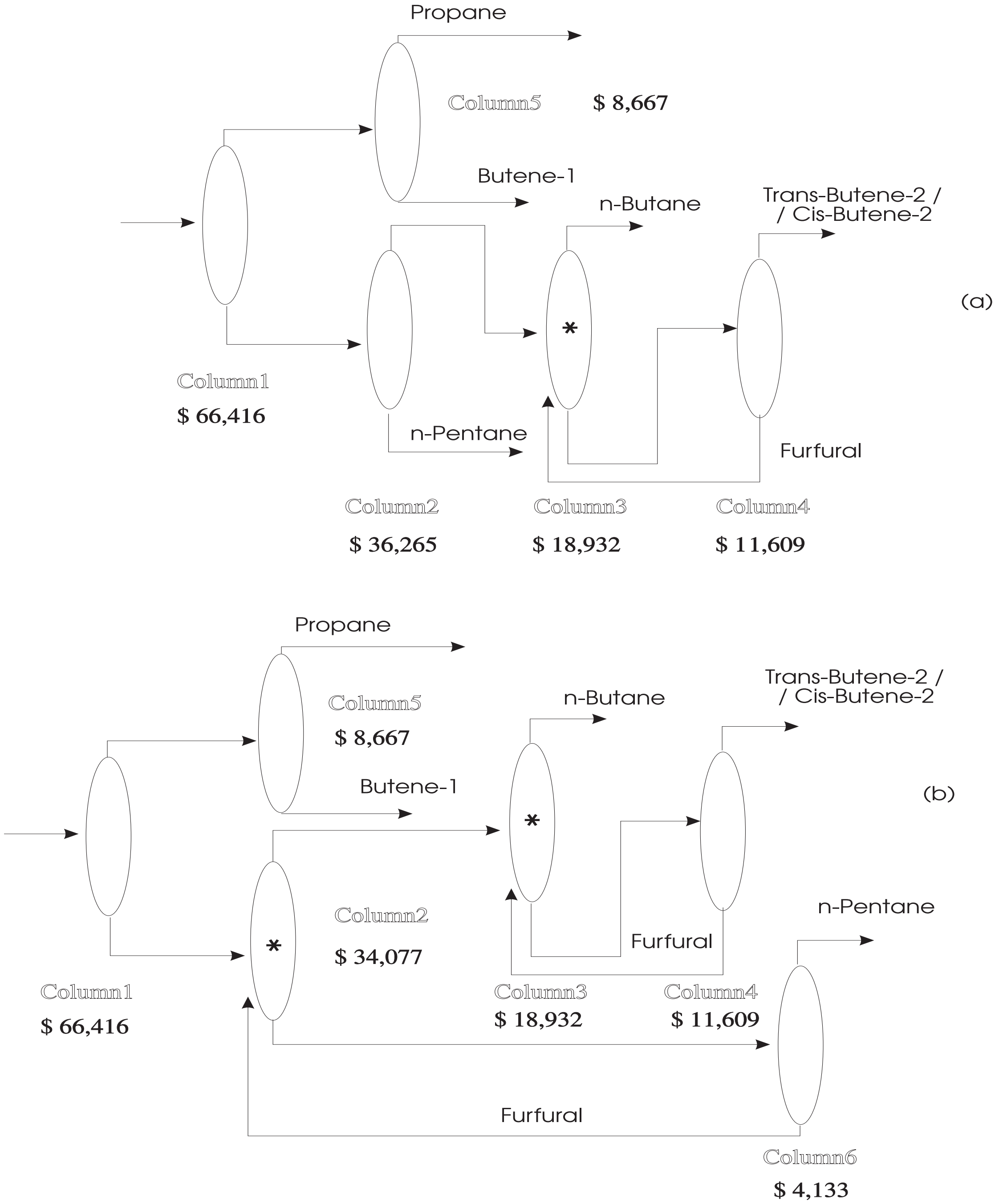


Figure 6.12: The last two flowsheets generated by the Nath & Notard method for the n-Butylene purification problem. The columns with an asterisk (*) indicate extractive distillation units.



Chapter 7

Conclusions

7.1 Contributions

The primary contributions of this thesis are:

- We describe DIAS, a computational framework that enables the transformation of informal accounts of innovative engineering design into precise computational models for performing the design task.
- We present a computational model for innovative design that views design as a sequence of design cycles.
- We describe a novel algorithm for orchestrating the use of physical and design knowledge in innovative design that is independent of heuristic or evolutionary design strategies.
- We develop a set of primitives for capturing design actions in DIAS.
- We demonstrate how to integrate a qualitative modeling language like QP Theory with numerical models and representations for heuristic knowledge in design.
- We describe a program (OUZO) which provides a platform for implementing different design strategies for separation systems.

7.2 Future Work

We have identified five areas for possible extensions of this research.

7.2.1 Generate explanations during the design process

DIAS focuses on the performance aspects of design and not on the generation of explanations for the design process. In general, while performance requires as much domain-specific knowledge as possible for efficiency purposes, explanation depends on representations relating the design results to the intuitions of the designer. Typically, in engineering design these intuitions can be expressed as commonsense rules balancing economic considerations against performance and functionality. For example, all of the heuristics for separation system design in Table 2.1 can be expressed as instances of more general guidelines for the design of filters (see section 3.5.6). Therefore, in order for our computational framework to support explanation generation, it will probably need to have a set of more abstract commonsense rules that describe the heuristics used in performing the design task.

Furthermore, the qualitative models in the physical knowledge component can form the basis for generating more intuitive explanations of the physical principles and the model construction process during the analysis phase of design.

7.2.2 Automate search for better design strategies

The development of evolutionary heuristics provides an example of how this approach might work. Most of these heuristic rules are developed by studying the effects that small structural changes introduce in configurations generated using an initial set of heuristics (see Table 5.1). The designer's task is to determine the application conditions for each modification that results with high enough frequency in superior designs. Machine learning techniques [56] can be used to automate the construction of evolutionary rules. Furthermore, planning techniques [2] can help to organize the application of new sets of initial and evolutionary heuristics in better design strategies.

7.2.3 Use more sophisticated model formulation techniques

At its current phase, DIAS supports compositional modeling techniques [15] for formulating physical knowledge models based on the current design description and the current sets of modeling and design assumptions. However, it is usually the case that the structure of the system and the assumptions for describing it are not the only parameters that determine how suitable for design is a physical model. Accuracy and computational efficiency play equally important roles in model selection. Therefore, the compositional modeling techniques in our approach need to be extended to capture the effects of these additional parameters during the analysis phase. Combining compositional modeling techniques with model sensitivity analysis [67] is a potential first step towards this direction.

7.2.4 Extend OUZO to other design domains

Different classes of process design problems in chemical engineering like control system synthesis, or total process flowsheets are possible new domains. Control system synthesis, in particular, presents a major challenge for our approach, since critical design decisions like the establishment of critical variables or the identification of significant interactions between design features and steady-state control are largely based on qualitative knowledge (e.g. order of magnitude reasoning) [45]. Other engineering domains where there is a singificant interest in using AI techniques (e.g. mechanical or aeronautical engineering) pose a new set of challenges for our approach.

APPENDIX A

Qualitative Process Theory: An Overview

Qualitative Process Theory (QP Theory) is a modeling language centered around the notion of processes as the agents of change in the world. The primitives used in QP Theory include *quantities*, *numbers*, *entities*, *views*, *perspectives*, *processes* and *influences*.

Quantities. The representation of a parameter of an object is called a *quantity*. Each quantity has two *numbers* associated with it: its amount (A) and its derivative (D). Each one of these numbers has a sign and a magnitude. A quantity belongs to at least one individual and it exists only when its owner does. Quantities are related with each other via a set of ordinal relationships. These include the greater-than, less-than and equal-to relations, their legal combinations and the *unrelated* relation. The later holds between two quantities when one of them does not exist.

Entities. Entities indicate instances of classes of objects. For example, various *container* entities represent instances of classes of objects that can be characterized as containers. Each entity contains the quantities of interest for the particular class of objects along with the causal dependencies and the inequality relations between them.

Views, Perspectives and Processes. Views and perspectives are conditional descriptions of state. Processes are conditional descriptions of behavior. Any instance of these model fragments exists for every combination of objects that match the descriptions of the *individuals* field in its description. Views and processes become active when their *preconditions* and *quantity conditions* are satisfied. Quantity conditions consist of sets of inequalities between quantities. Preconditions capture activation conditions that cannot be predicted solely by the dynamics of the situation (e.g. the actions of a human operator). Perspectives have no quantity conditions. They become active when their preconditions are satisfied.

Influences. Influences specifiy what can cause a quantity to change. There are two kinds of influences: *direct* and *indirect*. If a quantity is directly influenced, its derivative equals the sum of all the direct influences on it. There are two kinds of direct influences, I+ and I-, depending on whether the derivative of the quantity is monotonically increasing or decreasing with respect to the quantity it is influenced by. A quantity can be directly influenced only through a process. Processes and their influences are considered to be the causal origin of all change. If there are no active processes then all quantities must be constant.

An indirect influence occurs when a quantity is a function of some other quantity that is changing. There are three kinds of indirect influences, Qprop, Qprop- and Q=. The relation: (Qprop Q1 Q2) states that Q1 is monotonically increasing in Q2. The relation (Qprop- Q1 Q2) states that Q1 is monotonically decreasing

```
(defPerspective (Separation-System-Input ?system ?feed ?component)
  Individuals ((?system :Type Separation-System)
               (?feed :Type Multicomponent-Mixture
                      :Form (M-C-S ?components ?phase ?container)
                      :Conditions (Separation-System-Feed ?feed ?system))
               (?component :Type Substance
                           :Test (member ?component ?components)))
  Relations ((Non-Negative-Quantity (Feed-Flow ?component ?feed ?system))
             (Qprop (Total-Feed-Flow ?feed ?system) (Feed-Flow ?component ?feed ?system))))
```

Figure A.1: The input to a separation system for multicomponent mixtures. A term preceeded with a question mark (e.g. ?system) is a variable which is instantiated as part of the model construction process.

in Q2. Finally, the relation (Q= Q1 Q2) is translated as: (i) (equal-to (A Q1) (A Q2)), (ii) (equal-to (D Q1) (D Q2)) and (iii) (Qprop Q1 Q2).

A.1 An example

Figure A.1 provides an example of what the qualitative model fragment for the input for a separation system looks like in the domain theory. One more word about the syntax of QP Theory. As the example indicates the specification for each individual must have at least one of the following parts:

- The *:Type* part indicates that the next token is a unary predicate which must hold for an instance to exist.
- The *:Form* part constrains the possible bindings of the variable to those which unify with the form that follows.
- The *:Test* part signifies that the next form is going to be a LISP expression which should evaluate to a non-nil value for an instance to be created with the bindings so far.
- The *:Conditions* part indicates that all the remaining forms are additional statements that have to be believed for an instance to be created.

