Knowledge Based Flowsheet Modelling For Chemical Process Design

Douglas Hutton

Thesis presented for Degree of Doctor of Philisophy
University of Edinburgh
1990

Declaration

I declare that this thesis was composed by myself and that it describes my own work except where specifically stated in the text. The work was carried out between October 1987 and October 1990 in the Department of Chemical Engineering at the University of Edinburgh under the supervision of Prof. J.W. Ponton.

Douglas Hutton

Acknowledgements

I would like to thank ICI for funding this project and providing the expertise in design and flowsheeting which I was lacking. In particular, the time and effort of Alistair Struthers in developing CLAP and all its little nasties was greatly appreciated. I should also like to thank Prof. Ponton for initiating the project and allowing me freedom to explore. Thanks to Neil Skilling and Murray Laing for proofreading this thesis and to René Bañares Alcántara for some enlightening discussions.

Abstract

The aim of this work was to develop an experimental tool to perform flowsheeting tasks throughout the course of chemical process design. Such design proceeds in a hierarchical manner increasing the amount of detail in the description of the plant, and, correspondingly, in the mathematical models used to describe the plant. The models range from the simplest overall mass balance to rigorous unit models, and the calculations required in the course of a design may include the modelling of the complete plant or any of its constituent parts at any level of detail between these two extremes. Object oriented programming has been used to represent the hierarchy of units required throughout a hierarchical design.

A flexible modelling tool requires that models compatible with both the designer's intention and the context of the design are created. Sets of equations are defined in a generic form independent of process units with their selection as part of a model being dependent on the function and context of the unit being modelled. The expansion of the generic equation descriptions is achieved with reference to the structure of the unit, e.g. number of inlets and outlets, while the context of an equation determines the form of the equation to be applied, e.g. ideal or non-ideal behaviour. Equations are, therefore, represented as relations between a process item and its structural and contextual properties.

An increase in modelling flexibility is obtained by allowing the designer to interact with generated models. Different sets of equations can be selected within constraints imposed by the system. At a lower level, terms in individual equations can be modified for particular applications.

In chemical process design, many different analyses are performed. To demonstrate the application of different tools to a central model, the modelling system has been incorporated within a process synthesis framework.

The application of the system to simple design case studies is described.

Contents

Chapter 1

Introduction

The aim of this work has been to develop an experimental tool to perform flowsheeting tasks throughout the course of a design. Design proceeds in a hierarchical manner increasing the amount of detail in the description of the plant, and, correspondingly, in the mathematical models used to describe the plant. Models range from the simplest overall mass balance to rigorous, fully detailed unit models, and the calculations required in the course of a design include the modelling of complete plants or any of their constituent parts at any level of detail between these two extremes.

As a design develops, many alternative flowsheets may be generated reflecting the opportunistic way in which a designer works. The tool has been developed to support the numerical and heuristic evaluation of the alternatives, with the aim of producing, ultimately, a single design.

Most current flowsheeting technology is intended for modelling plants at a single level of complexity. In situations where several levels can be handled, there is no means of ensuring a consistency of data between them, or between the flowsheeting program and other programs used for design. This work aims to extend the applicability of flowsheet modelling as a tool to be used throughout the design process while considering the requirements of integrated process evaluation.

The justification for characterising the work in this thesis as knowledge based, is that, in comparison with more conventional programming techniques, the knowledge, both about the subject and the nature of the problem solving activity, has been coded explicitly in a modular manner. Inference is then possible based on the classification of the encoded knowledge.

The major area of work has been in knowledge representation. The representation of information must be able to describe the structure and function of plant items and the relationships between them. Relationships define topology, the position of a unit in a design hierarchy, its functional description and its modelling capability. Section 1.1 and Chapter 2 discuss the knowledge appropriate to flowsheeting and its representation. The application of the representation techniques to flowsheeting is discussed in Chapter 4.

The work described here has addressed the implementation of a solution method, together with its support facilities, appropriate for flexible flowsheet modelling. Chapter 3 discusses the advantages and disadvantages of the flowsheeting techniques currently in widespread use, providing the basis for the selection of the most suitable approach. The system could, however, be altered to accommodate a range of solution techniques as demonstrated in Chapter 3.

Integration with other design functions has been considered as part of knowledge representation. A tool for aiding in process synthesis has been developed to demonstrate the principle of wider integration. Process synthesis has a hierarchical structure providing an overall framework for design. The discrete levels suggested by the procedure allow a structured analysis of flowsheets, providing information for their subsequent modelling. The structure and its representation are discussed in Chapter 6.

1.1 Chemical Engineering Design

To provide a tool to facilitate process flowsheet modelling in the context of an integrated design, it is first necessary to define the scope of design and the level to which integration of different design functions is desirable and supportable.

Chemical engineering design is a multi-disciplinary exercise involving chemical engineers and chemists who define the process, mechanical and civil engineers who design process equipment and the site, and electrical and control engineers who provide power and control system design for the plant. Commercial pressures require that the design be completed in as short a time as possible. This can only be achieved by successful coordination of all these functions.

The different functions may view the same information in different ways, using terminology and notation unfamiliar outside their own fields. The coordination of information which is to be used from different viewpoints is itself a major research topic and has been given only brief consideration here. A discussion of the implications for a multi-disciplinary design environment has been presented by Westerberg et al [1] and Subrahmanian et al [2].

Design has traditionally been performed in two distinct phases [3] - conceptual design of the process and physical design of the items of plant. Initially, the process is defined by the chemists and chemical engineers. Specialists from other functions only become involved once the process definition is completed and their involvement is economically justifiable. This implies that once the design has reached the stage of involving the specialist functions, the process is fixed. Any fundamental alterations required as a result of consideration of the different functions are added to the design rather than integrated into the development of the process.

Process design is necessarily performed in an iterative manner. A number of designs will be considered initially and carried forward in enough detail to enable evaluation and, thereby, a choice to be made between them. The number of alternatives will be reduced as the degree of detail is increased, until a single process design may be chosen for progression to the plant and equipment design stage.

The present approach to design is to advance progressively from block flow diagram level (BFD) to the description of the individual unit operations. At this stage, a single design is accepted on economic grounds. Subsequently, a piping and instrumentation diagram (PID) is constructed, where control and layout are considered.

This is followed by a safety and hazard analysis where each line in the plant is investigated to see if variations from the specified conditions can occur and what, if any, hazards will result. Similarly, procedures for plant start-up and shut-down are determined.

Any of these evaluations could result in a major redesign of a plant which has been accepted without detailed consideration of the different functions. For example, a hazard analysis might reveal a failure resulting in an unanticipated explosive mixture of components. The solution to the problem at this stage of the design is to implement a control strategy ensuring that in the event of the identified failure, the explosive mixture is not formed. This may involve the specification of extra items of process equipment. If this eventuality had been considered earlier in the design, it may have been possible to define the process such that a failure of this sort would not occur.

An integrated approach to design would prevent such costly errors by encouraging the consideration of all facets of design throughout the hierarchical synthesis procedure (Figure 1.1), i.e. the iterative procedure would involve all disciplines rather than leave them until the process is fixed. By evaluating the process with respect to all these functions at every stage of design, redesign would be minimised. Advantages to be seen in completed processes should be: decreased physical size, greater energy efficiency, and improvements in safety and maintenance (see Preston [3]).

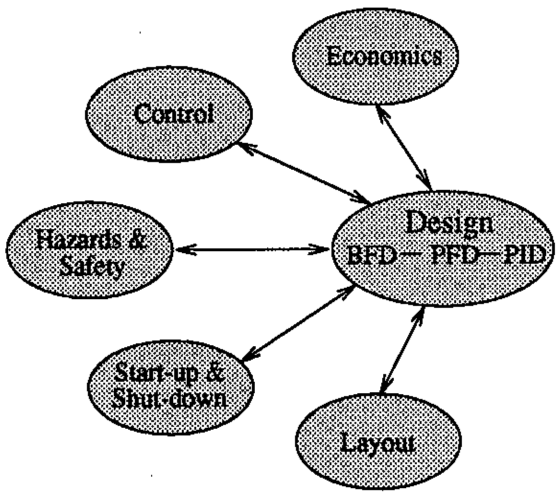


Figure 1.1: Integrated Approach to Design

1.1.1 The Role of Flowsheeting in Design

The synthesis of process flowsheets and the evaluation of their mathematical representations constitutes only a small part of chemical engineering design. "Flowsheeting" programs are used for evaluating flowsheet models in order to calculate heat and mass balances and equipment sizes. To establish the extent to which flowsheeting can be applied to a broader range of problems than attempted at present, a description of the design procedure and the role currently filled by flowsheeting is necessary.

The procedure is shown in Figure 1.2. Design is initiated by the need for a financially attractive product. Market models assess current product values and predict future trends resulting in the suggestion of a product or product area.

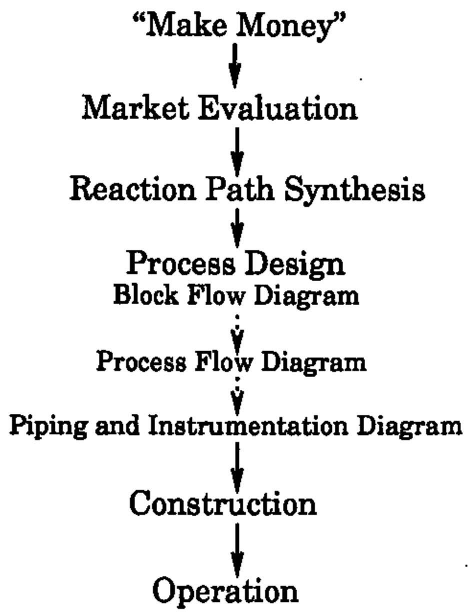


Figure 1.2: Procedure for Chemical Plant Design

If the product can be made using existing technology, then that technology is adapted to suit a particular site and required throughput. If, however, the technology does not exist, a chemical reaction sequence is required. Chemists provide information about the reactions involved, including kinetic equations, by-product and product distributions, etc.

There is already a range of possible processes to be evaluated. From this point on, increasingly more alternative processes can be generated. The alternatives must be evaluated to eliminate the less attractive options, resulting in a single completed design. It is infeasible, however, to produce completed designs for every alternative generated, so evaluation takes place at different levels of detail. Even at the level of minimum input, e.g. only basic reaction information, some options can be discarded, if, for example, the raw materials are more expensive than the products, or if a catalyst is rapidly poisoned and expensive to regenerate.

Throughout the process design phase more detail is added to the process description, starting with a block flow diagram of conceptual process operations and proceeding towards a process flow diagram where the unit operations of the completed plant have been defined, as shown in Figure 1.3. In parallel with this increase in process detail, more information is available for (and required of) the evaluation model.

The level of description adopted by flowsheeting programs is that of the pro-

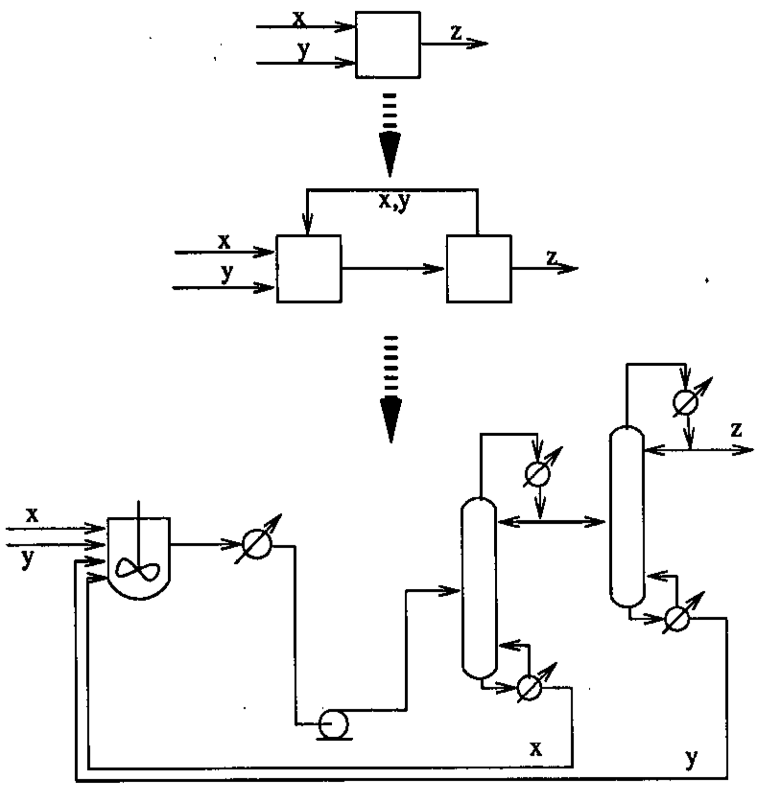


Figure 1.3: Hierarchical Approach to Design

cess flow diagram. This seemingly narrow range still accounts for thousands of program runs every year. It is, however, desirable to retain the knowledge accumulated thus far in the synthesis of the plant and extend it to incorporate the mechanical design of plant items, ensuring a consistency of information and ease of subsequent data manipulation. To do this in a traditional flowsheeting package would require an impractically large unit model library incorporating conceptual process models and detailed individual unit models for every possible unit operation. A different approach is necessary and is discussed in Chapter 4.

The concept of a hierarchical decomposition of a design is now well accepted, and should be incorporated in any representation of the design procedure. It allows the evaluation of process alternatives with the least possible effort being wasted on fruitless ones.

1.1.2 Design Knowledge

Modelling the design procedure, and encapsulating the knowledge identified as being used by experienced designers is a major research topic. Categorisation of the knowledge is necessary for the creation of a representative model which allows the designer to perform the required tasks in a flexible and natural way. The following types of knowledge used by engineers in design have been identified by Beltramini and Motard [4]:

1. Laboratory data describing chemical and physical behaviour of materials,
2. Data representing standards or specifications,
3. Mathematical models for mass balance, thermodynamic laws, chemical laws, costing, etc.
4. Heuristic and judgemental knowledge:
    a. subjective decision making exhibited by experienced engineers
    b. a way of combining information which might not conform to a mathematical model
    c. knowledge which directs the design towards an optimal solution
    d. skills in focusing on different parts of the design and deriving information necessary for decision making
    e. skills in resolving goal conflicts
    f. strategic knowledge for controlling the overall problem solving activity.

Some types of knowledge have already reached a high level of sophistication. For instance, numerical and procedural tasks, such as determination of physical and chemical property data for materials, and mathematical modelling for simulation are well established techniques. However, implementation of strategic and heuristic knowledge is comparatively limited.

In the encapsulation of strategic knowledge, design tasks can be broadly categorised into "routine" and "non-routine" activities [5]. Routine design encompasses problems which have a well defined design procedure that is essentially the same for any application. The steps in such procedures are known and any choices are limited to a known set of alternatives. Distillation column design and sequencing, heat exchanger network synthesis and the design of certain types of reactors fall into this category, because they require a small, manageable domain of knowledge.

Overall process design falls into the non-routine category because the required amount and diversity of engineering knowledge is extensive, and there is no well defined procedure for developing a problem specification into a complete plant design. Process synthesis, which is only part of the overall design activity, can be considered as a routine task since there are well developed theories for its practice. It is possible, therefore, for non-routine tasks to have routine operations within them.

The process of design can be viewed as having a hierarchical structure; a structure which has been utilised by many researchers in this field. A plant can initially be modelled by a simple block structure representing high level chemical processes (such as reaction and separation), and can subsequently be expanded incrementally to a more detailed description. Hierarchies have been implemented for:

1. directing the course of the design towards a solution, and
2. providing a structure for the different models which can be used throughout the design.

The hierarchical approach suggested by Douglas [6,7] fits into the first category, i.e. as a framework for directing the design. Douglas presents a hierarchy of decision levels which increases the detail of the process description in an evolutionary fashion starting with consideration of the reaction and feeds. This provides a hierarchical structure to the design. At each level, the whole flowsheet is assessed, increasing the amount of detail by the use of heuristics. In situations where no heuristics are available and, therefore, no discrete choice can be made, alternative processes are generated. The decision hierarchy consists of:

1 Batch vs continuous,

2 Input - output structure of the flowsheet,

3 Recycle structure of the flowsheet and reactor considerations,

4 Separation system specification,

  a. vapour recovery system,

  b. liquid recovery system,

5 Heat exchanger network.

The hierarchy puts a formal structure on the approach used by engineers in the preliminary stages of design.

A model of the design procedure itself has been proposed by Talukdar et al [8] who suggest a structure incorporating "tests", "aspects" and "operators". "Aspects" are the viewpoints of a design from initial specification to final production i.e. equivalent to levels in a hierarchy. "Operators" convert input aspects to intermediate and output aspects, i.e. perform the steps between levels which can be manual or automatic. "Tests" compare aspects for consistency and report the results in another aspect. This is similar to the procedure adopted by Douglas, but is intended for a wider range of application.

The second category above is discussed by Lien et al [9] who identify that different levels of model are used throughout the design, ranging from qualitative or order of magnitude models at the earliest stages, to rigorous mathematical models for later calculations. In the course of design, the designer moves from high to low level strategies and back as required. For instance, at one level in a design, a rapid, approximate calculation will determine the feasibility of distillation for the separation of two components. Once the decision has been taken that distillation may be used, a relatively simple model can be used to determine the dimensions of the distillation column and approximate capital and running costs. This provides the information necessary to evaluate the whole design on an economic basis. If that analysis is favourable, a detailed model may be used to determine mechanical information such as the dimensions of the column and its internal fittings. If, however, the design had proved unfavourable then another approximate calculation could be used to evaluate other possibilities. During this procedure, the design has only advanced one step, i.e. a separation may be carried out by distillation, but the detail of the models used to describe it is dependent on the task which the designer is trying to perform.

Lien et al describe this knowledge as models of the designers and of the available aids. The knowledge describes the tools available and their applicability to a given task. Ideally, the tools would be mathematical models of engineering principles, which, when combined in the correct manner, would provide a description of the design at an appropriate level of detail. However, many large organisations have extensive libraries of computer tools for use in the evaluation of design, ranging from material and heat balances for entire plants, to programs concerned with the detailed design of individual plant items. The utilisation of this existing design knowledge, in the form of programs and routines, is an alternative which is being investigated by some researchers (see Chapter 2).

The common theme of the research described above is the use of a central model to describe the design. The consistency of data, and the consistent use of data by different reasoning modules, requires a common store of information, including numerical values for properties and the relationships between the properties. An individual design task is then interacting with data which has been accumulated from other tasks, enabling the identification of contradictions within the data.

1.2 Computer Tools for Chemical Engineering Design

As discussed in Section 1.1, specialists from different functions are involved in coordinated, integrated design. The lack of integration, and hence the need for significant iteration, in process design is partly due to the range of specialisations and partly to the large amount of data generated.

Many specialist skills have now been encapsulated in computer programs, overcoming the difficulty of ensuring that these skills are available at the relevant time. Many tools have been developed in the CAD field as well as that of process design, and their coordination in large scale environments has been achieved (see Section 1.2.3 below).

Among the earliest tools developed in chemical engineering were "flowsheeting programs". The ability to perform accurate mass and heat balances rapidly on a scale which was previously impractical, was an important advance. The maturity of these programs has resulted in this becoming one of the most important tools available to a chemical plant designer. Programs for designing individual plant items, particularly heat exchange and distillation equipment, are also in widespread use. These programs have been complemented by databases of physical and chemical properties.

The development of "expert systems" has expanded the area of computer applications by considering problems which have no obvious numerical formulation, for example, equipment selection and materials of construction. Some problems have been approached by both mathematical methods and expert system techniques, each with their particular strengths. The result is a wide range of programs and tools applicable to different aspects of chemical engineering design.

With advances in computer hardware and software, the development of an integrated design environment has become an area of growing practical application. Importance is now being placed on coordination of the different aspects of design. Databases are now available to support the required management of data, but are not alone sufficient to capture "design knowledge". Developments in programming techniques, such as object oriented systems, have proved effective for managing both data and knowledge about design procedures in general, the design in hand and information about the range of tools which can be used.

1.2.1 The Role of Databases in Design

For many aspects of design large quantities of data are required. For example, physical properties of materials require tables of parameters for use by the different models of each property. Chemical data is required, along with information about hazards and legal limits for emissions. Further data is generated as a design proceeds. Every model created provides more data about the plant and its operation. Consistent management of the data is therefore essential for its use in large multi-user projects.

When considering integration of computer tools and the associated design data, the case for using a database to coordinate the information becomes very strong. The maintenance of a consistent store of data, whether it be fundamental or generated, is essential if a range of different programs and users is to interact effectively.

Commercial databases were developed for the tabulation of data containing few interactions and a large number of similar items. However, Cherry et al [10] describe process engineering data as being extremely complex where the number of similar items is small. The items also tend to be very strongly related. Chemical engineering design data is, therefore, not well suited to the use of commercial database management systems (DBMS). Databases specifically for chemical process engineering have been developed, which are different from their commercial counterparts. A review of engineering database systems has been published by Benayoune and Preece [11].

Early attempts at integration used databases to link individual programs. Branch [12] describes the implementation of an interlinked database architecture where a number of smaller databases, each relating to a particular application task, e.g. synthesis, plant layout, etc. were coordinated by a DBMS. Existing design programs were incorporated into the various tasks.

Britt et al [13] point out that such systems have not evolved as design tools, because they do not represent the creative, trial and error approach to conceptual design tasks performed by engineers. The systems lack the knowledge which an engineer uses to guide the design process. To this end, many researchers have concentrated effort on the development of "knowledge bases" and associated tools.

Databases are required for storing data generated during the course of a design, while knowledge bases store the strategies and assumptions employed by the engineer. The advent of knowledge bases, therefore, does not signify the demise of databases. Advances in the area of design will involve the combination of databases and knowledge bases.

The most common programming technique for the implementation of knowledge bases is object oriented programming (e.g. DESIGN-KIT (Stephanopoulos et al, [14]), KNOD (Beltramini and Motard, [4])), which has provided a format natural for use in engineering design due to its modularity. Object oriented programming (OOP) is described in detail in Chapter 2. Briefly, OOP is a language for representing groups of related information as "objects" and their "attributes". Objects can be used to correspond directly to the processing units in a plant. This representation is possible using a conventional database, defining each object as a table. However, the number and range of relationships employed would be very difficult to describe. For example, different properties of the unit can be related to each other by means of a mathematical model.

Recent advances in database technology have produced object-oriented databases, which provide a more powerful representation for relationships between stored items of information. However, databases cannot be used to describe all of the types of knowledge discussed above, much of which is methodological.

A hybrid approach is required with information being stored in a database (or databases) to maintain the general data, such as physical property parameters, as well as the data generated as the design proceeds including the various topologies investigated and results of the associated calculations. Knowledge bases should be used for the incorporation of the wide range of relationships which are required between the items of design information and for the methodological aspects of design. Relationships exist between different classes of data item, e.g. between process items and the programs which can be used for their design. In that way knowledge about the use of information and design procedures can be captured.

1.2.2 Process Synthesis

Many computer tools are available to help in the solution of routine design synthesis problems such as distillation sequences and heat integration networks [15]. Synthesis of process flowsheets can be considered to be a routine task, since the conversion of raw materials to products can be regarded as a mathematical transformation under a set of known operators i.e. unit operations.

Two general approaches exist for the automatic generation of flowsheets: algorithmic and heuristic. The algorithmic approach is mathematically rigorous and guarantees to locate an optimal process, but is expensive in computing time. The heuristic approach, however, does not guarantee a rigorous mathematical result, but tends to produce good flowsheets quickly. The heuristic approach, which uses rules of thumb based on past experience, ingenuity and the intuition of the designer, therefore more closely emulates the procedures adopted by an engineer.

An example of an algorithmic approach for deriving optimal process flowsheets is described by Johns and Romero [16] who combined dynamic programming and branch and bound techniques. A qualitative system for representing high and low properties was used because the program was intended to suggest alternatives at early stages of development.

Heuristic synthesis, while maintaining the principle of mathematical transformation at least in early programs, developed the concept of hierarchical synthesis, which has since been incorporated in tools for supporting the design procedure. Of the heuristic approaches, the hierarchy of decision levels proposed by Douglas [6] has proved most resilient. The expert system, PIP [17], discussed below, is a direct implementation of the method, while other authors, notably Lott [18], Stephanopoulos (MODEL.LA)[19]and Britt [13], have developed software which uses Douglas's approach as a basis for synthesis.

Siirola and Rudd [20] propose six steps combining synthesis and analysis. Each stage of analysis determines the implications of the previous synthesis step and provides appropriate information for the next. The six steps are reaction path selection, selection of raw material and product amounts, matching of source species to sinks, identification of tasks not involving separation, identification of required separations and conversion of specified tasks into items of plant.

AIDES (Siirola and Rudd [20], Siirola et al [21]) is the implementation of the above problem solving strategy. The authors have implemented five out of six steps of the synthesis and analysis procedure described above.

The procedure involves matching sources and sinks of components. For example, product streams and reactor inlets are sinks, while feeds and reactor outlets are sources. Where sinks and sources do not match, AIDES proposes unit operations to perform the required task.

Mahalec and Motard [22,23] suggest the use of techniques employed for mathematical theorem proving. The procedure starts with a set of goals (products) and attempts to derive conflicts among a set of facts and the desired goals. Redundant portions of the initially proposed flowsheet, which are obtained by a depth-first heuristic search, are eliminated using a "look-back" strategy. The structure is further improved by an evolutionary search based on a set of rules.

Kirkwood et al [17] have developed a hierarchical system using the decision levels proposed by Douglas. Heuristics are employed to select unit operations and identify process alternatives. Quantitative models calculate process flows, equipment sizes and cost information.

The program (PIP) produces alternative processes at points where it cannot make a decision. A depth-first search of the alternatives is used to locate profitable processes. If a process is profitable, more detail is added. If not, the sensitivity of the design to changes in product price is evaluated.

Lu and Motard [24] have combined heuristic production rules and a linear programming algorithm to generate a flowsheet structure. They adopt a hierarchical strategy, allowing the initial level of design to incorporate only process concepts, i.e. matrices representing goals and sources. The linear programming technique produces a preliminary flowsheet, which provides a basis for an evolutionary search to generate modifications to the flowsheet structure.

Since overall flowsheet synthesis can be regarded as a routine task, according to the definition of Davis [25], it can be considered as one of the standard tools used in design. In an integrated design system, the other tasks, such as safety evaluation and control system synthesis, would require separate tools. An integrated system requires interaction between all tasks at all levels of design, suggesting a hierarchical approach. The non-hierarchical programs can still provide useful process alternatives for evaluation, but are of less value in an integrated system. For that reason the larger design programs and environments which have been developed have supported a hierarchical synthesis procedure.

1.2.3 Design Environments

"Design environments" provide an interface between designer and tools, maintaining data in a consistent form. Environments are not intended to automate design of chemical plants. They are intended to provide a designer with access to the tools required throughout the integrated design process.

In the most basic sense, the coordination of tools can be achieved by providing a user interface to a suite of translation programs. An example of this type of mechanism is PROCEDE [26]. The interface allows the graphical description of a flowsheet to a high degree of sophistication. The designer is then at liberty to invoke any of the design tools in any order. Such flexibility is important, allowing the adoption of familiar design procedures rather than being constrained to a predefined framework.

The limitations of systems such as PROCEDE is the lack of information about the constituent tools. PROCEDE has a central data store, but no description of the relationships between process items or properties of process items. With no indication of the validity of the values of properties, or whether they are specified or calculated, tools can be invoked with inputs which may be contradictory or incomplete. The results of the operations are returned to the central store with no indication of where they came from and no means of ensuring consistency. For example, a flowsheet simulation may calculate a heat exchanger heat transfer area which is subsequently used in a package specifically to design heat exchangers. The package may select a tube length from a set of standard lengths which may require a revision of the heat transfer area. With no knowledge of the interaction between heat transfer and heat load, no indication can be given that the results of the simulation may now be inaccurate.

A framework incorporating knowledge about the tools, i.e. what can be achieved and what is required, has been developed by Daniell and Director [27] for the application of CAD tools. The tools have been represented by "objects" describing their general abilities and manage the low level programming details of their invocation. Not only does this representation allow the user the flexibility of choosing any tool with reference to its abilities, but, since this information is explicitly stated, it is possible to provide reasoned advice on tool selection.

The authors describe the implementation of a "blackboard model" to select and invoke the tool most appropriate to a given problem. A blackboard model is a representation technique where the knowledge is divided into sets of rules each concerned with a particular subject. These "Knowledge Sources" monitor the "Blackboard" of data about the problem, waiting for information which they can act on. When such a situation is recognised, a Knowledge Source places a bid on the Blackboard indicating that it can be used and what it can achieve. The program "Interpreter" then determines which of the competing Knowledge Sources to invoke. This representation is discussed in Chapter 2.

Allowing the Interpreter to determine the sequence of actions takes the direction of the design away from the engineer. To allow the designer to direct the procedure the same framework could be used merely to present the tools which could be invoked in a given situation. This would be of particular value if a large number of tools was available and a particular analysis required only a subset of these. Presenting the user with the subset of options could provide guidance for completing the analysis.

The ADVENT system, described by Britt et al [13], adopts a similar philosophy, and is intended to support process synthesis and optimisation for "pinch" synthesis of heat exchange networks. An object oriented executive allows the user to interact with application programs, in this case synthesis and analysis programs for heat exchange networks, as well as a simulator, and a graphical interface. The authors suggest the extension of the system to handle whole plants using the synthesis hierarchy of Douglas and a database.

One of the major objectives in the design of the above systems has been the ease of maintenance, i.e. the ease of adding new modules and removing old ones. Systems such as PROCEDE have little interaction between tools, implying that the addition of new tools only requires the correct mapping between the data structures of the central model in the environment and the application program. The system proposed by Daniell and Director, however, requires additional information describing the range of applications with associated input and output values. The work to incorporate new tools has, therefore, increased. In general, the more information that the environment has about its constituent tools, the more useful it can be to a designer. With the implementation of appropriate reasoning it can provide help about the tasks involved and tools for their solution. However, the increase in additional information also increases the effort necessary to interface new tools.

The use of existing application programs alone is not sufficient to provide all the flexibility the designer may require. Most programs only perform one task representing a single step in the design procedure, e.g. distillation column sequencing, flowsheet simulation, heat exchange network design, etc. The models which can be created in such systems are restricted to those provided by the modules, whereas a designer creates models of processes at different levels of abstraction and evaluates them against various criteria. Further, the user does not have access to the models used by the application programs. This means that the designer may not be able to create exactly the model required, or manipulate it to suit particular needs. In most cases, there will not be any way of displaying the model being used.

To provide a better environment for chemical engineering design, it becomes necessary to support flexible modelling and allow access to the models created. If the knowledge about the problem domain is adequately detailed, in that the properties of the items of interest to a designer are characterised along with their relationships to each other, then the definition of a mathematical model requires only the ability to define the sets of relationships between the properties which provide a description of the item. The models are, therefore, created by the designer for a particular application, but are available for evaluation by simulators or reasoning modules.

An environment based on this principle is DESIGN-KIT [14] developed using the artificial intelligence toolkit, KEE. The system is object oriented with plant items represented by objects within an inheritance hierarchy. The designer can select from a number of standard units or, more importantly, define the operation of a unit by selecting appropriate constituent parts. Rules are used to ensure the consistency of such selections. Further rules infer the configuration of the unit, and, hence, the associated mathematical model for the desired task. The mathematical model is, therefore, representative of the operation defined by the designer.

The system dynamically generates equations from the model definition and interfaces to a range of evaluation modules. These modules can perform, for example, a "degrees of freedom" analysis, to aid the selection of design variables, and symbolic differentiation prior to equation based simulation. Since the model is defined centrally, the same definition can be used to develop a semi-qualitative model based on the order of magnitude of unit attributes. Potentially, other types of model could be generated for other applications.

The evaluation modules for such a system must be general in order to accommodate the extra interaction designers might require with models. For instance, in DESIGN-KIT, instead of interfacing to a specific process flowsheeting program, the only part which is incorporated in its original form is the solution mechanism. Model libraries have been superseded by the mechanism for model definition. The flowsheeting "executive" has been replaced by rules which provide a more flexible interface to the solver. The removal of these rigid parts of the flowsheeting program allows interaction with the model, to evaluate, for example, design variable selection. The implication is that for systems based on this general modelling concept, there is a need to develop more tools specifically for the environment. Tools are required for formulating models for each application in order to allow interaction with the models at different levels. Either the application modules must be written specifically for the environment, or the fundamental evaluation parts of existing programs must be separated from their internal model representations, e.g. the solver from the simulator.

The result is an environment with a high degree of flexibility for the designer, who can define models rather than let the application programs develop them,. However, the disadvantage for the system developer is the inflexibility of the interfacing. The design tasks normally performed by application programs require either new custom made modules, or major alterations to the application programs.

The emphasis of this type of environment is on a central model definition which can be used to formulate models for a range of different types of evaluation.

Therefore, interfacing to tools in the manner described by Britt et al, is still an option. It is valuable to retain this option as a means of interfacing existing packages which a designer has experience of, or are company standard. The system can then support the design programs which are available at the point of application.

The model not supported by PROCEDE or DESIGN-KIT is a model of the design process itself. This can be used to provide a recorded structure of design development and associated decisions, and, from the programmer's point of view, to incorporate strategies for directing the design task. MODEL.LA [19,28,29] consists of a modelling language integrated with a model which supports design development (see Section 1.2.4) to provide this required extension to DESIGN-KIT.

More recently proposed design environments address more fundamentally the nature of design. Smithers [30] argues that to provide intelligent support or automatically reproduce the design process, it is necessary to understand how knowledge is organised, used and generated during design. He describes the development of an "exploration based" model of design. The model takes as its input a description of the initial requirement, which tends to be incomplete and inconsistent. The space of possible designs is "explored" to determine in what ways the initial statement is incomplete and inconsistent. These points are used to concentrate the design activity and thereby refine the initial description. This differs from a search problem in that the initial requirement cannot be regarded as a specification due to its inconsistency. A goal state is achieved when a point in the design space is found which specifies a design fully satisfying the, now revised, requirement. This approach is also taken by Bañares-Alcántara [31] in a proposed chemical engineering design environment.

Study of the nature of the design process has revealed the importance of social aspects. Design is typically performed by a team rather than one person. Individuals may perform separate tasks, but many are collaborative. Westerberg et al [1] and Subrahmanian et al [2] discuss the development of a design support environment called N-Dim intended to provide a medium for a multi-disciplinary team of designers to communicate on the common design.

In this system, individual designers may have different viewpoints pertaining to their specialities, and should, therefore, be able to access their own models.

Obviously, for consistency, only one model should be used. The authors suggest that, to overcome this apparent contradiction, individual designers should be able to access a common model but from their own viewpoint. Designers will be allowed to work on models in their own space, developing and experimenting. When a designer is ready to share a model with other workers, that model becomes permanent. The other workers may copy and modify the model in their own space, but the "cast in stone" model remains as a record of that state of the design.

The authors also discuss the development of a modelling language, ASCEND, which is tailored to suit large scale design activities (see Section 1.2.4) by incorporating the notion that data can be accessed with a different viewpoint. The language is intended for the formulation and solution of algebraic models, but Talukdar and Westerberg [8] discuss its implementation in the wider context of multi-user design. Three aspects of design are considered:

- The information gathering phase prior to the development of a new product.
- The modelling of the design procedure. The example discussed is the representation of test-aspect-operator diagrams.
- The modelling of the designed artifact in its various stages of development, which is the original purpose of ASCEND.

The discussion of the first two applications is mostly hypothetical, and would require the support of an environment such as N-Dim for multi-user access. The principle, however, is straightforward. Individuals can develop their own models of the artifact, the design or the information, and permit their use to others, at which point the models become "cast in stone". This allows different designers to create models from their own viewpoint, but accessing a common model.

As yet, these later systems (Smithers, Bañares-Alcántara, Westerberg et al) are in the development stage, but they already indicate that future design environments are going to support the designer in more than the use of a suite of tools. Modelling of the design procedure will be valuable, not necessarily in automatic design, but for support for the tasks and social interactions involved.