APPENDIX B

Numerical Models for Multicomponent Columns

The following section deals with two aspects of the numerical models that are used by the system, the calculation of relative volatilities and the generation of cost estimates.

B.1 Computing Relative Volatilities

In order to compute the relative volatilities for the various substances in a multicomponent mixture, the system uses a library of physical properties. The entries in this library include vapor pressures at various temperatures, boiling points at various pressures, the critical pressures and the Antoine equation coefficients for some of the substances. Figures B.1, B.2, B.3 and B.4 describe some of the equations that use property information to compute relative volatilities for the ordinary and extractive distillation cases.

In particular, Figures B.1 and B.2 show how the relative volatilities between two substances are calculated based on the equilibrium ratios (the *Relative-Volatility-Calculation-from-KValues* equation) or the vapor pressures between them (the *Relative-Volatility-Calculation-from-Vapor-Pressures* equation)[1]. We use the Antoine equation [72] for computing vapor pressures for substances, in case these are not directly available from the physical properties table. For the equilibrium ratios we use an equation from [54] (the *K-Ideal-Calculation-in-Reference-Conditions* equation in the figure). The *Liquid-Molal-Volume-in-Physical-Properties-Table* equation exemplifies the way physical properties are retrieved from the properties table. The table has an accessor function (*get-physical-property*) that gets evaluated using the *:EVALUATE* directive. The later informs the equation solver that the right hand side of the equation will not be algebraically solved but will be evaluated as a LISP expression. Most of these equations come in pairs that compute physical properties at reference and actual operating conditions.

Figures B.3 and B.4 describe the relative volatility calculations in the extractive distillation case. The computation of these parameters proceeds in a similar way with the ordinary case except that we have to multiply the relative volatility for the ordinary distillation case with the selectivity between the key components (the *Extractive-Alpha-LK-HK-Calculation-for-Keys-and-Polar-Solvent-in-Ternary* and the *Extractive-Alpha-LK-HK-Calculation* equations). The selectivity of two substances is defined as the ratio of the relative volatility of the key components in the mixture which are to be separated in the presence of the mass separating agent to their relative volatility before the addition of the agent. The formula we

[1] There are similar equations for the actual operating conditions for the column.

```
(defEquation Antoine-Equation
 ((Separation-Properties-in-Feed-Conditions ?feed ?column ?substance))
 ((Vapor-Pressure ?substance ?column :operating)
  (Alpha-1 ?substance)
  (Alpha-2 ?substance)
  (Alpha-3 ?substance)
  (Feed-Temperature ?feed ?column)
  (Critical-Pressure ?substance))
 (= (A (Vapor-Pressure ?substance ?column :operating))   ;; (in psia)
    (* (A (Critical-Pressure ?substance))
       (exp (- (A (Alpha-1 ?substance))
               (/ (A (Alpha-2 ?substance))
                  (+ (A (Feed-Temperature ?feed ?column)) (A (Alpha-3 ?substance)))))))))

(defEquation Liquid-Molal-Volume-in-Physical-Properties-Table
 ((K-Ideal-Substance-Properties ?substance ?op-temp ?ref-temp ?phys-temp ?system ?op-pres ?column))
 ((Liquid-Molal-Volume ?substance ?phys-temp))
 (= (A (Liquid-Molal-Volume ?substance ?phys-temp))
    (:EVALUATE (get-physical-property Liquid-Molal-Volume ?substance ?phys-temp))))

(defEquation Liquid-Volume-Calculation
 ((K-Ideal-Substance-Properties ?substance ?op-temp ?ref-temp ?phys-temp ?system ?op-pres ?column))
 ((Liquid-Volume-Constant ?substance)
  (Liquid-Molal-Volume ?substance ?phys-temp)
  (Critical-Temperature ?substance))
 (= (A (Liquid-Volume-Constant ?substance))
    (/ (A (Liquid-Molal-Volume ?substance ?phys-temp))
       (+ 5.7 (* 3.0 (/ (* 1.8 (+ ?phys-temp 273.15)) (A (Critical-Temperature ?substance))))))))

(defEquation K-Ideal-Calculation-in-Reference-Conditions
 ((K-Ideal-Substance-Properties ?substance ?op-temp ?ref-temp ?phys-temp ?system ?op-pres ?column)
  (Value-of (A (Atmospheric-Pressure ?env)) ?atm ?eqn))
 ((K-Value ?substance ?column :reference)
  (Vapor-Pressure ?substance ?column :reference)
  (Critical-Temperature ?substance)
  (Critical-Pressure ?substance)
  (Liquid-Molal-Volume ?substance ?ref-temp))
 (= (A (K-Value ?substance ?column :reference))
    (* (/ (A (Vapor-Pressure ?substance ?column :reference)) (/ ?atm 6894.8))
       (exp (+ (/ (* 0.4278 (- (/ ?atm 6894.8) (A (Vapor-Pressure ?substance ?column :reference))))
                  (* (expt (/ (+ ?ref-temp 459.67) (A (Critical-Temperature ?substance))) 2.5)
                     (A (Critical-Pressure ?substance))))
               (/ (* (A (Liquid-Molal-Volume ?substance ?ref-temp))
                     (- (/ ?atm 6894.8) (A (Vapor-Pressure ?substance ?column :reference))))
                  (* (+ ?ref-temp 459.67) 10.73)))))))
```

Figure B.1: Computing the vapor pressures and the equilibrium ratios of the substances in a multicomponent mixture for the ordinary distillation case.

```
(defEquation Relative-Volatility-Calculation-from-KValues
 ((Distillation-Features-at-Reference-Conditions
  distillation ?column ?h-k ?l-k ?pressure (M-C-S ?components ?phase ?stage) ?op-pres ?ref-temp)
  :Test (null (polar-solvent-in? ?components)))
 ((Alpha-LK-HK ?l-k ?h-k ?column :reference)
  (K-Value ?l-k ?column :reference)
  (K-Value ?h-k ?column :reference))
 (= (A (Alpha-LK-HK ?l-k ?h-k ?column :reference))
    (/ (A (K-Value ?l-k ?column :reference)) (A (K-Value ?h-k ?column :reference)))))

(defEquation Relative-Volatility-Calculation-from-Vapor-Pressures
 ((Distillation-Features-at-Reference-Conditions
   distillation ?column ?h-k ?l-k ?pressure (M-C-S ?components ?phase ?stage) ?op-pres ?ref-temp)
  :Test (null (polar-solvent-in? ?components)))
 ((Alpha-LK-HK ?l-k ?h-k ?column :reference)
  (Vapor-Pressure ?l-k ?column :reference)
  (Vapor-Pressure ?h-k ?column :reference))
 (= (A (Alpha-LK-HK ?l-k ?h-k ?column :reference))
     (/ (A (Vapor-Pressure ?l-k ?column :reference))
        (A (Vapor-Pressure ?h-k ?column :reference)))))
```

Figure B.2: Computing the relative volatilities of the substances in a multicomponent mixture for the ordinary distillation case.

use for calculating the selectivity (the *Selectivity-Calculation* equation) is based on an infinite dilution of the substances in the mass separating agent (the Weimer and Prausnitz method from [72]). This is an approximation since the polar solvent concentration of the solvent that we are actually using is .9 of the total volume of the feed to the extractive distillation column[2]. In an actual design system we would expect to have access to data that would allow us to assume non-infinite dilution for the selectivity parameter. Unfortunately we did not have access to this data for this work.

The equations refering to the reference conditions for each column are activated for all the separation alternatives. The set of equations dealing with the actual operating conditions are active only for the design alternatives that are found to be promising by the design knowledge.

[2] According to the *(Consider (Polar-Solvent-Concentration ?p-s .9))* statement in Figure 4.37.

```
(defEquation Weimer-&-Prausnitz-Method-Activity-Coefficients
 ((Extractive-Distillation-Features-at-Actual-Conditions
   ?column ?l-k ?h-k ?p-s ?pres ?temp (M-C-S ?components ?phase ?stage))
  (Substance ?s) :Test (and (not (eql ?s ?p-s)) (member ?s ?components)))
 ((Polar-Solubility ?p-s ?column :operating)
  (Energy-of-Interaction ?s ?p-s ?column :operating)
  (NonPolar-Solubility ?s ?column :operating)
  (NonPolar-Solubility ?p-s ?column :operating)
  (Molar-Volume ?s ?column :operating)
  (Molar-Volume ?p-s ?column :operating)
  (Gas-Constant ?env)
  (Infinite-Activity-Coefficient ?s ?p-s ?column :operating))
 (= (A (Infinite-Activity-Coefficient ?s ?p-s ?column :operating))
    (exp (/ (+ (* (A (Molar-Volume ?s ?column :operating))
                  (- (+ (expt (- (A (NonPolar-Solubility ?p-s ?column :operating))
                                 (A (NonPolar-Solubility ?s ?column :operating))) 2)
                        (expt (A (Polar-Solubility ?p-s ?column :operating)) 2))
                     (* 2.0 (A (Energy-of-Interaction ?s ?p-s ?column :operating)))))
               (* (* (A (Gas-Constant ?env)) ?temp)
                  (- (+ (log (/ (A (Molar-Volume ?s ?column :operating))
                                (A (Molar-Volume ?p-s ?column :operating)))) 1.0)
                     (/ (A (Molar-Volume ?s ?column :operating))
                        (A (Molar-Volume ?p-s ?column :operating))))))
            (* (A (Gas-Constant ?env)) ?temp)))))

(defEquation Selectivity-Calculation
 ((Extractive-Distillation-Features-at-Actual-Conditions ?column ?l-k ?h-k ?p-s ?pres ?temp ?feed))
 ((Infinite-Activity-Coefficient ?h-k ?p-s ?column :operating)
  (Infinite-Activity-Coefficient ?l-k ?p-s ?column :operating)
  (Selectivity ?l-k ?h-k ?p-s ?column :operating))
 (= (A (Selectivity ?l-k ?h-k ?p-s ?column :operating))
    (/ (A (Infinite-Activity-Coefficient ?l-k ?p-s ?column :operating))
       (A (Infinite-Activity-Coefficient ?h-k ?p-s ?column :operating)))))
```

Figure B.3: Computing infinite activity coefficients and selectivities for the extractive distillation case. The parameters for which we have supplied no equations in this figure (e.g. Energy-of-Interaction, NonPolar-Solubility, etc) are computed directly from the physical properties table.

```
(defEquation Extractive-Alpha-LK-HK-Calculation-for-Keys-and-Polar-Solvent-in-Ternary
 ((Extractive-Distillation-Features-at-Actual-Conditions ?column ?l-k ?h-k ?p-s ?pres ?op-temp ?feed)
  (Extractive-Ternary-Mixture-Underwood-Rel ?p-s ?elem1 ?elem2 ?l-k ?h-k ?feed ?column ?op-temp)
  (Substance ?s) :Test (member ?s '(,?elem1 ,?elem2)))
 ((Infinite-Activity-Coefficient ?s ?p-s ?column :operating)
  (Vapor-Pressure ?s ?column :operating)
  (Vapor-Pressure ?p-s ?column :operating)
  (Extractive-Alpha-LK-HK ?s ?p-s ?p-s ?column :operating))
 (= (A (Extractive-Alpha-LK-HK ?s ?p-s ?p-s ?column :operating))
    (/ (* (A (Infinite-Activity-Coefficient ?s ?p-s ?column :operating))
          (A (Vapor-Pressure ?s ?column :operating)))
       (A (Vapor-Pressure ?p-s ?column :operating)))))

(defEquation Extractive-Alpha-LK-HK-Calculation
 ((Extractive-Distillation-Features-at-Actual-Conditions ?column ?l-k ?h-k ?p-s ?pres ?temp ?feed))
 ((Extractive-Alpha-LK-HK ?l-k ?h-k ?p-s ?column :operating)
  (Selectivity ?l-k ?h-k ?p-s ?column :operating)
  (K-Value ?l-k ?column :operating)
  (K-Value ?h-k ?column :operating))
 (= (A (Extractive-Alpha-LK-HK ?l-k ?h-k ?p-s ?column :operating))
    (* (A (Selectivity ?l-k ?h-k ?p-s ?column :operating))
       (/ (A (K-Value ?l-k ?column :operating )) (A (K-Value ?h-k ?column :operating))))))
```

Figure B.4: Computing the relative volatilities of the substances in a multicomponent mixture for the extractive distillation case.

B.2 Generating Cost Estimates

All the numerical models described below are activated only for the alternatives that were identified as promising by the design heuristics.

The method for producing cost estimates is based on the Fenske-Underwood-Gilliland method for multicomponent distillation columns. Figures B.5, B.6, B.7 and B.8 describe all the equations in the version of the method that we use. The method starts by calculating the parameter *Phi* (in equation *Ternary-Mixture-Underwood-Phi-Calculation*, Fig. B.5) which is a special-purpose parameter used for computing the reflux ratio for a given separation using the Underwood equation [35]. Note that there is a constant relative volatility value used in this equation for each pair of substances. All of the shortcut design procedures rely on this assumption [13]. In addition, the sharp separation assumption in the preconditions of this equation allows us to use the feed flow rate for each substance instead of the distillate flow rate for that substance. Finally, we assume that the feed to the current column is always liquid.

The procedure *interval-halving-method* in the Ternary-Mixture-Underwood-Phi-Calculation equation implements an interval halving algorithm for finding through iteration the roots of the expression in tis third argument. The numerical intervals for the solution are indicated by the first (upper bounds) and second (lower bounds) arguments to the procedure. The fourth argument refers to the acceptable error margin for the method. The fifth argument is the quantity we are solving for and, finally, the sixth argument represents the monotonicity of the expression with respect to the variable we are solving for (+ for increasing and − for decreasing monotonicity).

The Phi parameter is used in the *Min-Reflux-Ratio-1-for-3* and *Min-Reflux-Ratio* equations for calculating the minimum reflux ratio (the quantity *Min-Reflux-Ratio*) for the column. The quantity *Min-Reflux-Ratio-1* is used to hold an intermediate result during this calculation. The Min-Reflux-Ratio-1-for-3 equation refers to ternary mixtures. There are similar equations for quadratic, quintic and hexadic mixtures. There is also a similar set of equations covering extractive distillation for all these types of mixtures. Finally, the *Sigma* expression in the Min-Reflux-Ratio-1-for-3 equation refers to the $\sum$ symbol for addition. The equation solver recognizes this special symbol and solves the composite equation accordingly.

The minimum reflux ratio is used in Figure B.7 in the *Operating-Reflux-Ratio-1* equation to compute an estimate of the operating reflux ratio (the quantity *Reflux-Ratio*) for the column. The particular value of the *Operating-Reflux-Estimate* used is 1.2 as suggested by [13]. This value is supplied to the model as one of the design assumptions in the problem specification.

The operating reflux ratio is then combined with the average vapor velocity calculated by the *Average-Vapor-Velocity-for-Ordinary-Distillation* equation taken from [49] to give an estimation of the diameter of the column that is going to achieve a given separation (the *Column-Diameter-1* equation taken again from [49]).

The *Min-Number-of-Stages* equation in Figure B.7 is used to estimate the minimum number of stages needed to achieve a given separation. This equation is the same with the Gilliland-Fenske equation for binary mixtures. The sharp separation assumption allows us to use it for multicomponent mixtures. The minimum number of stages is used in the *Column-Height-for-Ordinary-Distillation* equation to compute an estimation for the height of a column that achieves a given separation.

Finally, the estimates for the column and the diameter of the column are combined in Figure B.8 to give an estimate for the installation cost for the column (the equation is taken from [49]). We assume that the total cost for the column is equal to the installation cost, therefore we do not take into account the annual operating cost for each separation unit.

The program contains a similar set of equations that estimates all these parameters for the extractive distillation case.

```
(defEquation Ternary-Mixture-Underwood-Phi-Calculation
 ((Consider (Possible (Separation distillation (?l-k ?h-k) ?column)))
  (Consider (Reference-Component ?comp (M-C-S (?elem1 ?elem2 ?elem3) ?phase ?stage)))
  (Consider (Liquid-Phase-Feed (M-C-S ?components ?phase ?stage) ?column))
  (Actual-Conditions ?column distillation (Pressure ?op-pres) (Temp ?op-temp) (?l-k ?h-k)))
 ((Phi (?elem1 ?elem2 ?elem3) ?column)
  (Alpha-LK-HK ?elem1 ?comp ?column :operating)
  (Alpha-LK-HK ?elem2 ?comp ?column :operating)
  (Alpha-LK-HK ?elem3 ?comp ?column :operating)
  (Feed-Flow ?elem1 (M-C-S ?components ?phase ?stage) ?column)
  (Feed-Flow ?elem2 (M-C-S ?components ?phase ?stage) ?column)
  (Feed-Flow ?elem3 (M-C-S ?components ?phase ?stage) ?column))
 (= (A (Phi (?elem1 ?elem2 ?elem3) ?column))
    (:EVALUATE
     (interval-halving-method
      (3.6 2.4 1.3)
      (3.5 2.2 1.001)
      (+ (/ (* (A (Alpha-LK-HK ?elem1 ?comp ?column :operating))
               (A (Feed-Flow ?elem1 (M-C-S ?components ?phase ?stage) ?column)))
            (- (A (Alpha-LK-HK ?elem1 ?comp ?column :operating))
               (A (Phi (?elem1 ?elem2 ?elem3) ?column))))
         (/ (* (A (Alpha-LK-HK ?elem2 ?comp ?column :operating))
               (A (Feed-Flow ?elem2 (M-C-S ?components ?phase ?stage) ?column)))
            (- (A (Alpha-LK-HK ?elem2 ?comp ?column :operating))
               (A (Phi (?elem1 ?elem2 ?elem3) ?column))))
         (/ (* (A (Alpha-LK-HK ?elem3 ?comp ?column :operating))
               (A (Feed-Flow ?elem3 (M-C-S ?components ?phase ?stage) ?column)))
            (- (A (Alpha-LK-HK ?elem3 ?comp ?column :operating))
               (A (Phi (?elem1 ?elem2 ?elem3) ?column)))))
      .05
      0.0
      (A (Phi (?elem1 ?elem2 ?elem3) ?column))
      +))))
```

Figure B.5: Estimating the Phi parameter for ordinary distillation and a ternary mixture. The program uses similar equations for quadratic, quintic and hexadic mixtures as well as for the extractive distillation case.

```
(defEquation Min-Reflux-Ratio-1-for-3
 ((Consider (Possible (Separation distillation (?l-k ?h-k) ?column)))
  (Consider (Reference-Component ?comp (M-C-S (?elem1 ?elem2 ?elem3) ?phase ?stage)))
  (Distillate-Products ?products ?column)
  (Substance ?product)
  :Test (and (member ?product ?products)
     (member ?product '(,?elem1 ,?elem2 ,?elem3)))
  (Actual-Conditions ?column distillation (Pressure ?op-pres) (Temp ?t) (?l-k ?h-k))
  (Column-Feed (M-C-S ?components ?phase ?stage) ?column))
 ((Distillate-Component-Flow-Rate ?product ?column (?l-k ?h-k))
  (Total-Distillate-Flow-Rate ?column (?l-k ?h-k))
  (Alpha-LK-HK ?product ?comp ?column :operating)
  (Phi ?components ?column)
  (Min-Reflux-Ratio-1 (?l-k ?h-k) ?column))
 (= (A (Min-Reflux-Ratio-1 (?l-k ?h-k) ?column))
    (Sigma
     (/ (* (/ (A (Distillate-Component-Flow-Rate ?product ?column (?l-k ?h-k)))
      (A (Total-Distillate-Flow-Rate ?column (?l-k ?h-k))))
           (A (Alpha-LK-HK ?product ?comp ?column :operating)))
        (- (A (Alpha-LK-HK ?product ?comp ?column :operating))
           (A (Phi ?components ?column)))))))

(defEquation Min-Reflux-Ratio
 ((Consider (Possible (Separation distillation (?l-k ?h-k) ?column)))
  (Consider (Reference-Component ?comp (M-C-S ?substances ?phase ?stage)))
  (Actual-Conditions ?column distillation (Pressure ?op-pres) (Temp ?t) (?l-k ?h-k))
  (Column-Feed (M-C-S ?components ?phase ?stage) ?column))
 ((Distillate-Component-Flow-Rate ?h-k ?column (?l-k ?h-k))
  (Total-Distillate-Flow-Rate ?column (?l-k ?h-k))
  (Alpha-LK-HK ?h-k ?comp ?column :operating)
  (Phi ?substances ?column)
  (Min-Reflux-Ratio (?l-k ?h-k) ?column)
  (Min-Reflux-Ratio-1 (?l-k ?h-k) ?column))
 (= (A (Min-Reflux-Ratio (?l-k ?h-k) ?column))
    (+ (A (Min-Reflux-Ratio-1 (?l-k ?h-k) ?column))
       (/ (* (/ (A (Distillate-Component-Flow-Rate ?h-k ?column (?l-k ?h-k)))
                (A (Total-Distillate-Flow-Rate ?column (?l-k ?h-k))))
             (A (Alpha-LK-HK ?h-k ?comp ?column :operating)))
          (- (A (Alpha-LK-HK ?h-k ?comp ?column :operating))
             (A (Phi ?substances ?column)))))))
```