1.2.4 Process Modelling

The importance of modelling as a tool for design has been emphasised by Stephanopoulos et al [19] and Westerberg et al [1]. During the course of a design, engineers create models which reflect the state of the designed article. Traditionally, programs modelling flowsheets or individual plant items have predefined models which constrain the exploratory nature of model building. For this reason, systems have been developed to support the modelling task, notably: MODEL.LA [28,29], ASCEND [32] and ModAss [33]. A detailed comparison of the systems and the present work will be made in Section 5.3. This section outlines briefly the capabilities of each system.

The central theme of these systems is the hierarchical decomposition or construction of models. The mathematical description of large processes can be defined as a set of submodels describing parts of the process. The decomposition can continue down to the definition of a single variable. For example, the description of a flash vessel can be defined as the summation of the heat contributions of input and output streams, which can be further decomposed into enthalpies, then to temperatures, heat capacities, etc.

These tools are most useful when as many fundamental relationships as possible have been defined. The designer can then create high level models by combining lower level models, e.g. the flash vessel described above can be modelled by specifying that it requires mass balance, heat balance and vapour-liquid equilibrium models.

The techniques for describing and developing models are different for the three systems discussed.

ASCEND was developed for formulation and solution of algebraic models. The resulting language has some object oriented properties rather than it having been developed in an object oriented language (see Chapter 2). The structure of a model is similar to that of the concept of an object. Related information, including the specification of mathematical relationships between constituent properties, is classified under a general heading, e.g. a distillation column. Any examples of distillation columns then have the properties of the general model. The different parts of a model are discussed below. Figure 1.4 shows part of the definition of a distillation column in ASCEND.

The variables `nt` and `feed_tray` are declared as integers. If subsequent spec-

```
MODEL column;
   nt, feed_tray                IS_A integer;
   tray[integer]                IS_A generic_flash;
   tray[1]                      IS_REFINED_TO reboiler;
   tray[2..feed_tray-1]         IS_REFINED_TO stage;
   tray[feed_tray]              IS_REFINED_TO feed_stage;
   tray[feed_tray+1..nt-1]      IS_REFINED_TO stage;
   tray[nt]                     IS_REFINED_TO condenser;
   :
   FOR i: 1..nt-2
   CREATE
        tray[i+1].lout, tray[i].lin    ARE_THE_SAME;
        tray[i].vout, tray[i+1].vin    ARE_THE_SAME;
   END
   :
   FOR i: tray[feed_tray].feed.comp_name[1..tray[feed_tray].feed.nc]
   CREATE
     recovery[i] * tray[feed_tray].feed.F * tray[feed_tray].feed.y[i]
       = tray[1].lout.F * tray[1].lout.y[i]
   END
   :
END column
```

Figure 1.4: Partial View of a Distillation Column Model written in ASCEND

ification of the values is with a non-integer, the error will be identified. The specification of the trays shows how lower level models can be incorporated in a higher level one. The specification is made that all trays (`tray[integer]`) are of the type "generic_flash", which has been defined as a model in its own right, relating input component flowrates to output flowrates with vapour-liquid equilibrium relationships. The following lines identify the specific types of model which are associated with each tray. For example, the model for a stage is a refinement of a generic flash (by the IS_REFINED_TO operator) with two input streams. Similarly the refinement to a feed stage has three input streams.

The properties demonstrated are akin to concepts of object oriented languages. For example, the IS_REFINED_TO operator is similar to the notion of inheritance, whereby the properties of a more general operation, for instance, the reboiler, are inherited by a specialisation, i.e. tray[1]. The IS_A operator corresponds to the instantiation of specific models. The example in Figure 1.4

shows all trays being defined as instances of the generic flash operation. Their particular refinements are subsequently detailed. An instance of the column can be created by stating that a specific column IS_A example of this generic column, and then providing values for `nt`, the number of trays, and `feed_tray`, the number of the feed plate.

In the description of the relationships between connecting plates, the liquid flows between plates must be equated, as do the vapour flows. This can be achieved, either by creating equations, or, as here, by specifying that the variables ARE_THE_SAME. The equality is represented by one piece of data, in this case representing a flowrate, which can be accessed by two names, i.e. the same data item is accessed by `tray[3].lout` and `tray[2].lin`. This facility is important in multi-user projects where different people can refer to the same data item by their own chosen names.

Each term is also a path name. For instance, the item in `tray[2].lin` is an item called `lin` in the model of `tray[2]`. This avoids the redeclaration of property types in higher level models, thus simulating the accessing of an object's slot in object oriented programming.

The specification of an equation is shown at the bottom of Figure 1.4. The equation describes the relationships between the feed composition and the product composition in terms of recoveries.

In comparison, MODEL.LA has been developed entirely in an object oriented language. It was developed specifically for chemical process engineering, intending that the models be described in terms of the physical and chemical phenomena involved. The user does not appear to have direct access to symbolic equations. In specifying a model, the modeller creates a generic description of the phenomena in the unit. The generic template can then be instantiated to a specific instance of the unit in a manner similar to ASCEND.

Unlike ASCEND, a hierarchy of fundamental, domain specific relationships has been defined, providing the basic structure of the mathematical models. The relationships describe the balances which may be accommodated , e.g. mass balance, energy balance, phase equilibria, etc. The modeller interacts with this hierarchy by characterising the model in two ways. A description of the relevant conservation equations is made along with assumptions about the physical and chemical phenomena (and in the case of lumped vs distributed, a modelling phenomenon). Figure 1.5 shows how a model of a flash might be defined in MODEL.LA. The description has been abridged, to show only the basic relationships.

```
((FLASH IS_A UNIT)
 :
 ((INPUT1-FLASH IS_A CONVECTIVE_PORT)ENDMODEL)
 :
 (THE TYPE-OF-MODEL OF FLASH IS LUMPED)
 (THE BALANCE-EQUATIONS OF FLASH IS (SET.OF(MASS-BALANCE-EQUATION
                                       ENERGY-BALANCE-EQUATION)))
 (THE PHASES OF FLASH IS (SET.OF(VAPOUR LIQUID-1))
 (THE PHASE-EQUILIBRIUM-CHARACTERISTICS OF FLASH IS
       PHASE-EQUILIBRIUM)
 (THE PHASE-EQUILIBRIUM-MODEL OF FLASH IS UNIFAC)
 (THE THERMAL-CHARACTERISTICS OF FLASH IS (SET.OF(
       HOMOGENEOUS-TEMPERATURE ADIABATIC)))
 (THE PRESSURE-CHARACTERISTICS OF FLASH IS (SET.OF(
       HOMOGENEOUS-PRESSURE)))
 :
 ENDMODEL)
```

Figure 1.5: A Flash Vessel Model described in MODEL.LA

A set of rules has been defined to "translate" the above description into a functional specification. For example, the combination of the statement that the model is lumped and the balance equations include a mass balance equation, implies the use of a lumped mass balance equation. The balance equations are defined in a completely general form, with terms for any of the rates which might be included in them. The other statements perform two functions.

- They identify additional equations which should be added to the model. For example, the specification of homogeneous temperature identifies the necessity of the equality between the vessel temperature and the outlet temperature.

- They identify terms in the balance equations which can be removed according to the approximation which the assumptions represent. For example, the statement in Figure 1.5 that the thermal characteristics include adiabatic operation, removes the external heat source term in the energy balance.

Once low level functional descriptions have been defined, they can be combined into high level operations. For example, emulating the distillation column model created in ASCEND in Figure 1.4, a similar description can be defined in MODEL.LA as shown in Figure 1.6.

```
((COLUMN IS_A UNIT)
   :
   input and output definitions
   :
  (THE COMPONENTS OF COLUMN IS REBOILER)
  (THE COMPONENTS OF COLUMN IS COLUMN-SECTION)
  (THE COMPONENTS OF COLUMN IS FEED PLATE)
  (THE COMPONENTS OF COLUMN IS COLUMN-SECTION)
  (THE COMPONENTS OF COLUMN IS CONDENSER)
   :
  (THE USER-DEFINED-RELATIONSHIP OF COLUMN IS
                         (-(* RECOVERY MOLAR-FLOWRATE-FEED-PLATE)
                                       MOLAR-FLOWRATE-REBOILER))
  ENDMODEL)
```

Figure 1.6: A Distillation Column Model described in MODEL.LA

In ASCEND, models describing the operations of condensers, reboilers and plates can be defined as specialisations of a flash with a set number of inputs. Conceptually, the same can be done in MODEL.LA by combining a mass balance with a heat balance and a vapour-liquid equilibrium, but including no specification of the expected number of inputs or outputs. The description possible in MODEL.LA does not allow for the identification of individual terms, such as the number of inputs. The description of the flash includes a mass balance, which, on instantiation of the model, creates a set of symbolic expressions based on the specified inputs and outputs. Therefore, the generic flash operation cannot be used separately as a model, but once included as part of a higher level model where input and output ports have been defined, the mathematical description is complete.

The equations have been arranged in a hierarchy indicating specialisation. The distillation column model defined in ASCEND (Figure 1.4) includes a relationship between the input composition and product composition. This is desirable in the description of the column, but the relationship is not included in the hierarchy described in MODEL.LA. Requiring the user to place a newly defined relationship in the hierarchy is unacceptable because of the problems of maintaining the tree and the rules for selection without the specialised knowledge of the program developer. For that reason, a relationship has been provided in the hierarchy for user-defined expressions. It is not clear from the literature how a user-defined relationship is included in a model or placed in the hierarchy. The line in Figure 1.6 defining the above relationship is, therefore, conjecture, but is possible in some form.

The column model defined in MODEL.LA is shorter than that in ASCEND because the specification of the connections and relationships regarding transfer between model components is done at the submodel level, i.e. the connection between the reboiler to the first column section and the nature of the mass transfer is defined within the separate model instances. Column sections have been defined as submodels since, apparently, there, is no method for specifying variable numbers of parts, thus the technique employed by ASCEND cannot be used. The alternative is to state explicitly the plates to be used between reboiler and feed, and between feed and condenser. This then, essentially, becomes an instance of a column, a new one being required for each different configuration. The same problem exists for the column sections and it is not clear from the literature how this is overcome.

The third system, ModAss [33], despite little published information, seems to lie somewhere between MODEL.LA and ASCEND. Certain aspects of a model are implied automatically, notably mass and energy balances. These relationships are expressed in a very general format, as in MODEL.LA, and can be specialised by identifying particular phenomena, e.g. a chemical reaction or specifying ideal mixing.

For more specific modelling applications, a "model browser" has been created to allow the specification of mathematical expressions. These are placed in a hierarchy of specialisations and generalisations. For example, an ideal K-value is a specialisation of a general K-value. The model can be constructed by defining an expression or by referring to other expressions and models which are to be included. The generalisations of the model are also searched for their expressions which are then added to the set. This procedure is similar to that used in ASCEND.

The model solver is the mathematical tool-kit Macsyma, which is also capable of algebraic manipulation to simplify models. The example quoted is of six equations representing the specification of an ideal K-value. Manipulation reduces the six equations to the single familiar expression. However, the specification of the models to be included as part of the ideal K-value model suggest prior knowledge of the solution.

The structure of the models and available tools imply that a model of a distillation column could be constructed in a manner similar to ASCEND, with the inclusion of models representing plates, reboilers and condensers. However, insufficient information is available to establish whether or not set operations are possible, e.g. defining a set of n plates.

A multi-level approach has been adopted which, potentially, removes the necessity for describing a generic model of a distillation column with plates. Initially, the modeller would construct a model of a distillation column, perhaps with approximate models such as Fenske's equation, but also, conceptually, with a model such as that described above. The model of the column could then be decomposed into plates and ancillary equipment which would be modelled individually. This would preclude the necessity for defining a generic column model, since specific plates would be defined as parts of a specific column. The separate parts could then be modelled as specialisations of a generic flash operation as before.

In conclusion, the three systems, ASCEND, MODEL.LA and ModAss, promote modelling as a design activity, but with differing emphasis. ASCEND, as a pure modelling language, provides great flexibility in defining mathematical models in a hierarchical manner. MODEL.LA is intended less for the modeller than the designer, allowing descriptions of the physical and chemical phenomena to be made, from which symbolic descriptions can be inferred. The user seems to have very limited access to the symbolic expressions, suggesting two distinct modes of use: one for the system and model developer, and one for the designer to employ the developed models. ModAss attempts to combine the two approaches, allowing some automatic inference of models from fundamental phenomena, while providing full access to the model development facilities.

Chapter 2

Knowledge Representation Techniques

The scope of this work was presented in Chapter 1; its aim is to provide a flowsheet modelling tool which can be used throughout overall plant design. This chapter will describe representational formalisms suitable for the task and explain why object oriented programming (OOP) was chosen. A description of the principles of OOP is followed by a comparison with the properties of the programming language used.

2.1 Classification of Representation Techniques

Three main knowledge representation techniques have been categorised by Jackson [34]:

- Rule based systems
- Systems of structured objects.
- Logic based systems

All three representation schemes can, in principle, be used to represent the same information, but the important distinction of their use is the ease of application to particular classes of problem. These techniques are described in [34], and further techniques, which have found less application in chemical engineering, are presented by Rich [35].

Rule Based Systems

Rule based, or production systems, describe the problem domain by means of a set of rules which are condition-action pairs, e.g.

$$\text{if } A_1 \;\&\; ... \;\&\; A_n$$
$$\text{then } B_1 \;\&\; ... \;\&\; B_m$$

This rule is interpreted as "if conditions $A_1 ... A_n$ are true, then actions $B_1 ... B_m$ can be performed". The rule set is supplemented by an Interpreter which determines the rules to apply, and a Workspace where goals are stated and information is added to the data structure.

This formulation is sufficient for problems with a well defined domain of knowledge. In chemical engineering, however, there is no theory for the practice of overall plant design. Even if one could be determined, the scope of the problem is too great to be represented by a single set of rules. A rule based description, therefore, could not be developed. An extended discussion of the issue of representing chemical engineering design in a knowledge based system is presented by Struthers [36]. Reduced domains, however, have provided successful applications in areas such as physical property prediction, distillation column sequencing, catalyst selection and heat exchanger network synthesis.

The CONPHYDE system [37] is a prototype expert system applied to the domain of physical property prediction. The program provides advice on the selection of thermodynamic models appropriate to particular vapour liquid equilibrium situations. The authors describe the system as being constructed from their interpretation of textbook knowledge which has been encapsulated in a set of rules [38]. This supports the hypothesis that the problem space must be well defined for this type of implementation.

CONPHYDE contains only 37 rules. These, however, appear adequate to describe the domain of knowledge. For more complex situations, more rules are required, e.g. HEATEX [38], an expert system for aiding heat exchange network synthesis, requires 115 rules. As more rules are added to a system it becomes more difficult to maintain consistency due to possible interactions between them.

The Interpreter is required to choose the most appropriate rule from those applicable in a given situation. However, as the methods of selection become more complex, the program developer must consider how and when the new rules should be fired. This is a major disadvantage of the rule based approach. When the rule base is large, the addition of new rules creates difficulty in determining their effect on program behaviour.

It is therefore not possible to use simple rule based systems for overall design. However, due to the modular nature of design, strategic knowledge tends to be concerned with particular tasks. This implies that it can be partitioned into Knowledge Sources representing experts in a particular field. Such "blackboard systems", use the Workspace to maintain facts about the problem in its context, see Figure 2.1.

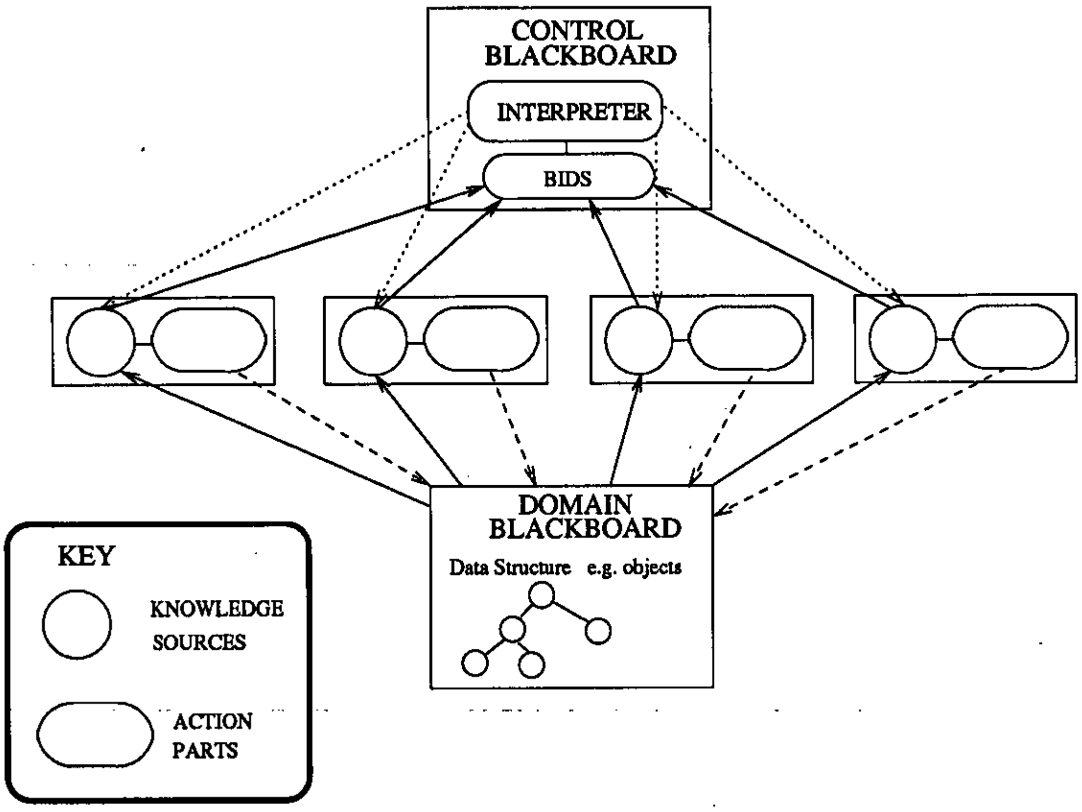


Figure 2.1: Structure of Blackboard Systems

The Knowledge Sources are continually checking the Workspace to identify information satisfying the conditions for their invocation. When such a situation is found, the Knowledge Source states its ability to fire and what it can achieve as a "bid". These operations are represented by solid lines in the figure. The Interpreter then determines which of the competing Knowledge Sources to implement. The dotted lines indicate the interaction between the Interpreter and the Knowledge Sources, and the dashed lines represent the addition of new information to the data structure by the action part of a Knowledge Source.

An example of a blackboard system is DECADE [39] which aids in the selection of a catalyst for a specified single step reaction. There are a number of Knowledge Sources concerned with topics such as the specification of the problem, thermodynamic consistency and the classification of the target reaction.

Systems Based on Structured Objects

Rule based systems lack any relational structure in their constituent knowledge bases. The rules are intended to be independent from each other to maintain modularity in terms of the knowledge they represent, so characterising relationships between items is not readily achievable in a rule based approach. Systems with an underlying structure defining concepts and the relationships between them are represented more suitably by object or frame based systems. Chemical processes naturally conform to a modular structure, with frames or objects able to represent operations. The ability to represent relationships between items, such as flowsheet connectivity, conceptual refinement and property inheritance, is also required in order to describe the knowledge used in design effectively.

The use of objects to describe a problem domain and rules to provide an inference structure is widespread in the context of large scale design. DESIGN-KIT is an object oriented implementation in KEE, an artificial intelligence (AI) toolkit, using rules to develop modelling descriptions. A broader viewpoint has been adopted by Lien [40] with AKORN D, which was developed with consideration of methods of design. Lien's argument is that while process design may conform to a seemingly procedural structure, it is not a rigid precedence ordering but merely a framework. The hierarchical structure of Douglas (see Section 1.1.2) may provide a guide to the overall problem solution, but a designer's interpretation may be different, moving freely between the tasks involved at the various levels.

AKORN D is a blackboard system which interacts with a domain knowledge base of frames implemented using the AI toolkit, Knowledge Craft. It is intended to provide a framework for solving problems in the opportunistic manner of designers, hence the implementation of the Blackboard, where bids by Knowledge Sources are the suggestions of members of a design team, and the control sequence is driven by the overall goals. AKORN D has been applied to the synthesis of distillation sequences in the program S^5 [33].

Logic Based Systems

The third representational technique is logic, which, in principle, can be used to represent any of the above information. The logic referred to here is a type of formal language which consists of syntactic rules for deduction. Logic programming as a basis for knowledge based systems uses a subset of predicate logic. Levesque and Brachman [41] claim that a tradeoff is required in the use of logic for representing knowledge, since, despite the undoubted representational power of a formal logic, its full implementation is computationally intractable. Therefore, the more complete the logic used, the more impractical it becomes. Logic programming languages, therefore, can only use a subset of a formal logic; the abilities of which, in a representational sense, are offset by the efficiency of its implementation.

Prolog [42] is an example of a language based on predicate logic possessing some considerable advantages over conventional programming languages for use in symbolic inference. The main features of Prolog are presented in Appendix A. Knowledge is represented as single statement Facts and multi-statement Rules. The Rules are constructed from Goals consisting of necessary Facts and Rules which must be satisfied. Proving a hypothesis, i.e. executing a Prolog Goal, utilises features such as recursion, backtracking, pattern matching and variable unification all of which are discussed in the appendix.

No significant systems have been developed entirely based on logic. However, since a logic knowledge base can be constructed similarly to a rule base, in which special cases are identified before more general ones, the combination of all three representational techniques can be incorporated into one system. This implies that the overall design problem can be decomposed into subproblems each of which can be represented by the most appropriate technique. The work described here has been implemented in an object oriented system which was written in Prolog. The combination is a powerful one for process design, allowing an object representation of the constituent operations and the relationships between them, plus access to the deductive capabilities of a logic based language.

The following sections describe a model-based approach to chemical engineering design as a method for structuring knowledge, followed by a discussion of object oriented programming which has been used to implement this structure.

2.2 A Model Based Reasoning Approach to Chemical Engineering Design

During the process of chemical engineering design, many models are created. Lien et al [9] identify three classes of model in design.

- **Models of physical things.** Such models include the mathematical descriptions of processes with the appropriate degree of complexity. They vary between qualitative models reflecting the tendencies of the model, and accurate models incorporating large sets of equations which may be algebraic, ordinary differential, partial differential or Boolean.

  Flowsheets are models in this sense, being constructed from individual unit operations. Flowsheets are also part of a model of the design which maintains the relationships between flowsheets, both alternatives and structural enhancements.

- **Models of the available tools.** Before strategies for solving design problems can be formulated, the tools available to the designer must be modelled, i.e. what the tools provide as an output and what their input requirements are.

- **Models of strategies.** For any automation of design or the facets of design, models of the methods are required. This may be for overall plant synthesis, design of an individual item of plant, or the formulation of a mathematical description of the process.

The model based reasoning advocated by Kunz [43] is a technique for structured development of knowledge based systems and representation of the domain knowledge. The basis of the approach is the concept of formal symbolic models in which structure and function are defined explicitly. Structure can be thought of as the attributes of items within a problem domain. In physical items, structure could include dimensions and capacities; conceptual items, such as theories, would include the parameters appropriate to its description. Function is a symbolic description of the item and what it is supposed to do, whether it be to react two components or solve a set of equations. Thus, model based reasoning is a method for solving problems by analysing the structure and function of symbolically modelled systems.

It is important to distinguish between formal symbolic models and the more traditional approaches of mathematical and heuristic models. Mathematical models, in general, are "black box" models, relating inputs to outputs. Intermediate calculations are not intended to be accessed by external models or programs and consequently no physical significance is associated with them. The function of such models, therefore, is explicit, but without any explicit statement of structure. The structure of the model is implicit in the mathematical representation, thereby limiting the reasoning possible with the model. For example, no information can be established as to reasonable input values or the interpretation of results, unless, of course, the mathematical model is one part of a formal symbolic model detailing this information.

Similarly, heuristic models relate inputs to outputs without explicit representation of the structure of the system being modelled. These different types of models are not mutually exclusive, and mathematical models and heuristic models can be complementary to a model based description. However, it could be argued that such models are only required when the knowledge of the function is incomplete or in the interests of efficiency. For an overall perspective and the representation of relationships between items of information, it becomes easier to reason when structure and function are stated.

A formal symbolic model of a system states explicitly the functional behaviour of the system and its structure, thus allowing reasoning providing, not only output information from input, but the internal states of the model. The symbolic description also has the advantage of being modular, resulting in a model description closely matching the problem decomposition of practitioners in the field of interest.

In order to communicate information about models and their associated deductions, a symbolic representation should reflect the natural idioms of the domain. Since, in chemical engineering particularly, information can be communicated more efficiently with the use of diagrams, a graphical representation of models can readily be implemented, either for displaying the results of some reasoning exercise or to allow input to the model.

As an example of the use of a formal symbolic description, consider the representation of a plug flow reactor. A black box mathematical model can provide only the type of information which can be obtained from a simulation, e.g. temperature profile, output composition, etc., which are explicit aims of the calculations. These models, however, require reasoned specification of input values and interpretation of results. Reasoning about the mathematical description can provide little information since the structure has not been stated explicitly, i.e. no significance can be attached to any variables in the model. For example, the relationship between catalyst deactivation, input temperature and composition may be incorporated in the model, but unless it has been identified and labelled, it cannot be used in any analysis other than simulation.

The mathematical model may determine some of its internal states in the course of calculation, which are not provided as output values. This information is not retained and may not be used for further reasoning. Other reasoning modules may require values for these internal states which may be established either by user specification or its own reasoning. This may introduce a conflict between the results of the mathematical model and the subsequent reasoning in situations where the internal states are different.

A heuristic model may provide the relationship described above, but again, reasoning about the structure of the reactor is impossible since heuristics would not describe a reactor in those terms.

A reactor is shown in Figure 2.2. Here, all structural aspects are available for reasoning. The functional description is provided by the statement that the reactor is of plug flow character, i.e. it has the properties of a reactor with refinements identifying it as plug flow. This description can be used to reason about the development of a mass balance. For instance, a reactor mass balance can be derived from the stoichiometric equation. In this case, the most efficient simulation can be achieved mathematically, but the mathematical model has been inferred from the specifications, thus the simulation accurately describes the designer's intentions.

Other types of reasoning could evaluate, for example, the relationship between volume, vessel pressure and vessel thickness.

The individual physical aspects of the model (e.g. vessel thickness, temperature profile) can be described in a similar manner, with facets such as the range

```
Function - is a : Plug Flow Reactor
           is a type of : Reactor
Structure -stoichiometric equation : A + B → C + D
           conversion : f(T, P)
           catalyst : ...
           catalyst deactivation : f(T, composition)
           inlet stream : ...
           outlet stream : ...
           temperature profile : ...
           volume : ...
           vessel thickness : ...
           pressure : ...
```

Figure 2.2: Formal Symbolic Description of a Plug Flow Reactor

of expected values.

A further advantage of the approach is the provision of a structured problem domain. An overall chemical engineering design problem domain is shown in Figure 2.3. This corresponds to the structure of integrated design in Figure 1.1, but here illustrates the information maintained in the overall domain and, thus, available to all reasoning modules.

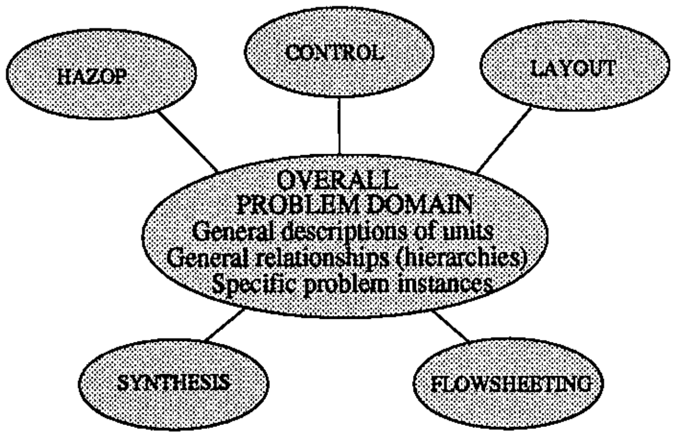

HAZOP
CONTROL
LAYOUT
OVERALL
PROBLEM DOMAIN
General descriptions of units
General relationships (hierarchies)
Specific problem instances
SYNTHESIS
FLOWSHEETING

Figure 2.3: Chemical Engineering Problem Domain

The central problem domain is the model of a current design which contains generic information about the subject, e.g. generic descriptions of unit operations, and specific information pertaining to individual problems specified by the designer, i.e. specific plant topologies.

The modules connected to the central domain are reasoning modules which can be thought of as models of theories. For example, the flowsheeting "theory model" may contain models not necessarily a part of (or of any use to) any other modules.

The flowsheeting theory model can be expanded as shown in Figure 2.4. The solution methods can be items with, as their function, the solution of equations. Their structure consists of their requirements, e.g. mathematical model, required formulation, specifications, etc.

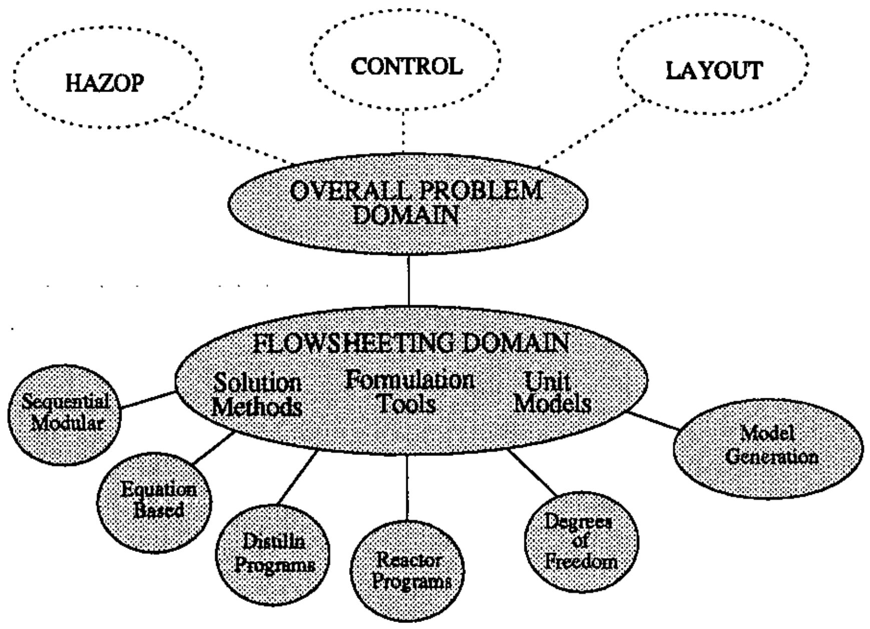


Figure 2.4: Flowsheeting Domain Model

The purpose of this theory model is to solve sets of equations appropriate to the status of the design, which is to be established by the equation generation module (Figure 2.4).

The representation of the models must include unit operations, with their structural and functional attributes, and conceptual items such as solvers.

The reasoning entails generating equations appropriate to the level of design and performing calculations appropriate to the structure of the equations, i.e. deciding which of the available techniques to use for a particular problem.

In summary, a formal symbolic description provides a consistent description of systems in a modular manner allowing different types of reasoning to be performed on it. This is a natural representation for chemical engineering where the domain is decomposed conveniently into unit operations. A model based description also provides a structured domain of knowledge which can be divided into models of theories. Chemical engineering design, with the different types of reasoning appropriate for different evaluations of a process (see Figure 2.3) is well suited to such a description.

2.3 Object Oriented Programming

Object oriented programming provides a highly modular structure for storing information in a manner conceptually similar to the physical description of a chemical process. OOP also supports a model based approach, allowing the representation of models in terms of their structure and function. The central organisational theme of OOP is the use of structured "objects" containing "slots" requiring "fillers" which can be data or procedures, to describe items of knowledge. The objects are related by an inheritance mechanism allowing a prototypical description of concepts reflecting their default states.

2.3.1 Storing Data in Objects

Object oriented programming is a language for representing groups of related information as objects and their attributes. Objects may represent physical items, such as a heat exchanger with its attributes being heat load, heat transfer area, etc. (see Figure 2.5), or conceptual items such as graphs with attributes including axes, labels, ranges, points, etc.

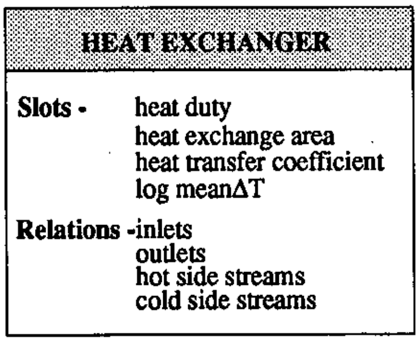


Figure 2.5: Object Oriented Description of a Heat Exchanger

The attributes of an object can be split into data values which can be stored in "slots", and those concerned with relationships to other objects, which are stored in "relations". This distinction, however, is not made in all OOP languages. For example, the AI toolkit Knowledge Craft supports the creation of user-defined relationships, but the MODEL.LA project [28] is an implementation using another toolkit, KEE, where the mechanism of such relationships had to be defined by the developers. Figure 2.5 shows the conceptual division of properties into slots and relations. Attributes such as heat transfer area and heat load which are properties of the object itself, are defined as slots, but the connections to the unit refer to objects representing streams and so are relations.

As objects have slots, slots can have "facets" which are the properties of the piece of data stored there. For example, the slot for heat exchange area has the defining properties that it is a number, and its expected value will be greater than zero. Similarly if a valve object had a slot for status, it would expect the value to be an atom and one of the set of enumerated types, "open" or "closed". Facets can be described within the structure of the object or separately depending on the syntax of the system. Figure 2.5 shows an object without facets as part of the structure. An example of a slot definition is shown in Figure 2.6.

slot definition:
    slot name - heat transfer coefficient
    object - heat_exchanger
    value type - number
    units - W/m^2 K
    value range - 0-10000
    default value - databank call...

Figure 2.6: Definition of a Slot

Facets can also contain pieces of procedural code, called "demons", which are to be invoked whenever the slot is accessed, either to set it or check it. The intended function of demons is to perform data validation and verification, but they can be used for a wider range of operations. The code may involve manipulating the value contained in the slot, or informing other objects and slots of a change in value. Demons can wreak havoc, however, if not strictly regulated, because the execution, or "firing", of one demon may involve setting a slot in another object, which in turn may set another slot, making the creation of loops almost inevitable. There is essentially no control over the firing of demons, so they are used with caution.

The "methods" which are used to manipulate an object are also stored in its slots. The code to perform the manipulation is accessed when a "message" is passed to an object requiring a particular method to be fired. This is analogous to calling a routine or function in a conventional language. For example, a method could be used to model the object. It makes sense, therefore, for the code to be associated with the object since different items can be modelled by different solution techniques.

Object oriented programming and other frame systems were developed from earlier graph based representations (see Jackson [34]) where nodes and links were used to represent concepts and their relations. The concept is not as simple in practice as it sounds, due, in part, to the ambiguity of node assignments. For example, a node labelled "car" could be referring to the concept of a vehicle, the class of all cars, a typical car or a specific car. Jackson illustrates part of this argument by describing the class of all cars with general properties, such as being constructed from a set of wheels, a chassis and an internal propulsion unit. An example of a typical car could be a BMW with four wheels and an internal combustion engine, as opposed to a three wheeled electric car. In addition to these properties, this typical car could be associated with a number of miles per gallon of petrol consumed, which may differ from that obtained by a particular instance of a BMW.

The human understanding of the situation is, therefore, dictated by the collection of prototypical structures constructed from previous experience of the subject. Rich [35] describes the analysis of new experiences as the evocation of the stored structures which are then filled in with the details of the current event.

Frames and objects were developed in an attempt to capture this type of knowledge. Objects are defined in "classes" defining the attributes common to all members of the prototypical class. For instance, the "class" or "generic" object representing a heat exchanger would contain all slots required by any heat exchanger, e.g. slots for heat exchange area and duty. The slots also represent prototypes of the properties which, therefore, can contain default values. Figure 2.5 is a depiction of a generic heat exchanger object.

A particular heat exchanger or "instance" is a copy of the generic template, including the default values for the attributes. The instance is likely to represent an exception to the prototypical heat exchanger, thus the slots will contain values consistent with the application to the specific situation, e.g. a heat exchanger called E101 may incorporate the general attributes of the class object retaining, for example, the default value for the heat transfer coefficient, but the values for the heat duty and area slots will refer to its current situation.

Classes are arranged in a hierarchy where subclasses represent specialisations of particular classes. For instance, a heat exchanger is a specialisation of an enthalpy change device, which is analogous to a typical car being a specialisation of the class of cars in the example above. The enthalpy change object represents the concept of operations that exchange heat (which can include heaters, coolers, boilers and condensers), and the exchanger object describes the class of all exchangers. Further specialisations could be plate exchangers and shell and tube exchangers. Following the car example, these can be viewed as the typical examples of heat exchangers.

The hierarchy allows inheritance of the prototypical properties of conceptually more general objects by the specialisations. This implies that information describing an enthalpy change device need not be contained in the description of a heat exchanger, i.e. the information is inherited by the subclasses. The specialisation represents an increase in the functional description of the object, so, for example, the enthalpy change device may incorporate the concept of a temperature set point, but include nothing about the heat load. The heat exchanger object which specialises it may contain the heat load information, but still has access to the concept of a temperature set point. An example of an inheritance network is shown in Figure 2.7.

Slots containing methods can also be inherited. The context of inherited methods determines the nature of the action. For example, a high level method associated with an object called "unit operation" may be to model the process. From the viewpoint of overall control, this may be the limit of a designer's interest. Objects defined as specialisations of a unit operation, e.g. a distillation unit, might require specific modelling instructions, such as incorporating a Fenske distillation model. The high level viewpoint need not be concerned exactly how each model is performed, but merely with the decision that this action should be taken.

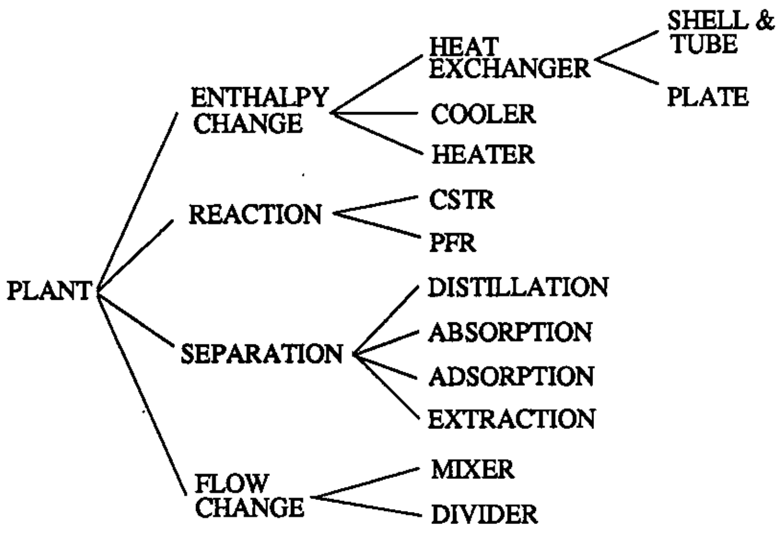


Figure 2.7: Example Inheritance Network

2.3.2 Program Control by Message Passing

The flow of information in object oriented systems is determined by the methods associated with an object's slots. Methods are fired by passing a "message" to the object, normally from another object. An immediate application can be seen in simulation, particularly of the sequential modular category. Each object can model a flowsheet item and pass messages to downstream objects to start their calculations. Loops can be resolved by allowing the natural iteration to take place until the convergence requirement is met, in which case message passing would cease.

Methods and message passing differ from demons in that demons are procedural attachments to slots which provide an action when a slot is accessed. They are intended to enhance functionality by maintaining slot consistency, rather than to control the running of the program. Methods are supposed to provide the structure for information flow by manipulating their host objects and passing messages to other objects. Obviously, for complex problems, a great deal of planning is required to ensure methods function as intended and avoid destructive interactions with each other (or with demons). The problem is similar to that experienced with rule based systems. The addition of new objects and new methods requires fundamental knowledge of the existing structure, both of objects and program control.

Hierarchical inheritance systems should, in principle, allow this to be done incrementally, by classifying new objects at the correct position in the hierarchy thereby defining their functionality. The nature of knowledge, however, is not clearly divisible into concepts that do not interact, which makes convenient classification more difficult than initially suggested. The problem can be solved in part by multiple inheritance, where a method can be constructed by inheriting methods from other specialisations in the hierarchy. To conform to any convention, this information should be inherited from the direct line of ancestors, but this is not necessarily the case. For example, the definition of a jacketed tank would inherit methods from objects representing a jacket in the class of heat exchange equipment, and a tank in the class of storage equipment. This can solve complicated situations, but loses the convention of inheritance in a structured hierarchy.

2.3.3 Worlds or Contexts

"Worlds" or "Contexts" are intended to allow the manipulation of objects for a particular purpose without affecting information elsewhere. In Knowledge Craft [44], a root context contains the knowledge base representing the problem domain. Contexts for different versions of the domain contain new information plus any changes from the root context. In this manner, modelling and testing of different situations can be performed in separate Contexts. For example, a Context may be used to accommodate synthesis procedures, while another may be used for hazard studies. Objects created by the synthesis world may have to be manipulated and changed for the purposes of assessing hazards, but since they are in separate worlds only a copy of the objects need be altered, the original objects being unchanged and available for further synthesis. The Contexts can then be "merged" to incorporate the changes into the original objects.

Contexts are structured in a tree, thus any Context can spawn child Contexts, with the intention that as they are developed they automatically copy any objects requested from the parent Context. Consistent management is required to ensure that when merging is attempted, only a complete version of the object is copied, i.e. there are no other versions in child Contexts in a different state. The tree structure is equivalent to an inheritance hierarchy for objects.

2.4 CLAP - Combined Logic and Procedures

CLAP is an object oriented programming language developed in Edinburgh by A. Struthers [36]. It is written in Prolog, providing full access to its logic programming capabilities, traditional object oriented facilities as well as high level procedural programming constructs in the CLAP language (such as loops, if-then-else, etc.) and an extended interface to the C programming language.

CLAP supports objects as described in Section 2.3.1 with slots, facets and demons. The conceptual difference between slots and relations is realised with different modes of use for each. Slots have facets and demons as discussed above, whereas relations can have inference techniques associated with them (see Section 2.4.1 below).

The differences between CLAP and traditional object oriented languages stem from its engineering origins. Engineering contains many activities more suited to a procedural representation than the concept of message passing incorporated in most object oriented languages. In traditional OOP, control is achieved through the firing of methods in an undetermined order, or by achieving global constraints with the use of global variables. CLAP retains the advantages of message passing and objects, but differs from traditional OOP in that methods are removed from the objects. This allows a high level control mechanism to be used, i.e. methods are fired in a determined order.

CLAP also displays differences in the use of message passing. Traditionally, messages are sent to objects invoking one of the methods in the objects slots. In CLAP, messages are used to allow objects to communicate with each other and, more unusually, with methods. A message consists of a piece of information, which may be symbolic, numerical or a combination, which can be attached to objects or methods and subsequently used to make inferences about the state of the object or method. When a method checks what an object has been told, a piece of inferential code can be invoked which can provide some conclusion based on all the information received by the object up to the point the inference was made.

The provision of worlds or contexts in CLAP is different from the implementation in conventional OOP discussed in Section 2.3.3. Each context is held entirely separate from every other. There is no organisational structure, so no automatic inheritance is possible. Copies of entire contexts are possible, or the copying of individual objects. The operations then carried out do not affect the original context. No automatic merge facility is provided, so significant changes must be noted and implemented by the programmer.

Instances of CLAP code appear in various examples throughout this thesis, so a brief discussion of the syntax of the language will be given here. Unless otherwise stated, any word appearing in an example preceded by a "$", is a CLAP keyword. Any term that begins with a capital letter is a variable according to the convention of Prolog.

Generic objects are defined by a type name, a relation to a parent object in the inheritance hierarchy, a list of slots, a list of relations and an optional list of display commands. Figure 2.8 shows the heat exchanger object from Figure 2.5 as a CLAP generic object.

```
object(heat_exchanger) :-
     self - _,
     variables - [Q,A,U,DT,I,O,HS,CS,Pts],
     slots - [is_a - enthalpy_change device,
               heat_duty - Q,
               heat_exchange_area - A,
               heat_transfer_coefficient - U,
               log_mean_T_diff - DT,
               screen_location - Pts],
     relations - [inlets - I,
                  outlets - O,
                  hot_side_streams - HS,
                  cold_side_streams - CS],
     display - [draw_exchanger(Pts)].
```

Figure 2.8: CLAP Representation of a Generic Heat Exchanger Object

The variable list allows unification of slot values. For example, when the value for the "screen_location" slot is set, the variable `Pts` in the display call is automatically unified to the new value.

To create an instance of an object, the command is as follows:

```
$create_object heat_exchanger-e101 $in world1
```

which creates an instance of the heat exchanger object called `e101` in context `world1`. The specification of the context is optional, the default being the current context.

To set and check slots and relations, the commands are of the following form:

```
$set slot-e101-heat_transfer_coefficient-800
$set relation-e101-inlets-[s1, s3]
$check slot-e101-(heat_duty-H)
$check relation-e101-(hot_side_streams-S)
```

Note that the form of the slot and relation values can be in any format, here numbers and lists. This can be constrained by using the facet construct, where each slot can be defined in a manner similar to generic objects.

Slots and relations can have "meta-slots" which describe a particular view of a slot. For instance, a flowrate can be described using a range of different units, e.g. kg/s, tonnes/yr, lbs/s, etc. Different tools and different users may wish to access such a slot from different viewpoints. The stored data should be consistent, e.g. all SI units. Pieces of code can then be written to provide the conversion from one set of units to another. To access a slot by a meta-view the call is as follows:

```
$check slot-stream1-(flowrate-Flow@@kgs)
```

These examples are sufficient to illustrate the form of CLAP calls. Further descriptions will be given as necessary, or see the CLAP reference manual [45]. The facilities receiving particular attention in the remainder of this chapter are the range of relations available with their associated inference capabilities, and the implementation of CLAP methods with their extension for loosely defined procedures.

2.4.1 CLAP relations

There are three types of relation available in CLAP: inheritance relations, standard symbolic relations and user defined relations.

Inheritance provides a functional relationship between objects, as described in Section 2.2, allowing reasoning about the operation of an object. Specification of the parent is achieved by placing an "is_a" in the slot list with its value set to the generic parent object. This can be left as a variable and subsequently set dynamically, but this seems to defy the reason for having it. Objects related in this way inherit the slots and relations of their conceptual ancestors along with their associated facets and defaults in the case of slots, and inference techniques in the case of relations. Thus, if a shell and tube exchanger is defined as a specialisation of the heat exchanger object, it will access its own slots as above and also the generic slots of a heat exchanger object.

Standard symbolic relations are defined within an object in the same manner as slots. However, the value stored in the relation can be reasoned about using a separate piece of code to make some inference. The inference technique can be loaded from a separate file, so can be in any format, e.g. rules, C functions, etc. For example, Figure 2.8 shows a heat exchanger with relations for its inlets and outlets. There is also a relation to store the hot side stream information, which the user would not explicitly have to provide a value for, since an inference technique could be used to infer the information from the specification of the other relations. In this case, checking the temperatures of all the connecting streams would indicate which stream was on the hot side.

Inference techniques and the relations they operate on, are inheritable. However, they only apply to the local level of inheritance. The inference method takes as one if its arguments, the relation list of the object to allow the use of combinations of the relations in the decision mechanism. Using the above example, the shell and tube exchanger may inherit the relations and inference methods of the generic heat exchanger, but these are not combined with the local relations in any inference, i.e. the four relations in Figure 2.8 cannot be used in conjunction with those defined locally for the shell and tube exchanger.

The third type of relation is defined independently from any object. The resulting relation can then be applied to any object to which the relation is applicable. There are five fundamental types of these relations arranged in a hierarchy as shown in Figure 2.9. The root relation has four subclasses: symbolic, constraint, specialise and operator.

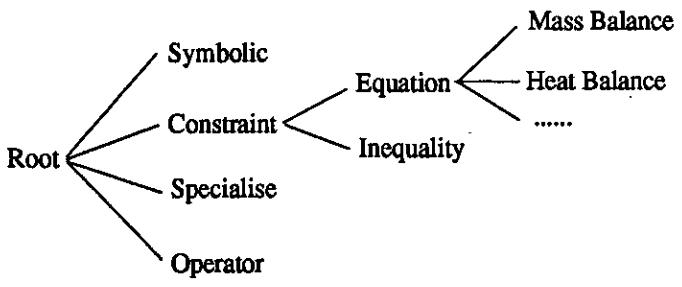


Figure 2.9: CLAP Relation Hierarchy

Symbolic relations are similar to standard symbolic relations and can be used to represent concepts such as "upstream of" or "connected to". The specification of a user defined relation automatically implies its inverse. If a label for the opposite definition is given, it is used to describe the opposite relationship, otherwise CLAP creates a relation called "inverse". For example, if objects a and b are connected by the relation "upstream_of" such that a is "upstream_of" b, the inverse is also true. If the name of the inverse is supplied, it can be invoked as, for instance, b is "downstream_of" a. If it is not provided, the relation will be automatically, b is "upstream_of_inverse" a.

The second group, constraint relations, are ones describing the form of a constraint of an object, typically an equation or expression. When a constraint, such as a mass balance, is defined, it is placed in the relation hierarchy as a specialisation of the constraint relation, i.e. as a "specialise relation". Code written to manipulate the constraint relation can therefore be used to manipulate all of its specialisations. Chapter 5 describes the use of constraint relations to provide flowsheeting equations and balances.

Constraint relations are defined similarly to objects, for example the mass balance relation shown in Figure 2.10.

```
relation(mass_balance,Unit-Val) :-
    variables - [Unit,Form,Bindings],
    bindings - [I = inlets $of Unit,
                O = outlets $of Unit],
    return_form - (sum_of(mass_flowrate $of I, $over I)
                =  sum_of(mass_flowrate $of O, $over O)),
    return_type - equation,
    slots - [is\_a - constraint].
```

Figure 2.10: Example Constraint Relation Describing a Mass Balance

Constraint relations have two major parts:

- the "return form" which is the generic description of an equation or expression,
- the "bindings" which is a list of variables contained in the equation with the objects, slots and relations to which they correspond.

Relations can be in any of the forms described above, so they could contain reference to further mathematical expressions. The implications of this are discussed in Chapter 5.

The generic constraint form is expanded into a specific form of the constraint applied to a particular object by the third user defined relation type, the specialise relation, e.g. the mass balance of a particular mixer. The constraint form of a mass balance shown in Figure 2.10 can be applied to the mixer by a specialise relation which expands the generic equation according to the bindings specific to the instance of the mixer. If, for example, the mixer had two inlets, s_1, and s_2, and one outlet, s_3, the specialised form of the relation would be:

$$In_1 + In_2 = Out_3$$

$$\begin{aligned} \text{where: } In_1 &= \text{mass\_flowrate \$of } s_1 -> Val1, \\ In_2 &= \text{mass\_flowrate \$of } s_2 -> Val2, \\ Out_3 &= \text{mass\_flowrate \$of } s_3 -> Val3. \end{aligned}$$

Individual specialise relations for each constraint relation can be written, but the default expansion of the generic form is normally sufficient.

It should be noted that at this point the expression has not been evaluated in any way. No specifications have been included. This implies that one specialised form can be used to obtain solutions for different problem specifications.

The fourth subclass, the operator relation, evaluates the specialised form as the range of the relation. The default operator relation checks all bindings in the specialised form (see above), checking the slots and relations to find values for specifications and further expansions of relations. These are placed in the Value part of the above binding expressions. The fully specified equations are created by unifying the Value part with the variable part also present in the equation (e.g. $Val1$ with In_1). Solution can then be achieved using code in the `active_code` slot of the operator relation, which can then supply the equations to a solution method or package.

2.4.2 CLAP Methods

Methods are pieces of procedural code used to manipulate objects. The CLAP language allows the code to be made up of procedural items such as loops and if-then-else tests, CLAP calls to set and check slots or make inferences, and Prolog calls. It is also possible to define relations which call C subroutines.

The single argument to a method is a list of objects whose generic type or types are defined in advance. This is intended to provide specific applications for specific objects. For example, a method could be used to model a distillation operation by the McCabe-Thiele procedure [46].

Extra information required by a method can be "attached" in a similar manner to slots. This is the CLAP definition of message passing which is in contrast to the interpretation used in other object oriented systems (see Section 2.3.2). This information can be attached by demons or as the result of an inference. More unusually, it allows communication between methods. For example, once the McCabe-Thiele method has been completed and the number of plates is known, it could send a message to another method for performing distillation column plate to plate calculations. The plate to plate method would ensure that a message had been received in which case it could proceed. Otherwise it would call McCabe-Thiele itself or return, having failed. A section of such a method is shown below.

```
method(plate_to_plate, distillation-C),
    variables - _ - _,
    type - program,
    program( $if not($attached (number_of_plates-N) $to self)
                  $then $call mc_cabe_thiele-C,
             $else....).
```

Inference techniques can be used to determine the best method by which to model a particular object. The inference returns the name of the appropriate method and automatically runs it with the object under scrutiny as its argument. For example, if it were required to calculate the theoretical number of trays in a distillation column and methods were available to perform the calculation using McCabe-Thiele, Ponchon-Savarit or Fenske, the appropriate method could be inferred by checking the specifications and other information available to the distillation object.

2.4.3 Extended Methods

Standard CLAP methods provide a means of implementing procedures which are well defined in terms of their constituent actions and possible alternatives. The steps which make up a procedure and points where a choice can be made are known, along with the alternative courses of action.

This description corresponds with the definition of routine design discussed in Section 1.1.2. However, non-routine tasks cannot be represented in this fashion. The constituent parts of such tasks are known but are not performed in any particular order, or, more likely, the order is dictated by a particular designer and a particular design. The choice points are numerous and the alternatives prohibitively many to represent using procedural constructions.

Non-routine procedures can, however, be given some procedural goals even if the goals are not achieved by sequentially ordered subgoals. For example, the synthesis procedure of Douglas discussed in Section 1.1.2 is a set of procedural goals i.e. the input-output structure must be defined before the recycle structure can be considered. The goals themselves consist of a wide range of flowsheet synthesis decisions, including flowsheet creation and the analysis and evaluation of alternatives. These subgoals, however, are not normally performed in a procedural manner. The designer will switch between the various options as necessary.

A new type of extended method has been developed to describe non-routine procedures. Figure 2.11 shows the structure of an extended method.

```
extended_method - Method :-
        calling_sequence - call1..call2..call3..
        guards - guard1..guard2..guard3..
        macros - ....
        assertions - assertion1..assertion2..assertion3..
        loopback_points - point1..point2..point3..
        object_of_interest - Object
        other_slots - .....
```

Figure 2.11: General Structure of Extended Methods

The calls which constitute the method, whether they be sequential or otherwise, are contained in the calling sequence. If the calls can be made sequentially they are simply connected in a dotted sequence (a CLAP structure of the form: a..b..c). If, however, some calls have no defined order, they can be represented and performed "in parallel". Parallel calls can be included as part of a sequence, thus allowing procedural progression and parallel decision making.

In this context, "parallel" refers to tasks which, when grouped together, represent the options available at a particular stage and can be performed in any order any number of times. The physical representation to the user is as a menu displaying the courses of action relevant at that point. Ideally, there would be different windows for the individual tasks, the user selecting the window, and hence the operation to be performed. The graphics package supported in CLAP has proved a major limitation in this area, resulting in only a single window being used and operations being selected from a menu.

The relevance or appropriateness of any operation can be assessed by providing associated "guards", which are conditions that must be met before a call can be performed. In the case of a sequential call this prevents any further progress until the conditions are met, and in the case of parallel calls this results in exclusion from the menu of options. This is not entirely satisfactory since the designer may complete a parallel call unaware that some options are unavailable. The intention is to display the excluded calls in the menu to indicate their presence at the level of interest but prevent their selection.

The structure of extended methods shown in Figure 2.11 allows the replacement of excessively long guard conditions by macros. This makes written methods more legible.

Once a call has been completed it is useful to be able to make some assertion about the status of the procedure. This may be in the form of setting slots, relations or facts, and may involve the extended method or other relevant objects. Similarly to guards, assertions are contained in a dotted sequence, each one corresponding to a call in the calling sequence.

When a guard is not satisfied and the execution of its corresponding call is prevented, it is desirable to know the reason for the failure and what further action to take. The explanation is entirely up to the programmer, while extended methods provide the means for obtaining the information required by the guard. Each guard is associated with a single call and in turn with a loopback point which returns the program to an earlier point in the extended method (or in an earlier extended method) where the information can be obtained. This allows the user to provide the appropriate information before proceeding.

One, normally high level, object is provided as an argument to extended methods thus providing a centre of interest and source of information. The method may manipulate the object, and the guards may access its slots and relations.

Further slots are provided to accommodate the status of the method and a predicate to update the status, plus slots for information about displaying the method.

An application of extended methods in synthesis, in particular the Douglas procedure, is discussed in Chapter 6.

Chapter 3

Techniques for Solving Flowsheets

Mathematical modelling programs used in chemical engineering design can be broadly divided into one of two categories: those used for overall process flowsheet modelling, and those used for modelling specific unit operations. Both types of program are used extensively throughout design. One aim of this project was to establish the extent to which existing programs can be incorporated in a modelling environment to be used during the design procedure. The flexibility required by a designer in constructing mathematical descriptions of items of plant or entire processes must be weighed against the ease with which it can be provided. This chapter discusses the approaches currently used for solving flowsheets and how significantly the flexibility must be compromised for the implementation of each.

Flowsheeting programs perform mass and heat balances on plant models specified by a designer or engineer. Additional calculations also provide information about the sizing of the items of equipment included in the flowsheet.

The structure of a typical flowsheeting program is shown in Figure 3.1. It consists of an executive which reads the specified plant model from an input file and formulates a mathematical model appropriate to the solution method available. This formulation uses a library of predefined unit models and physical property relations. The model is solved and the solution placed in an output file.

Conceptually, the simplest and most general way of using existing programs in conjunction with an environment is to provide an interface to the input and output files. It is the most general approach because any program operating an input/output file format can be accommodated by specifying a mapping between

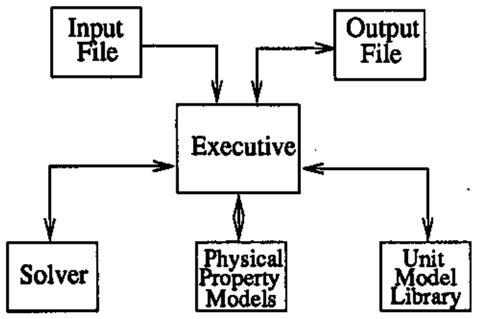


Figure 3.1: Structure of Flowsheeting Programs

the symbolic data structures in the environment and the format required by the program. This is the approach adopted by PROCEDE (see Section 1.2.3). For such systems, some correspondence must be found between the process defined by the designer as a set of conceptual operations, and the existing library of unit models. This constitutes a major drawback. For instance, PROCEDE claims to have a library of over 200 standard symbols for the conceptual description of the process. It is unclear how the system resolves the problem of interfacing to a flowsheeting program with, for instance, a library of 50 unit models.

The model libraries place some undesirable restrictions on the specification of the plant model. Whatever solution method is used (see Section 3.1), the libraries provide a range of fixed format mathematical models, whether they be routines or sets of equations. The models must be of sufficient detail to provide an accurate simulation of the unit operation in question, but if a more approximate model is required, either a new unit model must be provided, or the more detailed one has still to be used. In the latter case, the input information required for the full model must still be provided. For example, the evaluation of a heater may be necessary with the intention of calculating its heat load. The library model may incorporate the equation:

$$Q = UA\Delta T \tag{3.1}$$

for the calculation of heat transfer area, A. However, it requires a value for heat transfer coefficient, U, which may not be available or is inappropriate for the model of the plant and, therefore, not of interest. In this case an estimate must be provided before the model can be used, which, if the estimate is not revised, may lead to errors in later stages of design.

Due to the wide variety of equipment items used in chemical plants and the limited number of models available in a unit library, it will often be the case that there is no unit model for the particular process. This may require the combination of two or more models which together match the necessary performance. For instance, in Esspros, discussed in Section 3.3, a model of a nitric acid absorption column can be created by the combination of a mixer, two reactors and a separator.

In other cases a different model may provide a reasonably accurate description of the process being modelled. For example, during a series of interviews designed to uncover such problems, a situation was found where an engineer had required a model for a condenser. There had not been one available, so a flash had been used instead with the outlet vapour flowrate set to zero. The official response held that while this was acceptable, a mixer model would have been better.

The solution suggested for the latter problem is not acceptable in a design situation where the information generated as part of the continuous design procedure is to be maintained with the decisions taken. There is a loss of functional knowledge when one conceptual operation is replaced by a completely different one, e.g. the replacement of a condenser by a mixer which is conceptually an unrelated operation.

The implication of the above is that maintaining a library of unit models provides a restriction on the specification of a flowsheet description. To develop a knowledge based flowsheeting system using an existing library, the mathematical relationships incorporated inside each model must be known. Situations such as the condenser/flash/mixer example described above can then be resolved, since the fact that the mixer model contains the most complete or consistent heat balance equations would be encapsulated in the system knowledge base. The appropriate combination of models representing the nitric acid absorber could also be inferred.

The determination of suitable models has gone beyond the functional description of both conceptual flowsheet operation and unit model. The suitability of models is being determined at the level of equations. Section 3.3 addresses such an approach where a model composed of library units is deduced from a set of equations.

Due to the restrictions of the model library format, much of this work has been concerned with developing an alternative approach, which is discussed in Section 5.4.2. Removal of the library implies that models must be generated by the tool as opposed to the flowsheeting program. Thus, a more flexible modelling format must be provided to solve the problems discussed above.

The part of the flowsheeting program remaining from the structure shown in Figure 3.1 is the solver. The remainder of this chapter discusses methods of flowsheet solution and how different techniques satisfy the requirements of a flexible flowsheeting tool.

3.1 Flowsheeting Solution Techniques

There are three main types of flowsheeting program: sequential modular, equation based and the slightly less well known simultaneous modular. A general discussion of process flowsheeting techniques can be found in Westerberg et al [47], and a review of flowsheeting programs is presented by Flower and Whitehead [48,49].

Sequential modular programs perform calculations on a unit by unit basis. Each unit in turn has its outputs calculated from its inputs. The units consist of subroutines containing the input/output relationships describing the process and the method with which to solve them, i.e. modelling equations are represented implicitly. When a recycle is located, a selected stream is "torn" and estimates are provided for the variables in that stream. There may, of course, be more than one recycle and hence torn stream. Algorithms are available for the selection of optimal tear streams, e.g. Lee and Rudd [50]. A pass is made through the flowsheet which provides new calculated values for the estimated variables. These can, in principle, be used as new estimates, or various standard convergence techniques can be used to bring the estimates closer to the true solution. When the recycle values match their previous estimates, the recycle is solved.

The main failing of the sequential modular approach is the difficulty of applying design constraints. Since outputs are calculated from inputs it is not possible to constrain a unit's output directly. This can be overcome by using "control units" which allow the user to apply constraints while varying selected design variables.

Most commercial flowsheeting systems are based on sequential modular architecture, some examples of which are: FLOWPACK (ICI) [51], PROCESS

(Simulation Sciences) [52], FLOWTRAN (Monsanto) [53] and ASPEN (Aspen Technologies) [54].

Equation based programs have not been widely used in industrial practice. Until recently, no acceptable program was available. The earliest used only linear models [55] which were inadequate for describing complex unit operations. Development of robust non-linear packages has been rather slow.

Examples of academic equation based systems are: SPEED-UP (Imperial College) [56], ASCEND II (Carnegie Mellon University) [57] and QUASILIN (Cambridge University) [58,59]. An extensive survey of equation based systems is given by Shacham et al [60].

Equation based programs explicitly represent the equations of unit operations, compared with the implicit representation of the sequential modular approach. When a simulation is performed, these equations are solved simultaneously, usually by a Newton method. This produces a sparse matrix of the coefficients of linear and linearised non-linear equations. This is solved by Gaussian elimination.

The equation based approach has the advantage of being able to accommodate design constraints which, being equations themselves, are simply added to the equation set. This represents one of the disadvantages of this approach, since the arbitrary specification of consistent design variables in systems of hundreds of equations is highly intractable. Poor selection of design variables can result in a singular matrix and thus no unique solution to the problem can be obtained.

The main drawback of the equation based approach is the requirement for initial values for every variable in a given problem. Poor initial estimates often result in non-convergence, thus initialisation represents a crucial step in equation based solution. Initialisation of such a large number of variables must be performed by a program.

Simultaneous modular programs are a combination of the above methods. Unit models contain detailed descriptions of input/output relationships, as in sequential modular systems, and simpler linear relationships with adjustable parameters. A sequential pass through the flowsheet is made, providing coefficients for the linearised unit models. The linear equations are then solved simultaneously as in an equation based system. The sequential pass overcomes the problem of variable initialisation for the equation based part of the method.

The simultaneous modular approach was developed to introduce the desirable aspects of equation based flowsheeting to existing sequential modular programs. The large investment in the sequential modular approach has resulted in users being reluctant to change to the equation based techniques. The application of this combined approach, however, provides some of the speed of an equation solver, and the ability to place design specifications in a more flexible way, while retaining the numerical stability of the sequential modular method.

There are few examples of successful simultaneous modular systems, but examples include the work of Perkins [61] and Mahalec et al [62].

A similar system has been proposed by Johns [63,64] where, instead of linear models, simplified non-linear representations of the "rigorous" equations contained in the sequential modular units, are used for simultaneous solution. The method of solution performs an initial iteration using the rigorous models which generate parameters for the simplified models. Iteration on the simplified models is then performed until convergence is achieved or the model is no longer valid. A further iteration of the detailed model is performed before the simplified model is used again. This procedure continues until convergence is achieved.

3.2 Solution Method Requirements for Flexible Modelling

This work has been concerned with developing flexible modelling tools, rather than creating the definitive flowsheeting algorithm. Appropriate strategies have been developed for formulating problems for solution by suitable techniques. Before describing formulation methods it is necessary to describe the implementation of different solvers and their specific requirements. For example, an equation based solution technique needs methods for checking design specifications and variable initialisation. This discussion also leads to a choice between equation based and sequential modular solution approaches for flexible modelling applications.

From a modeller's viewpoint, development of mathematical descriptions is aided by the ability to express relationships as equations. Mathematics has a widely known syntax, so models created as sets of equations should be understood by model users as well as developers. If models are created using the "model based" techniques described in Section 2.2 then each term in each equation will be available to be reasoned about. In this respect, sequential modular unit models are "black boxes". The equations are stated implicitly. Thus the terms are not available for reasoning. A model user, therefore, has no access to the equations in the model and so has little scope for modifying them.

An equation based description, however, allows flexible model development and use. Mathematical syntax can be used to define sets of equations to represent particular operations, such as vapour-liquid equilibrium. These sets can then be used as the basis of high level descriptions of flowsheet units, e.g. a flash unit can be modelled by a vapour-liquid equilibrium, a mass balance and a heat balance. Using a "model based" description, the individual terms of the equations can be accessed for modification by the user of the model. For example, a vapour pressure may be supplemented by a fugacity coefficient.

A "model based" approach to mathematical modelling implies that, of the flowsheeting solution methods described in Section 3.1, an equation based technique is most appropriate. However, existing programs with a structure as shown in Figure 3.1 do not permit flexible access to their constituent parts. Allowing modellers to create descriptions which can be reasoned about for different applications, e.g. symbolic linearisation and graphic presentation, and allowing users to modify these descriptions, implies that the unit model library is no longer of any use. The executive can also be replaced by a reasoning module which permits the access to the equations described above, as well as performing the required formulation tasks. The input and output file format can also be discarded in favour of an interactive system. Therefore, the only required part of as existing equation based program is its solver, in which case it may be easier to develop a solver specifically for such a modelling system than to modify an existing one.

If a sequential modular solver is used, more of the program structure is retained. This is a supporting point for this approach which should be considered since it is the most widespread of those discussed. Unit models are fundamental to the solution technique, so a unit library must be maintained. Since this is the case, and no access to the equations can be gained once the model is created, no interaction with the formulation executive shown in Figure 3.1 is necessary. Thus, problem specifications can be mapped from the symbolic data structures of the modelling tool to the input file format of the solver. Similarly, solutions can be interpreted from the resultant output file. Despite the restriction of not being able to modify models once they are created, it is still useful to be able to build composite models from library units. A method for generating appropriate sequential modular models from an equation based description is described in Section 3.3.

Solvers are not only required for flowsheeting calculations. Throughout design, models of whole flowsheets and smaller models of their constituent parts are created. The "model based" approach described above allows construction of mathematical descriptions for different levels of complexity, since, throughout, they are similarly defined as sets of equations. An equation solver can be used to evaluate any such formulation. However, a sequential modular solver is restricted to the models available in its library. To describe detailed behaviour of individual operations, access to the equations is required. For this reason, an equation based solver is required to supplement the sequential modular technique for the accurate simulation of such situations.

In summary, a flexible modelling format, for both modellers and users, can be developed using equations. Access to individual terms of each equation can be permitted using a "model based" representation. An equation based solver is the most appropriate of those discussed for evaluating the generated model, but most, if not all, of the structure of existing equation based programs should be discarded and replaced by the proposed system. A sequential modular program could be used with little alteration, but does not provide adequate access to modelling equations.

For these reasons, an equation based technique has been used as the primary solution method. The flexibility obtainable from such a technique outweighs the robustness achievable by other methods. It should be recognised, however, that both equation based and sequential modular methods have been implemented. The representation of flowsheet information is such that any solver is merely manipulating structured data describing a design. The data can be formulated in a manner solvable by a sequential modular approach even if it does not match the flexibility of equation based techniques.

The following sections discuss the implementation of both sequential modular and equation based solution techniques. The solvers all take a set of equations as their input.

3.3 Implementation of a Sequential Modular Solver - Esspros

Sequential modular programs retain most of the original program structure as discussed in Section 3.2. Most systems are written in FORTRAN, although the program used, Esspros, is also available in Fortran8x and C. The reasoning provided, converts an equation based model into the Esspros input format. This is achieved with a set of rules written in Prolog.

Esspros [65] is a flowsheet modelling program for performing mass balances based on a sequential modular architecture. The unit model library consists of a set of fundamental operations which can be used to construct more complex processes. For instance, in the synthesis of nitric acid, an absorption column is used to react the NO_2 and NO obtained from the reactions, with water, and thereby separate them from N_2 and O_2. The Esspros model of the absorber involves a mixer, two reactors and a separator (see Figure 3.2).

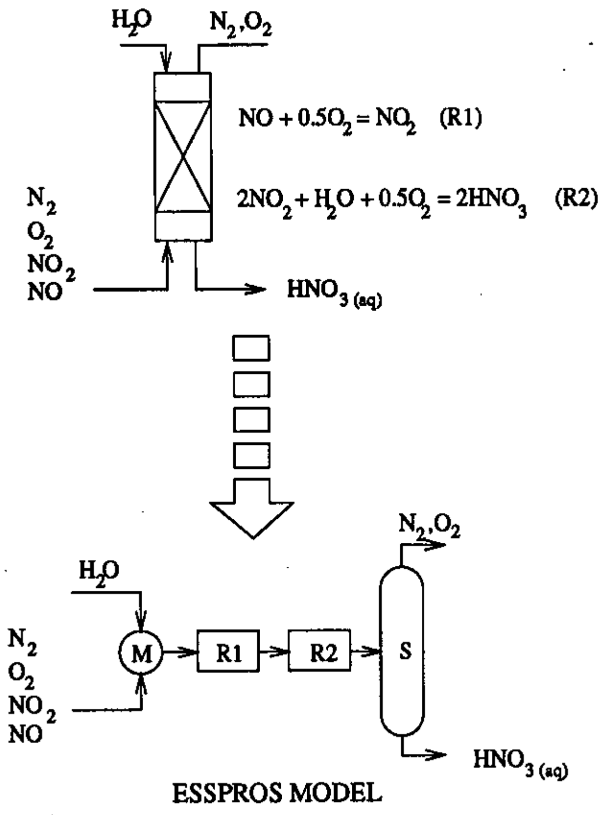


Figure 3.2: Esspros Decomposition of a Nitric Acid Absorber

The input to Esspros is a file which is written as a program. Declarations are made for the number of components and, optionally, what they represent. The individual items are written as subroutine calls with parameters defining the streams in and out, and, where appropriate, additional specifications.