Figure B.6: Estimating the minimum reflux ratio for a given separation using the Underwood method for the ordinary distillation case.

```
(defEquation Min-Number-of-Stages
 ((Distillation-Features-in-Actual-Conditions
   distillation ?column ?h-k ?l-k ?pressure (M-C-S ?components ?phase ?stage) ?op-temp))
  ((Min-#-of-Stages ?column (?l-k ?h-k))
   (Distillate-Component-Flow-Rate ?l-k ?column (?l-k ?h-k))
   (Distillate-Component-Flow-Rate ?h-k ?column (?l-k ?h-k))
   (Bottom-Component-Flow-Rate ?l-k ?column (?l-k ?h-k))
   (Bottom-Component-Flow-Rate ?h-k ?column (?l-k ?h-k))
   (Alpha-LK-HK ?l-k ?h-k ?column :operating))
  (= (A (Min-#-of-Stages ?column (?l-k ?h-k)))
     (/ (log (* (/ (A (Distillate-Component-Flow-Rate ?l-k ?column (?l-k ?h-k)))
                    (A (Distillate-Component-Flow-Rate ?h-k ?column (?l-k ?h-k))))
                (/ (A (Bottom-Component-Flow-Rate ?h-k ?column (?l-k ?h-k)))
                   (A (Bottom-Component-Flow-Rate ?l-k ?column (?l-k ?h-k))))))
        (log (A (Alpha-LK-HK ?l-k ?h-k ?column :operating))))))

(defEquation Average-Vapor-Velocity-for-Ordinary-Distillation
 ((Distillation-Features-in-Actual-Conditions
   distillation ?column ?h-k ?l-k ?pressure ?feed ?op-temp)
 ((Average-Vapor-Velocity ?column (?l-k ?h-k)))
 (= (A (Average-Vapor-Velocity ?column (?l-k ?h-k))) (* .761 (sqrt (/ 1.0 (/ ?pressure 101330.0))))))

(defEquation Operating-Reflux-Ratio-1
 ((Distillation-Features-in-Actual-Conditions
   distillation ?column ?h-k ?l-k ?pressure ?feed ?op-temp)
  (Consider (Operating-Reflux-Estimate ?val)))
 ((Reflux-Ratio (?l-k ?h-k) ?column)
  (Min-Reflux-Ratio (?l-k ?h-k) ?column))
 (= (A (Reflux-Ratio (?l-k ?h-k) ?column))
    (* ?val (A (Min-Reflux-Ratio (?l-k ?h-k) ?column)))))

(defEquation Column-Diameter-1
 ((Distillation-Features-in-Actual-Conditions
   distillation ?column ?h-k ?l-k ?pressure ?feed ?op-temp)
  (First-Column ?system ?column)
  (Reference-Conditions ?column (Pressure ?ref-pres) (Temp ?ref-temp)))
 ((Column-Diameter ?column distillation (?l-k ?h-k))
  (Average-Vapor-Velocity ?column (?l-k ?h-k))
  (Reflux-Ratio (?l-k ?h-k) ?column)
  (Total-Feed-Flow ?feed ?column)
  (TBoil ?l-k ?system :reference)
  (Total-Distillate-Flow-Rate ?column (?l-k ?h-k)))
 (= (A (Column-Diameter ?column distillation (?l-k ?h-k)))
    (sqrt (* (/ 4.0
                (* pi (A (Average-Vapor-Velocity ?column (?l-k ?h-k)))))
             (A (Total-Distillate-Flow-Rate ?column (?l-k ?h-k)))
             (+ 1.0 (A (Reflux-Ratio (?l-k ?h-k) ?column)))
             22.2
;; APPROX: Because there is a large concentration of only one desired product and an almost atmospheric operating pressure
;; it is assumed that the dew point of the vapors at the distillate is nearly equal to the boiling point of the desired product.
             (/ (+ (A (TBoil ?l-k ?system :reference)) 273.15)  ;;; from oC to oK
                273.0)
             (/ 1.0 (/ ?pressure 101330.0)  ;;; from Pa to atm
             (/ 1.0 3600.0))))))
```

Figure B.7: Estimating the required diameter for a column that achieve a given separation for the ordinary distillation case.

```
(defEquation Column-Height-for-Ordinary-Distillation
 ((Distillation-Features-in-Actual-Conditions distillation ?column ?h-k ?l-k ?pressure
   ?feed ?op-temp)
  (Consider Stage-Efficiency ?efficiency))
 ((Min-#-of-Stages ?column (?l-k ?h-k))
  (Column-Height ?column distillation (?l-k ?h-k)))
 (= (A (Column-Height ?column distillation (?l-k ?h-k)))
    (+ (* .61 (/ (A (Min-#-of-Stages ?column (?l-k ?h-k))) ?efficiency)) 4.27)))

(defEquation Min-Column-Cost-for-Ordinary-Distillation
 ((Distillation-Features-in-Actual-Conditions distillation ?column ?h-k ?l-k ?pressure
   ?feed ?op-temp))
 ((Column-Height ?column distillation (?l-k ?h-k))
  (Column-Diameter ?column distillation (?l-k ?h-k))
  (Installation-Cost ?column (?l-k ?h-k)))
 (= (A (Installation-Cost ?column (?l-k ?h-k)))
    (* 43.4 7620.0 (A (Column-Diameter ?column distillation (?l-k ?h-k)))
       (expt (/ (A (Column-Height ?column distillation (?l-k ?h-k))) 12.2) .68))))
```

Figure B.8: Computing the height and the final cost estimate for the current column in the ordinary distillation case.

APPENDIX C

Configuration Synthesis Rules in OUZO

```
(ConfigSynthRule Determine-Feed-Flows-for-Distillate-Feed
 ((Missing-Products ?desired-products ?prev-feed (?l-k ?h-k) ?column)
  (Distillate-Products ?distillate-products ?column)
  :Test (subsetp ?desired-products ?distillate-products)
  (Distillate-Feed :From ?column :To ?system)
  (Separation-System-Feed ?feed ?system)
  (Value-of (A (Distillate-Component-Flow-Rate ?product ?column (?l-k ?h-k))) ?d ?d-eqn)
  :Test (member ?product ?desired-products))
 (unless (member '(assign (A (Feed-Flow ,?product ,?feed ,?system)) ,?d) *problem* :Test #'equal)
    (assign-in-problem '(A (Feed-Flow ,?product ,?feed ,?system)) ?d)))

 IF ?desired-products have not been recovered yet
    AND ?desired-products are a subset of the distillate products for ?column
    AND separation system ?system feeds from the distillate products for ?column
    AND ?product is one of the distillate products for ?column
          for which we know its distillate flow rate
 THEN the feed flow for ?product in ?column will be equal to the distillate flow rate for ?product.
```

Figure C.1: Assert the feed flow rates for a separation system that feeds from the distillate products of the current column. There is a similar rule for the bottom products for the current column. A pair of similar rules are also used for the distillate and bottom products for the extractive distillation case.

C.1 Overview

Configuration synthesis rules contain methods for changing the current design description and monitoring the state of design. Figures C.2, C.3, C.6, C.7, C.5, C.9, C.10, C.11, C.1, C.4, C.12, C.13, C.14 and C.15 describe these rules in detail. Each figure contains the actual encoding of the rule along with a summary of its content (in English).

The majority of these rules encode procedural knowledge on how to update the current flowsheet. Consequently, they have a strong procedural flavor as indicated by the frequent use of LISP code in their body. Most of this code is hidden behind procedure calls. Apart form the primitives described in section 5.1.2 there are three major procedures used in the bodies of these rules.

The procedure *delete-assertion* removes an assertion that unifies with its argument from **scenario**. *Assign-in-problem* is used to assign numerical values to parameters in the design description. The global variable **problem** holds all these assignments. Finally, *clear-eqn-table* removes all the equations from the list of active equations *(*active-eqns*)* that refer to the column or the separation system specified in its argument.

```
(ConfigSynthRule Establish-Equilibrium-Temperature
 ((Value-of (A (Equilibrium-Temperature ?component ?pressure)) ?temp ?eqn) :Var ?f1)
 (unless (find-if #'(lambda (x)
                     (and (eql (car (second x)) 'Value-of)
                          (equal (second (second x)) (second (car ?f1)))))
                   *scenario*)
    (assert-in-scenario (car ?f1))))

(ConfigSynthRule Establish-K-Ideal-Temperatures
 ((Value-of (A (K-Ideal-Reference-Temperature ?component)) ?temp ?eqn) :Var ?f1)
 (unless (find-if #'(lambda (x)
                     (and (eql (car (second x)) 'Value-of)
                          (equal (second (second x)) (second (car ?f1)))))
                  *scenario*)
     (assert-in-scenario (car ?f1))))

(ConfigSynthRule Establish-Boiling-Points
 ((Value-of (A (TBoil ?substance ?system :reference)) ?boil-point ?eqn) :Var ?f1)
 (unless
  (find-if #'(lambda (x)
               (and (eql (car (second x)) 'Value-of)
                    (equal (second (second x)) (second (car ?f1)))))
           *scenario*)
    (assert-in-scenario (car ?f1))))

For each substance establish its boiling point and the reference
temperature under which the calculation of the equilibrium ration is based.

(ConfigSynthRule Establish-System-Reference-Temperature
 ((Value-of (A (Reference-Temperature ?system)) ?ref-temp ?eqn) :Var ?f1)
 (unless (find-if #'(lambda (x)
                     (and (eql (car (second x)) 'Value-of)
                          (equal (second (second x)) (second (car ?f1)))))
                  *scenario*)
    (assert-in-scenario (car ?f1))))

Insert the reference temperature for the current separation system in the design description.
```

Figure C.2: Insert physical properties for each substance and separation system in the design description.

```
(ConfigSynthRule Establish-K-Value-Relations
 ((Value-of (A (TBoil ?substance1 ?system :reference)) ?boil-1 ?eqn-1)
  (Value-of (A (TBoil ?substance2 ?system :reference)) ?boil-2 ?eqn-2) :Test (< ?boil-1 ?boil-2))
 (let ((n1 `(A (TBoil ,?substance2 ,?system :reference)))
       (n2 `(A (TBoil ,?substance1 ,?system :reference))))
   (unless (or (member `(assertq (greater-than ,n1 ,n2)) *scenario* :test #'equal)
               (member `(assertq (less-than ,n2 ,n1)) *scenario* :test #'equal))
     (if (equal n1 (car (qpe::find-qrel n1 n2)))
         (pushnew `(assertq (greater-than ,n1 ,n2)) *scenario* :test #'equal)
       (pushnew `(assertq (less-than ,n2 ,n1)) *scenario* :test #'equal)))))

 IF ?substance1 has lower boiling than ?substance2
 THEN assert the relation between the boiling points in the design description.
```