The library contains the models: mixer, splitter (divider), separator (three types), reactor, flash, distillation and two stream types - feed and recycle. Most models require some additional parameters which effectively fulfill the degrees of freedom or indicate the mode of operation (e.g. the separator has three modes where the product specifications can be made as recoveries, flows or mole fractions).

These models, therefore, contain the basic equations for mass balances, which can be associated with particular equations and sets of equations provided by the model generation phase. The types of equations generated also indicate the mode of operation of the models.

As an example, consider the absorption column discussed above. The representation in CLAP is a single absorber with two associated reactions (see Figure 3.2). The model generation phase returns a set of mass balance equations incorporating the two feeds and the two product streams with the two reactions. The specification of the product mole fractions (it could apply equally well to flows or recoveries) is inherent in the mass balance equations, i.e. the required quality of the nitric acid.

The general form of the mass balance equations is sufficient to imply the use of a mixer as the first operation, i.e. there are two inlet streams. Equally, the two outlet streams imply a splitter or separator. The specification is in terms of mole fractions and not stream ratios which, therefore, denotes a separator and its mode of operation. The mass balance includes a specification of two stoichiometric equations, thus indicating two reactor models (with the required stoichiometric parameters). Also included will be an expression indicating reaction conversion, which is required by the reactor models.

In this manner, large scale models can be constructed. There are, however, significant reservations about this inference mechanism. For example, Esspros requires a separator to have one output stream which is completely specified, i.e. the mole fractions of all components in either the top or the bottom stream. The flexible specification encouraged by the equation based system allows these specifications to be mixed as long as they are consistent. Esspros, therefore, requires some major searching for specifications or a fundamental restructuring of the model building facilities to prevent such mixed specifications from being placed. Thus, a suite of tools is required to aid in the formulation of sequential modular problems, on top of those required for the equation based modelling of the individual items.

One such tool required for the Esspros implementation was a method for the recognition of recycles and choice of the optimal tear streams. Currently, an Esspros user must identify the recycle streams and specify them as such. An algorithm has been implemented in Prolog for locating recycles [66]. Since the Esspros description of the process generally contains more units than the CLAP representation, the algorithm operates on the CLAP structure.

Once recycles have been identified, other algorithms are available for selecting the optimal tear streams (see Rudd and Watson [67]).

3.4 Implementation of Equation Based Solvers

As discussed in Section 3.2 the implementation of an equation based solution technique requires not only a solver, but also additional tools to support formulation. For example, a method for assessing the suitability of design variables has been implemented.

A tool of the type proposed should have a suite of solution methods at its disposal. Equation based solvers are well suited to solving models of entire plants, but are unwieldy for solving individual units or individual equations. In these situations small, simple solvers are more appropriate. There are also programs specifically for performing calculations on one particular item of equipment.

Two small solvers, intended as rapid calculators, have been used to augment the capabilities of the large scale flowsheet solvers. Their range of applicability is small, but they provide an example of the use of different solution techniques.

3.4.1 Degrees of Freedom and Sensitivity

In specifying the performance of a plant, section or unit, it is necessary to choose variables which can be assigned values without over-constraining the unit. This is not straightforward in equation based techniques where constraints can be placed arbitrarily. In sequential modular programs each unit must be fully specified since outputs are calculated from inputs. The use of control units to place specifications necessitates the "freeing" of one variable before another can be constrained, making it is easier to identify the effect of a single specification.

As discussed in Section 3.2, the main solution technique is equation based, so a check is required of a designer's specifications to ensure their feasibility as design variables. This is measured against several criteria by the following algorithm:

1. The number of degrees of freedom is calculated from the equation:
$$N_D = N_v - N_e \tag{3.2}$$
where N_D = the number of degrees of freedom, N_v = the number of variables in the system, and N_e = the number of equations in the model. If the number of specified design variables is greater than N_D, then the procedure stops.

2. The variables are then checked using a variation of the Lee, Christensen and Rudd algorithm [50] to ensure the specifications do not contradict each other. The full algorithm is presented in Appendix B.

3. If there is a contradiction, the offending variable is found and an alternative suggested.

4. If there are any remaining degrees of freedom, suitable design variables are found.

5. The system of equations is assessed to determine whether or not sequential solution is possible given the current set of specifications. This is determined by the Lee, Christensen and Rudd algorithm in 2. If sequential solution is not possible, the equations are checked to see if the selection of different design variables would allow sequential solution. In the simple case where one solver can only be used for such sequentially soluble problems, this approach provides the information necessary to choose the appropriate solution procedure.

This algorithm has been written in Prolog. The result is a compact implementation which bears a marked similarity to matrix reordering methods. However, Prolog is slow in operation and, as the method is mostly array manipulation, it is probably more appropriately written in C.

A degrees of freedom analysis can only check the correctness of the structure of the problem. It cannot detect whether values of variables will interact in a destructive manner or are not feasible. In sparse matrix solution such problems are common and result in non-convergence. They should, therefore, be identified by an analysis of sensitivity before solution commences.

3.4.2 Analytical Manipulation

The simplest solver implemented is used to manipulate the analytical expressions obtained from the model generation phase. The principle of the method is the rearrangement of equations to make a single unsolved variable the subject of the equation, whereupon it can be solved for directly. A system containing any equations which must be solved simultaneously, e.g. one containing a recycle, cannot be solved by this technique.

The set of equations is delivered to the solver unordered, so the first task is to locate an equation containing a single unsolved variable. A set of manipulating rules is applied to any such equation in an attempt to make the variable the subject of the equation. If successful, the variable is evaluated directly. If not, the equation can still be solved numerically, in this case by passing the equation to the one-dimensional Newton method described below. If non-linear equations can be rearranged resulting in one variable being the subject, then they too can be solved analytically.

The method has been written in Prolog because it facilitates the manipulation of symbolic expressions by way of pattern matching. A further advantage is due to the equations having been generated with Prolog variables for each term. When any variable is evaluated, Prolog's automatic unification instantiates the value in all expressions containing that variable. The procedure can then continue searching for equations recursively with the variables being updated automatically, until no more can be solved.

3.4.3 One-Dimensional Newton

The purpose of implementing a solver for single equations was for design situations when a request could be made for a particular property as opposed to the solution of a complete unit model. For instance, when evaluating a stream for distillation, the K-values might be requested. Instead of formulating the model for the large solvers, which for such simple calculations becomes time consuming, a simple equation solver is more appropriate. The manipulative solution approach, above, was used for this purpose, but many expressions require some iteration, e.g. bubble and dew points. This numerical solver was developed for such situations.

Problem equations are linearised using a symbolic differentiation module implemented in the full Newton's method (see Section 3.4.4 below). The expression is then formulated for a single-dimensional Newton's method iteration written in Prolog. Since only single equations are being evaluated, the solution is rapid enough that an implementation in C is not required. The technique suffers badly, however, if a reasonable initial estimate is not given.

3.4.4 Newton's Method

No general purpose equation solver was available at the beginning of this work so the requirement of an uncomplicated implementation for experimentation was met by the choice of Newton's method. The formulation of the problem involves three stages, all of which have been written in Prolog.

Equation Linearisation

Initially, the equations describing the problem of interest are linearised by taking the first term of the Taylor expansion and then rewriting them in the form required by the normal multi-variable Newton-Raphson iteration scheme.

The equations are differentiated symbolically using a set of 13 differentiation rules which cover most cases: trig, logs, powers, arithmetic functions, etc. The form of the linearised equations can be symbolically manipulated to provide an understandable written expression, but the likelihood of anyone wanting to see these equations is very small. The facility is available for this presentation, but the form used internally allows simplified manipulation by the matrix reordering algorithms.

Equation Ordering

The linearised equations are ordered using the P4 or SPK2 algorithms [68], producing an implicit bordered lower triangular matrix of coefficients. This phase is also performed in Prolog, but, as mentioned above, it may be more efficiently implemented in C.

Variable Initialisation

In Newton-based methods, initial values must be supplied for all variables. Apart from the design variables which must be set before a solution is possible, it is unlikely that the designer will want to supply starting values for the variables which the program is supposed to calculate. In problems containing thousands of variables this is clearly an infeasible request. It is reasonable to assume, therefore, that any equation based flowsheeting program should be able to provide all necessary initial values.

In a hierarchical design procedure variable initialisation is readily implemented, since the design is proceeding in stages of increasing complexity. At a given stage of the design, many of the variables can be initialised using values calculated in earlier simulations. If a single model is being simulated under different conditions e.g. different feed rates, then the previous solutions can be used to initialise the new ones.

This is achieved when the expanded relation representing the balance equations is passed to the solver. The specifications have been made and are, therefore, already unified throughout the model. The bindings associated with the equations are then checked for variables not already instantiated. Any such located terms are matched against the solution of the specific variable in the stored, solved form of the "perform" relation evaluated most recently. The model in which this search is performed, must be of the same relation class as the one being initialised.

It is possible for the model to have changed since earlier versions were evaluated. In this case, many of the terms will still be present, but some will not. The ones not present can then be matched with related variables in simulations performed at a level of less functional detail, where applicable. This is also the approach used for new simulations at levels of greater functional detail.

Certain key properties can be matched between levels, mostly belonging to equivalent streams. This is particularly useful for recycles which will require iteration to solve. All streams defined at a level of lesser detail have an equivalent at the level of greater detail (though the inverse does not necessarily hold). This correspondence is defined by the designer as the flowsheet is created.

This variable initialisation procedure may not be able to provide estimates for all variables, however, so some will still require generated values. In most cases, though, these variables can be calculated by a forward elimination pass which is made simpler, since the reordering step has already been performed. The only exceptions are the spikes or tear variables located by the reordering algorithm, which can be estimated based on other local values.

Two approaches have been studied for the estimation of the remaining variables. Both involve writing the unsolved equations as inequality constraints, e.g.

$$w_o = w_1 + w_2 \tag{3.3}$$

implies $w_o \geq w_1$
and $\quad w_o \geq w_2$
assuming all terms are greater than zero.

Unknown variables are estimated based on the order of magnitude of the other variables in the equation. The estimates are then sequentially substituted into the unsolved equations, generating values for all other unknowns. The generated constraints are then checked to ensure the estimates are within the feasible region. If they are not, estimates are provided from other equations and the procedure repeated.

The basic assumption of this method is that in an equation, expressions subjected to the same operator are of the same order of magnitude. For example, in Equation 3.3 above, if w_1 had a value of 100, it is assumed that w_2 is of the same order of magnitude i.e. 100. The equation can now be evaluated and, if only one estimate is required, so can the remaining equations. The constraints generated form the unsolved equations can then be checked. In practice, this assumption is too basic to be of practical use. The number of possible permutations of equations and specifications would require a very large number of special conditions to be assessed for even a small number of equations.

The second approach takes an initial estimate for an unknown variable and substitutes it into the unsolved equations. This provides values for the constraints which are then checked to see what, if any, change is required in the variables for a feasible starting point to be found. If no change is necessary then a feasible point has been found, otherwise the suggested change is implemented and the substitution repeated. Using the above example, the feasible point found is where: $w_o = 101, w1 = 100$ and $w_2 = 1$.

The final stage of formulation is the writing of two C routines using the ordered, initialised equations. The Newton's method iteration is a C program which requires routines to provide the functions for calculating the updated values of the variables' coefficients from previous iteration values. The other provides initial values of all design variables. The routines are written by Prolog calls.

Matrix Solution

The sparse system is solved in C using the CBS algorithm [69], a modification of Gaussian Elimination. The solution is placed in a vector accessible from both C and Prolog.

Figure 3.3 shows the sections implemented in Prolog and those in C.

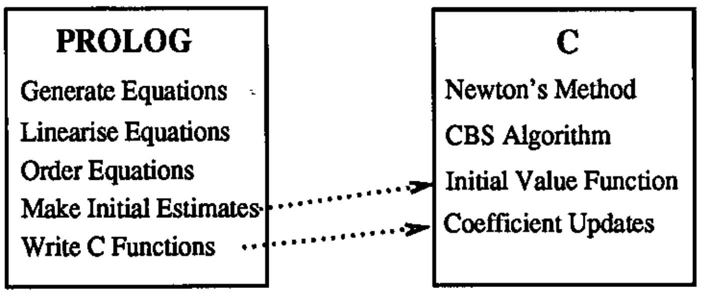


Figure 3.3: Implementation of Newton's method in C and Prolog

3.5 Conclusions Concerning Solution Strategies

Both equation based and sequential modular solution methods have been investigated to assess their strengths as part of a modelling tool. Implementations of both approaches have been achieved, requiring different levels of modification to existing program structures.

Flexible specification of problems requires an alternative to the unit model library used in flowsheeting programs. The concept of a library is central to the sequential modular approach, implying that the library should be retained. However, to provide a reasoned model, a decomposition of the process is required at the equation level. For both approaches, therefore, an equation generation module is the starting point in problem formulation.

The major differences between the implementations stem from this point. The equation based method requires only a mapping between the equation representations of the symbolic model and the solver. This applies to both large and small problems. The sequential modular solver, however, requires a reasoning module to deduce an appropriate unit model, or combination of models, corresponding to the generated equation description. For simulations of individual items of plant, more detailed models are required, as well as the ability to alter the models for particular situations. This suggests, certainly in the case of Esspros, the necessity of additional modelling capabilities, most appropriately achieved with an equation based method. In conclusion, the equation based method allows more flexible specification of problems and, consequently, more accurate translation into a mathematical description than is achievable with a sequential modular strategy.

The two approaches require a range of tools to support model formulation. Both require a method for checking the degrees of freedom, thus constraining the specifications to suit the particular solver. A matrix method has been implemented for the equation based approach (Section 3.4.1), while the sequential modular method requires a more rigid syntax to be imposed on the CLAP structures, restricting the range of specifications that could be made on particular unit operation models. A procedure is also necessary for the initialisation of variables by the equation based implementation, whereas the modular approach needs a tool for identification of optimal tear streams.

A tool of the type proposed should have different solution methods available for different applications. For instance, equation based or sequential modular solvers are well suited to solving models of entire plants, but for solving individual units or individual equations, small solvers are more appropriate. Programs are also available specifically for simulating one particular item of equipment. For this reason, two small solvers, intended as rapid calculators, have been used to augment the capabilities of the large scale flowsheet solvers.

A solution strategy has been implemented specifically for the equation based solver. The evaluation of the degrees of freedom (see Appendix B) indicates the solver which is most appropriate for the generated equation set. The method identifies irreducible blocks in the matrix which require simultaneous solution. For large problems this is most often the case. The full Newton's method is automatically invoked for such situations.

Where no irreducible blocks are present, for instance in some models of individual unit operations, the simpler solvers can be used to achieve an analytical solution where possible. The method involving symbolic manipulation is used for such cases. If this method identifies equations which cannot be solved analytically, the one-dimensional Newton is used.

The specification of a solution strategy could also be used where programs are available for the modelling of individual unit operations. Depending on the range of tools, the method would initially select programs on the basis of the type of unit being modelled. Subsequently, programs could be invoked when the specifications on the CLAP representation match the input requirements of a particular solver.

Chapter 4

An Object Oriented Representation for Flowsheeting

The knowledge required by a flowsheeting tool must include information about:

- a range of solution methods and their required formulations
- how to create a consistent mathematical model based on specified information
- use of physical property models
- the concepts of the design procedure i.e.
  - hierarchical synthesis
  - evaluation of alternatives
  - integrated process evaluation

Mathematical methods for solving flowsheets have reached a high level of sophistication and new algorithms are not necessary. Different formulations are required for different solution techniques, but all are merely different configurations of a single problem. It is, however, desirable to demonstrate that the representation of the process and its specifications can be manipulated to provide valid formulations for different solution methods, as described in Chapter 3. This chapter addresses the representation of processes in a central model to allow consistent formulation for different evaluations.

Physical property models are also well established with many databases and packages either developed by individual companies or commercially available, e.g.

PPDS [70]. The constants and parameters for the models should be available to the proposed system. However, representation of the associated mathematical relationships must be consistent with equations used for describing unit operation models. The format for defining mathematical expressions is discussed in Chapter 5.

A model based approach has been proposed using object oriented programming (OOP) to represent the process and support the concepts of design discussed above.

In the creation of any knowledge based tool there are five areas for consideration which should be concurrently advanced to maturity (see Kunz [71])

- purposes
- representation
- reasoning
- interfacing
- testing

Kunz advocates an opportunistic control scheme for selecting the topic which should be considered at any point in time. As soon as an issue gives the developer cause for concern, it should be addressed. Periodically, a review of the points covered gives an indication of the areas neglected and thus overdue for consideration.

The five areas indicated are of equal importance and while it cannot be demonstrated that the concurrent-opportunistic control method was used throughout this work, the importance of the five topics will be indicated with respect to the development of a useful design tool.

Purposes. The aim of this project was the development of an experimental flowsheeting tool suitable for use in calculations throughout the course of a chemical engineering design. Such support for a designer involves the creation of mathematical models of plant items at wide ranging levels of detail and with different foci.

Representation. The object oriented language, CLAP [45], was chosen for the representation of the problem because of its ability to represent not only structural items but also the relationships between them. CLAP is a procedural language, which has access to Prolog logic programming and C for rapid numerical manipulation and evaluation. CLAP is itself written in Prolog.

Flowsheeting deals with items which can be separated into distinct classes. The classes include the following:

- flowsheets
- unit operations
- streams
- chemical species
- modelling equations.

CLAP objects have been used to represent all of the above classes except modelling equations which have been represented in user defined relations and are discussed in Chapter 5.

Since the tool is intended to be used throughout design, the stages of a design must also be represented. Process alternatives must be described and the relationships between them, e.g. which are developments or refinements of others.

Reasoning. The reasoning must take the place of the engineer for the task of creating a mathematical model and formulating it for a particular solver or other external program. This requires, therefore, a model of the solver (for equation based simulation) or the flowsheeting program (where the solver is sequential modular).

Interface. The interface to the designer is normally an addition to the program tacked on at the end. However, for a final product to be of use to an engineer, constant consideration must be given to the aspects of the design, such as decisions, calculations and tools appropriate to a given situation, which a designer will wish to see. The limitations of the available software and the disproportionate amount of time required for the development of a graphical interface have meant that while interfacing has been considered throughout, it has not always been possible to implement a display exactly as envisaged.

Testing. Examples for testing the various proposed ideas always bring to light different aspects of a problem. Therefore, many small examples and some larger examples have been considered during development. Some examples appear in the text to illustrate points more clearly, while more detailed tests appear in appendices D, F and E.

4.1 Unit Operations

Objects have been used to represent a range of unit operations for a flowsheeting tool which is to be used at different levels of process design. These objects range from conceptual processes representing whole plants to detailed unit models, related using an inheritance mechanism as described in Section 2.3 and shown in Figure 2.7.

Slots within the objects have been defined for properties useful in flowsheeting and synthesis procedures. Further attributes may be necessary for the implementation of other evaluation tools. Inheritance provides a means of collating common attributes of related units, e.g. a heater and cooler may both require slots for heat load and the utility source, so they are contained within a more abstract object describing enthalpy change. In this context heaters and coolers imply instances where the requirement for heat exchange has been identified but insufficient information is available to specify the source or sink of heat. Since objects such as heaters and coolers are ostensibly identical, it could be argued that there is only a requirement for one object representing the two operations. However, the two operations are conceptually opposite functions, so using one object reduces the amount of reasoning possible about the function of the object. For this reason, both objects are retained in the inheritance hierarchy.

Only a subset of the range of operations required in a chemical plant has been represented. Many units are similar in character, varying only in small details, so creating a unique object for each one is an impractical task. Apart from the development work required to do this, a system supporting a large number of flowsheet items becomes unwieldy to use.

One possible solution to this problem is to use multiple inheritance which allows objects to be defined as belonging to more than one class and thus able to inherit the properties of more than one object. For example, a flash unit incorporating external heating could be defined as being a combination of a flash and a heat exchanger. However, this approach presents difficulties where a combined unit may not need all the properties available from the constituent operations, which indeed may contradict each other. The structured classification of the objects is also disrupted, reducing the reasoning possible based on a tree as described in Section 2.3.2.

Multiple inheritance is not used in this system. Instead, objects are described with sufficient flexibility to represent a range of flowsheeting applications. Providing a mathematical model for units involves reasoning about the specified properties of the object and constructing a model based on the specifications. Using the above example of a heated flash, the appropriate object describes a generic flash operation but incorporates slots for heat load and other heating properties. Thus the flash object is still only a specialisation of the object representing vapour-liquid equilibrium separations. If no heating properties are required by the designer, then these slots will not have values. When the mathematical model is constructed the slots are checked, and if they have values, then a model incorporating external heating will be constructed, otherwise only equations describing the flash operation are prescribed. This approach removes the mathematical description from the object and thus overcomes the problem of a rigid format of model described in Chapter 3. The models constructed here are based on the information provided both functionally, in the selection of the unit operation, and structurally according to the information provided about the unit.

The most difficult task in representing a problem using OOP is to decide to what level operations are to be broken down, and to which objects given information belongs. This is demonstrated by the different characterisations developed by other workers in this field, e.g. objects in DESIGN-KIT [14] differ from those developed here. The representation adopted here is not definitive, but performs adequately for use in flowsheeting and simple synthesis. However, the implementation of other reasoning modules may require revision of the inheritance tree.

Objects are used to represent streams as a separate entity for several reasons. In the course of a design, it is more natural to refer to streams and properties of streams than unit operations. Unit operations can be thought of merely as mathematical functions mapping a set of inlet stream properties to a set of outlet stream properties.

Secondly, without the use of stream objects, each unit operation would have to incorporate slots for the properties of all inlet streams and all outlet streams. The incorporation of a variable number of streams would create problems for identifying their individual properties.

Finally, stream objects provide a better functional description of concepts such as temperature than unit operation objects. Within a unit operation object there may be several slots to hold different temperatures, which makes it difficult to recognise each one as belonging to the single concept of temperature. If, however, temperatures are stored in separate streams, labelling individual properties becomes straightforward.

Stream objects are used to represent connections between flowsheet objects, and contain the properties common to the connected units, e.g. temperature out of one object is assumed to be the same as the temperature into the next. The connection is indicated in the objects by setting a slot to contain the relevant stream number (Figure 4.1). The objects representing unit operations access the stream object when they require values associated with their connecting streams.

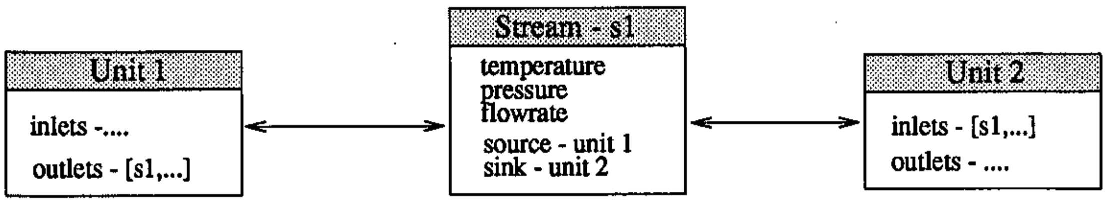


Figure 4.1: Representation of Streams as Objects

4.2 Chemical Species

In general, chemical species might be represented by objects in two different ways:

1 One generic object, "chemical_species" is defined, containing slots for physical property parameters which can be used to generate appropriate physical property equations as required. All components used in a design are single instances of the generic object. For example, a stream object could be created which contains components benzene and toluene. Objects are created for benzene and toluene containing their physical property parameters. These objects could then be referenced by any other stream in the design.

2 Each chemical component is represented by a generic object which, apart from containing physical property parameters, contains slots for properties

such as temperature and pressure which could be set in local instances. In the above example, each component in each stream would be represented by an instance of the generic object for the particular component used.

Both representations have advantages. The first representation is the more natural expression of the concept of streams and components. In this description, a stream has a temperature, pressure, flowrate and constituent components as opposed to each component having a temperature and pressure. However, some calculations require a different emphasis. If, for example, it is required to calculate the relative volatilities of two components, it is normal to compare vapour pressures which are calculated based on temperature. In that case, the second representation has the advantage of being able to compare two components directly, with local temperatures and pressures contained in their own slots.

The representation used is the natural description, i.e. streams have temperatures and pressures, and instances of component objects contain only physical property parameters. The relative volatility example described above is accommodated by creating an instance of a stream which may (or may not) be temporary. The components are then associated with the stream, and, therefore, with local values of temperature and pressure. The problem of referencing different objects for temperatures and pressures is discussed in Section 5.1. The parameters should be available in a physical property database, but such a facility has not been incorporated. For the sake of testing programs, values have been provided manually.

The system includes a method of determining physical property data for substances not detailed in a database [72]. The technique involves estimating critical properties from group contributions. The calculated values are adjusted in the light of a comparison with the results of known compounds recognised as being similar to the compound of interest.

4.3 Object Representations for the Design Process

To provide a practical tool for use throughout design, whether it be a design environment or a supporting tool, a representation must be provided describing the steps taken. This should include the generation and evaluation of alternative processes. Apart from providing consistent support for the development of alternatives, the knowledge incorporated in such a characterisation reflects the decisions made, from the selection between process alternatives to the placing of specifications on the plant operation. Such information is useful for reviewing the design, especially if the reviewer had no part in the project itself, but wishes to determine the reasoning behind the completed design.

Lien et al [9] describe an example of preliminary design illustrating the flexibility that will be demanded of such a representation. The designer continually moves from levels of greater abstraction to more detailed levels and back again, developing models for different evaluations which supply information necessary to make synthesis decisions. The designer takes advantage of implicit relationships between levels while modelling and making decisions. For example, the choice between a liquid phase and vapour phase reactor is deferred at a high level until the associated separation is modelled in sufficient detail to indicate which is more economical. The modelling involved is also subject to levels of detail. For instance, a simple model of the separations may be sufficient to select between reaction schemes. It may be necessary, however, to model the same separations at a greater level of detail to provide enough information.

The discussion of expert system approaches to design in the work by Lien et al highlights the key representational issues, but does not, however, discuss implementation:

- the description should be hierarchical, incorporating levels of increasing detail as design proceeds,
- alternative process structures should be accommodated and maintained until the optimal structure can be determined,
- movement should be possible between levels of detail, taking advantage of the relationships between structural abstractions and refinements,
- data should be shared between levels.

Two further important points arise from this discussion. Firstly, the authors have difficulty in separating the synthesis task from representational issues. The representation must be able to support synthesis tools, whether they be algorithmic or heuristic, as any other tool would be supported. However, the generic objects used to describe the steps taken during a design are independent of any particular tool. The application of tools will ultimately shape the use of specific instances, rather than their generic classes.

The second point is that at any level of detail, there is a range of models describing the items depicted at the chosen level at various degrees of complexity, e.g. a distillation column can be modelled by an split fraction balance, a Fenske-Gilliland-Underwood model or a plate model (which, in turn, may incorporate different levels of detail). The abstraction of a process description into models of differing levels of complexity is discussed in Section 5.4.

The following sections describe in detail different representational models of a hierarchical design with a discussion of the model implemented.

4.3.1 Hierarchical Representation of Design

A common representational medium for hierarchical design is the concept of design states as nodes in an undirected graph. One view is to consider a graph of structural refinements, i.e. explicitly representing a relationship such as "has_parts" (and its inverse "is_part_of") as edges connecting nodes describing design states, see Figure 4.2

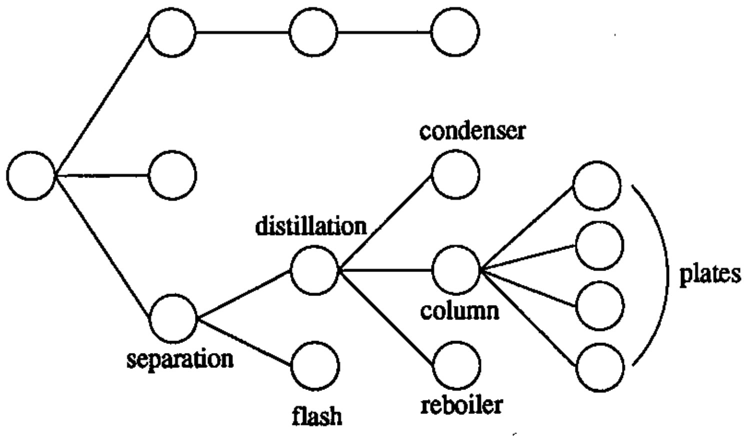


Figure 4.2: Graph of Design States Related by Refinement

In this example, the separation operation has been refined as a flash and a distillation operation. The distillation operation has been refined to the constituent parts of the column.

This graph adequately describes process disaggregation if the design does not involve any process alternatives. The expansion of the graph in Figure 4.2

to accommodate the alternative structures which the designer might wish to explore requires a third dimension to be added. For example, if the separation could equally be carried out by two distillation columns, the graph would appear as in Figure 4.3

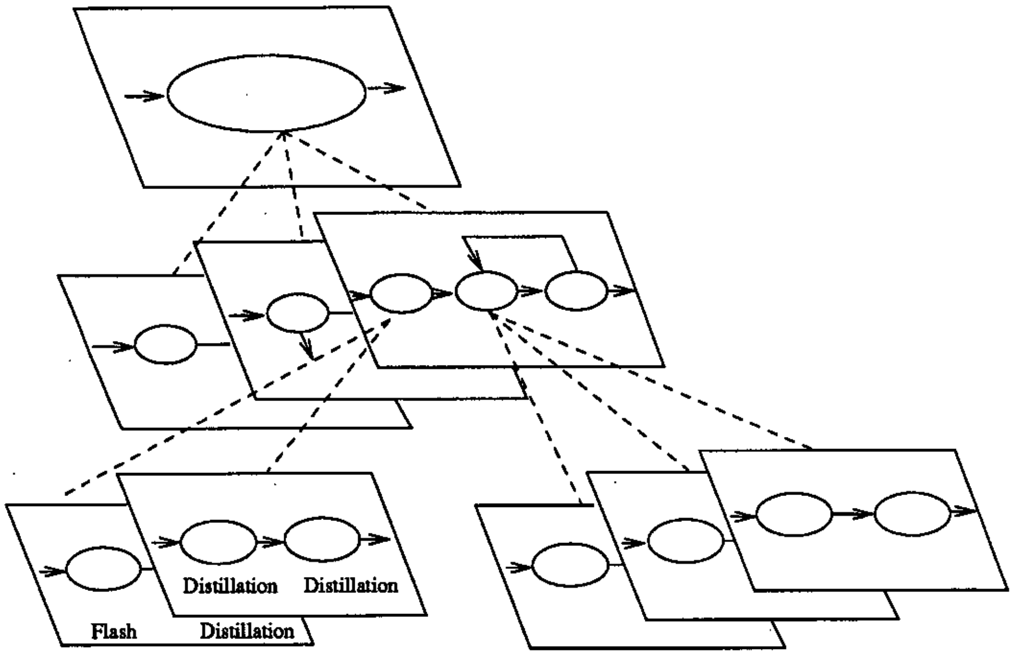


Figure 4.3: Graph of Design States Representing Structural Alternatives

The graph now represents the notion of items being refined to more detailed parts and accommodates the idea of alternatives being exclusive. The representational issue not incorporated is the concept of a flowsheet. Using the representation suggested by Figure 4.3, flowsheets can be constructed from the nodes of the graph by selecting between the alternative refinements of a given node. This allows the flexibility of being able to construct models incorporating a wide range of complexities, e.g. modelling a reaction section together with detailed distillation models. However, the onus is on the designer to define a model every time one is required. In its current form, the graph can be used to construct models of any combination of structurally related parts of a design. The extensive flexibility of this representation is discussed by Bañares-Alcántara [73] who emphasises its clean semantics as being a benefit for the designer's understanding of the development of the design.

A potential source of inconsistency arises within this representation in cases where certain combinations of parts are invalid. In a simple example, a process may be developed with the intention of comparing two reaction schemes, one being liquid phase, the other vapour. The reactions may each require specific separations. Figure 4.4 shows the graph as represented by relationships of refinement and constituent parts. Potentially, each reaction scheme could be combined with each of the separation schemes which were intended to be exclusive. If the alternatives are subsequently refined, the confusion is compounded as to which structural combinations are intended to be valid. The exclusivity is difficult to represent, but Bañares-Alcántara argues that there may be some benefit to be gained from being able to model such combinations and regards this as an area of future research.

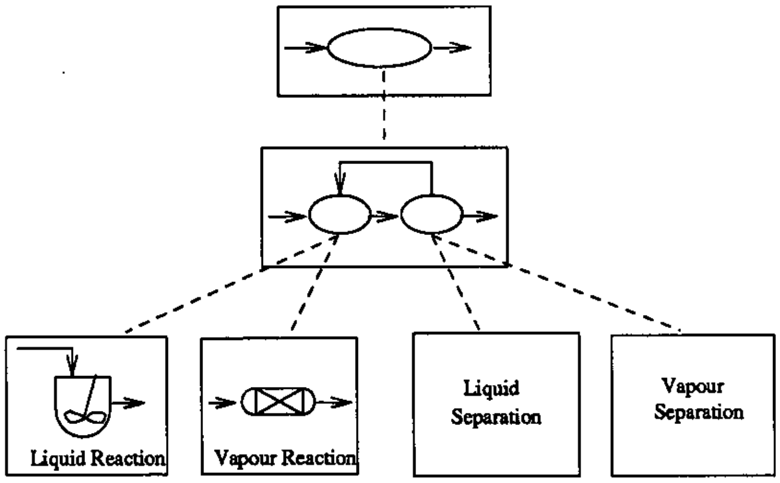


Figure 4.4: Graph Showing Potential Combination of Inconsistent Parts

The implemented representation incorporates relationships describing the notions of refinement and constituent parts. To overcome the inconsistency of alternative refinements, the concept of the flowsheet has been used to constrain the potential explosion of combinations. The refinement relationship now refers to flowsheets resulting in a hierarchical development of whole processes. It is, however, necessary to be able to design the constituent parts of a flowsheet separately, so it must still be possible to support the modelling of incomplete flowsheets. The use of the flowsheet to constrain the number of potential combinations reduces the exploration of the problem space to considering valid combinations of units. The hierarchy of flowsheets also provides a record of decisions, such as the combination of a vapour phase reactor with a vapour phase separation as opposed to a liquid separation.

This is similar in concept to the notion of "cast in stone" described by Westerberg et al [1] and Subrahmanian et al [2] with respect to the sharing of information in a group design activity. In the environment proposed by Westerberg et al, the authors discuss the large amount of information shared by the members of a design project team. While each member maintains their own models and data, with the ability to delete it or alter it at will, any shared information must remain unaltered, hence "cast in stone". Similarly, in the development of flowsheets, many structural alternatives may be modelled, but the maintenance of a hierarchy retains the final proposed structures. The information associated with the nodes in the hierarchy includes modelling results, relationships reflecting topological components and relationships describing refinement from previous levels.

Figure 4.5 shows the implemented construction of nodes in a design graph. The nodes represent flowsheets related by "is_refined_to" from high levels of abstraction to levels of greater detail. The inverse relationship is "refinement_of". Between levels of abstraction the relationship of "has_parts" and its inverse "is_part_of" indicate which process items in levels of greater detail correspond to which more abstract processes, e.g. the liquid phase reactor is a part of the reaction section. This relationship is dependent on the current viewpoint of the design. Since alternative structures are independent, the designer may only consider one at a time. Therefore, when the design is viewed from node (A), the liquid phase reactor is considered as a part of the reaction section, and when viewed from node (B), it is the vapour phase reactor.