Figure C.3: Include the relations between the boiling points of substances in the design description.

```
(ConfigSynthRule Determine-Polar-Solvent-Feed-Flow
 ((Consider (Possible (Separation extractive-distillation (?l-k ?h-k) ?column)))
  (Bottom-Feed :From ?column :To ?system)
  (Separation-System-Feed ?feed ?system)
  (Polar-Solvent ?p-s)
  (Attracts ?p-s ?h-k)
  (Consider (Polar-Solvent-Concentration ?p-s ?c))
  (Value-of (A (Total-Feed-Flow ?prev-feed ?column)) ?f ?f-eqn))
 (unless (or (member
              `(assign (A (Feed-Flow ,?p-s ,?feed ,?system)) ,(* ?c ?f)) *problem* :Test #'equal)
             (exists `(Value-of (A (Feed-Flow ,?p-s ?f ,?system)) . ?y)))
    (assign-in-problem `(A (Feed-Flow ,?p-s ,?feed ,?system)) (* ?c ?f))))

 IF extractive distillation takes place in ?column
    AND separation system ?system feeds from the bottom products for ?column
    AND ?p-s is the polar solvent used in ?column
    AND the total feed flow for ?column is ?f
    AND the desired concentration for ?p-s is ?c
 THEN the feed flow for ?p-s in ?system will be equal to the product of ?c and ?f.
```

Figure C.4: Assert the feed flow rates for the polar solvent in a separation system that feeds from the bottom products of the current column.

```
(ConfigSynthRule Update-Num-of-Stages
 ((Steady-State-Column-Design-Features
   ?column ?c-stage ?c-phase ?r-stage ?r-phase ?distillate-flow
   ?bproduct-flow ?substance1 ?substance2)
  (Consider (Partial-Condenser ?column))
  (Consider (Partial-Reboiler ?column))
  (Condenser-Stage ?condenser-stage)
  (Reboiler-Stage ?reboiler-stage)
  (Value-of (A (Num-of-Stages ?column)) ?num-of-stages ?eqn))
 (let ((Cur-Stage nil)
       (stage-list `(,?reboiler-stage ,?condenser-stage))
       (prev-num-of-stages (second (car (find-if #'(lambda (x) (in? x))
                                               (exists '(Num-of-Stages . ?x)))))))
   (unless (and prev-num-of-stages (= prev-num-of-stages (truncate ?num-of-stages)))
     (assert-in-scenario `(Num-of-Stages ,(truncate ?num-of-stages)))
     (dotimes (i (- (truncate ?num-of-stages) 2)) ;; 2 because the reboiler and condenser are already there
        (setq Cur-Stage (gentemp "stage"))
        (assert-in-scenario `(not (Feed-Stage ,cur-stage))))
        (assert-in-scenario `(Stage ,Cur-Stage))
        (assert-in-scenario
         `(Contained-Binary-Liquid-Mixture (2-C-S (,?substance1 ,?substance2) liquid ,Cur-Stage)))
        (assert-in-scenario
         `(greater-than (A (Amount-of (2-C-S (,?substance1 ,?substance2) liquid ,Cur-Stage))) ZERO))
        (assert-in-scenario
         `(Contained-Binary-Gas-Mixture (2-C-S (,?substance1 ,?substance2) gas ,Cur-Stage)))
        (assert-in-scenario
         `(greater-than (A (Amount-of (2-C-S (,?substance1 ,?substance2) gas ,Cur-Stage))) ZERO))
        (assert-in-scenario `(Gas-Path ,Cur-Stage ,?condenser-stage))
        (assert-in-scenario `(Liquid-Path ,?condenser-stage ,Cur-Stage))
        (assert-in-scenario
         `(Consider (Gas-Flow-Between (2-C-S (,?substance1 ,?substance2) gas ,Cur-Stage)
                                      (2-C-S (,?substance1 ,?substance2) gas ,?condenser-stage))))
        (assert-in-scenario
         `(Consider (Liquid-Flow-Between (2-C-S (,?substance1 ,?substance2) liquid ,?condenser-stage)
                                         (2-C-S (,?substance1 ,?substance2) liquid ,Cur-Stage))))
     (assert-in-scenario `(Gas-Path ,?reboiler-stage ,Cur-Stage))
     (assert-in-scenario `(Liquid-Path ,Cur-Stage ,?reboiler-stage))
     (assert-in-scenario
      `(Consider (Liquid-Flow-Between (2-C-S (,?substance1 ,?substance2) liquid ,Cur-Stage)
                                      (2-C-S (,?substance1 ,?substance2) liquid ,?reboiler-stage))))
     (assert-in-scenario
      `(Consider (Gas-Flow-Between (2-C-S (,?substance1 ,?substance2) gas ,?reboiler-stage)
                                   (2-C-S (,?substance1 ,?substance2) gas ,Cur-Stage))))
     (delete-assertion (find-fact-in-list 'Distillation-Column-Stages *scenario*))
     (assert-in-scenario
      `(Distillation-Column-Stages ,(second (find-fact-in-list 'Distillation-Column *scenario*))
                                   ,stage-list)))))
```

IF the steady-state design features for the column are active
 AND the column has a partial condenser and a partial reboiler
 AND the particular stages for these units have been defined
 AND the value for the number of stages in the column is *?num-of-stages*
 AND the current design description does not correspond to a column with *?num-of-stages* stages
THEN create a new design description for a column with a partial condenser, a partial reboiler and *?num-of-stages* stages.

Figure C.5: Update the number of stages in a binary column.

```
(ConfigSynthRule Initialize-Separation-System
 ((Separation-System ?system)
  (Consider (Sharp-Separation-Model-For ?system))
  (Separation-System-Feed (M-C-S ?components ?phase ?feed-stage) ?system)
  (Value-of (A (Feed-Pressure (M-C-S ?components ?phase ?feed-stage) ?system)) ?pressure ?eqn)
  (Value-of (A (Feed-Temperature (M-C-S ?components ?phase ?feed-stage) ?system))
           ?temperature ?eqn-1))
 (unless (exists '(First-Column ,?system ?column))
    (let ((new-column (gentemp "COLUMN")))
      (assert-in-scenario '(Distillation-Column ,new-column))
      (assert-in-scenario '(Consider (Sharp-Separation-Model-For ,new-column)))
      (assert-in-scenario '(Column-Feed (M-C-S ,?components ,?phase ,?feed-stage) ,new-column))
      (assert-in-scenario
       '(Consider (Liquid-Phase-Feed (M-C-S ,?components ,?phase ,?feed-stage) ,new-column)))
      (assert-in-scenario '(First-Column ,?system ,new-column))
      (assert-in-scenario
       '(Value-of (A (Feed-Pressure (M-C-S ,?components ,?phase ,?feed-stage) ,new-column))
                 ,?pressure :CS-Rule))
      (assert-in-scenario
       '(Value-of (A (Feed-Temperature (M-C-S ,?components ,?phase ,?feed-stage) ,new-column))
                 ,?temperature :CS-Rule))
      (assert '(First-Column ,?system ,new-column))
      (assign-in-problem
       '(A (Feed-Pressure (M-C-S ,?components ,?phase ,?feed-stage) ,new-column)) ?pressure)
      (assign-in-problem
       '(A (Feed-Temperature (M-C-S ,?components ,?phase ,?feed-stage) ,new-column)) ?temperature)
      (dolist (examine (exists '(Examine ?x))) (delete-assertion (car examine)))
      (assert-in-scenario '(Examine ,new-column))
      (assume-in-scenario '(Focus ,new-column)))))
```

IF there exists a separation system *?system*
  AND we assume sharp separators for *?system*
  AND we know the feed mixture for *?system*
  AND we know the operating conditions for *?system*
THEN if there are no columns defined for *?system*
  then instantiate the description for its first column *new-column*.
  New-column will have the same feed and operating conditions as *?system*.
  We assume a liquid feed and a sharp separation approximation for *new-column*.
  Finally, the Examine predicate directs the qualitative analysis to focus on *new-column*
  and the Focuspredicate directs the heuristic analysis to evaluate
  alternatives for *new-column*.

```
(ConfigSynthRule Column-System-Desired-Products
 ((Desired-Products ?products ?system)
  (First-Column ?system ?column) :Test (null (exists '(Desired-Products ,?products ,?column))))
 (assert-in-scenario '(Desired-Products ,?products ,?column)))
```

IF we know the desired products for separation system *?system*
  AND the desired products for the first column *?column* in *?system* have not been determined
THEN the desired products for *?column* will be the same with the ones for *?system*.

Figure C.6: Create the description for a new column in a separation system.

```
(ConfigSynthRule Column-Products
 ((not (Consider (Design-Complete ?system ?tcost)))
  (First-Column ?system ?column)
  (Column-Feed (M-C-S ?components ?phase ?stage) ?column)
  (Value-of (A (Reference-Pressure ?system)) ?ref-pres ?pres-eqn)
  (Consider (Possible (Separation distillation (?l-k ?h-k) ?column)))
  :Test (null (exists '(Destroy (Connections-to ,?column)))))
 (unless (exists '(Column-Products ?x ?y ,?column))
    (let ((top-products nil)
          (bottom-products nil))
      (mapc #'(lambda (x)
                (when (and (less-volatile? ?l-k x ?ref-pres) (not (eql x ?h-k)))
                   (push x top-products)))
            ?components)
      (mapc #'(lambda (x)
                (when (and (less-volatile? x ?h-k ?ref-pres) (not (eql x ?l-k)))
                   (push x bottom-products)))
            ?components)
      (pushnew
       '(assertq (Column-Products ,top-products (,?l-k ,?h-k) ,?column)) *scenario* :test #'equal)
      (assert '(Column-Products ,top-products (,?l-k ,?h-k) ,?column))
      (assert '(Distillate-Products ,top-products ,?column))
      (pushnew '(assertq (Distillate-Products ,top-products ,?column)) *scenario* :test #'equal)
      (pushnew
       '(assertq (Column-Products ,bottom-products (,?l-k ,?h-k) ,?column)) *scenario* :test #'equal)
      (assert '(Column-Products ,bottom-products (,?l-k ,?h-k) ,?column))
      (assert '(Bottom-Products ,bottom-products ,?column))
      (pushnew '(assertq (Bottom-Products ,bottom-products ,?column)) *scenario* :Test #'equal))))
```

IF a solution to the design problem for separation system *?system* has not been found
AND and we know the feed and the reference pressure for the current column *?column*
AND the heuristics have decided on an ordinary distillation process for *?column*
AND *?column* is not scheduled for removal by the evolutionary heuristics
THEN assert in the design description the distillate and bottom products for *?column*.