4.3.2 Implementation of the Design Development Graph

A design graph as described above has been implemented in the MODEL.LA project [19] using the Context tree structure available in KEE. The resulting graph differs from the implementation achieved in CLAP only in the support provided for the maintenance of the graph. A discussion of the differences is in Section 4.3.3.

Contexts in CLAP are not related to each other at all, unlike KEE and Knowledge Craft. For this reason, objects have been defined that are, conceptually, intended to perform the same function. The simplified CLAP description of a generic design node object is shown in Figure 4.6. The objects, called

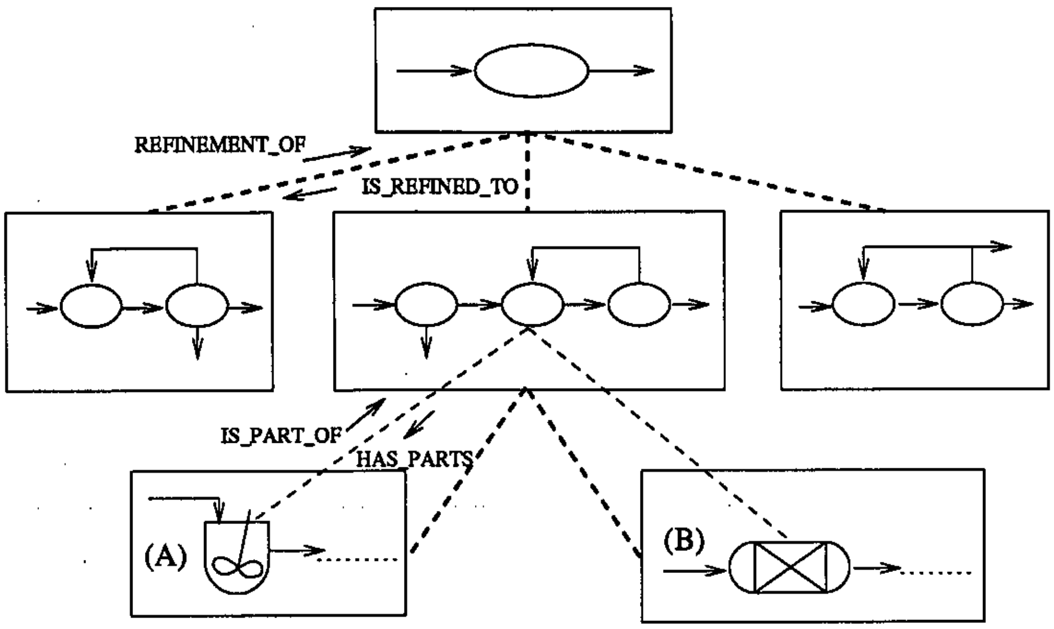


Figure 4.5: Graph Representing Hierarchical Development of Process Flowsheets

design_nodes, have relations containing information describing the position of the node in the graph, i.e. what it is a refinement of and what its refinements are, similar to the construction of a Context tree in KEE or Knowledge Craft. In instances of design nodes, the refinement_of relation contains a single term, the name of the only node of which it is a refinement. The is_refined_to relation may contain a list detailing the alternative refinements of the current node.

It should be stressed that the design nodes in this implementation are intended to be used for structural alternatives. Any numerical alternatives are achieved within the design node, i.e. one node is used to contain all models related to the same process structure. For instance, a model can be created with a flowrate specified as 10 kg/s. Subsequently the specification may be altered to be 15 kg/s. This does not require a separate design node. The models are stored as relations, either to the node itself or to an item within the node, thus they can each be reviewed within the common context of the flowsheet. The storing of models as relations is discussed in Section 5.1.

The design node slots are used to describe the topology of the node. The objects slot contains a list of the process items contained within the node, and the streams slot identifies all streams. These slots effectively restrict the use of the constituent instances of process items and streams to the one node, i.e. the

```
object - design_node :-
    self - _,
    variables - [ ... ],
    slots - [objects - Objects,
             streams - Streams,
             equations - Equations],
    relations - [refinement_of - Ref,
                 is_refined_to - Rfs].
```

Figure 4.6: Generic Design Node Object Used to Construct Design Graphs

designer can only access these objects when viewing the design at this node.

The concept of parts is maintained in the process unit objects as a relation between the parent unit and its list of parts. For example, an object representing separation has a relation containing the name of the object in the parent node of which it is a part. In this example this could be a specific object representing a plant. It also contains a relation which contains a list of parts in the child node. The relation is updated as new objects are added to the child node. Here, this might include distillation columns and auxiliary equipment.

This information is also reflected in the `objects` slots of the nodes. Each process in the list is paired with its abstraction in the node above the current one. For example, in Figure 4.5, the liquid phase reactor is paired with the reaction section. This implementation reduces the amount of computation required to establish complex relationships from more fundamental semantics.

The `equations` slot has been included to facilitate the mathematical modelling of design nodes, which corresponds to the modelling of flowsheets. It is a relatively simple task to construct such models since the object also contains all topology information.

The relationships linking the design nodes and their constituent parts define the structure of the design graph. This provides a high level of functionality which should encompass all operations that could be required in the development of a design based on the graph structure.

A generic object, called "design", has been defined to aid in the management of the problem hierarchy and movement within it. The object is shown in Figure 4.7. The object represents the designer's current viewpoint of the design. Its slots store the current design node (`where`) and the section, if any, within the flowsheet (section). By changing the value of the where slot, the refinement relationship edges in the design graph (Figure 4.5) are traversed. Similarly, altering the value of the section slot locates the design object within the graph of part relationships.

```
object - design :-
    self - _,
    variables - [ ... ],
    slots - [where - W,
             section - S,
             levels - L],
    relations - [technology - Tech].
```

Figure 4.7: Generic Design Object

In addition to the two objects, extended methods have been written to provide a menu of options to develop the topology of the graph and flowsheets within the nodes. These provide the ability to:

- Refine nodes, i.e. create new nodes related to the current node by refinement relations,
- Move up and down the hierarchy of nodes via the "refinement" relations and using the "parts" relations to define the section of interest. The design object is used to store the current position.
- Create objects in nodes for process items, and relate them to abstractions in higher nodes ("part_of" relations).
- Specify connections by creating stream objects, and relating them to the equivalent stream in the higher node if necessary. Only some streams are related to higher nodes. These are identified as streams which cross the boundaries of sections defined at the higher level of abstraction, e.g. at the higher level, objects may be specified for a reaction section and a separation section. In the node at the level below, streams connecting distillation columns are within the boundaries of the separation section and are, therefore, not related to any streams in higher nodes. The stream connecting the reactor to the first distillation column, however, crosses the boundaries

of the reaction section and separation section and is, therefore, related to the stream connecting the two at the higher level.

- Delete a process object, removing all objects related to it by "part_of". This includes all objects which are part of the parts. Any streams connected to the objects deleted must also be removed. Since this action is so drastic, a warning is given before commencing. However, if the node has been refined to a further structurally complete node, the delete operation will do nothing. The decisions incorporated in the graph resulting in the items exclusion from further consideration are important when reviewing the design. This information describes why this particular combination of items was rejected.

- Display the processes and streams in a given node or section of a node.

- Copy the generic description of an equivalent section to a separate node. In situations where several alternatives have been created at the same level, it might be desirable to copy the particular refinement of one section to another node. This is only a copy of the generic objects and their connections. Any numerical slot values are ignored since, even though the scheme is the same, the instance must be considered as entirely separate. For example, two reaction schemes may be developed and two possible separation schemes, each of which is compatible with either of the reaction schemes. It is, therefore, desirable to be able to copy the developed structures into separate nodes for subsequent combination and evaluation (see Figure 4.8).

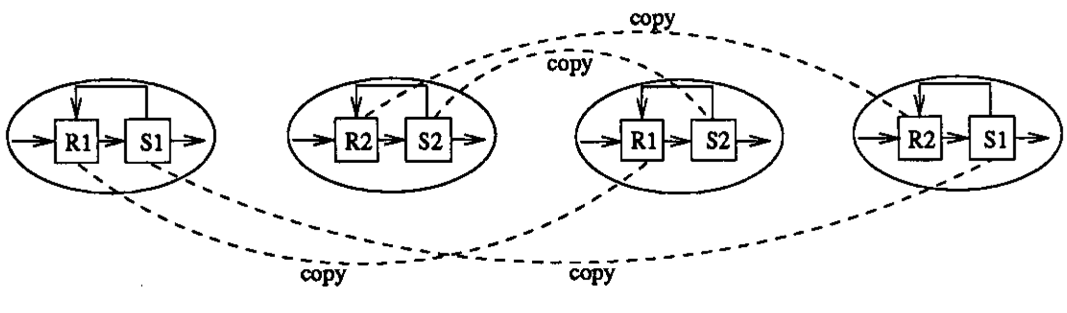


Figure 4.8: Illustration of the Desirability of Copying Sections

- Move between sections in a node by identifying the objects related by the higher level abstraction.

Sections are defined by the contents of all "ancestor" design nodes i.e. the parent node and all its precursors. For example, Figure 4.9 shows a node containing a model of a distillation column described by a set of plate objects.

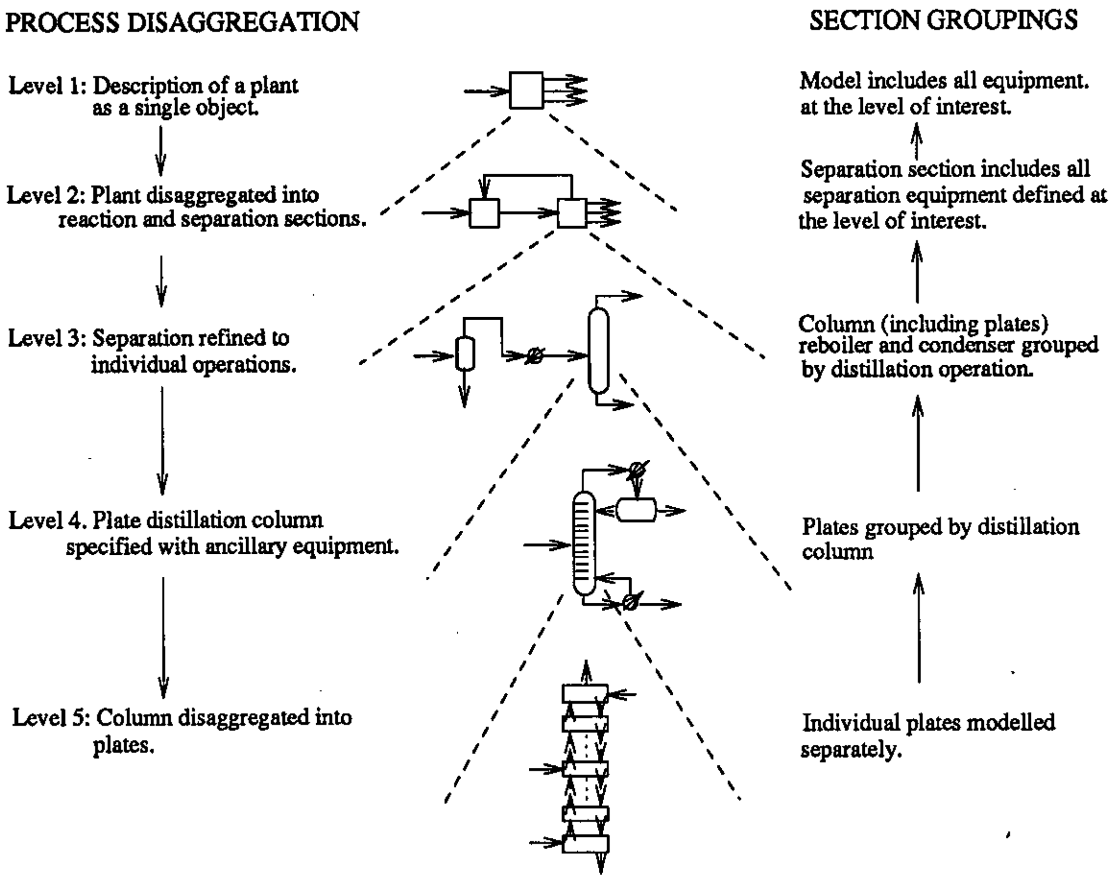

PROCESS DISAGGREGATION
SECTION GROUPINGS
Level 1: Description of a plant as a single object.
Level 2: Plant disaggregated into reaction and separation sections.
Level 3: Separation refined to individual operations.
Level 4. Plate distillation column specified with ancillary equipment.
Level 5: Column disaggregated into plates.
Model includes all equipment. at the level of interest.
Separation section includes all separation equipment defined at the level of interest.
Column (including plates) reboiler and condenser grouped by distillation operation.
Plates grouped by distillation column
Individual plates modelled separately.

Figure 4.9: Illustration of the Different Sections Containing Distillation Trays

The flowsheet in that node can have sections defined at several levels. At the lowest level, the individual plates could be modelled separately. The set of plates may then be modelled together if, as shown in Figure 4.9, the plates are defined as "parts of" a distillation column object in the parent node. The parent node of that node may include condensers and reboilers as part of a distillation operation, thus indicating a broader section. At a higher node still, a separation section may have been defined which includes other separations, pumps, etc. The limit of section encapsulation would be at the root design node which contains an object representing the whole plant. The specification of this as the section boundary would include all objects in the flowsheet. In this example, the most detailed node could have sections defined to contain: all plates in a distillation column, all distillation equipment, all separation equipment, and finally all plant equipment.

To summarise, the representational issues identified by Lien et al and discussed in Section 4.3 have been accommodated in the implementation discussed above. The concept of a hierarchical development with increasing detail as the design proceeds is encapsulated within the definition of the design graph created in CLAP. The graph also supports the separate development of structural alternatives which are, however, related to previous levels of abstraction. Methods have been defined in CLAP to allow ready movement through the graph, up and down and between alternative structures, by utilising the relationships defined between nodes and the processes within the nodes. The notion of sharing data between levels has been introduced with the idea of relating corresponding streams. A more complete discussion of sharing data between levels of abstraction is given in Section 4.3.3.

4.3.3 Consistency Maintenance

While it is desirable to have a tool for flexible specification of processes and their mathematical descriptions, some constraints must be applied to ensure the definition of the process remains consistent as it is developed. Since a graph of flowsheet refinements and alternatives is being supported, information from the abstract levels of detail can be used to constrain the more detailed development of the process to conform to the original conception of its "purpose". Thus if any future refinement contradicts the initial aims, this can be viewed either as a failed design, or a valid alternative to the high level specification, i.e. an alternative set of aims has been identified.

A mechanism for consistency maintenance has been constructed based on the concept of the purpose of processes. Any refinement of a process, i.e. to its parts, is constrained by the purpose of the abstract definition. Obviously, being more abstract, this constraint is not a great imposition on the flexible specification of the refinements, but a guide for development. For example, the high level purpose of a plant may be the formation of a particular product by the reaction of specified feed components. The purpose would be constrained further by placing a requirement on throughput and product quality. This constrains subsequent refinement of the node to incorporating a reaction mechanism and meeting the product specification by including a separation operation.

Consistency of the refinements is checked at two levels: functional and structural. Functional specifications are communicated by comparing the classes of objects defined at more abstract levels with their refinements at more detailed levels. Any differences identified in the purposes of the two levels are relayed to the designer who must decide if the contradiction discounts the newly stated design, or if it should be implemented as an alternative design node at a higher level.

A process is evaluated by locating the separate sections, represented by individual objects, in the parent node, e.g. a reaction section and a separation section. The associated parts of each are located in the new, refined node using the "part_of" relation. For example, the separation section may have pumps, heat exchangers and a distillation column as its parts. The identification of the conceptual relationship between the distillation column and the separation is achieved by reference to the inheritance hierarchy, i.e. a distillation column is a specialisation of a separation. Any number of valid separation devices may be included in the refined section, but at least one must be present to ensure a separation is being performed.

Functional properties are checked by a CLAP method when the designer specifies the addition of a child node from the current one.

The second level of checking is made on the structural specifications of the parent node. This ensures the specifications placed on the more abstract definitions of the design are still being met by the more detailed ones. This is achieved with the use of demons on the slot values. If a contradiction is found, the user is informed, again with the option of changing the current model or creating an alternative at a higher level. Either way, the designer is made aware of a deviation from the project aims.

The structural check is based on the streams which have been defined at the two levels. When streams are created at a level of increased detail, the context of the new stream is compared with the previous level. If the new stream corresponds to a connection at the earlier level, then the user is asked to confirm it. As an example, the initial description of a plant may be a single unit with two inlets and three outlets (see Figure 4.10). The next level might consist of reaction

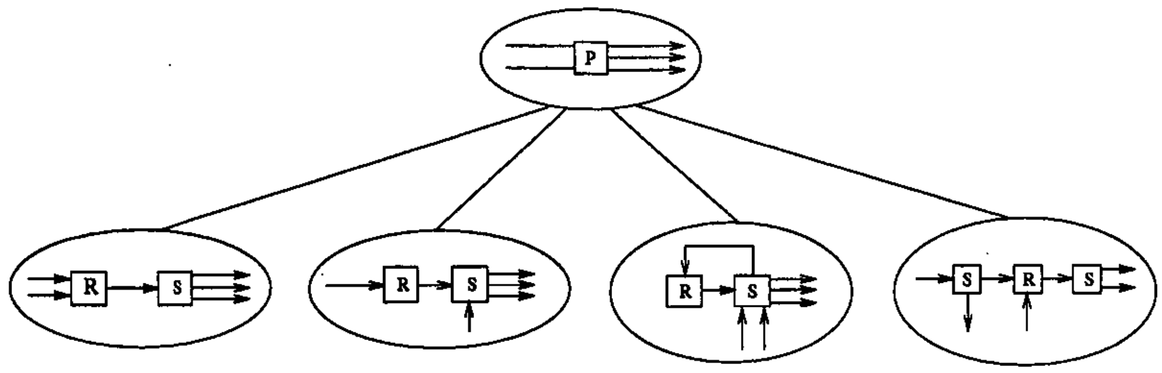


Figure 4.10: Example of Possible Alternative Refinements of a Plant

and separation sections. The feeds to the plant may be in several different places, e.g. both to the reactor, both to the separation, or one to each. In whatever way this is achieved, there must be two feeds. The two correspond to the two feeds specified for the plant, which enables the specifications placed at that level to be communicated to the new level. The same procedure is performed for the outlets.

The reason for requiring new alternatives to be created at a high level of abstraction rather than communicating changes back up the tree, is to maintain consistency for all alternatives developed. All of the possible structures are based on the original specifications, both functionally and structurally. If one refinement requires the alteration of these specifications, then the alternatives to that structure, e.g. the different feed situations illustrated in Figure 4.10, may be in contradiction to them. These structures may be entirely valid under the old constraints which should, therefore, be rigidly maintained. Thus the new structure requiring the alteration of the original (or more accurately, any previous) constraints should be accommodated in a new branch of the design graph.

MODEL.LA [29] approaches consistency in a slightly different way. As discussed in Section 5.4.1, the generation of models is based most appropriately on the assumptions made in the specification of the problem. For mathematical descriptions, MODEL.LA requires the user to state the physical and chemical phenomena occurring in a particular unit, which implicitly (and in some cases explicitly) identifies the associated assumptions.

A similar technique is used for maintaining consistency in the development of a model hierarchy, which is stored in a KEE Context tree. Three areas are considered for compatibility between levels: topology, structure and the constituent physical and chemical phenomena. The assumptions upon which compatibility is determined are implicit, in this case, contained in the description of the design at a particular level.

"Structural compatibility" is established by using relationships of the "has_parts" type. As in the CLAP implementation, the user identifies which processes in a new Context constitute a disaggregation of which processes in the parent Context. This provides a frame of reference for further compatibility checks and also for multi-level modelling, i.e. using detailed models as parts of more abstract flowsheets.

Checking "phenomena compatibility" is equivalent to checking a process's purpose, as described above. The inference determining the correspondence between levels is performed automatically, as in CLAP.

"Topological compatibility" is equivalent to the assessment of structure described in the CLAP implementation above. The streams are matched between levels; every stream in an abstract level having an equivalent in every one of its child Contexts. As in CLAP, when one to one associations are identified, they are related automatically. In situations where items have more than one inlet or outlet and the correspondence with the child process is ambiguous, the user is required to distinguish the links. The difference between MODEL.LA and CLAP is demonstrated when child Contexts have more streams than the parent in association with a particular unit. CLAP resolves this by requiring the addition of a new node at a higher point in the design graph. MODEL.LA accommodates the inconsistency by providing a "one-to-a-set" mapping between the single stream in the parent Context and the set of streams in the child Context (see Figure 4.11). In the example, the stream A has been mapped to the set of streams, C and D.

The intentions of consistency checking in MODEL.LA are the same as in CLAP. Both attempt to ensure consistency of design development and to provide a structure for inheriting information between levels. Propagating information *via* one-to-one mappings is quite straightforward. For instance, streams can be copied to provide default data in a new Context. Any process unit involved in a one-to-one mapping, however, must be a more specialised form, or a copy, of the object in the parent Context. The values contained in the parent object can

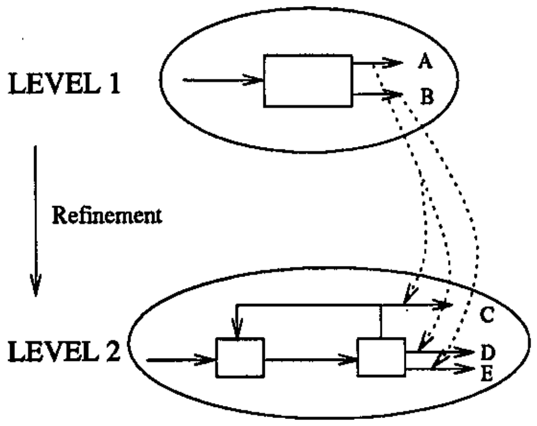


Figure 4.11: Illustration of a One to a Set Mapping for Streams

then be supplied as defaults to the child object in a manner similar to streams.

Since MODEL.LA supports one-to-a-set links, these can also be used by child Contexts to inherit information from parents. To do this, however, the user is required to define mapping functions, relating all relevant variables in the individual parent object with the appropriate variables in the set of child objects in each specific Context. In the above example, a mapping function could be used to relate the flowrate of stream A to the sum of the flowrates of C and D. This is implemented by using constructs in the modelling language, and may constitute a more natural approach to design development than is possible in the CLAP implementation.

It is also possible for information to be inherited from child Contexts to parent Contexts. This has not been implemented in CLAP because values inherited from a model of a specific child Context are not necessarily consistent with values calculated in other child Contexts. The view of the design at the parent node would, therefore, be incompatible with all but one of the child Contexts.

For similar reasons, a distinction must be made between specified and calculated values. Any model created at a parent node is based on assumptions made at its associated level of detail. Any child nodes are refinements, both in a functional sense, and in terms of the assumptions. Therefore, any calculated values are dependent on the model generated and hence the assumptions made. Since there is a refinement between nodes, calculated values will be incompatible between levels and, therefore, should not be inherited. The exception to this rule is for variable initialisation for the equation solver (see Section 3.4.4). This is permissible since the initial values are almost certain to change in the course of iteration.

To ensure the conceptual division between specified and calculated values is maintained, the two groups are stored separately. In order to facilitate access to specifications, these are stored directly as slot values in the appropriate objects. Calculated values are stored in "operator" relations after solution. This also allows specifications to be changed and new solutions generated, each solution being stored separately.

In conclusion, consistency is maintained by ensuring the structure and function of new design nodes remains compatible with specifications made at the parent node. Any inconsistencies imply the creation of a new alternative node at a more abstract level. The consistency of numerical values contained in generated models is ensured by storing specifications and calculated values separately. This allows inheritance of specifications, but not calculated values, except in the instance of variable initialisation for equation based solution.

4.3.4 Complexity Maintenance

As a process design is developed, the design graph will support several levels of increasing complexity and, potentially, a large number of alternatives. If every possible variation is represented by a separate node in the graph, it will be very unwieldy to use, some nodes being distinguished in some instances only by a small detail, e.g. the feed to a distillation column being to different plates. Aside from ease of use, the amount of computer memory required to store such a graph will be very large. Although computer memory is relatively inexpensive (£60 per Mbyte), it is desirable to provide tools in order to reduce the size of the graph. The computing power required to run such a system will not only be more reasonable, but the resulting design graph will be more understandable for the user.

Douglas [6] proposes a hierarchy of decision levels for proceeding with a design. The decisions have been outlined in Section 1.2.2. At each level in the hierarchy Douglas identifies possible structural synthesis steps. Figure 4.12 shows the structural possibilities developed at the input-output structure level based on the decisions described by Douglas. If each of the alternatives was completely evaluated the resulting design graph would be impractically large. The simple example in the figure results in 4! alternatives at the first synthesis step.

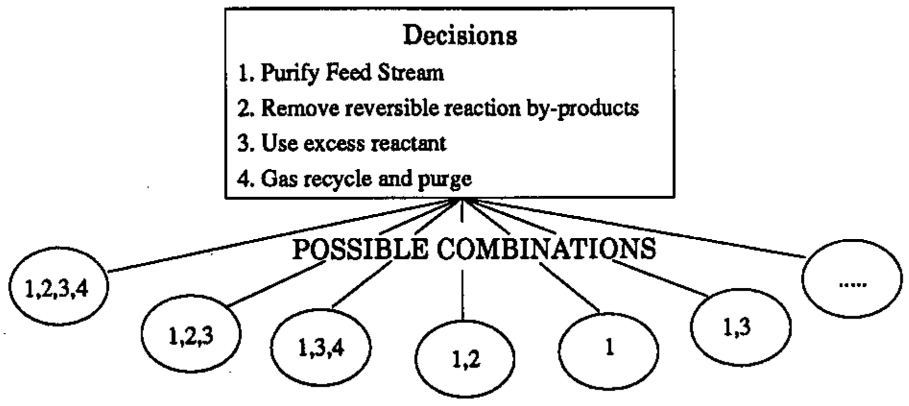


Figure 4.12: Possible Structural Alternatives Based on High Level Decisions

To reduce the number of alternatives that should be evaluated, Douglas describes heuristics and shortcut calculations indicating the most promising line of development. The intention is to guide the designer through the explosion of alternatives with the minimum effort in order to establish the economic viability of a process. The premise is that if the process developed using the heuristics is not going to be profitable, no more time need be wasted evaluating the alternatives.

Some decisions, however, may not have suitable heuristics to allow a decision to be made. In these cases, nodes representing the alternatives would be added to the design graph. Subsequent calculations and evaluations may provide the means to distinguish between the alternatives.

Douglas's method provides a high level means of decreasing the number of nodes created in the design graph and consequently reducing the amount of data. Section 6.1 describes the implementation of the decision level concept using CLAP extended methods. This technique focuses the approach to design in what can be considered a depth first evaluation of the process.

A lower level approach is required for evaluating alternatives which should reflect the ability of designers to determine the effect of a structural enhancement in the context of a more abstract process. For example, if a distillation column is being designed, the specification of feed location can be categorised as a source of structural alternatives. The natural approach is to create different models of the distillation column incorporating the structural differences. The models are initially evaluated separately, whereupon it is possible to eliminate some choices without going any further. The nodes containing the distillation column need contain no more than is necessary, e.g. only the column, or the column plus its auxiliary equipment; condensers and reboilers. It may then be desirable to model the alternatives as part of the flowsheet. This is achieved by replacing the high level distillation operation with the low level model. See Figure 4.13.

The method involves creating a design node for each structural alternative, but only to contain the items relevant to the evaluation. Eliminating some nodes by evaluating them separately, reduces the number of possible combinations. By allowing the modelling of a detailed node within the flowsheeting context of a more abstract node, combinations can be evaluated and perhaps rejected without having to store them all. The mathematical models should be retained, however, as a record of the evaluation of the relationship between the detailed and abstract parts.

Only combinations which cannot be distinguished by the desired criteria, or the designer decides have promise, need to be maintained as complete design nodes. For example, Figure 4.13 shows a high level node with two distillation columns. The structural alternatives to be evaluated are:

- whether the first column is a packed or plate column,
- there is also a choice of sidestream location in the second column with a corresponding choice of side stream stripper or rectifier.

The more detailed alternatives are represented in the sub-nodes. The diagram shows the relationships between the detailed and abstract levels, the two columns being maintained separately. The models which can be constructed, shown as boxes in the figure, display the full range of combinations and their association with the abstract node.

On the basis that the column with the sidestream stripper has been rejected after individual calculation, only two models of the combinations have been created. If one combination can be rejected, only one node need be maintained as a full flowsheet as shown in Figure 4.14. The shaded parts denote nodes which have been eliminated. If, however, neither is rejected, two nodes are required to be expanded fully.

In conclusion, two approaches have been used to reduce the amount of data requiring to be stored. An implementation of Douglas's hierarchy of decision levels focuses the evaluation of process alternatives by using heuristics with the

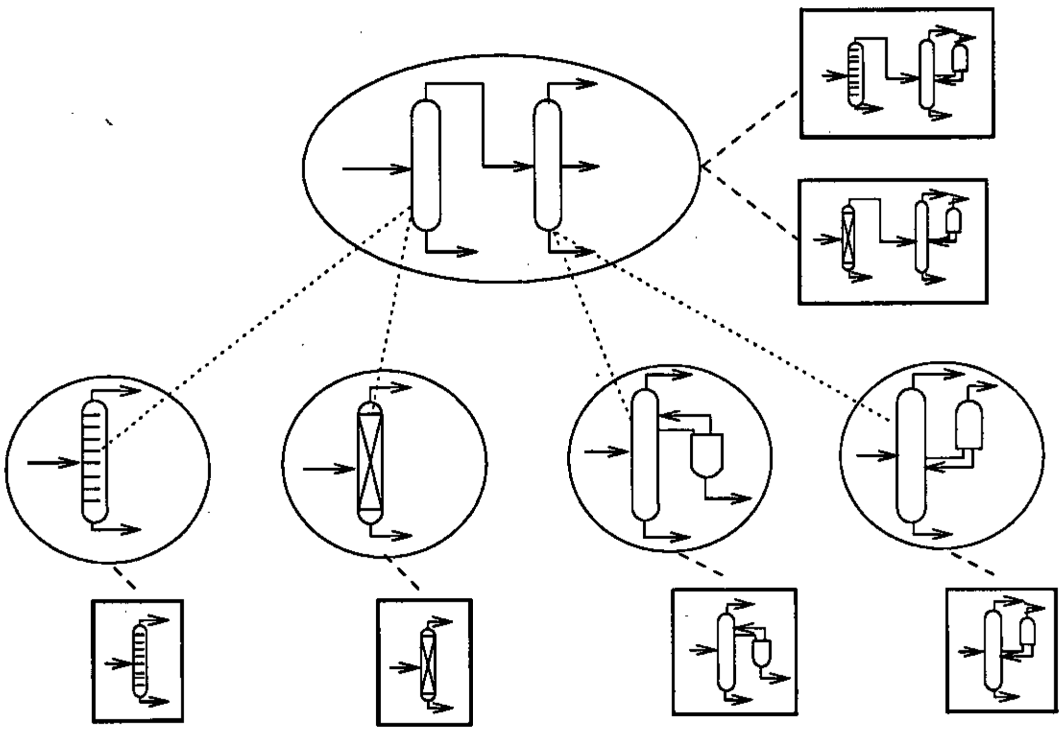


Figure 4.13: Illustration of the Interaction of Models in Abstract and Detailed Flowsheets

aim of providing a good base case design. Many alternatives can be eliminated without requiring mathematical modelling.

The second proposed approach limits the number of complete designs required to be held in computer memory by allowing modelling of alternative structures, initially separate from other items, and subsequently in the context of a higher level flowsheet, i.e. at the sub-node level in the Figure 4.13.

The relationship between the items in the abstract node and the detailed nodes is the has_parts relation discussed above. The implementation of the modelling aspects of the approach is discussed in Section 5.5.3.

4.4 Summary of Object Representation for Flowsheeting

A range of unit operations has been represented using objects related by an inheritance hierarchy. Since mathematical models are constructed based on the specifications and context of a flowsheet unit, attributes of objects are defined with sufficient flexibility to represent a range of applications. Objects are also

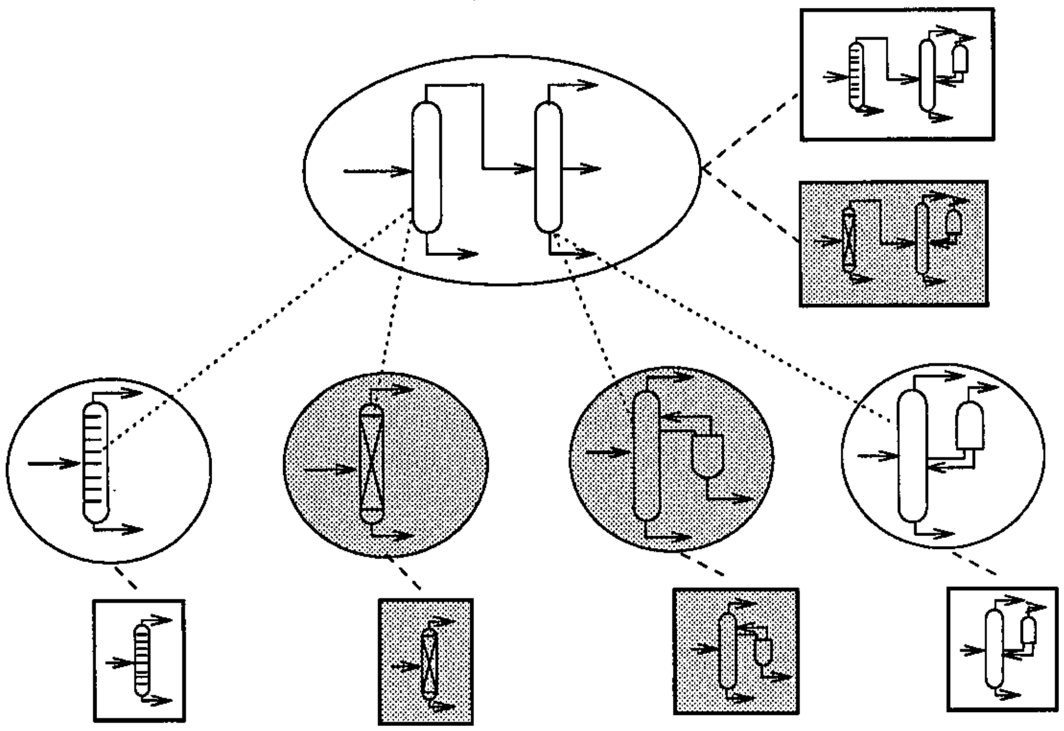


Figure 4.14: Possible Elimination of Process Alternatives by Modelling

defined for streams and chemical species.

Flowsheets representing refinements and alternatives generated during design are described by nodes in a graph. Each node is a CLAP object containing information describing the associated flowsheet and its position in the graph. Hierarchical development of a process is denoted by edges, i.e. an edge linking two flowsheets indicates that one flowsheet is a refinement of the other. This relationship is stored in the node objects.

Units in one flowsheet may represent parts of a more abstract unit in a less refined flowsheet. For example, a distillation column may be a part of a separation section. This relationship is also stored in the design node objects.

These two relationships have been used to provide a range of facilities for supporting the graph, for example, adding new nodes, adding flowsheet items, traversing the graph, etc. They also provide the basis for ensuring consistency and complexity in the graph. Consistency is maintained by checking the structure and function of objects related by "part of", thus ensuring new design nodes are compatible with specifications made at their parent nodes. Complexity is reduced by using the Douglas hierarchy of decision levels to eliminate some alternatives without creating a flowsheet. The "part of" and "refinement" relations are also used to allow modelling of alternative structures without having to store their full flowsheets.

Chapter 5

A Representation for Modelling Flowsheets

Development of a representation for mathematical models of flowsheets must consider the requirements of model developers and model users. A symbolic, equation based description provides a well known syntax enabling developers and users to understand models developed on this basis.

Automatic generation of models of unit operations is possible based on the information specified in the unit objects by the designer. This provides the designer with a model which is complete and consistent without having to be an expert on modelling. Using "model based" techniques, each term in each equation is accessible either by reasoning modules or by the designer which provides a means of modifying models for specific applications.

5.1 Equations as Constraint Relations

It is possible to associate high level models with generic unit operation objects, but the result is equivalent to a unit model library which is supposed to be replaced, as discussed in Section 3.2. The structure of the model is dependent not only on the function of the object, but the context it is in and the specifications made on the unit operation. For this reason, a full set of equations representing the object should not be defined within its generic description. The internal structure of the high level model should be determined only when a request is made to model the object. A choice can then be made between the modelling options available, or some combination of models may be selected.

Small sets of equations are defined separately and combined to represent a particular unit operation. This also avoids needless repetition of commonly used mathematical descriptions, such as heat balance equations.

It is possible to describe equations as objects with slots for the form of the equation, the variables involved and, potentially, linearised and integrated forms. However, equations are more appropriately defined as relationships between the properties of objects.

The definition of individual process items involves specification of their function and assignment of values to their structural attributes. Function determines the type of operations which are likely to be performed by the process, and the values of structural attributes indicate the potential range of parameters which will be incorporated. For example, the specification of a flash unit with one inlet, two outlets and three components, indicates the use of vapour-liquid equilibrium relationships and mass balance equations applied to a particular number of streams and components.

Equations must, therefore, be defined in a general form in order to produce a set of equations for a process where the number and composition of inlet and outlet streams is unknown. This applies to any situation where a variable number of equations may be generated or there is a summation of a variable number of terms.

A high level description of the equations is possible using predicate calculus. For example, the mass balance for a mixer can be written as:

$$in \in Inlets, out \in Outlets, \exists eqn \in Eqns,$$

$$eqn \,\underline{\triangle}\, \sum_{Inlets} in.mass\_flowrate = \sum_{Outlets} out.mass\_flowrate \tag{5.1}$$

where :	$A \in B$	means "A is a member of set B",
	$\exists$	means "there exists...",
	$A \,\underline{\triangle}\, B$	means "A is denoted by B",
	$\sum_A$	means "summation over the set A".

This description explicitly specifies the existence of an equation. In the context of algebraic model generation this is implicitly understood, so the definition can be reduced to:

$$in \in Inlets, out \in Outlets,$$

$$\sum_{Inlets} in.mass\_flowrate = \sum_{Outlets} out.mass\_flowrate \qquad (5.2)$$

Initially, a modelling system was developed incorporating this general description of equations using macros to describe the predicate calculus operations of $\forall$ ("for all"), $\sum$, $\exists$ and $\in$. The interpretation of general models applied to specific objects provided a set of equations applicable to the description of the object. This work was later combined with the structured user defined relation hierarchy implemented in CLAP (Section 2.4.1).

An equation of the form shown above is a constraint relating the properties of objects, here, the component mass flowrates in the inlets and outlets of a unit. This form can be applied generally to all units, providing mass balances for specific applications. Equations are represented in this form by CLAP "constraint relations". Written as a constraint relation, the description is represented as in Figure 5.1.

```
relation(overall_mass_balance, Unit - Range) :-
     domain - _,
     variables - [ Unit, Form, Bindings ],
     bindings - [I = inlets $of Unit,
                 O = outlets $of Unit],
     active_code - [ ],
     return_form - ( sum_of((mass_flowrate $of I), $over I)
                   - sum_of((mass_flowrate $of O), $over O)
                     = 0),
     return_type = equation,
     slots - [is_a - constraint].
```

Figure 5.1: CLAP representation of an overall mass balance relation

The slot `return_type` in Figure 5.1 indicates that the relation is representing an equation. The slot can also have values of `expression` and `list $of equations`. The difference between an equation and an expression is that an equation contains an equality and an expression does not. The practical implications of this are that the equation form is used to represent the highest level modelling constraints, e.g. conservation of mass and conservation of energy, while the expression form is used to represent the parts of these relations corresponding to properties of an object, e.g. vapour pressure or heat content can either be ascribed values as properties or be represented by mathematical expressions for their evaluation from more fundamental properties. The relationship between these two forms is further discussed in Section 5.2.

The remaining form, `list $of equations` is self-explanatory. It allows the description of more than one equation in a single relation.

The `return_form` is the direct CLAP equivalent of generic equation (5.2). The form can include summation terms, as in Figure 5.1, and specification of a set of equations (normally indicated by $\forall$), which both include the concept of set membership.

The `bindings` slot in Figure 5.1 allows reference to a list of properties or a list of other units, in this case a list of inlets and a list of outlets. When the relation is interpreted, the variables representing these properties (here, `I` and `O`) will be instantiated and, therefore, unified throughout the relation. The corresponding variables in the `return_form` will then contain the set of properties over which mathematical operations are to act, e.g. summation and "for all".

It should also be noted that the `bindings` slot provides a mechanism for accessing the slots of objects directly associated with the unit of interest. In this case, the description uses the flowrates of streams which are connected to the unit of interest. The simple example of a relation representing an overall mass balance requires only a simple mode of reference, i.e. the only accessible slots belong to the unit itself or to objects associated to the unit directly by static relations.

A more complex mode of reference is required if it is necessary to access slots which are related to one another within an object or are in different objects. For example, if a component balance is required, it is necessary to be able to write:

$$\forall comp \in Components, in \in Inlets, out \in Outlets,$$

$$\sum_{Inlets} in.comp.molar\_rate - \sum_{Outlets} out.comp.molar\_rate = 0 \tag{5.3}$$

or:

$$\forall comp \in Components, in \in Inlets, out \in Outlets,$$

$$\begin{aligned} &\sum_{Inlets} (in.molar\_flowrate \times in.comp.mole\_fraction) \\ - &\sum_{Outlets} (out.molar\_flowrate \times out.comp.mole\_fraction) = 0 \end{aligned} \tag{5.4}$$

In both cases it is necessary to refer to a property in a stream corresponding to a component. The properties, component molar flowrate and component mole fraction, are not considered generic attributes of a component (see Section 4.2), but as properties of the stream to which they belong, i.e. composition is a stream property despite the temptation to associate properties such as flowrate, mole fraction and temperature with component objects.

The access of such properties, therefore, requires a different mechanism to the simple reference shown in Figure 5.1. Stream objects have slots defined for components, mole fraction, component molar flowrates, etc. which is the logical place for such slots. The slots contain lists of corresponding values, e.g. if the `components` slot is set to contain `[benzene, toluene]` and `mole_fraction` contains `[0.4,0.6]`, the implication is that the mole fraction of benzene is 0.4 and toluene 0.6.

The properties corresponding in this way are accessed in a constraint relation by writing:

```
mole_fraction $corresponding_to components-C $of Stream
```

where `C` is instantiated to a component in the list of components in object `Stream`. The bindings and return form of a component balance are as follows:

```
bindings - [C = components $of Unit,
            I = inlets $of Unit,
            O = outlets $of Unit],
return_form - (
  set_of( sum_of( component_molar_rate
                  $corresponding_to components-C $of I, $over I)
        - sum_of( component_molar_rate
                  $corresponding_to components-C $of O, $over O)
        = 0,
         $over C)).
```

which is the direct translation of Equation (5.3).

A further mode of reference has been implemented for conceptually similar representational situations, but concerning relations rather than slots. For example, in the study of a stream it might be desirable to evaluate the vapour pressures of the components of the stream. Using the Antoine equation:

$$\ln P^* = A - \frac{B}{T + C} \tag{5.5}$$

where: P^* = vapour pressure
A, B, C = Antoine coefficients
T = temperature

A, B and C are all properties of a component, but vapour pressure and temperature are properties of the stream, as previously discussed. One solution to the representational problem is to make A, B and C stream properties in separate lists as with mole fraction, and then access them using `$corresponding_to`. However, it is equally important to maintain A, B and C as component properties as it is to maintain vapour pressure and temperature as stream properties.

A better solution is the introduction of a reference unit or object. This object contains the information not immediately obtainable with direct reference to the subject of the relation. In the vapour pressure example, vapour pressure would be defined as a property of a component (which is not particularly desirable, as discussed previously) at certain reference conditions. The reference unit concept provides a degree of flexibility which justifies its use, in that it can be either an existing object (in this case the stream of interest) or a separate entity containing only the test conditions. This can be used for evaluating a property separately from the flowsheet being studied, e.g. if it was required to know the vapour pressure of a component at a temperature different from the stream it was in. This second application is useful for independent calculations to provide information for decision making e.g. the calculation of relative volatilities to indicate the ease of separation of components by distillation. An example of a vapour pressure relation written using a reference unit is shown in Figure 5.2.

A more acceptable solution to the problem, and the one implemented, has been to adapt the relation mechanism to produce a list in the manner of the corresponding slots situation. The mode of reference is subtly altered. Using a reference unit, the vapour pressure would be described by:

```
vapour_pressure $of Component,
```

In the corresponding slot definition, the expression is written:

```
relation(vapour_pressure, Component - Range) :-
     domain - component,
     variables - [ Component, Form, Bindings ],
     bindings - [ ],
     active_code - [reference_unit(Unit)],
     return_form - (
         exp(  antoine_a $of Component
            - (antoine_b $of Component
              /(temperature $of Unit + antoine_c $of Component))
            )),
     return_type = expression,
     slots - [is_a - constraint].
```

Figure 5.2: A relation describing vapour pressure using a reference unit

```
vapour_pressure $corresponding_to component-Component $of Stream
```

The second case has defined vapour pressure as a property of a stream as opposed to a component, and, therefore, the relation has direct access to the stream slots including temperature. This description has been used to represent vapour pressures, K-values and relative volatilities. The bindings and return form of a vapour pressure relation with a stream as the subject, are:

```
bindings - [C = components $of Stream],
return_form - (set_of(
        exp(antoine_a $of C - antoine_b $of C
        /(temperature $of Stream + antoine_c $of C)),
                      $over C)),
```

5.2 Expansion of Generic Equation Descriptions

The expansion of generic constraint relations to specific relations modelling particular objects is achieved using a specialise relation, which associates a relation with an object.

The default expansion provided in CLAP expands the description of the mathematical relationship into equations (or expressions, see 5.1) containing Prolog variables. It is important to note that at this stage no evaluation has taken place.

This provides the opportunity to perform some reasoning about the form of the equations generated for a particular unit, e.g. to check the degrees of freedom of the model and its specifications, or the creation of the linearised form of the equations. Individual specialise relations can be defined for individual constraint relations, but it has not been found necessary to do so.

The association of a relation to an object is achieved with a normal `$set relation` call. The name of the specialised form of the relation adds the prefix `has_` to the constraint relation name. For instance, to associate an overall mass balance, as in Figure 5.1, with a mixer, m100, the call is:

```
$set relation-m100-has_overall_mass_balance-Name.
```

The name of the stored relation is returned in the argument `Name` for subsequent recovery and analysis.

In this way a single unit can be described by as many equations as desired. In a mathematical modelling sense, this gives rise to the possibility of the combination of what would normally be considered redundant equations describing a single unit. However, any relation applied to a unit is a separate entity unless explicitly combined in another constraint relation, e.g. the combination of heat and mass balances in Figure 5.3.

```
relation(heat_and_mass_balance, Unit - Range) :-
     domain - _,
     variables - [ Unit, Form, Bindings ],
     bindings - [ ],
     active_code - [ ],
     return_form - [overall_mass_balance $of Unit,
                     overall_heat_balance $of Unit],
     return_type = (list $of equations),
     slots - [is_a - constraint].
```

Figure 5.3: Combination of heat and mass balance relations

Being able to associate a range of equations with a single unit means many different modelling options are possible. For example, for modelling a distillation column, two different mass balances can be used. It is possible to provide a split fraction model for simple flowsheeting calculations, or detailed plate models for rigorous simulation of the performance of the column. In both cases it is possible to perform mass balances only or to add a heat balance as required. Further equations can be supplied for the determination of the number of plates and minimum reflux ratio as design calculations.

The two different mass balances which can be applied to the column are mutually exclusive, i.e. they cannot be combined into a single model without including redundant equations, since the modelling of the individual plates will inherently contain an overall mass balance which is stated explicitly in the split fraction model. The inclusion of a heat balance is a different matter. A heat balance can be defined for the whole column, and the same generic heat balance can be associated with each of the plates, i.e. one heat balance constraint is written which can be applied to a whole column as well as each of its plates. It is possible, therefore, to have three modelling combinations:

- split fraction balance and overall heat balance,
- plate mass balances and overall heat balance,
- plate mass balances and plate heat balances.

In the simplest case, one generic mass balance and one generic heat balance can be used to create the specific forms for the column and each of its plates.

The flexibility this provides is an important asset for a design tool. During design, calculations take many different forms, even in the design of a single item, such as the distillation column discussed here. The proposed modelling representation provides the ability to characterise the design data for a process item in a single structure and then model that data as required.

The specialise relation expands the constraint to a specific form and bindings corresponding to the object being modelled (see Section 2.4.1), the form being the specific equation, or equations, and the bindings being a list of terms identifying the variables in the equation with the location of the stored values. For example, if the overall mass balance relation in Figure 5.1 is associated with a mixer with two inlets, the form and bindings are as follows:

Form: $In_1 + In_2 = Out$

Bindings: $[In_1 =$ mass_flowrate \$of $stream_1 -> Val1,$
$In_2 =$ mass_flowrate \$of $stream_2 -> Val2,$
$Out =$ mass_flowrate \$of $stream_3 -> Val3].$

The values of the variables in the bindings are obtained by the "operator" relation. The slots and relations indicated by the middle term of each binding are checked, the values being unified with the *Val* part of the expression.

In the case of slots, this will either be a value, if the slot contains one, or the variable will remain uninstantiated, presumably to have its value calculated. It is also possible to access other user defined relations which have been defined in the `return_type` slot as "expressions". The return form of the user defined relation will be unified with the *Val* part of the binding and any bindings generated from its expansion will be appended to the overall list of bindings. It is, therefore, possible to construct complex expressions from high level descriptions. For example, an overall heat balance could be described by:

$$\sum_{in \in Inlets} in.heat\_content = \sum_{out \in Outlets} out.heat\_content \qquad (5.6)$$

In CLAP, this is written:

```
relation(overall_heat_balance, Unit - Val) :-
:
bindings - [ In  = inlets $of Unit,
             Out = outlets $of Unit],
return_form - (sum_of( heat_content $of In, $over In)
             - sum_of( heat_content $of Out, $over Out)
             = 0),
:
```

When the relation is expanded to its specific form, the bindings will contain references to the `heat_content` of all the unit's streams. When the bindings are checked, the `heat_content` may have a specified value, in which case the variable will be replaced by a numerical value. Otherwise, the constraint relation for `heat_content` is expanded and the expansion placed in the specialised form of the heat balance equation. The expression is defined by:

$$stream.flowrate \times \sum_{comp \in Components} (comp.enthalpy \times stream.comp.mole\_fraction) \qquad (5.7)$$

Similarly, the component enthalpies may have values, in which case no further expansion is necessary, or enthalpy could be expanded to the more fundamental expression for a component:

$$\Delta H_{ref} + \int_{Tref}^{T} CpdT \tag{5.8}$$

Again, it is possible for a value of Cp to be known, otherwise it is expanded to:

$$A + BT + CT^2 + DT^3 \tag{5.9}$$

As with numerical values, the expressions ultimately replace the variables in the higher level equations and expressions by Prolog unification. Thus, part of the overall heat balance relation above may eventually be described as:

$$\begin{aligned} stream.flowrate &\times ((comp.\Delta H_{ref} + comp.Cp(stream.T - comp.T_{ref}) \\ &\times stream.comp.mole\_fraction) + ...) \end{aligned}$$

It is possible for such expansion to take the model to a level of complexity that is inappropriate for a particular evaluation.

There are two mechanisms for controlling the depth of expansion:

- providing a value for a relation, e.g. here, specification of a component enthalpy would remove the requirement for its expansion.
- explicitly requesting the truncation of an expansion.

Truncating the expansion of a relation by specifying a value raises an interesting point. If, for example, the component enthalpy had been assigned a value and its expansion, therefore, was not performed, there would be no expression containing temperature as a variable. If one of the purposes of the calculation was to determine a temperature, it would now be impossible. For this reason, a mechanism has been provided to allow completion of the expansion even though a value has been specified for the relation. For example, a heat balance may be defined for a unit with one component, where the inlet enthalpy is 500 and the outlet enthalpy is 600. The resulting set of equations may then look like:

$$inlet.flowrate \times (500 \times 1) - outlet.flowrate \times (600 \times 1) = 0$$

$$600 = \Delta H_{ref} + Cp(T_1 - T_{ref})$$

With known values of ΔH_{ref} , Cp and T_{ref}, the temperature of the outlet stream can be determined.

An example application of this type of expansion and truncation of the expansion, is in the representation of a flash. The vapour product may contain identified light components, but the composition of the liquid may not be fully known. In this case, the vapour heat capacities would be expanded to the component level. Since the components of the liquid are not known, this expansion would not be desirable, but a value for the heat capacity of the whole liquid stream may be available. The value would be provided at the level of Equation 5.8, thus truncating the expansion.

Explicitly requesting the truncation of an expansion allows the definition of models of a known structure. This can be achieved either as part of a high level modelling algorithm, where the form of the equations is defined for a particular application, or by reasoning about the high level relation in the context of the design, i.e. the level of synthesis and the relationships with other plant items may define a level of modelling complexity.

The truncation is specified with the use of a "meta-slot". The specialise form can then be written:

```
$set relation-m100-has_overall_heat_balance@@level2.
```

This example will, therefore, only expand as far as component enthalpies and not as far as the polynomial Cp expression.

The expansion of user defined relations may result in the formation of programming loops. For example, the definition of mole fraction in a stream relates the quantities of the individual components present and the total quantity. A subsequent expansion of component quantities may involve mole fractions and the total flowrate, in which case a loop has been created. CLAP overcomes this by preventing a single expression being expanded more than once in a series of expansions.

5.3 Alternative Modelling Representations

The description of equations as relations is a powerful tool for the development of mathematical models. Once equations have been defined it is a simple task to combine them to create larger models ranging from individual component properties to models of whole plants.

The construction of models in this manner is similar to the approach adopted by ASCEND[32], where low level routines are written to model plant operations and subsequently combined to form larger models. Similar flexibility is obtained by being able to construct models from any parts appropriate or desired. Strict type matching prevents obvious errors, e.g. a liquid stream cannot be modelled as a vapour stream.

The representation of equations in the two approaches differs, with ASCEND using a procedural language in which equations fill the role of subroutines, compared with the symbolic predicate calculus description used here. It is possible to do more reasoning about equations written in a symbolic mathematical form. For example, if a sequential modular package is to be used to solve the model, it is possible to generate the sequential modular code by reasoning about the equations describing the problem (see Section 3.3). If an equation based solver is to be used, the same equations can be used to generate the linearised equations required by the solver. Since ASCEND has been written for an equation based solver only, this has not been considered as part of its development.

The differences between the representations stem from subtly different aims. ASCEND was developed as a modelling environment to allow a designer to develop models rapidly, whereas this project has been to create a flexible flowsheeting tool. The two aims appear to be very similar, since flowsheeting is the modelling of whole plants. The similarity is further compounded with the inclusion here of the aim of allowing modelling of individual units. Both modelling representations could be used to write unit models which form part of a flowsheeting tool. The subtle difference between the two approaches lies in the interaction with the designer. A designer using a flowsheeting program is not necessarily interested in creating the model of the plant item or, in some instances, even in all the equations constituting the model. In ASCEND, the intention is for the user to construct the models of process items and use them to construct higher level models of plants.

This is a significant disadvantage in its wider applicability, since it requires the user to be familiar with the concepts of modelling in such an environment, i.e. abstracting the unit operation into a complete modelling description. Engineers are familiar with many aspects of modelling, but few engineers are required to construct complete (and hence necessarily consistent) models of items of process equipment. Facets of the operation of a unit may be modelled separately. For instance, modelling a distillation column, an engineer may do a split fraction balance followed by a Fenske-Underwood-Gilliland calculation for the number of plates. This may be followed by plate to plate calculations. It is not immediately obvious how these different models interact and it is, therefore, unfortunately simple to construct an over or under-constrained model inadvertently. In this example, the combination of a plate to plate model and an overall component balance will almost certainly contain some redundant equations. For this reason, ASCEND is likely to be of more use to experienced modellers than engineers and designers.

The aspect of flowsheeting making it more widely accepted is the property of such programs of allowing the designer to specify the problem and leaving the flowsheeting program to provide the model.

MODEL.LA [28,29] has taken the position of the designer rather than the modeller. Allowing the designer to specify the phenomena in a chemical system provides a more natural means for a designer to define a model. However, the phenomena-based reasoning is, necessarily, domain specific. The resulting system cannot fully define the domain of chemical engineering modelling, so some interaction with the designer is required in order to provide mathematical expressions for circumstances not covered by the reasoning. For example, the distillation column definition in Figure 1.6 requires the additional expressions relating feed compositions, recoveries and product compositions. This option is assumed to be available to the designer, but no information is provided about the specification of such relationships.

Phenomena-based reasoning provides a different problem for the designer. If a model is being written in the script format of MODEL.LA described in [28], no guidance is given as to what phenomena are involved. For example, if it is not stated that the model is to be lumped, it is not clear what conclusion would be reached about the type of mass and energy balances to be included. For this reason, a comprehensive model editor has been created which identifies the particular areas that should be considered with a range of alternative specifications.

It is, therefore, important for the designer to be able to create and modify models in two ways. Automatic generation of a model based on system phenomena is valuable to provide the structure of the model without requiring the designer to have expert knowledge of modelling principles. Access to the mathematical description is also vital, enabling visual evaluation of the model and the possibility of modifying the symbolic expressions for cases outwith the scope of automatic generation. ASCEND provides the latter functionality, but only very limited "reasoning" by which detailed models of high level operations can be constructed from low level models. MODEL.LA places the emphasis on the first aspect, but provides limited access to the symbolic expressions. ModAss [33] attempts to cover both facets by providing some automatic generation and access to the fundamental modelling aspects. The automatic generation appears to be restricted, however, to mass and energy balances.

A concept implemented in both MODEL.LA and ModAss is a model of the state of the design. MODEL.LA supports the development of a design graph, as discussed in Section 4.3, which is similar to the CLAP implementation with only slight differences. ModAss, however, supports hierarchical decomposition but does not accommodate structural alternatives. The discussion in Section 4.3 indicates the importance of alternatives in process development and documentation. The discussion also indicates a restructuring of the data is required for the incorporation of alternatives. ASCEND does not contain the concept of a model of the design except, to a limited extent, in the decomposition of models, but it is discussed as being part of a design environment [1] which would, presumably, incorporate such a model. In that context it is possible that some degree of reasoning about models be implemented.

The modelling representations discussed here cope differently with the notions of generic models and model instantiation. The CLAP representation contains generic objects describing process operations, separate generic model descriptions and the ability to create multiple instances of the models. The multiple instances allow the designer to create different formulations interactively and store the different results for subsequent review and evaluation.

ASCEND approaches the concept by defining a novel interpretation of generic objects and instances. A generic model can have refinements which, in turn, can be further refined. This is conceptually the combination of inheritance and the creation of model instances. For example, a model of a generic flash can be RE-

FINED_TO a model of a distillation feed tray, an operation which incorporates the notion of inheritance. The feed tray, however, can still be treated as a generic model allowing a further refinement to be made. This refined model may have a specified feed flowrate which would, in object oriented terms, normally be considered an instance of the model. An instance of a model in ASCEND is created when it is being compiled for solving. This approach has not been required in CLAP with the separation of the concepts of unit operation objects and modelling relationships. If ASCEND is to be incorporated in a larger environment, as has been suggested, the separation of these two individual concepts may require some modification of the language in order to provide a relationship between, for example, a flash model and a flash object, which could be described as entirely separate entities.

In MODEL.LA, the approach has been to define generic modelling properties associated with generic process objects. The implication of this is that instances of process objects have the same modelling equations. It is not clear whether or not the generic description can be altered dynamically for new instances of the operation, or for modifications of an existing one. It would seem unlikely because, if so, there is no correspondence between an instance of a model and its generic description in a case where this description has been altered.

It is valuable to be able to represent different instances of a conceptual process object by a range of modelling formulations, e.g. model one instance by a shortcut method, and another by a more rigorous method. It is equally important to be able to model a single instance by different formulations, e.g. to model a distillation column by a Fenske-Underwood-Gilliland model and subsequently model the same column with a plate-to-plate model. The MODEL.LA description seems to imply that these require different generic descriptions. In cases where the only difference between models is a single phenomenon, it would necessitate the creation of two generic models for what is, conceptually, a single process. During the course of modelling it is quite likely for such a set of models to be required, which, initially, provide a level of approximation but on modification, provide a more complete realisation of the process.

In CLAP, defining the models separately from the process operations allows model instances to be created reflecting the state of the process object. In effect this is equivalent to creating an instance of a model and then further refining it to an instance of a solution.

The approach adopted in ModAss incorporates models in a manner similar to CLAP. The models are separate from flowsheet items until specified as being included. Every model created is treated as an instance in that it applies specifically to an instance of a unit. It would appear that the modeller is able to alter the model and then recompile and solve it. It is not clear, however, if these different solutions are maintained along with the description of the model at the time of compilation as is done in CLAP.

The facets of modelling desirable within the context of design and which have been identified in the above discussion, are:

- The ability to provide some automatic model generation, but also to allow access to models for modification and specification.
- That provision must be made for dynamic modification of models rather than accepting a generic model for a generic process, which would be equivalent to accepting a library model albeit a custom made one.
- Instances of models should be retained with their associated solution to provide a record of the modelling formulations evaluated.
- A model of the design is required to provide full scope for evaluation of different structural options generated in a design. The model also provides a record of the decisions taken.

The implementation of a design graph in CLAP for maintaining process decomposition and structural alternatives has been discussed in Chapter 4. The definition of the modelling relations allows for instances of relations for different formulations of process items. It also allows for multiple instances of solutions which maintains the record of evaluations, as described above. The restrictions of simply providing a library model have been discussed in Chapter 3, indicating a requirement for some compromise. That compromise is as follows.

1. A model is generated automatically from the specifications made on individual process items, thus removing the requirement of the engineer to have knowledge about available models. The equation representation also removes the restriction of a library of fixed models, since models in this system are individually constructed from selected equations.

2. The model can be edited at a high level to provide extra modelling flexibility. Since the model has been constructed by reasoning about the specifications, the alternatives to the chosen model are known and can be presented to the user for selection, e.g. a general flash model can be replaced by a split fraction model.

3. The model can be edited at a low level to provide extra detail, e.g. a K-value calculation using vapour pressure can be adapted to use fugacities.

4. Entire unit descriptions identified as being functionally equivalent can be interchanged to provide a different level of structural detail. For instance, an object representing a distillation column may be modelled using Fenske-Underwood-Gilliland equations. A greater level of detail could be obtained by creating objects for the plates of the distillation column and modelling each plate as a separate flash operation. The model of the column as plates can be included into the model at the level where it is represented as a column, thus achieving the greater level of model detail at the higher level of abstraction.

In conclusion, the three systems discussed here in comparison with CLAP modelling capabilities demonstrate most of the key issues which must be considered in the development of a tool for flowsheet modelling within the context of a design environment. The three systems, ASCEND, MODEL.LA and ModAss, do not individually provide the full required functionality, but many of the points considered in the development of the tool in CLAP can be found in one or other of the systems.

From a model developer's viewpoint, ASCEND and CLAP user-defined relations both allow flexible modelling of unit operations. Both representations support the expansion of high level descriptions to low level descriptions, and both allow the selection of alternative models for a single application, e.g. short-cut or rigorous calculations.

The differences stem from the reasoning to be provided for the intended application. As part of a flowsheeting tool intended to be used within a design environment, the representation in CLAP has taken into account more general aspects of the description of design. Objects have been created in CLAP to represent process items which can be reasoned about, here, for modelling purposes, in Chapter 6 for synthesis and hypothetically for control, layout and safety (see Section 2.2). ASCEND uses object-like structures which are purely for modelling purposes, which are, therefore, limited to reasoning about equations. As mentioned previously, the representation of equations in predicate calculus may allow more reasoning about the equations themselves.

It should be noted, however, that the definition of modelling equations in CLAP is independent of the objects describing unit operations. This suggests the possibility of accommodating the ASCEND approach into a wider environment in a similar fashion.

For widespread use, both ASCEND and CLAP descriptions rely ostensibly on experienced modellers to create a suite of consistent models which could then be used by engineers who are not expert modellers. Mechanisms have been provided in CLAP to allow non-expert modellers to take advantage of the flexibility of the modelling description. A consistent model is generated automatically, thus performing the "expert" tasks, leaving the engineer free to customise the model within the limits imposed by the need for consistency.

The important facet of the MODEL.LA language is the phenomena-based reasoning which provides a structure for the model without requiring expert knowledge of modelling principles. The representation of generic models of specific phenomena differs from the CLAP implementation since, in CLAP, an instance of an object can have several different formulations depending on the specifications made. The MODEL.LA representation seems to require separate instances of generic models which must be defined for the full range of applications.

MODEL.LA is built round a graph describing the development of the design. This representation can be used as a basis for broader considerations of other design tasks.

The ModAss modelling tool provides two major facilities for the development and use of models. The models are partly generated automatically, although with consideration of principles other than mass and energy balances, as in MODEL.LA and CLAP (see Section 5.4), the scope of these models could be widened. The user also has general access to the models and the parts of the models, but again could provide some reasoning about their formulation (as in CLAP in Section 5.4.4). The existing reasoning is limited to inheritance and reasoning about the mathematical properties of the models, e.g. degrees of freedom.

5.4 Selection of Unit Models

Traditional flowsheeting programs model plant items at a predefined level of detail. If a unit model is incorporated in a flowsheet then the mathematical description contained in the library associated with the specified unit is added to the overall plant description. This assumes a level of mathematical detail associated with a level of functional detail, e.g. the specification of a heater indicates the use of a mass balance, heat balance and pressure balance at a level of detail appropriate to its operation despite the requirements of the flowsheet.

Ideally, there should be unit models to represent every operation required at every stage of a hierarchical design procedure. The models provided should also be flexible enough to allow a variety of input specifications.

In order to provide a feasible number of unit models at a level of practical use, flowsheeting program writers have necessarily restricted the models provided to ones corresponding to the process flow diagram stage of process development. Such a restriction makes the modelling of a hierarchical design awkward using conventional flowsheeting software. Attempts have been made, however, with limited success, to apply an industrial flowsheeting package to such a procedure, e.g. FLOWPACK, the ICI flowsheeting program [18].

Expert knowledge of the package is necessary for the correct selection of the models to represent the high level processes of a block flow diagram. The problem of model selection extends beyond the block flow diagram stage e.g. the condenser-flash-mixer problem in Chapter 3. Further knowledge is required to specify the process models involved correctly, so as much of the available information is utilised ensuring the degrees of freedom are satisfied. Since the library of models is defined at the process flow diagram level, the degrees of freedom are likely to be higher than is required for the modelling purpose in question. Certain input parameters, therefore, must be estimated to satisfy them. It should not be necessary to provide estimates which, apart from annoying the user, may be forgotten and subsequently mistakenly used as real data.

The designer should not have to formulate a design problem to suit a unit model library, thus compromising the functional knowledge contained in the process description with the limited set of models available. For this reason, modelling equations are maintained separately from generic process descriptions.

Once a process object has been created and included in a flowsheet, the modelling equations can be selected from a bank of equation models. This allows the designer to specify the unit operations required without having to consider which models contain which equations. The specification of process structure and function is sufficient to allow the selection of a set of equations (or fundamental unit operation models in the sequential modular case) describing a unit operation.

The selection of equations appropriate to the level of functional detail can be divided into two stages:

- selection of high level descriptions, e.g. identification of a requirement for mass balances, heat balances, phase equilibria, chemical equilibria, etc.,
- identification of the low level form of the high level descriptions applicable to the unit being modelled, e.g. if a K-value is required, the expression may or may not include a Poynting Correction factor depending on the pressure maintained in the unit.

It is possible to classify both stages under the second heading. The selection of high level descriptions can be thought of as being the identification of the form of the highest level balance description. The two have been separated, however, to allow the use of generic modelling information to ensure that the high level descriptions being constructed are consistent across a flowsheet and their constituent relations are compatible with each other. The selection of the appropriate form of the equations is, therefore, a separate problem, describing the specified function of the unit in a consistent framework of high level relations.

The two stages differ in implementation in that the high level description of a process unit is a single relation constructed from several lower level relations in the manner of the heat and mass balance relation in Figure 5.3. The selection of the form of the equation, however, involves identifying the terms of the expressions taking different forms depending on the unit and its specifications.

While it is valuable to have the automatic generation of a mathematical model of a unit operation, it is possible for the generated model not to be what was intended by the designer. It can also be argued that automatic model generation is almost as inflexible as the system it is supposed to replace. The designer is still required to provide certain input parameters which are unavailable, or the model may not provide the detail required. For this reason, it is also important that, while a model may be generated automatically thus performing the "expert" task of creating a consistent and complete model, the designer still has the freedom to modify it to suit a particular application.

The following subsections discuss the implementation of an automatic model generation tool and the interactive manipulation facilities available to the designer. This is illustrated with some examples.

5.4.1 Selection of High Level Model Descriptions

The automatic selection of high level descriptions of flowsheet units is intended to perform the "expert" tasks of creating complete and consistent models of process items.

The function of the object can be used to identify all possible types of equation that can be used by an object, i.e. mass balances, component balances, phase equilibria, heat balances, etc. However, once the equation set has been identified, the set which best describes the operation of the object at its state of specification must be ascertained.

It is possible to associate all user defined relations with a domain class of objects using their "domain" slot, see Figure 5.1. The relation can then only be applied to objects which are instances of the specified class. The default interpretation mechanism for relations does not check all "is_a" relations and so equations could not be inherited. A mechanism could be provided to perform this operation, but in its application several problems would arise:

- the search space of applicable relations is large and not ordered, making any search techniques time consuming,
- all relations which could be applied to a particular unit would be located, but would not necessarily constitute a consistent model,
- alternative models for a unit would not be recognised as being separate entities.

Several approaches have been investigated with the aim of generating a model based on the function of the units contained in the flowsheet and tailored to suit the specifications made.

Initially, the slots of the streams connected to a plant item object were investigated in an attempt to identify the operations occurring within the object. For example, if there were components in the output streams of an object which were not in the inputs, then one operation of the object is reaction. In this manner, various fundamental operations could be identified: reaction, separation, mixing, temperature change, pressure change and stream division. However, each individual unit in the flowsheet would have to be completely specified in order to provide a complete functional description. This would over-specify the mathematical model and so is not a feasible approach.

Instead of analysing the streams into and out of objects, the same information can be derived from the functional description of the objects, i.e. its purpose and internal structure. For instance, as described above, specifying the use of a heater implies the modelling requirement for a mass balance, heat balance and pressure balance. The important consideration is to establish which are applicable for a given level of detail.

An engineer creates models based on assumptions made at the higher levels of a design. As the design progresses, the assumptions either become supported and can be accepted as facts, or the evidence accrued contradicts the assumptions made, thus forcing the designer to alter earlier decisions. In an attempt to emulate this approach, the selection of appropriate modelling equations was based on the assessment of assumptions. Classes of equations were associated with assumptions fundamental to the use of the equations, e.g. Fenske's equation was based on constant relative volatility.

At the start of a design, a set of high level assumptions can be asserted with the intention of assessing the evidence supportive or contradictory to them which is amassed as the design proceeds. For example, very simple high level assumptions could include: constant composition (no reaction), constant enthalpy (no specified change of heat) and constant pressure (no specified pressure change). As objects are created and slots set, the assumptions are altered in the light of the new information. Since the equations are associated with assumptions, a set applicable to the level of assumption can be constructed.

An implementation of the proposed scheme was attempted, whereby assumptions were asserted at the beginning of a design and evidence to support or contradict these assumptions was accrued as the design developed. The work did not provide a satisfactory method for automatic model selection, but provided a valuable insight into the associated problems.