Figure C.7: Include the products of the current column in the flowsheet for the ordinary distillation case. There is a similar rule for the extractive distillation case.

```
(ConfigSynthRule Describe-Product-Splits
 ((Distillate-Products ?d-products ?column)
  (Bottom-Products ?b-products ?column)
  (Desired-Products ?desired-prods ?column)
  :Test (and (intersection ?desired-prods ?d-products) (intersection ?desired-prods ?b-products))
  (First-Column ?prev-system ?column)
  (Value-of (A (Product-Specification ?desired-prods ?prev-system)) ?p-r ?eq-1)
  (Connects-to ?column ?system)
  (Separation-System-Feed (M-C-S ?components ?phase ?stage) ?system))
 (unless (or (exists '(Desired-Products ,(intersection ?desired-prods ?d-products) ,?system))
             (exists '(Desired-Products ,(intersection ?desired-prods ?b-products) ,?system)))
   (assert-in-scenario '(Blend ,?d-products ,?b-products ,?desired-prods ,?system))
   (assert '(Blend ,?d-products ,?b-products ,?desired-prods ,?system))
   (unless (= (length ?d-products) 1)
      (let ((desired-products (intersection ?desired-prods ?d-products))
            (ff nil))
        (when (subsetp desired-products ?components)
          (assert-in-scenario '(Desired-Products ,desired-products ,?system))
          (dolist (comp desired-products)
             (setq ff (value-of-param '(A (Feed-Flow ,comp ?x ,?column))))
             (assign-in-problem
              '(A (Feed-Flow ,comp (M-C-S ,?components ,?phase ,?stage) ,?system)) ff))
             (assign-in-problem '(A (Product-Specification ,desired-products ,?system)) ?p-r)
             (assert '(Desired-Products ,desired-products ,?system)))))
   (unless (= (length ?b-products) 1)
      (let ((desired-products (intersection ?desired-prods ?b-products))
            (ff nil))
        (when (subsetp desired-products ?components)
          (assert-in-scenario '(Desired-Products ,desired-products ,?system))
          (dolist (comp desired-products)
             (setq ff (value-of-param '(A (Feed-Flow ,comp ?x ,?column))))
             (assign-in-problem
              '(A (Feed-Flow ,comp (M-C-S ,?components ,?phase ,?stage) ,?system)) ff))
             (assign-in-problem '(A (Product-Specification ,desired-products ,?system)) ?p-r)
              (assert '(Desired-Products ,desired-products ,?system))))))
```

IF some of the desired products *(?desired-prods)* for *?column* have been split between
the distillate and the bottom products
AND *?column* is part of separation system *?prev-system*
AND the desired recovery for *?desired-prods* in *?prev-system* is *?p-r*
AND *?column* connects to separation system *?system* for which we know its feed
THEN assert in the design description that the split products should blend at some point
and include in the description for *?system* the product specifications and feed flows
for the components of the split products that we want to recover.

Figure C.8: Detect product splits.

```
(ConfigSynthRule Define-New-Separation-System-For-Distillate-Products
 ((Missing-Products ?desired-products ?feed (?l-k ?h-k) ?column)
  (Distillate-Products ?distillate-products ?column)
  :Test (subsetp ?desired-products ?distillate-products)
  (First-Column ?prev-system ?column)
  (Value-of (A (Reference-Pressure ?prev-system)) ?r-pres ?r-pres-eqn))
  (Value-of (A (Reference-Temperature ?prev-system)) ?r-temp ?r-temp-eqn))
 (unless (exists '(Distillate-Feed :From ,?column . ?x))
    (let ((?system (gentemp "SYSTEM")))
      (assert '(Distillate-Feed :From ,?column :To ,?system))
      (assert-in-scenario '(Separation-System ,?system))
      (assert-in-scenario '(Distillate-Feed :From ,?column :To ,?system))
      (assert-in-scenario '(Connects-to ,?column ,?system))
      (assert-in-scenario '(Consider (Sharp-Separation-Model-For ,?system)))
      (assert-in-scenario '(Value-of (A (Reference-Temperature ,?system)) ,?r-temp ,?r-temp-eqn))
      (assert-in-scenario '(Value-of (A (Reference-Pressure ,?system)) ,?r-pres ,?r-pres-eqn))
      (assert '(Connects-to ,?column ,?system))
      (assert '(Separation-System ,?system))
      (assert '(Distillate-Feed :From ,?column :To ,?system))
      (assert '(Consider (Sharp-Separation-Model-For ,?system)))
      (let ((prev-focus (exists '(Focus . ?x))))
        (when prev-focus (delete-assumption (caar prev-focus)))))))
```

IF *?desired-products* have not been recovered yet
  AND *?desired-products* are a subset of the distillate products for *?column*
  AND we know the reference conditions for the separation system for which *?column* is part of
THEN create the description for a new separation system that feeds from the distillate products for *?column*.

Figure C.9: Instantiate a new separation system to isolate the desired products that are included in the distillate for the current column. There is a similar rule for the bottom products of a column.

```
(ConfigSynthRule Determine-New-Products-For-Distillate-Feed
 ((Missing-Products ?desired-products ?feed (?l-k ?h-k) ?column)
  (Distillate-Feed :From ?column :To ?system)
  (Distillate-Products ?distillate-products ?column)
  :Test (subsetp ?desired-products ?distillate-products)
  (First-Column ?prev-system ?column)
  (Connects-to ?column ?system)
  (Value-of (A (Product-Specification ?desired-products ?prev-system)) ?r ?r-eqn))
 (unless (member '(assertq (Desired-Products ,?desired-products ,?system)) *scenario* :Test #'equal)
   (assert-in-scenario '(Desired-Products ,?desired-products ,?system))
   (assert '(Desired-Products ,?desired-products ,?system))
   (assign-in-problem '(A (Product-Specification ,?desired-products ,?system)) ?r)))

 IF ?desired-products have not been recovered yet
    AND separation system ?system feeds from the distillate products for ?column
    AND ?desired-products are a subset of the distillate products for ?column
    AND we know the desired recoveries for ?desired-products
 THEN indicate in the design description that ?desired-products are among the
        desired products for ?system and set their desired recoveries equal to those
        for separation system ?prev-system which contains ?column.
```

Figure C.10: Assert the desired products and their recoveries for a separation system that feeds from the distillate products of the current column. A similar rule is used for the bottom products.

```
(ConfigSynthRule Determine-New-Feed-Properties-For-Distillate-Feed
 ((Distillate-Feed :From ?column :To ?system)
  (Distillate-Products ?new-feed ?column)
  (Value-of (A (Feed-Pressure ?initial-feed ?column)) ?pressure ?f-pres-eqn)
  (Value-of (A (Feed-Temperature ?initial-feed ?column)) ?temp ?f-temp-eqn))
 (unless (exists '(Separation-System-Feed ?x ,?system))
    (let ((?feed-stage (gentemp "FEED-STAGE")))
       (assert '(Separation-System-Feed (M-C-S ,?new-feed liquid ,?feed-stage) ,?system))
       (assert '(Multicomponent-Mixture (M-C-S ,?new-feed liquid ,?feed-stage)))
       (assert-in-scenario '(Multicomponent-Mixture (M-C-S ,?new-feed liquid ,?feed-stage)))
       (assert-in-scenario '(Separation-System-Feed (M-C-S ,?new-feed liquid ,?feed-stage) ,?system))
       (assert
        '(Value-of
          (A (Feed-Pressure (M-C-S ,?new-feed liquid ,?feed-stage) ,?system)) ,?pressure :CSR))
       (assert
        '(Value-of (A (Feed-Temperature (M-C-S ,?new-feed liquid ,?feed-stage) ,?system))
                  ,?temp :CSR))
       (assign-in-problem
        '(A (Feed-Pressure (M-C-S ,?new-feed liquid ,?feed-stage) ,?system)) ?pressure)
       (assign-in-problem
        '(A (Feed-Temperature (M-C-S ,?new-feed liquid ,?feed-stage) ,?system)) ?temp))))
```

IF separation system *?system* feeds from the distillate products for *?column*
AND *?new-feed* are the distillate products for *?column*
AND we know the actual conditions for *?column*
THEN assert in the design description all the relevant properties for *?new-feed*.

Figure C.11: Assert the feed properties for a separation system that feeds from the distillate products of the current column. There is a similar rule for the bottom products of the current column.

```
(ConfigSynthRule Design-Complete?
 ((Separation-System ?system)
  :Test (null (exists `(Connects-To ?column ,?system)))
  (Separation-System ?recent-system)
  (First-Column ?recent-system ?recent-column)
  (Column-Products ?products ?keys ?recent-column)
  (Distillate-Products ?d-products ?recent-column)
  (Bottom-Products ?b-products ?recent-column)
  (Value-of (A (Installation-Cost ?column)) . ?z))
 (unless (or (member `(assume! '(Consider (Design-Complete ,?system ?x))) *scenario* :Test #'equal)
             (exists '(Consider (Exchange . ?x)))
             (exists `(Connects-to ,?recent-column ?x)))
    (let ((Desired-Products
             (remove-duplicates
              (mapcar #'(lambda (x) (sort (copy-list (second (car x))) #'< :key #'sxhash))
                      (exists '(Desired-Products . ?x))) :Test #'equal))
            (Distil-Products
             (mapcar #'(lambda (x) (sort (copy-list (second (car x))) #'< :key #'sxhash))
                     (exists '(Distillate-Products . ?x))))
            (Bottom-Products
             (mapcar #'(lambda (x) (sort (copy-list (second (car x))) #'< :key #'sxhash))
                     (exists '(Bottom-Products . ?x))))
            (Blends (mapcar #'(lambda (x) (sort (fourth (car x)) #'< :key #'sxhash))
                            (exists '(Blend . ?x)))))
        (when (every #'(lambda (x) (or (member x Distil-Products :Test #'equal)
                                       (member x Bottom-Products :Test #'equal)
                                       (member x Blends :Test #'equal)))
                     Desired-Products)
        (cover-specifications)
        (assume-in-scenario `(Consider (Design-Complete ,?system ,(compute-cost))))))))
```

IF *?system* is the separation system in the initial design specifications
   AND *?recent-system* is the current separation system
   AND *?column* is the current column
   AND the products and the cost for *?column* have been determined
THEN if all the desired products in the initial design specification have been recovered
   then indicate that a complete design for *?system* has been found.