1 Evidence in support of, or against, assumptions could be regarded as conclusive or inconclusive but significant. This appeared to correspond with functional and structural aspects respectively. The creation of a new object alters the purpose of the flowsheet functionally and, therefore, irrefutably alters the assumptions, e.g. the creation of a reactor object implies a change of composition and species, hence removing the assumption of constant composition. The setting of slots provides structural information which tends only to support or contradict assumptions without being conclusive. For example, specifying the temperature of a stream does not imply non-constant enthalpy thereby signifying the requirement for a heat balance, but merely indicates that it may be the case. Further specifications of temperatures make the evidence more conclusive.

2 Most assumptions apply locally rather than globally, e.g. the creation of a reactor does not signify the use of a reaction model for every item in the flowsheet but only for reactors. In another case, a flash unit may be modelled using an equation of state, while another flash unit may not have sufficient data for that level of modelling.

3 The identification of every slot and object and their interactions with each assumption is combinatorially too large to represent.

4 In many cases it is difficult to assess precisely what affects an assumption. Consequently, it may be more advantageous to allow the designer to state when assumptions should be added or removed. For example, in the early stages of design, initial estimates of design variables can be achieved by making assumptions which are known not to be true, e.g. the assumption of ideal gas behaviour. However, their use is justifiable as long as models developed on this basis provide an acceptable approximation of the process being designed. In the case of an ideal gas model, it would almost certainly have to be the designer who specifies when this assumption is no longer valid.

5 Although basing the selection of equations on explicitly represented assumptions proved infeasible, the approach did, however, identify some of the requirements of sets of equations for their inclusion in a model.

In summary, the selection of modelling equations based on the level of assumption incorporated in the flowsheet description is a valid approach. The combinatorial nature of such an approach makes it infeasible to maintain all assumptions and their interactions with slots and objects automatically (3 above). The determination and evaluation of some interactions is exceptionally difficult, implying that the user will have to define the use of a large number of assumptions (4 above). This, however, will present the user with an unwanted barrage of questions as to the applicability of assumptions for the modelling of each item in the flowsheet. The approach adopted for model selection is intended to provide the user with a complete and consistent model of the flowsheet without requiring any further information over and above the design information already available. The model created may not necessarily be exactly what the designer was intending, but it will be consistent and will allow the alteration of the model within the constraint of consistency. The alteration of the model is discussed in 5.5.

5.4.2 Representation of a Model Library

The method eventually proposed for model selection corresponds to the unit model library of traditional flowsheeting programs (see Chapter 3) with the addition of heuristics to select the model which best fits the specification of the flowsheet. The recognition of the existence of local and global assumptions (2 above) suggests a representational mapping close to the unit object hierarchy discussed in Section 4.1. This, consequently, implies the use of an inheritance mechanism relating the assumptions implicit in the functional description of a unit operation to the applicable models.

An object has been defined to describe a "model" which contains descriptions of all high level flowsheeting models applicable to the level of functional description associated with the object. The information contained in the object includes the conditions for the use of the equations and allowable assumptions which can be made when a model is invoked.

Model objects have been associated with generic unit operation objects with a relational mechanism, thus allowing inheritance of models from higher levels of functional description. This mechanism is used for instances where the model suggested by the function of the unit is not applicable because insufficient structural information is available. For example, the creation of an object representing a flash suggests the use of a vapour-liquid equilibrium model. However, if the temperature, pressure or heat load have not been specified, then a model containing only composition balances need be provided.

Figure 5.4 shows the correspondence between the unit model hierarchy and the model objects. The diagram shows the expected model enhancements between the specification of a plant object and, for instance, a separation object. It also shows the further property that a model object need not be associated with every generic unit operation object. In the example shown, the mixer unit has no model associated with it. Consequently, it will inherit the modelling information of the flow change object and ultimately the plant object, in this case, an overall component balance. The divider object, however, has an associated model incorporating divider ratio expressions. If no divider ratio has been specified, an overall component balance is required. See Section 5.5 for a discussion of the editing of the model to allow the use of a ratio model without having ratios specified.

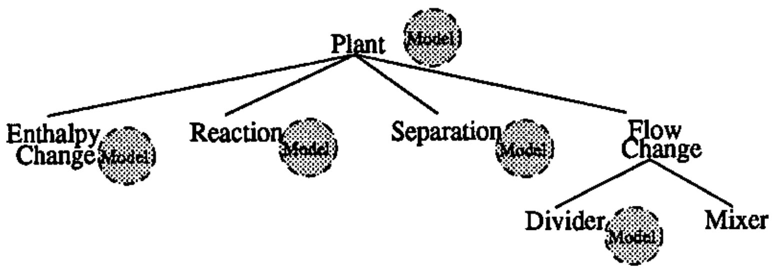


Figure 5.4: Correspondence Between Hierarchy of Process Items and Models

To prevent the inclusion of contradictory equations, e.g. a reaction mass balance and an overall component balance, "guards" have been associated with each model, thus allowing the selection of a single model from a list of potential models. The guard expressions must be completely satisfied for the inclusion of a model. The expressions normally include the checking of slots and relations of the unit being modelled, but may also include checks of the context of the model, such as the level of detail of all separators in the flowsheet. This is, effectively, the checking of local and global assumptions respectively. In the example here, if any separator is only at a conceptual stage, i.e. not defined as a unit operation, then a general heat balance cannot be implemented since the heat requirements of a "separator" are unknown.

An inheritance mechanism for model refinement implies that more conceptual functional descriptions should be continually updated by lower level modelling equations as the level of functional detail increases. However, not all modelling equations are refined with every increase in functional detail. For instance, a flash separator may have sufficient detail to allow an overall heat balance which is only available at the functional level of vapour-liquid separation when the type of heating is known. However, its composition may only be modelled by an overall component balance which is inherited from the "plant" model. For this reason, modelling equations have been split into areas of application, each one individually inheritable. The areas are:

- reaction
- composition
- energy
- flow
- pressure

Thus the flash object would inherit the energy model - overall heat balance - from the vapour-liquid separation model object, and the composition model - overall component balance - from the plant model object.

The structure of a "model" object is shown in Figure 5.5.

```
object(model) :-
   self - _,
   variables - [ReacEqs,CompEqs,EnEqs,Feqs,Peqs,Assum,Unit,Info],
   slots - [reaction_eqns - ReacEqs,
            composition_eqns - CompEqs,
            energy_eqns - EnEqs,
            flow_eqns - Feqs,
            pressure_eqns - Peqs,
            assumption_level - Assum],
   relations - [unit_model - Unit,
                required_info - Info].
```

Figure 5.5: Generic Model Object

The two relations contained in the description of the model object are used in the interpretation of the object. When the unit_model relation is instantiated to the name of the object being modelled, Prolog unifies the name throughout the model object and, thus, into the guard expressions. The required_info relation is provided for convenience. Several guards within one object may check the same slots in the same unit operation object. If the guard fails, backtracking uninstantiates the variables requiring the slots to be checked again in subsequent guards, The required_info relation is for the checking of slots which would otherwise be repeated, thus saving computing effort.

A example of a model object corresponding to a "plant" object is shown in Figure 5.6.

```
model - plant_model :-
    variables - [
  /*reaction*/       reaction_model
                       - (not(var(Reacs)))
                       - (conversion $of reactions $of Unit = 1),
  /*composition*/    overall_component_balance
                       - (var(Reacs)
                          $and expand_components(Unit))
                       - inlet_zero_rates,
  /*energy*/         $null - $null - $null,
  /*flow*/           $null - $null - $null,
  /*pressure*/       $null - $null - $null,
                 $null,
                 Unit,
                 [($check relation-Unit-(reactions-Reacs))]].
```

Figure 5.6: Example Instance of a Model Object

Note that the variable Unit appears throughout the object, so on its instantiation it is unified throughout the model. The required_info is obtained by checking the reactions relation of Unit. The value Reacs subsequently appears in two guards which would otherwise have had to be checked for each equation model.

The expressions corresponding to each equation slot have been divided into three parts:

- a dotted list of equation models (see Section 2.4.3)

- a corresponding dotted list of guards
- a corresponding dotted list of local assumptions

The example in Figure 5.6 has only one member in each dotted list, but the interpretation mechanism takes each term as a dotted list and processes the terms in the corresponding positions in the three lists.

There are two levels of assumptions which can be incorporated in the model object: those only applicable to the functional description associated with the model object; and those applicable to any object invoking the model. For example, in Figure 5.6, the local assumption corresponding to the equation `reaction_model` is that the conversion of reactions of the unit of interest is equal to 1. This only applies to the modelling of "plant" objects and not to any other objects inheriting the model. The `assumption_level` slot in Figure 5.5 is for inheritable assumptions.

The model in Figure 5.6 potentially allows the inheritance of a reaction model and an overall component balance, which as has been previously mentioned, are mutually incompatible. The corresponding guards, however, ensure they remain so. The reaction model is included in cases where a reaction has been specified for the unit, and a component balance in cases where a reaction has not been specified. This has implications for the inheritance mechanism. It is possible for any object which has had a reaction specified to inherit a reaction model instead of a component balance. For example, a heater or a flash model could incorporate a reaction mass balance, if so specified. This provides great flexibility for the mathematical description of the processes involved in a flowsheet.

The proposed mechanism for selecting equations is similar to that employed by MODEL.LA [28]. MODEL.LA uses the specification of physical and chemical phenomena to provide the modelling definition of a generic object. Here, however, the concern is with modelling specific instances of objects. The implication of the MODEL.LA work is that when a process is defined, the model is created by selecting the generic types of the items of process equipment. The modeller, therefore, must know to what degree of detail the resulting model must be taken *a priori*. Further, to provide a range of models of differing detail for a single process requires the specification of a generic object for each one.

The method outlined here is intended to provide the designer with a means of defining a mathematical description by specifying the function and structure of process objects. The function is provided by the generic class of the object, but, unlike MODEL.LA, a single instance can be described by a range of different models. The structure is detailed by the provision of values for certain key properties (in slots). The advantage of allowing models to be defined in this way is that the user is performing a natural task - specifying functional detail and providing values for design constraints.

Part of the hierarchy of models is shown in Figure 5.7. Nodes in the graph in parentheses have no associated model and thus inherit models from higher levels.

5.4.3 Example of Model Selection

The selection of modelling equations based on specifications is best illustrated by an example. Consider the description of a flash vessel. In the first instance, an object will be created with its connections detailed. If these are the only features attributed to the object, then the model which will be invoked is an overall component balance relating the components in the inlet and outlet streams.

This decision is based on the local specification of the object. However, there is another part of the model which could be included which is dependent on the context of the object. As part of the flowsheet, the degree of detail of other objects determines some aspects of the mathematical description of the individual items. In this case, for example, a heat balance will not be proposed unless all separation processes have been characterised to the level of a unit operation. Since the energy requirements of a "separation" cannot be determined until the type of operation is realised, a heat balance would serve no purpose to the complete flowsheet. If however, the scope of the model includes no such indeterminate processes, an energy balance can be included.

Depending on the further detailing of the constraints on the flash, the model will be constructed accordingly. If split fractions are provided, the object adopts the level of a simple separator with its associated mathematical description. If, however, a temperature or pressure is supplied, the intention is to model the vapour-liquid equilibrium of the vessel. Two equilibrium models have been defined. One has simple convergence properties but is limited to processes with single inputs. The other more general description is not as robust. For this reason, the more specific model has been retained, but is only used for processes with a single inlet.

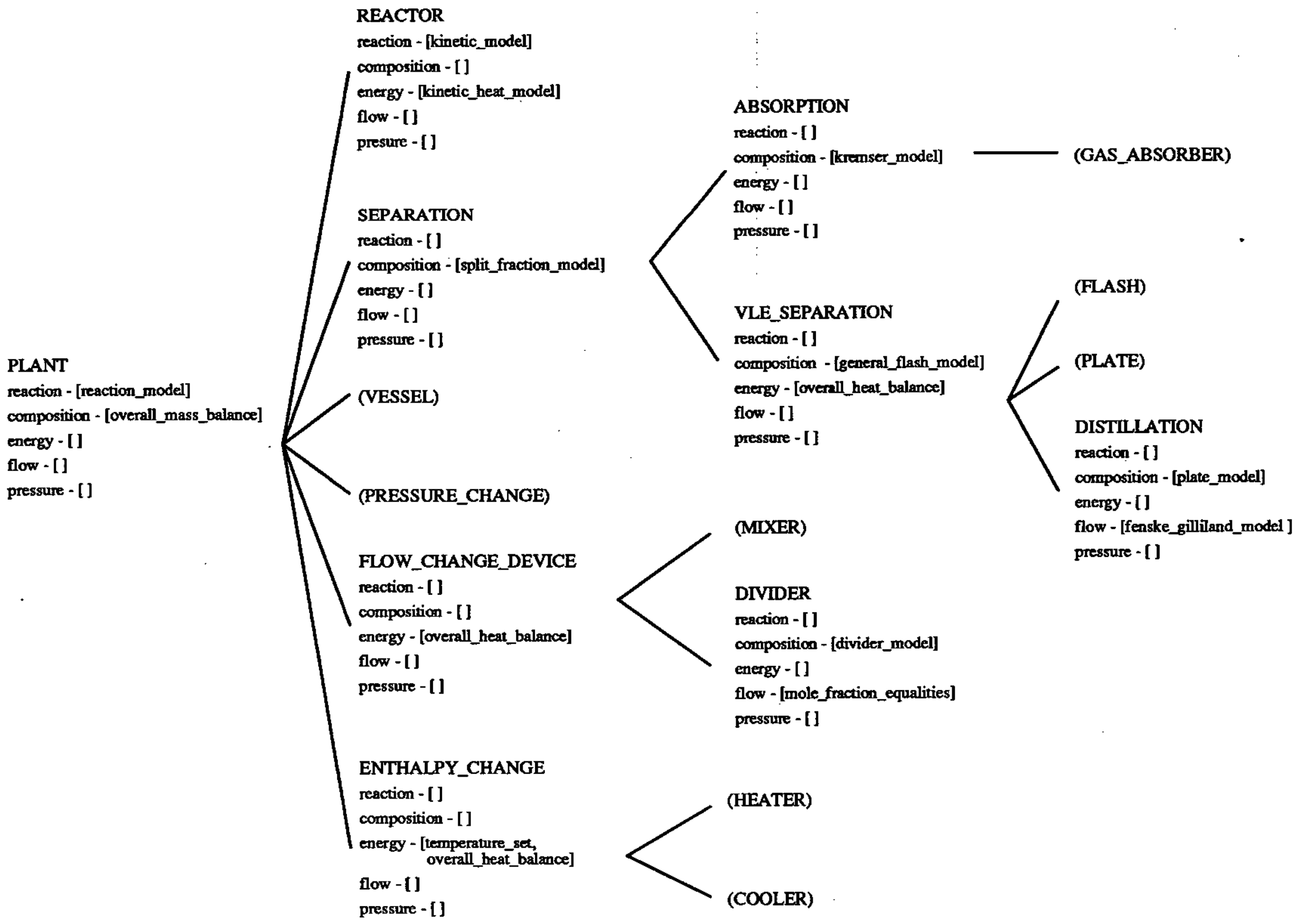


Figure 5.7: Part of the Implemented Hierarchy of Models

Once the flash has reached this level of detail, no further choices of model are available. The form of the selected equation can be altered, however, to widen the scope of their use in order to generate applications for specific processes (see Section 5.4.4). This level of model generation concerns the selection of the terms of the equations which are relevant to the specifications made on the process.

5.4.4 Selection of Equation Form

Section 5.4.2 described reasoning about models at a high level. This reasoning ensures model completeness in terms of the calculations achievable in a particular context, and consistency by ensuring the interactions between the different parts of the model do not result in redundancy or contradiction.

Within the framework of completeness and consistency, there are certain aspects of the model which can be reasoned about. In situations where alternative models exist for a single property, it must be possible to select the appropriate definition for the current context. For example, the Nusselt number correlations for the evaluation of heat transfer coefficient differ for laminar and turbulent flow. If it is possible to determine the flow regime at the time of model generation, it should be possible to select the appropriate definition. There are occasions, however, where several, seemingly equivalent, definitions are available with no means of distinguishing between them. For example, there are several alternative empirical correlations of Nusselt, Prandtl and Reynolds numbers for the same flow regime. In such cases, a choice should be made, but revision of this choice should be allowable when more information becomes available.

A similar case can be made for approximate models. As mentioned in Section 5.4.1 the assumptions supporting the use of approximations are very difficult to justify, so inferring their applicability is equally difficult to implement. If it is not possible to define a set of conditions for the use of an approximate model, it could be used in the first instance allowing revision of the model at a later stage.

The inference mechanism for selecting the form of equations is discussed in this section. The interactive revision of models with illustrative examples is presented in Section 5.5.

The mechanism for inferring alternative forms of equations involves defining an extra call in the active code of the parent relation. The call corresponds to a single undetermined term of the return form expression, therefore requiring a call for each such term. The calls are Prolog clauses establishing the form of the unknown term, generally by checking slots and relations. The clause, `select_eqn_type`, has four arguments:

1 the Prolog variable in the return form which is to be unified to the inferred form,

2 a reference to the term being sought, e.g. fugacity_coefficient,

3 the object that is the domain of the relation, which, in general, is being checked for appropriate slot values,

4 the reference unit, if applicable.

The reference term (2) is used to locate the correct Prolog goal and is also used for reporting the generic form of the equation.

When the relation is invoked as part of a model, the active code is run before the generation of the return form. The `select_eqn_type` calls are made, instantiating the undetermined variables in the return form expression which is then expanded as normal.

For example, consider the definition of the vapour-liquid distribution coefficient:

$$K_i = \frac{\gamma_i \times f_i \times \Phi_i \times P_i^*}{\phi_i \times P_T} \qquad (5.10)$$

where: γ_i = vapour activity coefficient,
f_i = fugacity coefficient,
Φ_i = Poynting correction factor,
P_i^* = vapour pressure,
ϕ_i = partial fugacity coefficient,
P_T = pressure.

If ideality is assumed, values of γ_i, f_i, Φ_i and ϕ_i can all be approximated to 1. If this assumption is invalid, the specification of an appropriate equation of state can provide values for γ_i, f_i and ϕ_i. In this example, it will be assumed that if no equation of state is specified, the system can be approximated to ideality. The Poynting correction factor is approximated to 1 except where a high pressure is used. Provided some heuristic notion of the definition of "high" can be made, the use of the factor can easily be inferred from the pressure of the unit being modelled.

```
relation(k_value, Unit-Val) :-
    domain - _,
    variables - [Unit, Form, Bindings],
    bindings - [C = components $of Unit],
    active_code - (
         select_eqn_type(Gamma, vapour_activity_coefficient, Unit, C),
         select_eqn_type(Phi, fugacity_coefficient, Unit, C),
         select_eqn_type(Poynting, poynting_correction, Unit, _),
         select_eqn_type(PartPhi, partial_fugacity_coefficient, Unit, C)),
    return_form - (set_of(
         (Gamma * Phi * Poynting *
         (vapour_pressure $corresponding_to components-C $of Unit))/
         (PartPhi * (pressure $of Unit)),$over C)),
    return_type - expression,
    slots - [is_a - constraint].
```

Figure 5.8: Definition of K-value Relation

The definition of the relation for K_i is shown in Figure 5.8. The terms in the return form which are undefined and are thus left as Prolog variables are each represented in the active code by a select_eqn_type call. When the specific local form of the relation is expanded, the inferred form is generated. In the case where no equation of state has been specified, the select_eqn_type calls return a value of 1 for the undefined terms as shown in equation 5.11. The non-ideal (but low pressure) situation is shown in equation 5.12 where the undetermined terms have been replaced by the non-ideal variables. These can simply be referring to slots, or, as in this case, to further relations. An example showing the full expansion of the expression is given in Appendix C.

$$\frac{1 \times 1 \times 1 \times P_i^*}{1 \times P_T} \tag{5.11}$$

$$\frac{\gamma_i \times f_i \times 1 \times P_i^*}{\phi_i \times P_T} \tag{5.12}$$

In this manner, a single relation can be defined to represent an equation in its most general and necessarily rigorous form. The approximations of the general form which can be made can be incorporated into the relation along with the required assumptions, where such definition is possible. In the above example, the definition of the single relation for the distribution coefficient, K_i, removes the requirement for separate definitions for ideal, non-ideal and high pressure situations.

5.5 Interactive Model Modification

Section 5.4.2 discussed the representation of the equivalent of a unit model library. High level models have been defined which are selected to describe flowsheet items based on the context of the item and its specifications. The particular form of the selected equations is also inferred locally at the time of generation.

Combining the two aspects of model selection provides a tool which supplies the designer with a model reflecting the level of detail in the design and the specifications placed upon it. It can be argued that this amount of reasoning constitutes little improvement over conventional flowsheeting software where the designer selects the high level model to be used. The selection of such models is a reasonably expert task, since several models can be available for one unit operation. It may be, for example, that several distillation routines exist, each with a particular specialisation. Automatic selection of the high level model ensures its consistent and correct use.

Low level reasoning about the form of individual equations provides a significant advance over conventional representations. In the approach described here, complex models (e.g. non-ideal) are only used when appropriate data is available. In existing flowsheeting packages, this requires either a range of high level models of different degrees of complexity, or one model requiring a large number of parameters which may not always be available or significant.

It is recognised, however, that while the two facets of automatic model selection provide an important benefit to the designer, it is highly unlikely for such a system to be able to provide exactly the model required for every application. As discussed in 5.4.4, it is not always possible to define rules for the automatic selection of the low level form of the equations. It is also common for a designer to be trying to calculate a value for a particular property. This would require knowledge of the models available and the specifications to make to invoke the correct model involving the desired property. DESIGN-KIT [14] provides a mechanism for such situations. The user can specify a property to be calculated, in their example, effluent composition. The inference method identifies the equation in the data model of the object under consideration containing the desired property. If the equation is fully specified, the value of the property can be calculated. If not, further equations are added containing the required unspecified variables of the original equation. This analysis and expansion, however, must be terminated by the user if a fully specified model cannot be inferred. Specifying the boundary of the problem is, effectively, defining the high level model, e.g. if the boundary is the limits of a reactor, then the high level specification of a reactor model will suffice. The advantage is that the model developed will contain the required property.

This example illustrates the desirability of being able to specify what is required of a model. It is also desirable for the designer to be able to modify models to provide the required mathematical description. This encompasses the ability to specify the particular property required of a calculation.

The following sections describe the methods whereby the designer can interact with the model generator. This extra functionality is a more significant increase in flowsheeting and modelling ability than automatic model selection. The designer can now define the model required for a particular task. The two aspects of model selection provide separate foci for interaction. A third is provided by considering the relationships existing between nodes in the design graph.

5.5.1 Modification of High Level Models

Increases in functional detail imply an increase in detail of the model, as discussed in Section 5.4.1. The mechanism for inferring the appropriate parts of the model divides the model into several sections, reasoning about each one separately. The model corresponding to the level of functional detail closest to the process unit in question is incorporated if its conditions are met. If the conditions are not met, levels of increasingly less functional detail are considered until an acceptable model is located.

Since each aspect of the model is considered individually, i.e. mass balances, energy balance, pressure balance, etc., any combination of the parts is valid. For interaction with the user, therefore, each part of the whole model can be modified by selecting any of the available models under a particular heading. For example, the available mass balance models for a distillation column, in descending order of complexity, are: a plate to plate model, a Fenske-Gilliland-Underwood model, a split fraction model and a simple overall component balance. If a different mass balance model was required, any of the above would be acceptable.

The symbolic description of model usage includes the conditions required for a model to be considered valid. This means that the modification procedure can be presented to the user in an understandable format by reasoning about the model objects. When the user selects which of the five parts is to be modified (reaction, composition, energy, flow or pressure) the inference mechanism locates all models under the selected heading appearing in the model objects of corresponding (and less) functional detail. For example, if a distillation column is being modelled, model objects corresponding to distillation, vapour-liquid equilibrium separation, separation and plant are considered (see Figure 5.7). These models are then presented to the user for selection.

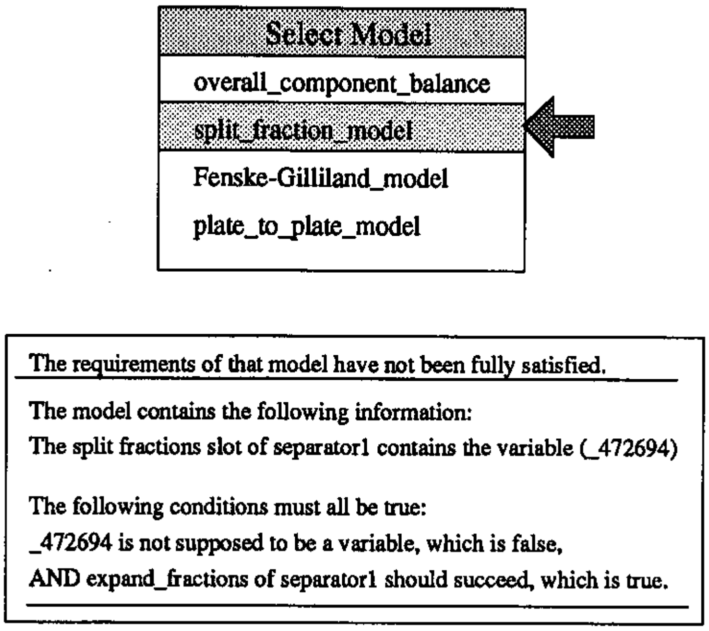
The selected model may not have the requisite information for its use, so it cannot be accepted without checking its corresponding guard conditions and assumptions. If all conditions are met, the model replaces its equivalent in the list of equations contained in the model being modelled.

When the conditions are not met, the reason for not including the chosen model must be presented to the user. This is achieved by interpreting each call in the guard and displaying the result. Figure 5.9 shows an example of an application to distillation. The user has requested a split fraction model which cannot be used because no split fractions have been specified. The unacceptability has been displayed in the window as a breakdown of the information in the model (which correspond to an interpretation of the `required_info` slot in the model object) and the individual interpretations of each guard.

The user is then able to return to the process description and provide the information necessary in order to implement the desired model.

The selection of a valid model results in a report of its acceptance.

The selection of the new model has been achieved within the framework created for automatic model selection. This implies that any model which is modified is still consistent and complete, because the checks required for automatic selection must also be satisfied by the modified model.



The requirements of that model have not been fully satisfied.

The model contains the following information:
The split fractions slot of separator1 contains the variable (_472694)

The following conditions must all be true:
_472694 is not supposed to be a variable, which is false,
AND expand_fractions of separator1 should succeed, which is true.

Figure 5.9: Example of High Level Model Modification for Distillation

5.5.2 Modification of Models at a Low Level

The interactive modification of the low level form of models is based on the low level reasoning about the form of equations. The aspects of the equations which can be modified are the terms which have been reasoned about. As discussed in Section 5.4.4, it is sometimes difficult to reason about the application of certain equations or terms in equations, particularly approximations. In such situations, it is safer to apply assumptions thereby instituting the associated approximation, than to expand the general expression in full. The approximated model will provide solutions, in most cases, which are close to the more rigorous solution, but involving significantly less computing effort.

In many situations only an approximate solution is required. For example, there is little point in applying the full, general K-value expression (in equation 5.10) as part of an evaluation of Fenske's equation, which is itself an approximation, in that it assumes constant relative volatility.

The selected equations, whether they be approximations or alternative empirical correlations, may not be adequate for the description of some problems. For example, K-values based on ideality may be insufficient for the modelling of a flash vessel, requiring modification of the model to incorporate non-ideality. It is of value, therefore, to be able to interact with the reasoning mechanism which selects the form of an equation.

The symbolic representation of equations as relations allows reasoning about their structure, in this case, the `select_eqn_type` calls. The calls are stored in the active code of the relation and contain a reference to the property they represent, e.g. `fugacity_coefficient`. The reference is also used to locate the appropriate Prolog goals which are used to establish the form of the term in the equation. By calling the Prolog clause, the current value can be established. Interpreting the remaining goals corresponding to the same term reveals the different values that the form of the term can take and also the conditions required (if any) to invoke them.

Conditions may have to be met if the alternative form is to replace an approximation. In cases where an equivalent choice exists, however, there may be none. For example, the non-ideal terms of the K-value expression can only be evaluated if a suitable equation of state has been specified, which constitutes a condition on the selection of this form. In the case of alternative equations of state, there may be no sound basis for selection and thus no conditions to satisfy.

The conditions involve the checking of slots and relations. This provides a simple syntax for reasoning about the conditions and for supplying any missing values. If unsatisfied conditions involve properties of objects, it is a straightforward task then to edit the object appropriately. If a Prolog goal fails, however, it is more difficult to inform the user of the correct action to take.

To illustrate the modification procedure, consider the alteration of a vapour liquid equilibrium calculation described by the equation:

$$y_i = K_i.x_i \tag{5.13}$$

When the decision has been taken to modify the model at a low level, the relation describing the model is reasoned about. The evaluated instance of the model is not used. It represents one modelling option which has been explored and remains unaltered as a record of the model development. The generic relation describing the high level model is used to convey two points to the user: the facets of the equations which can be expanded, e.g. the equilibrium relation contains a K-value expression which can be expanded as in equation 5.10, and the facets which can be modified, i.e. ones corresponding to `select_eqn_type` calls.

In this example, the option is taken to expand the K-value relation. The same options are again presented. The relation can be further expanded to vapour pressures, or the non-ideal terms in equation 5.10. Selection of the inclusion of fugacity coefficient in the K-value expression instigates the reasoning described above. The current value of the approximation, 1, is displayed along with the fact that an equation of state is required. If an equation of state is supplied, the form of the subsequently generated model includes the non-ideal terms in the K-value expression. The terms are expanded according to the specified equation of state.

5.5.3 Refining Model Detail in a Simulation

Section 4.3.4 described an approach for reducing the number of structural alternatives required to be stored and therefore maintained. During design, models are constructed at different levels and combined across several levels, so it is important to be able to use related models in this manner. Rather than imposing a modelling philosophy on the designer, this option aims to provide a facility which accommodates normal practices.

Figure 4.13 shows an example where two distillation units have been described at one level and then expanded to two different distillation alternatives each at a level of greater detail. The hypothesis is that, instead of creating four complete flowsheets representing the four alternative combinations, the designer would evaluate each individual process separately before considering interactions. The effect of the proposed alternatives on the complete flowsheet would then be assessed by including the new, more detailed model in the higher level flowsheet. The figure is not a particularly good example of this, but the distillation units are intended to represent a subsection of a large process.

The design graph relationships which have been discussed so far are the notions of refinement and parts. Refinement links flowsheets at different levels of detail. The detailed nodes containing the distillation column alternatives are, therefore, refinements of the flowsheet at the top level. The refinements are, however, incomplete in that they do not contain a description of the whole flowsheet. The concept of parts is used to identify the evolution of individual items in a flowsheet, thus the more detailed operations within the low level nodes are each considered to be parts of the high level distillation operations.

The concept of parts has been used to implement a facility allowing the inclusion of a low level model in a higher level flowsheet. The low level model must completely incorporate the structure and function of the high level operation (see Section 4.3.3 for details of how this is established). The streams which correspond between the two levels are used as a guide, since any stream entirely within the more detailed description are irrelevant to the more abstract flowsheet.

A model is constructed by temporarily associating the streams from the specified operations at the abstract level with the detailed operations replacing them. The model is generated as described in Section 5.4.2, but now the equations representing the detailed operations have been incorporated into the model of the flowsheet at the level of greater abstraction.

To instigate this facility, the designer selects an option from the modelling menu which then presents a choice of the lower level nodes which include the detailed enhancement of the operation being modelled. Any flowsheet models subsequently created will include the description of the refined operation until the selection is reversed.

To illustrate this capability, consider the simulation of a simple distillation column. This can be modelled individually, by, for example, a Fenske-Underwood-Gilliland model or a plate-to-plate model. However, it may be that the model does not correspond to known data about the column. A new node can be added to the design graph describing the column as a series of plates, a condenser and a reboiler, each a separate object. The generation of the model may use the general flash model for each plate. The combination is effectively the same as the general plate-to-plate model already implemented, so no advantage has yet been achieved. By modelling the plates separately, the model of each plate can be edited, as described in Sections 5.5.1 and 5.5.2, by altering, either the type of flash model used, or a low level facet, such as the equation of state. Once this model is functioning satisfactorily with respect to the known data, it can be included in the high level flowsheet.

In the current implementation, the designer selects the menu option indicating a change in the model of the distillation column. A further selection instigates the modelling of the column by its parts. The current node object is accessed to locate its refined nodes. An investigation of these nodes reveals those that have parts corresponding to the distillation column. The choice is then presented to the user. In this case only one option is available, that of the plate, reboiler and condenser objects. The single feed to the distillation column is then linked with the stream connecting to the feed plate object (this is merely an instance of a plate object with an extra input stream). The column feed stream object is then temporarily changed to indicate that its sink is the plate object. The output streams are similarly altered, but by changing their sources.

A model of the flowsheet can then be generated which now includes the plates in place of the distillation column.

The example above illustrates the power of this modelling facility. Not only does it reduce the amount of data to be stored (see Section 4.3.4), but it also allows a further degree of model manipulation. The generic model was insufficient to describe the operation, so it can be broken down into lower level operations which can be altered individually to provide the required mathematical description. Being able to model operations separately before inclusion in a flowsheet reflects a natural design practice and is, therefore, a valuable tool for modelling.

5.6 Model Results

The presentation and interpretation of model solutions is important for determining what, if any, modifications are required of the model. It is useful to be able to compare results from previous simulations and also to view the mathematical representation of the model. These options have been incorporated in extended methods.

Results are presented in three tables. The first displays the high level relations which define the model. This is useful for comparing models where the basis for the solution must also be evaluated.

The second table is a stream table which is an accepted method for presenting stream data for a flowsheet, so the table and the flowsheet should be viewed together. A graphical display of flowsheets has not been incorporated, but in principle, should be available. In the stream table, the rows correspond to chemical components, and columns to streams. The entries can be flowrates or mole fractions. Total flowrates, temperatures and pressures can also be included if data is available.

Stream tables do not include other stream properties or attributes of process units, such as dimensions. These values are displayed in the third table. In principle, these attributes can be divided into groups corresponding to process units and the groups displayed individually.

Presentation of information to the user is an important consideration in the development of an acceptable product and constitutes a research topic on its own. For example, stream tables are an accepted means of displaying flowsheet information. However, grouping all data for a stream, including flows, vapour pressures, etc., may be more suitable in a different situation. The conclusion of this discussion is that data can be presented to the user in a range of styles, textual and graphical, so the user should not only be able to define problems in different ways, but should also be able to examine the solutions from different viewpoints.

It is important to be able to review the mathematical model being used. This provides a means of determining which terms and equations in the model should be modified. Westerberg and Benjamin [74] suggest useful properties of such a tool. For example, the documented model could be structured as a user's manual with chapters and sections. This enables access to individual parts of the model through a table of contents.

A tool has been implemented to display models in this manner. The "specialise" form of a user-defined relation contains the expanded equations and a list of the bindings to object attributes. This relation is interpreted symbolically in order to write a formatted input file for LaTeX, a document preparation system[75]. The hierarchical decomposition of equations described in Section 5.2 is used to structure the document, the different levels corresponding to chapters, sections, subsections, etc. A table of contents is generated automatically.

Specifications are displayed as part of the model, but not calculated values. Three example documents are shown in Appendix D.

5.7 Summary of Modelling Functionality

A symbolic representation for modelling equations has been developed in which equations are defined as relationships between attributes of process unit objects. Equations can also be related to each other in a hierarchical structure where individual terms in an equation can be defined by further expressions. For example, a heat balance equation contains enthalpy terms, where enthalpy is defined in a separate expression. This structure, and the ability to express equations symbolically, are useful properties in model development.

A mechanism has been developed for automatically generating flowsheet models from high level definitions such as "component balance" and "heat balance". The details of model definition should be determined by an expert modeller along with the rules for their selection for flowsheeting applications.

A designer, however, uses models rather than creates them. The symbolic representation allows reasoning about equations to provide a flexible way of using models without necessarily having to create new ones. Models are created automatically which the designer can modify for specific applications. Access to the high level selection method enables the designer to select alternative definitions for individual parts of a model. This alteration of the model is constrained by the structure developed by the modeller. At a low level, the designer can select different definitions for equations or individual terms.