Figure C.12: Determine whether we have a complete design.

```
(ConfigSynthRule Evolve-Structure-1
 ((Destroy (Connections-to ?column))
  (Connects-to ?column ?system)
  (First-Column ?system ?next-column)
  (Column-Feed ?feed ?next-column))
 (progn
     (setq *scenario*
          (delete-if #'(lambda (x) (or (occurs-in? ?system x)
                                       (occurs-in? ?next-column x)
                                       (occurs-in? ?feed x)))
                     *scenario*))
     (clear-eqn-table ?system)
     (clear-eqn-table ?next-column)
     (setq *assigned-parameters*
          (delete-if #'(lambda (x)
                        (or (occurs-in? ?system (Parameter-Form x))
                            (occurs-in? ?next-column (Parameter-Form x))))
                     *assigned-parameters*))
   (setq *problem*
        (delete-if #'(lambda (x)
                       (or (occurs-in? ?system x) (occurs-in? ?next-column x)))
                   *problem*))
   (assert `(Destroy (Connections-to ,?next-column)))))
```

IF the evolutionary heuristics have indicated that all the connections to *?column* must be removed
AND *?column* is connected to separation system *?system*
AND *?next-column* is part of *?system*
AND the feed for *?next-column* is *?feed*
THEN remove from the current design description all the statements and the equations for *?system*, *?next-column* and *?feed* and indicate that all the connections to *?next-column* should be removed.

```
(ConfigSynthRule Evolve-Structure-2
 ((Destroy (Connections-to ?column))
  :Test (null (exists `(Connects-to ,?column ?next-system))))
 (progn
  (let ((new-focus-env (create-environment *Focus-Assumptions*)))
    (dolist (quant sg::*quantities*)
      (unless (implied-by? `((Quantity ,quant) . :True) new-focus-env)
         (setq *problem*
               (delete-if #'(lambda (x) (equal (second x) `(A ,quant))) *problem*))
         (setq *assigned-parameters*
               (delete-if #'(lambda (x) (equal (Parameter-Form x) quant))
                          *assigned-parameters*)))))))
```

IF the evolutionary heuristics have indicated that all the connections to *?column* must be removed
AND *?column* is not connected to anything else
THEN update the ATMS focus environment to reflect the new flowsheet and remove all the quantities and their values that are not implied by the current focus.

Figure C.13: Evolve the current flowsheet.

```
(ConfigSynthRule Evolve-Seader-&-Westerberg-Scenario
 ((Consider (Evolutionary-Strategy-For ?init-system Seader-&-Westerberg))
  (Consider
   (Exchange (Consider (Possible (Separation ?method1 (?lk-1 ?hk-1) ?col1)))
    :With (Consider (Possible (Separation ?method2 (?lk-2 ?hk-2) ?col2)))))
  (First-Column ?system ?col1)
  (Column-Products ?d-prods ?keys ?col1) :Var ?f1
  (Distillate-Products ?d-prods ?col1) :Var ?f2
  (Column-Products ?b-prods ?keys ?col1) :Var ?f3
  (Bottom-Products ?b-prods ?col1) :Var ?f4)
 (progn (clear-eqn-table ?col1)
        (assert-in-scenario '(Examine ,?col1))
        (assert '(Destroy (Connections-to ,?col1)))
        (delete-assertion (car ?f1))
        (delete-assertion (car ?f2))
        (delete-assertion (car ?f3))
        (delete-assertion (car ?f4))
        (delete-assertion '(Consider (Possible (Separation ,?method1 (,?lk-1 ,?hk-1) ,?col1))))
        (assert-in-scenario '(Consider (Possible (Separation ,?method2 (,?lk-2 ,?hk-2) ,?col1))))
        (let ((prev-column (second (caar (exists '(Connects-to ?c ,?system))))))
          (update-design ?col1 ?method2 ?lk-2 ?hk-2 prev-column))))
```

IF the evolutionary heuristics in the Seader & Westerberg method have indicated that
the separation in *?col2* should take place in *?col1*
THEN update the design description to reflect the proposed change.

Figure C.14: Evolve the current flowsheet according to the Seader Westerberg strategy.

```
(ConfigSynthRule Establish-Separation-Task
 ((Examine ?column)
  (Consider (Possible (Separation ?method ?keys ?column))) :Var ?f1)
 ((establish-separation-method ?method ?keys ?column)))
```

IF *?column* is the current column
  AND a separation task has been chosen for *?column*
THEN update the design description to reflect this design decision

```
(ConfigSynthRule Evolve-Design
 ((Consider (Design-Complete ?system ?tcost)) :Var ?f0
  (Establish-New-Product-Set ?alt-product-set ?products ?recovery ?system)
  (First-Column ?system ?column)
  (Evolve-Design-From ?column))
 ((establish-new-product-set ?alt-product-set ?products ?recovery ?system)
  (evolve-design-from-scratch ?column ?f0)))
```

IF a design satisfying the problem specifications has been found
  AND the evolutionary heuristics have suggested to split one of the original product sets
  AND *?column* is the first column in the completed design
  AND the evolutionary heuristics have suggested to evolve the current design starting from *?column*
THEN establish a new product set for design evolution and start modfying the design starting from *?column*

```
(ConfigSynthRule Evolve-Design-2
 ((Evolve-Design-From ?column ?sep-task ?sep-system))
 ((evolve-design-from ?column ?sep-task ?sep-system)))
```

Evolve the current design starting from column *?column* in separation system *?sep-system* with current separation task *?sep-task*.

```
(ConfigSynthRule Schedule-Alternative
 ((Schedule-Alternative ?sep-task ?sep-system))
 ((schedule-alternative ?sep-task ?sep-system)))
```

Schedule alternative *?sep-task* in *?sep-system*.

Figure C.15: Implement design decisions and evolve the current design.

APPENDIX D

Creating Representations for Heuristics in OUZO

This section provides an example of the method we followed for developing the representations for the heuristic rules in OUZO.

Let us consider the following general heuristic for separation systems:

> 'Remove a mass separating agent from one of the products in another, subsequent separation process.'

Before we attempt to write the representation for this rule we have to understand what types of actions it supports. Some background on the properties of mass separating agents explains why this a *rejection* heuristic. In particular[1], a mass separating agent (MSA) is an extra component that is added to a mixture in order to facilitate the separation of a mixture by extractive distillation. This agent modifies the equilibrium relations between the vapor and the liquid phases in the mixture in a direction that favors the desired separation. However, an MSA is an extra component that is not mentioned as a desired product in the design specifications, therefore at some point in the design we have to isolate this component from the mixture. For this reason, we always choose a MSA that is much less volatile than the original species, so that the separation of the MSA from the rest of the mixture will always be an easy and cheap separation. This means that if we have a column that accepts as input a mixture containing an MSA, a separation scheme with the MSA as the heavy key will be better than any other separation scheme (so we can reject the rest of them), except for the case in which the input mixture contains more than one MSAs (the current column feeds from two or more extractive distillation columns). In the later case we have to use the rest of the heuristics to choose between the separation schemes that involve the MSAs. Rejection of design alternatives is possible in OUZO with the *reject* primitive, therefore we use this primitive in the action part of the rule.

In order to write the conditions of the rule we have to determine how the the feed to the current column relates to the products of the previous columns in the sequence. This relation (for the case of a distillate feed) is described by the configuration synthesis rules in Figures C.9, C.11 and C.6 according to which, if the products of a separation task do not correspond to any of the desired products in the design specifications, they are used as inputs to a subsequent column in the separation sequence. Therefore, in order to represent this rule in OUZO we have to find the terms that represent the input to a column, along with the existence of a mass separating agent in the products of a previous column in the sequence. The qualitative model provides two predicates for this. The *(Column-Feed ?mixture*

[1]See chapter 3.

```
(defHeuristic Reject-non-MSA-Removal-Splits
  :Class Remove-MSA-From-Products-in-Subsequent-Operation
  :Conditions ((Mass-Separating-Agent ?p-s ?prev-column)
               (Connects-to ?prev-column ?system)
               (First-Column ?system ?column)
               (Column-Feed (M-C-S ?components ?phase ?stage) ?column)
               :Test (member ?p-s ?components)
               (Possible (Separation ?method (?l-k ?h-k) ?column)) :Var ?f1
               :Test (not (polar-solvent ?h-k)))
  :Action ((reject ?f1)))

 IF a mass separating agent was used in some previous column
    AND the previous column is connected to a separation system
    AND the separation system is connected to the current column
    AND the current column takes a multicomponent mixture as feed
    AND the mass separating agent is one of the components of the feed
    AND a possible separation scheme is proposed for the current column
    AND the heavy key is not a polar solvent in this scheme
 THEN reject this separation scheme.
```

Figure D.1: Remove a MSA from one of the products in another, subsequent separation process.

?column) predicate represents the input mixture to a column, while the *(Mass-Separating-Agent ?msa ?column)* form introduced by the Distillation-Features-in-Actual-Conditions view (Figure 4.39) represents the fact that a mass separating agent belongs to the products for ?column.

Furthermore, the sequencing order of the various column in a separation system is represented using two predicates in the qualitative model. The *(Connects-to ?column ?separation-system)* predicate indicates that some of the products of ?column are used as inputs to the separation system ?system. The *(First-Column ?system ?column)* form suggests that ?column is the first column of the separation system ?system.

Finally, each possible separation task in a column is represented in the design knowledge component using the *(Possible (Separation ?separation-method (?light-key ?heavy-key) ?column))* predicate (see Figure 5.2). In order to make sure that we do not reject any separation scheme with a MSA as a heavy key we use the procedure *polar-solvent* which checks whether the heavy key is a polar solvent and therefore a mass separating agent.

The final representation for this rule in `OUZO` is shown in Figure D.1.

BIBLIOGRAPHY

[1] Aelion, V., Cagan J., Powers, G. J., Inducing Optimally Directed Innovative Designs from Chemical Engineering First Principles, *Computers & Chemical Engineering*, vol. 15, no. 9, 1991.

[2] Allen, J., Hendler, J., Tate, A., *Readings in Planning*, Morgan Kaufmann Inc., 1990.

[3] Avallone, E. A., Baumeister III, T., *Marks' Standard Handbook for Mechanical Engineers*, McGraw-Hill, Inc., New York, 1987.

[4] Birnbaum, L., Collins, G., Remindings and Engineering Design Themes: A Case Study in Indexing Vocabulary, Panel Discussion on "Indexing Vocabulary", *Proceedings of the DARPA Case-Based Reasoning Workshop*, Morgan Kaufmann Inc., 1989.

[5] Brown, D. C., Chandrasekaran, B., *Design Problem Solving: Knowledge, Structures and Control Strategies*, Morgan Kaufmann Inc., 1989.

[6] Catino, C. A., Grantham, S. D., Ungar, L. H., Automatic generation of qualitative models of chemical process units, *Computers chem. Engng.* 15, 583-599, 1991.

[7] Chandrasekaran B., Design Problem Solving: A Task Analysis, *AI Magazine*, vol. 11, no. 4, Winter 1990.

[8] Cheung, J. T-Y, Stephanopoulos, G., Representation of Process Trends - Parts I-II, *Computers & Chemical Engineering*, vol. 14, no. 4//5, 1990.

[9] de Kleer, J., An assumption-based truth maintenance system, *Artificial Intelligence*, 28, 1986.

[10] de Kleer, J., Brown, J. S., A qualitative physics based on confluences, in *Readings in Qualitative Reasoning about Physical Systems* edited by Weld, D. S. and de Kleer, J., Morgan Kaufmann Inc, 1990.

[11] Collins, J. and Forbus, K. D., Reasoning about Fluids via Molecular Collections, *Proceedings of the National Conference on Artificial Intelligence*, Seattle, July 1987.

[12] Collins, J. and Forbus, K. D., Building Qualitative Models of Thermodynamic Processes, Unpublished manuscript.

[13] Douglas, J. M., *Conceptual Design of Chemical Processes*, McGraw-Hill, Inc., 1988.

[14] Falkenhainer, B., *Learning from Physical Analogies: A Study in Analogy and the Explanation Process*, PhD Thesis, University of Illinois at Urbana-Champaign, Urbana, Illinois, December 1988.

[15] Falkenhainer, B. and Forbus, K. D., Compositional Modeling: Finding the Right Model for the Job, *Artificial Intelligence*, 51, 1991.

[16] Falkenhainer, B., Forbus, K. D., Gentner, D., The Structure-Mapping Engine: Algorithm and Examples, *Artificial Intelligence*, 41, 1989.

[17] Forbus, K. D., The qualitative process engine, in *Readings in Qualitative Reasoning about Physical Systems* edited by Weld, D. S. and de Kleer, J., Morgan Kaufmann Inc, 1990.

[18] Forbus, K. D., Qualitative Process Theory, in *Readings in Qualitative Reasoning about Physical Systems* edited by Weld, D. S. and de Kleer, J., Morgan Kaufmann Inc, 1990.

[19] Forbus, K. D., de Kleer, J., Focusing the ATMS, *AAAI-88*, August 1988.

[20] Forbus, K. D., de Kleer, J., *Building Problem Solvers*, The MIT Press, 1993.

[21] Forbus, K. D. and Falkenhainer, B., Self-Explanatory Simulations: Scaling up to large models. *AAAI-92*, July 1992.

[22] Foust, A. et al., *Principles of Unit Operations*, R. E. Krieger Co, Malabar Florida, 1990.

[23] Franke, D. W., Deriving and Using Descriptions of Purpose, *IEEE Expert*, April 1991.

[24] Franks, R., *Modeling and Simulation in Chemical Engineering*, John Wiley & Sons, New York, 1972.

[25] Gero, J. S., Design-Prototypes: A Knowledge Representation Schema for Design, *AI Magazine*, vol. 11, no. 4, Winter 1990.

[26] Goel, A., *Integration of case-based reasoning and model-based reasoning for adaptive design problem solving*, PhD Thesis, Dept. of Computer and Information Science, The Ohio State University, Columbus, Ohio, 1989.

[27] Grossmann, I. E., Westerberg, A. W., Biegler, L. T., Retrofit Design of Processes. Reklaitis G. V., Spriggs, H. D. (Eds), *Foundations of Computer-Aided Process Operations*, New York, NY: CACHE/Elsevier, 1987.

[28] Hammond, K. J., On Functionally Motivated Vocabularies: An Apologia, Panel Discussion on "Indexing Vocabulary" *Proceedings of the DARPA Case-Based Reasoning Workshop*, Morgan Kaufmann Inc., 1989.

[29] Hanna, A. S., Willenbrock, J. H., Sanvido, V. E., Knowledge Acquisition and Development for Formwork Selection System, *Journal of Construction Engineering and Management*, ASCE, Vol. 118, No. 1, March 1992.

[30] Hendry, J. E., Hughes, R. R., Generating Separation Process Flowsheets, *Chemical Engineering Progress*, vol. 68, no. 6, 1972.

[31] Henley, E. J., Seader, J. D., *Equilibrium-stage separation operations in chemical engineering*, Wiley, New York, 1981.

[32] Hinrichs, T. R., Kolodner, J. L., The Roles of Adaptation in Case-Based Design, *AAAI-91*, July, 1991.

[33] Huang, Y. W., Fan, L. T., An Adaptive Heuristic-Based System for Synthesis of Complex Separation Sequences, in *Artificial Intelligence in Process Engineering* edited by Mavrovouniotis, M. L., Academic Press Inc., 1990.

[34] Joskowicz, L., Williams, B. (eds), *Working notes of the AAAI Fall Symposium on Design from Physical Principles*, Cambridge, Mass, 1992.

[35] King, C. J., *Separation Processes*, McGraw-Hill, Inc., 1971.

[36] Luyben, W. L., *Process Modeling, Simulation, and Control for Chemical Engineers*, McGraw-Hill, Inc., 1989.

[37] Mahalec, V., *Procedures for the Initial Design of Chemical Processing Systems*, PhD Thesis, Univ. Houston, Texas, 1976.

[38] Mahalec, V., Motard, R. L., Evolutionary Search for an Optimal Limiting Process Flowsheet, *Computers and Chemical Engineering*, 1, 149, 1977.

[39] Mavrovouniotis, M. L. (ed), *Artificial Intelligence in Process Engineering*, Academic Press Inc., 1990.

[40] Mayer, A. K., Lu, S. C-Y., An AI-Based Approach for the Integration of Multiple Sources of Knowledge to Aid Engineering Design, *Journal of Mechanisms, Transmissions and Automation in Design*, Transactions of the ASME, Vol. 110, September 1988.

[41] Mitchell, T. M., Steinberg, L., Kedar-Cabelli, S., Kelly, V., Shulman, J. and Weinrich T., An Intelligent Aid for Circuit Redesign, *AAAI-83*, August 1983.

[42] Motard, R. L., Westerberg, A. W., Process Synthesis, *AIChE Advanced Seminar Lectures Notes*, New York, 1978.

[43] Nath R., Motard, R. L., Evolutionary Synthesis of Separation Processes, *AIChE Journal*, vol. 27, no. 4, July 1981.

[44] Navinchandra, D., Sykara, K. P., Narasimhan, S., Design Synthesis with Qualitative Influence Graphs: Steps Towards Multi-State Dynamical Devices, *Working notes of the AAAI Fall Symposium on Design from Physical Principles*, Cambridge, Mass, 1992.

[45] Nishida N., Stephanopoulos G., Westerberg, A. W., A Review of Process Synthesis, *AIChE Journal*, vol. 27, no. 3, May 1981.

[46] Olson, D. G., Erdman, A. G., Riley, D. R., A Systematic Procedure for Type Synthesis of Mechanisms with Literature Review, *Mechanism and Machine Theory*, Vol. 20, No. 4, 1985.

[47] Pahl, G., Beitz, W., *Engineering Design*, The Design Council, London, 1984.

[48] Pikulik A. and Diaz, H. E., Cost Estimating Major Process Equipment, *Chemical Engineering*, vol. 84, no. 21, 1977.

[49] Rathore, R. N. S., van Wormer, K. A., Powers, G. J., Synthesis of Distillation Systems with Energy Integration, *AIChE Journal*, vol. 20, 1974.

[50] Ressler, A., A Circuit Grammar for Operational Amplifier Design, AI-TR-807, MIT AI Lab, January 1984.

[51] Riesbeck, C. K., Schank, R. S., *Inside Case-Based Reasoning*, Hillsdale, NJ: Lawrence Erlbaum, 1989.

[52] Roylance, G., A Simple Model of Circuit Design, AI-TR-703, MIT AI Lab, May 1980.

[53] Rodrigo, F. R., Seader, J. D., Synthesis of Separation Sequences by Ordered Branch Search, *AIChE Journal*, vol. 21, no 5, 1975.

[54] Seader, J. D., Westerberg, A. W., A Combined Heuristic and Evolutionary Strategy for Synthesis of Simple Separation Sequences, *AIChE Journal*, vol. 23, no. 6, November 1977.

[55] Sgouros, N., M., Integrating Qualitative and Numerical Models in binary distillation design, *AAAI Fall Symposium on Design from Physical Principles*, Cambridge, MA. October 1992.

[56] Shavlik, J. W., Dietterich, T. G., *Readings in Machine Learning*, Morgan Kaufmann Inc., 1990.

[57] Siletti, C. A., Design of Protein Purification Processes by Heuristic Search, in *Artificial Intelligence in Process Engineering* edited by Mavrovouniotis, M. L., Academic Press Inc., 1990.

[58] Sriram, D., DESTINY: A Model for Integrated Structural Design, *Journal for AI and Engineering*, October 1986.

[59] Steinberg, L. I., Design as Refinement Plus Constraint Propagation: The VEXED Experience, *AAAI-87*, July 1987.

[60] Stephanopoulos G., Artificial Intelligence in Process Engineering - Current State and Future Trends, *Computers & Chemical Engineering*, Vol. 14, No. 11, 1990.

[61] Stephanopoulos, G., Johnston, J., Kriticos, T., Lakshamanan, R., Mavrovouniotis, M. and Siletti, G., DESIGN-KIT: An object-oriented environment for process engineering, *Computers & Chemical Engineering*, vol. 11, no. 6, 1987.

[62] Stephanopoulos, G., Henning, G., Leone, H., MODEL.LA. A modeling language for process engineering - Parts I-II, *Computers & Chemical Engineering*, vol. 14, no. 8, 1990.

[63] Stephanopoulos, G., Westerberg, A. W., Studies in Process Synthesis - II. Evolutionary Synthesis of Optimal Process Flowsheets, *Chemical Engineering Science*, 31, 195, 1976.

[64] Thompson, R. W., King, C. J., Systematic Synthesis of Separation Schemes, *AIChE Journal*, 18, 941, 1972.

[65] Ulrich, K. T., Computation and Pre-Parametric Design, AI-TR-1043, MIT AI Lab, Sept. 1988.

[66] *Webster's Ninth New Collegiate Dictionary*, Merriamm Webster Inc, 1984.

[67] Weld, D. S., Reasoning about model accuracy, *Artificial Intelligence*, vol. 56, no 2-3, August 1992.

[68] Weld, D. S., *Theories of Comparative Analysis*, MIT Press, 1990.

[69] Williams, B., *Invention from First Principles via Topologies of Interaction*, PhD Thesis, MIT, June 1989.

[70] Williams, B., Qualitative analysis of MOS circuits, *Artificial Intelligence*, vol. 24, December 1984.

[71] Williams, B. C., de Kleer, J., Qualitative reasoning about physical systems: a return to roots, *Artificial Intelligence*, vol. 51, no 1-3, October 1991.

[72] Winkle, M. V., *Distillation*, McGraw-Hill, Inc., 1967.

[73] Yang, B., Datseris, P., Datta, U., Kowalski, J., An Integrated System for Design of Mechanisms by an Expert System - Domes: Applications, *Journal of Mechanical Design*, Transactions of the ASME, 24, vol. 113, March 1991.