Models are associated with the object representing the subject of the model, e.g. a node for a flowsheet or an individual distillation column. A symbolic description of any model is available to the user for display or documentation. Solutions of any previous model can also be reviewed.

Chapter 4 described the development of tools for supporting generation of process flowsheets in a hierarchical manner and maintenance of the synthesised hierarchy. Chapter 5 addressed the automatic generation of flowsheeting models and their modification. The tools developed have been integrated in an network of extended methods which are shown schematically in Figure 5.10. The groupings represent parallel calls within the indicated extended methods. The *analyse flowsheet* method can be accessed from both *analyse* and *topology*.

Most of the options for topology management including movement in the design graph have been described in Section 4.3.2. The options under the *analyse flowsheet* heading of *up level, down level, list level, list section, switch section* and *copy equivalent section* are also described there.

The analysis option of *analyse components* displays a selected component as a table of parameters obtained from a database. The *analyse stream* call displays stream attributes, enables editing of their specifications and allows calculation of some properties. The operations under *analyse unit* include:

- *edit object* which displays the unit and allows editing of specifications. The associated call *display object* displays the unit but does not allow editing.
- *model object* which evaluates the current model of the unit. If no model is specified, one is generated. Models provided by other design modules will

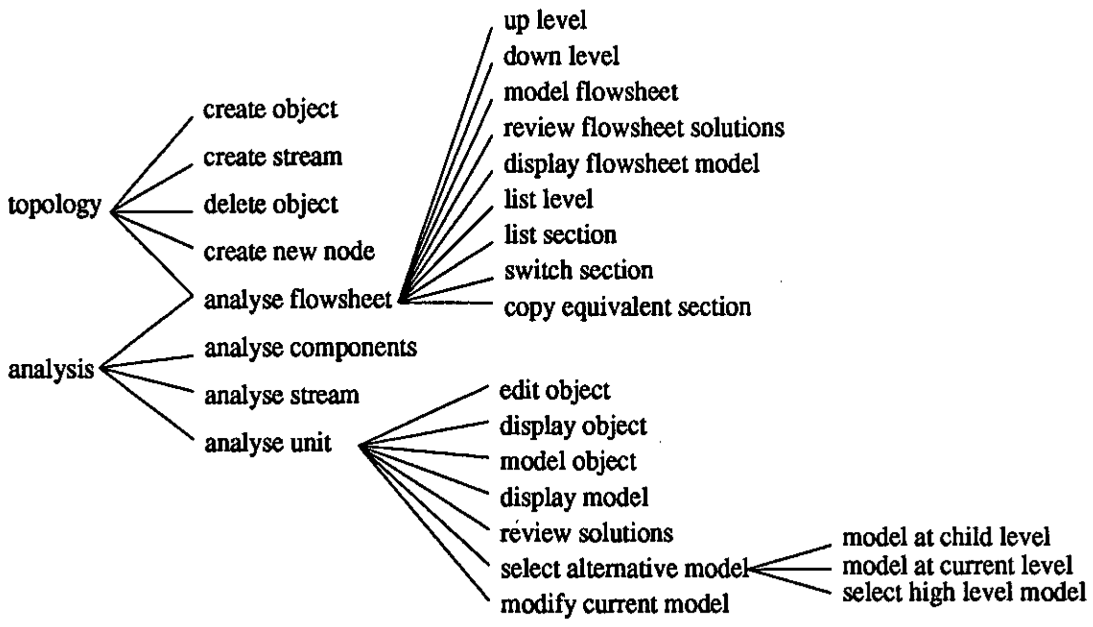


Figure 5.10: Schematic Representation of Modelling Extended Methods

be evaluated without modification (see Section 6.2). The equivalent call of *model flowsheet* in the options of *analyse flowsheet* performs the same task for a whole design node, but does not allow evaluation of separately generated models which may be inconsistent with those generated locally.

- *display model* generates a formatted LaTeX document [75] describing the model in its expanded form and its general equation form. Dependent equations are related by sections and subsections in the text. Variables and values are identified by their associated slots and relations. Three examples are shown in Appendix D. The flowsheet command, *display flowsheet model* does the same for a flowsheet model.

- *review solutions* allows browsing of the solutions of all evaluated models of a unit, or in the case of flowsheets, *review flowsheet solutions* does the same. The solutions are divided into the high level relations used in its definition, a stream table and a table of other values calculated by the model (see Section 6.3.4).

- *select alternative model* has three lesser choices. Modelling the unit at a level of greater detail as described in Section 5.5.3 is achieved by *model at child level.* Reverting to the current level is performed by *model at current* *level.* The high level modification of models discussed in Section 5.5.1 is accessed through *select high level model.*

- *modify current model* performs the low level model modification described in Section 5.5.2.

These options provide access to a range of modelling facilities which allow modelling of flowsheets and individual units. Automatically generated models are available for high and low level modification to represent specific situations. Solutions of models can be reviewed and the modelling equations can be documented. Appendix F shows an example of the different options in use.

Chapter 6

Modelling of Design Strategies

This chapter discusses strategies used in the course of design. Many of these strategies are not well defined. An integrated design environment requires the support of different techniques to evaluate the state of a process design. Evaluation may concern the physical ability of a design to meet given specifications, its economic feasibility, control and operability, safety and layout. Many of these evaluations are procedural, involving detailed algorithms. In other cases only a statement of high level goals is possible with little detail to indicate how the goals should be achieved.

The emphasis of this work has been on numerical flowsheet evaluation, incorporating algorithms for solution of equations as well as strategies for model formulation. There are few aspects of this which can be considered as procedurally ill-defined. However, the inclusion of such a modelling tool in a large environment requires consideration of how and when it is to be accessed during design, which is not well defined.

The example which have been considered here are as follows:

1. Overall flowsheet synthesis. This can be represented as a hierarchy of high level tasks with great flexibility required to achieve them.

2. Design of individual unit operations. This can also be described by a hierarchy of tasks.

3. Formulation of problems for solution by different solvers, a task which is well defined and mostly algorithmic.

These three examples display a range of representational issues. Their implementation must consider how much can be productively achieved automatically and how much interaction with the user is required. The following sections describe the implementation of these examples with respect to the efficient representation of the procedures while providing the required interaction with the user.

6.1 Modelling Process Synthesis

Section 1.2.2 described overall process synthesis as a routine operation in that it can be characterised by a procedure or algorithm. Algorithms and heuristic procedures have both been used in other work to create synthesis tools, as discussed in Section 1.2.2. These tools have only limited criteria for evaluating generated processes. An economic assessment is the normally only basis for selection of the optimal flowsheet in such tools. Other analyses for e.g. control and hazards, follow once a base case design has been accepted. The hierarchical decomposition of decision levels proposed by Douglas [7] provides a high level framework for synthesis. By implementing this approach in a flexible manner, analysis by a range of evaluation modules, such as control and hazard analyses, becomes possible. The work presented here demonstrates this principle by accommodating the modelling facilities described in Chapters 3, 4 and 5 within a synthesis framework.

The implementation of Douglas's decision level approach requires consideration of the user interface to determine the level and type of interaction required with a designer. The hierarchical decomposition of synthesis suggests a number of goals which must be attained at each level. The goals are achieved by performing the tasks in an order depending on the design and the designer. Within each level the designer may also want to evaluate the process by means of different models. This implies wide integration of tools. Therefore, the synthesis tool cannot be a single self-contained procedure but must be available throughout process development. This is true for all other evaluation tools.

Section 4.3 identified the requirement for the designer to be able to move up and down in the graph of flowsheets. This reinforces the proposed model, whereby the synthesis framework is applied to a flexible central representation of the design. The fact that synthesis is concerned with enhancing the detail of the models does not mean that movement within the hierarchy is not required. On the contrary, many decisions require the assessment of the design at more than one level. The decision levels, therefore, do not correspond directly to levels in the design graph.

From the designer's viewpoint, the implementation must allow selection from the range of available tools within the constraints applied by the synthesis hierarchy. For example, it is desirable to restrict the type of process item that can be selected at the early stages of synthesis. Some structure to the synthesis procedure must be visible for the designer to determine the steps which are appropriate, and help should be available if this is insufficient. In situations where a particular piece of information is required the user can be informed what the missing information is and instructed how to provide it.

The recognition that synthesis is a procedure with intermediate aims but no strict path through the network of decisions led to the work on extended methods described in Section 2.4.3. The existing methods in CLAP were intended for strict, well defined procedures and were, therefore, of limited use. Extended methods allow the specification of general goals and, where applicable, the order of execution. The implementation of the synthesis procedure in extended methods is constrained by placing guards on the calls, allowing progress only when certain conditions have been met. Loopback points provide a method of specifying where missing information can be obtained if a guard fails.

An example extended method is shown in Figure 6.1 which represents the decomposition of synthesis into the decision levels of Douglas. In principle, different extended methods can be defined for batch operation and solids processing, but have not been implemented. Douglas suggests procedures for such processes. The extended method in Figure 6.1 refers to continuous fluid processes.

Each call in the calling sequence is to a further extended method incorporating the decisions required at the associated level. As described in Section 2.4.3, each member of the dotted list representing the calling sequence has corresponding entries in the dotted lists of "guards", "assertions" and "loopback points". In this example, the guard corresponding to the input output structure decision level prevents its execution if the design does not have a specified *process_chemistry* object, which is stored as a relation to the "design" object. The *process_chemistry* object must also have a value for reaction path. If these conditions are not met, the corresponding loopback point indicates that the information can be obtained, or specified in the extended method *collect_input_information.*

On successful completion of a call, the corresponding "assertion" can be made.

```
extended_method - continuous_fluid :-
        variables - [input_output_structure..
                     recycle_structure..                                  Calling Sequence
                     reactor_system..
                     separation_system..
                     energy_integration, ]

            ($check relation-Design-(process_chemistry-Chem)
                      $and $check slot-Chem-(pathway-Path)..
            check_boiling_point_data..
            ($guard - full_reaction_spec)..                               Guards
            ($guard - separation_spec)..
            $null..
            ($guard - utilities_established)),

                    [full_reaction_spec - .......
                     separation_spec - .....                              Guard Macros
                     utilities_established - .....],

        $set slot-design-synthesis_level-(recycle_structure-start)..
        $set slot-design-synthesis_level-(reactor_system-start)..         Assertions
        $set slot-design-synthesis_level-(separation_system-start)..
        .....

                synthesis-collect_input_information..
                continuous_fluid-input_output_structure..                 Loopback Points
                continuous_fluid-recycle_structure..
                continuous_fluid-reactor_system..
                .....

                     Design,                                              Object of Interest
                $null,Status,Surface,Info,Display].                       Other Slots
```

Figure 6.1: Extended Method Representing the Synthesis Procedure of Douglas for Continuous Fluid Processes

The assertions here simply inform the design object illustrated in Figure 4.7, of the position within the synthesis procedure.

In order that the synthesis procedure may interact with other tools, a general extended method has been defined which contains calls to modelling and evaluation facilities as well as synthesis decisions. These general tools are then available whatever stage of synthesis has been reached. Methods for allowing the creation of flowsheets i.e. unit specification, connections, and flowsheet editing, have also been provided along with analysis facilities e.g. numerical evaluation and editing and reviewing of specifications. These options are all straightforward operations and, therefore, can be written as standard CLAP methods or Prolog clauses.

The decisions associated with each level in Douglas's hierarchy can be divided into two classes:

1. those requiring action
2. those corresponding to advice.

For example, before the recycle structure can be judged complete, a reaction section and a separation section must be defined. This corresponds to a decision requiring essential action which must be carried out before proceeding to the next decision level.

Decisions involving non-essential action are heuristic evaluations of alternative courses of action which can then be presented as advice. For example, in the early stages of synthesis, the designer should consider whether or not the feeds to the reactor require purification. The heuristic evaluation may suggest purification of the feeds, but the designer should still be able to evaluate alternatives. The designer may decide that this decision is not applicable for a particular design and not even consider it. Specific action is, therefore, not essential for this case. The designer can accept the heuristic decision or implement an alternative.

Such choices can be evaluated interactively by presenting them to the designer as a menu of choices at the appropriate point in the synthesis hierarchy. The designer is then aware of the decisions which can be made even if none of them are selected. Those which are selected provide advice indicating heuristically a potentially optimal process. The designer can accept this or evaluate any desired alternatives.

The extended method for general process evaluation accommodates different evaluation tools as well as non-essential decisions in a framework of high level goals. Part of the extended method is shown in Figure 6.2. The user obtains access to the non-essential decisions along with other tools in a "parallel" call, as described in Section 2.4.3. The essential actions are checked once the parallel call is exited, i.e. the designer determines that all that can be achieved at the current level has been completed.

The implementation utilises Prolog unification to invoke the decisions of the appropriate synthesis level while providing access to all other tools.

```
extended_method - synthesis :-
    variables - [current_call..
                 call_information(Syn, CallName, CallDecisions,
                                                    CallChecks)..
                 pcall(CallName,
                          topology..
                          analysis..
                          help..
                          collect_input_information..
                          store_to_file..
                          CallDecisions)..
                 CallChecks..
                 final_check,
/** GUARDS **/
         ($check slot-design-(synthesis_level-(Syn-_))
                $and calls_and_guards(Tech, Seq, Guards)
                $and corresponding_term(Syn, Seq, Guard, Guards))..
          where(Node)..
          (        (Syn \== batch_v_cont)..*
                   (Syn \== batch_v_cont)..
                    $null..$null..$null..$null)..
           Guard..
          ($check slot-design-(synthesis_level-(synthesis-complete))),
/** MACROS **/
          :
/** ASSERTIONS **/
           ($set slot-design-synthesis_level-(Syn-start))..
           ($set slot-Node-synthesis_level-Syn)..
          :
```

*\== is the Prolog inequality test.

Figure 6.2: Part of an Extended Method for General Process Evaluation

The first call in the calling sequence in Figure 6.2 informs the designer of the current level in the synthesis hierarchy by checking the pertinent slot in the design object. The three parts of the corresponding guard locate this information and, using the generic synthesis method in Figure 6.1, identify the guards appropriate to the current level. The guards are instantiated in the variable `Guard` which appears further down the list of guards, corresponding to the checks for essential action. The unification of the term provides the correct guards for the synthesis level.

The second call, `call_information`, identifies three labels associated with the synthesis level. `CallName` is used to provide a tag for the parallel call which follows. `CallDecisions` refers to an extended method containing the decisions involving non-essential action. This variable is unified to a term in the parallel call. `CallChecks` corresponds to an extended method providing verification of the essential actions. This is unified with the call following the parallel call. The guard on `call_information` finds the name of the current node in the design graph. The "assertion" invoked on completion of the call sets a slot in the design node detailing the synthesis level which was applied there. This is updated if a new synthesis level is subsequently applied.

When moving between levels of the design graph, it is important to have access to the decisions relevant to the node currently being investigated, otherwise inconsistencies could occur. For example, if the synthesis procedure has reached the separation system structure level, a review of design nodes evaluated at previous synthesis levels cannot be allowed access to the unit operations available at the separation system level. This would invalidate any assumptions used in their modelling. Associating the design nodes with a decision level also provides a means of reviewing the decisions which were available at earlier stages in a design.

The third term in the calling sequence is a "pcall"; a list of operations which can be performed in "parallel". The name of the "pcall" is unified with the variable `CallName` which is used as a label for the menu of options and as a loopback point if the call corresponding to the essential actions is not completed successfully. This menu provides access to the main evaluation tools. The menu contains the following options:

- `topology`. This is an extended method providing options for adding and deleting process items and streams from flowsheets. The method also includes procedures for adding nodes to the design graph and moving between them. This call has a guard which prevents its use when a decision is being made between batch and continuous operation.

- `analysis`. This is an extended method allowing analysis of flowsheets, i.e. movement in the design graph, displaying and modelling of all or part of flowsheets. This option provides access to some of the same operations available under `topology` because they are appropriate to both tasks. Individual units can be analysed separately, for editing specifications, modelling and reviewing models. This option also allows reviewing of streams, with access to their specifications, and displaying chemical component properties. As with `topology`, this call is blocked during the batch versus continuous decision.

- `help`. This is a Prolog call to aid in the completion of a synthesis level. The checks on essential action are invisible to the user, so the help option attempts to describe the information that is required. The call accesses the extended method containing the checks and displays a menu containing the calls in the calling sequence. Each call corresponds to an individual goal for a particular decision level and has been given a name which describes this aim. On selection of a goal, its associated guards in the extended method are interpreted and presented to the user as a list of conditions to be met.

- `collect_input_information`. This is an extended method providing access to fundamental objects describing the overall design, e.g. products, purity, process licenser, etc., the process chemistry and the site.

- `store_to_file`. This is a Prolog call which stores all objects used in the construction of the design graph including nodes, unit operations and streams. All user-defined relations are also stored. This allows interruption and recommencement of the design procedure.

- non-essential decisions - an extended method which corresponds to the current level of synthesis. The method typically includes a parallel call of decisions and heuristic evaluation options for the current level.

The fourth call in Figure 6.2, corresponds to an extended method detailing the essential action which should be taken for this synthesis level to be deemed complete. The name of the method is supplied by unification with the `CallChecks` variable in the `call_information` call. Its guard is also obtained by unification.

The last call, `final_check`, is a Prolog goal which always succeeds. Its guard, however, checks the `synthesis_level` slot of the design object for a value indicating that the synthesis procedure is complete. If this is true, the extended method exits, which has the effect of advancing one level in the synthesis hierarchy. If not, the corresponding loopback point returns the procedure to the first call in the method, i.e. `current_call`.

As a whole, the extended method in Figure 6.2 provides access to evaluation tools and information modules, which, in this case, include steady-state modelling and synthesis decisions respectively. The synthesis procedure of Douglas provides a framework for developing the design. At any particular level in the synthesis hierarchy, a constraint is imposed on what can be achieved, primarily by restricting the description of a flowsheet to process items relevant at that level. For example, at the input-output structure level, only objects describing plants and storage are available. At the recycle structure level, additional objects describing reaction sections, separations, stream division, mixing, etc. are provided.

This constraint encourages the designer to complete decision levels. Only a limited amount can be achieved with a restricted number of process items, thus requiring the designer to exit the "pcall" in order to advance. The subsequent verification determines whether the level is complete or not, indicating the ability to proceed or a requirement for more information.

Decisions which do not require action are implemented within the "pcall" along with other evaluation tools. This allows the designer to adopt an opportunistic approach to process development, i.e. moving between the tasks as necessary. As an example, part of the extended method describing the decisions at the input-output structure level of the synthesis hierarchy is shown in Figure 6.3.

```
extended_method - input_output_decisions :-
    variables - [decision_level_intro(input_output_structure)..
                 pcall(input_output_options,
                           no_of_product_streams..
                           purification_of_feeds..
                           by_product_treatment..
                           purge_requirement),
/** GUARDS **/
               $null..
               (    ($guard - ...)..
                    ($check ...)..
                    ($check relation-Tech-(process_chemistry-Chem)
                        $and $check slot-Tech-(feeds-Feeds)
                        $and $check slot-Chem-(by_products-Bys))..
                    :
```

Figure 6.3: Part of an Extended Method Representing Non-Essential Decisions

The first call provides a short piece of text describing the current synthesis level. The following "pcall" contains four options which, if selected, invoke pieces of code which can be methods, extended methods or Prolog clauses providing heuristic evaluation of synthesis problems. For example, in the figure, `no_of_product_streams` tabulates all chemical components which are present into products, by-products, reactants, intermediates, etc. and uses their boiling points to suggest a number of potential output streams from the process. The call `purification_of_feeds` requires additional qualitative information from the designer, which is obtained by a series of questions. Heuristics are then used to determine whether or not the feeds should be purified and where they should be fed to.

In general, the guards on the decision calls are not inhibitive and protective as for essential actions, but are helpful for allowing presentation of appropriate information. For example, the decision about `by_product_treatment` is only relevant if there are by-products identified. This is normally the case, but if a problem had no by-products the user should not see that option. The guard, therefore, removes it from view. A better example of this is given in Appendix F, where options for evaluating design of a distillation column are presented as the relevant data becomes available.

When the designer can achieve no more at a particular level, the termination of the parallel call, i.e. selecting "finish" on the menu, instigates the checks for essential action. This is performed by a separate extended method associated with the decision level. Figure 6.4 shows an example of an extended method for verifying the completion of the input-output structure level.

The calling sequence consists of a list of Prolog goals, all of which always succeed. They provide a high level statement of the checks being performed, which can then be used by the help facility described above.

The tests are performed by the guards associated with the high level goals in the calling sequence, i.e. here, the guards inhibit progress. For example, in Figure 6.4 the third goal, `output_streams_classified` has a guard consisting of five tests:

1. `streams_classified` which ensures that the chemical species in the plant have been classified into products, by-products, etc. and a number of product streams has been identified.

```
extended_method - input_output_checks :-
    variables - [units_for_purpose..
                 input_for_all_feeds..
                 output_streams_classified,
/** GUARDS **/
         (current_level(Objects, Streams)
             $and plant_exists(Objects))..
         (($check relation-Tech-(feeds-Feeds))
             $and current_system_inputs(Streams, Inputs)
             $and dotlength(Feeds, FL)
             $and input_length_chk(Inputs, FL)
             $and streams_match_materials(Inputs, Feeds, feed))..
         (streams_classified
             $and current_product_streams(Streams, Outputs)
             $and correct_number_of_outputs(Outputs)
             $and $check relation-Tech-(products-Products)
             $and streams_match_materials(Outputs, Products, product)),
         :
```

Figure 6.4: An Example Extended Method For Verification of Synthesis Level Completeness

2. `current_product_streams` which locates the streams in the plant which constitute outputs.

3. `correct_number_of_outputs` which checks that the specified number of output streams matches the calculated number. If fewer outputs are specified than are required, additional streams must be provided. If there are more than the required number, a warning of excess is given.

4. A CLAP call which checks the `products` relation of the "technology" object `Tech`. The object represents the original design remit and the relation contains a statement of the products the plant aims to produce.

5. `streams_match_materials` which ensures that the materials that have been specified as products are present in the output streams.

In general, the associated loopback points return execution to the "pcall" where the missing information can be provided.

In summary, the tools required for synthesis must be available throughout design along with other evaluation modules. This has been accomplished by creating a general extended method providing access to evaluation tools and synthesis decisions in the opportunistic manner required by designers. The "pcall" construct in extended methods supports this control mechanism.

The decisions in the synthesis method of Douglas have been divided into those requiring action and those providing a heuristic reduction of the search space for a base case design. The decisions involving essential action are used to ensure that a level in the synthesis hierarchy has been completed. This is achieved with a decomposition of the goals which are then written as guards in extended methods. The decisions providing advice are represented in a "pcall" in another extended method which displays the choices to the user as a menu.

The use of extended methods supports the opportunistic control mechanism used by designers in process synthesis and evaluation. Since the structure is based on Prolog, variable instantiation and unification can be utilised to provide the internal details of the general extended method appropriate to the synthesis level.

This facility is particularly useful when moving between design nodes, which have a synthesis level associated with them corresponding to the level most recently applied there. The general extended method can then adopt the synthesis level appropriate to particular nodes in different branches of the design graph. If for example, one branch is developed to the level of separation system structure, and alternative branches are only at the recycle structure level, when the designer moves between them, the decisions pertinent to the node can be made available.

An example application of the synthesis tool is shown in Appendix E.

6.2 Design of Unit Operations

Design activities can be categorised as either "routine" or "non-routine". Routine design encompasses problems which have a well defined design procedure which is ostensibly the same for any application. Design of distillation columns, heat exchange equipment and certain types of reactor fall into this category. The different tools which may be used to complete the various stages of the routine design are coordinated by the aims, and methods to achieve the aims, which are known in advance. Myers et al [5] discuss this approach in the design of distillation columns. Their method involves defining all possible courses of action and the points where choices are made in the procedure. In this way, the possible range of starting conditions and the different design considerations can all be represented and included as a path through the decision tree.

This type of approach aims to provide automatic design of unit operations, the example considered being distillation. In order to achieve this, the design of the item must be at a stage where the designer has confidence in the basis of the automatic design. For instance, in the case of distillation, once the number of plates has been decided, the design of the individual trays can be performed automatically with reasonable confidence. The crucial point is the confidence that the designer has in the calculated number of plates. Myers et al state that the starting point for automatic column design is once the type of column, type of tray, reboiler, condenser, performance requirements, etc. have been specified. However, at this stage, much of the design of a distillation column has been completed.

There is, therefore, a requirement for a strategy linking the overall synthesis procedure to the point where automatic design becomes acceptable. This section addresses an approach which extends the overall synthesis method to consider the design of individual unit operations. Specialists in design are required to formulate pertinent strategies for particular unit operations, so the work presented here only demonstrates the general principle.

The design of individual unit operations can be considered as an extension of the overall synthesis method because an opportunistic approach using different modelling methods including the evaluation tools of general synthesis is required. Extended methods, therefore, are the most appropriate representational technique. The example unit operation considered here is distillation.

The problem can be decomposed into two subproblems:

1. Identifying the most suitable point in the synthesis procedure at which the unit operation design procedure should be made available.

2. Formulating a strategy for developing a loosely defined unit operation into a detailed specification. This can then be used to complete the design automatically, e.g. by an approach such as Myers et al, or at this point the completion of the design becomes the concern of another design function.

In the case of distillation design, the starting point of the strategy is at the liquid separation level of Douglas's decision hierarchy which is detailed in Section

1.1.2. At this point, individual separations are identified and possible distillations are evaluated. The procedure begins with the calculation of relative volatilities which can be used to decide heuristically whether or not distillation is feasible. One of the heuristics described by Douglas is if the relative volatility of the components in the desired split is less than 1.1 then it is unlikely that distillation will be economical.

Assuming that distillation is economical, the next decision suggested by Douglas is to determine column conditions. For example, if possible the column should be operated at, or slightly above, atmospheric pressure. This depends on the ability to condense the tops and reboil the bottoms. It is preferable to have the tops condensed by cooling water and the bottoms reboiled by low pressure steam. To determine suitable column conditions, the bubble point of the tops and bottoms are calculated. These indicate whether or not the streams can be respectively condensed by cooling water and reboiled by low pressure steam at the chosen pressure. If they cannot, a pressure where this is possible can be calculated.

The calculations can all be performed procedurally, requiring no interaction. The decision about the selection of column conditions still lies with the designer, since some trade-off may be required between pressure and heat exchange media.

Once column conditions are fixed, the number of plates can be calculated. There are several different models with different levels of assumption which can be used. Initially, an approximate number of plates and reflux ratio can be determined by a Fenske-Underwood-Gilliland method. In the case of a binary mixture, McCabe-Thiele and Ponchon-Savarit procedures could also be used. When a number of plates has been calculated, a plate-to-plate model can be used for more accurate simulation.

This particular strategy can only be used if the specifications are on the product composition. If the number of plates has been defined, the order of the tasks is different.

The example described above identifies the important points of what can be considered an extension of a general synthesis procedure:

1. There are high level procedural goals, e.g. ensuring that distillation is going to be feasible, determination of column conditions and calculation of the number of plates and reflux ratio.

2. Evaluation of the design remains opportunistic, particularly since different specifications can result in revision of the order of the tasks. The designer can still instigate different models and modify them as required. Interaction with other tools may also be required to move between nodes in the hierarchy, for example, to create a plate model of the column.

3. Models specific to the design of the unit operation are required, e.g. Fenske-Underwood-Gilliland, McCabe-Thiele, etc., but interaction with general modelling facilities is still required. For instance, the Fenske model may be evaluated at a range of temperatures. Other modifications of the model may also be necessary.

4. Some models have a specific range of application, e.g. those applying to binary separation, and should only be presented to the user in relevant situations.

These points are similar to those identified in the development of overall process synthesis, implying that the same representation of extended methods can be used here.

The extended method for distillation design is accessed as a decision which does not require action. This allows the opportunistic use of other tools available in the extended method for general process evaluation. The models created in the evaluation of distillation can then be modified and reviewed like any other models.

The models created for distillation evaluation, particularly for calculating the number of plates, may not correspond to the decomposition of models described in Section 5.5.1. Such a situation could arise if the tools are developed independently by domain experts, as suggested here. The models will then have been tailored for a particular purpose. For example, in distillation, Fenske's equation can be used to calculate the minimum number of plates, Underwood's equation determines minimum reflux ratio and the Gilliland correlation estimates an actual number of plates.

In a flowsheeting context the same equations may be used, but in conjunction with mass and heat balance equations. The approach for automatic model selection in Section 5.4.1 was developed specifically for flowsheeting, and, while this facility is available throughout the synthesis procedure, it is not directed towards a particular design task such as estimating the number of plates in a distillation column. When models are used for such tasks, it is unlikely for them to conform to the decomposition into five subjects used for flowsheeting . The models do not have to be complete in a flowsheeting sense, i.e. they do not necessarily incorporate mass and heat balances around a unit.

For this reason, models developed for design tasks are considered as separate from flowsheeting models even though they may both contain some of the same CLAP relations. Therefore the mechanism for automatic model selection has been extended to identify groups of relations that do not conform to the five topic decomposition. When such situations are identified, the context of the model and its application are assessed to ensure the model's acceptability. For instance, if a unit is being evaluated individually, then no alteration of the model is required. If, however, it is part of a flowsheet then a model based on the decomposition replaces the non-standard one.

This is also the case if the designer attempts to modify the high level model description as discussed in Section 5.5.1. It cannot be assumed that a non-standard description conforms to any part of five point decomposition, so the whole model is replaced to ensure the consistency of modifications. When the user decides to modify a model, the current description and a recommendation are displayed. If a modification of a non-standard model is attempted, it is replaced by the recommended one, the change then being made.

Appendix F shows a worked example of the interaction of a distillation design procedure with the general evaluation method. Different models of a distillation column are created for the calculation of particular properties. Subsequent modification of the model requires a standard flowsheeting description to be generated.

Certain classes of model have a particular range of application and it is important for the designer only to have access to them in relevant situations. For example, the example implemented for distillation design includes two models for calculating the number of plates. One provides a shortcut estimate for the specific case of a binary separation. The other - the Fenske model - is more widely applicable. For multicomponent separations, therefore, the binary model should not be displayed to the user. For binary separations, both can be displayed, unless some criteria for selection can be determined.

The modelling tasks are represented as a network of extended methods, one of which is for the calculation of the number of plates. A parallel call is used to present the modelling options to the designer as a menu. The display of the options is, therefore, controlled by guards, in this case to restrict the shortcut method to binary separation only. The construction of the parallel call and its accompanying guards is shown in Figure 6.5.

```
pcall(calculate_number_of_trays,
          fenske_underwood_gilliland..
          shortcut_binary),
:
($null..
 $check slot-Sep-(components-[_,_])),
```

Figure 6.5: Guard for Selection of a Binary Separation Model

The guard on the shortcut binary method checks the `components` slot of the object, `Sep`. On calling the method, `Sep` will be instantiated to the name of the distillation column. If the value of the slot is a list with only two members (`[_,_]`) the separation is binary and the model usable.

In conclusion, strategies for the design of individual unit operations can be developed as extensions of a general synthesis procedure. A similar structure of extended methods can be used for representation., providing high level goals and a mechanism for their opportunistic realisation. A domain expert should develop the hierarchy of tasks required for a particular unit operation. Mathematical models may be developed for specific tasks in the design of a unit operation, e.g. the number of plates in a distillation column. These models should be considered incompatible with those developed elsewhere, such as flowsheeting. They can, however, be evaluated using the set of tools available, e.g. low level modification and re-evaluation with different specifications. When interaction with other models is required, for instance, in a flowsheeting context, the model should be replaced.

6.3 Interface Between Design Data and Flowsheet Solvers

In comparison with the applications of design synthesis discussed in the previous sections, model formulation requires little interaction with the designer. The interface between a specified problem and flowsheet solvers should be invisible to the user. The approach to solution is algorithmic as opposed to opportunistic. The specification of the problem and the modification of models requires a greater degree of flexibility than in model formulation. In all, the numerical evaluation procedure can be divided into four tasks:

1. Identification of the scope of the problem to be solved.

2. Generation of a mathematical model appropriate to the problem.

3. Solution of the generated model.

4. Presentation of results.

This procedure is normally iterative. Stage 4 allows a review of the model and its basis, which provides an opportunity to return to step 2 and modify the model. To accommodate this flexible approach, the high level formulation tools have been written as extended methods. The constituent parts have been written variously as Prolog goals, CLAP methods, extended methods and C routines, whichever was most appropriate.

6.3.1 Problem Scope

The scope of the problem ranges between the calculation of a single property and the simulation of whole flowsheets. A mechanism has been developed to allow the specification of the size of the problem, which depends on the application. It is important, therefore, to identify the types of calculations that are required and in what situations.

Individual units may require independent calculations of particular properties, e.g. calculation of the number of plates in a distillation column. This type of evaluation can be associated with design methods for particular unit operations as described in Section 6.2 above. Such calculations can then be presented to the designer at the appropriate stage of the process development.

In an opportunistic development of a design, however, individual properties of streams are often calculated. Streams retain a single level of functional and structural detail throughout process synthesis, and so can have a range of calculable properties associated with them. For example, valuable attributes include: bubble and dew points, K-values and vapour pressures. This information is made available to the designer in a parallel call in an extended method, allowing analysis of streams (see Section 5.7). The use of the models can be restricted to the later stages of synthesis by the use of guards. The parallel call and associated guards from the extended method for stream analysis is shown in Figure 6.6. In the example, the synthesis level must be either at the reactor system level or the separation system level.

```
pcall(stream_options,
        edit_stream..
        calculate_stream_properties)..
:
($null..
 (temp_and_pressure_known(Stream)
      $and member(Syn,[reactor_system, separation_system])))..
   :
```

Figure 6.6: Part of Extended Method for Stream Calculations

In the wider context of steady state flowsheet modelling, calculations can be performed for whole flowsheets, sections and individual units, all of which are represented in the design graph. This decomposition of flowsheets can also be used as the subject of other evaluation modules, such as control system design or hazard assessment.

Figure 5.10 shows the division of analysis tasks into four groups, including those of flowsheets and units. Selecting analysis of an individual unit defines that as the scope of the problem. Any modelling performed under this option incorporates only the specified unit. Selection of the flowsheet analysis tool, provides procedures for moving through the design graph, i.e. selecting a whole flowsheet, modelling a node i.e. again, a flowsheet, or identifying a section, as described in Section 4.3.2.

Throughout design, access to mathematical models of flowsheets and their constituent parts is required, but not in a predefinable order. Therefore, general steady state modelling tools have been implemented in extended methods. Different modelling situations have been identified:

- Steady state flowsheet modelling, described in Chapters 3, 4 and 5
- Calculation tasks as steps in a design procedure, particularly for individual

unit operations discussed in Section 6.2

- Calculation of stream properties.

Appendix F describes an example of the interaction of the different modelling facilities.

6.3.2 Model Generation

Model generation constitutes an "expert" task and should, therefore, be automated as far as possible, as discussed in Chapter 5. The user should be able to describe the design in familiar terms, that is, functional descriptions of the processes involved and specifications for their operation. The automatic model generator then determines a mathematically consistent equation based description from the process definition.

Process definition, in terms of process units and specifications, is achieved flexibly by using extended methods, as described in Section 5.7. Translating this definition into a mathematical model is achieved in two stages:

1. Inferring the high level model description, as discussed in Section 5.4.1.

2. Generating the equation set from this description.

This second step corresponds to the creation of the "specialise" form of the relation from the "constraint" form. The operation involves creating equations containing Prolog variables corresponding to the terms indicated in the constraint relation, including the expansion of summations and "for all" statements, e.g.

$$\sum_{in \in Inlets} in.mass\_flowrate \rightarrow In_1 + In_2 + \ldots + In_n$$

A corresponding list of bindings is generated to associate each Prolog variable with a slot or relation, e.g. In_1 may correspond to the *mass_flowrate* slot of stream $s1$.

The manipulation required in this process uses pattern matching to identify terms such as summations. The procedure is also recursive in that a description of an equation can be decomposed by identification of operators used in its construction. For example, the expression:

$$\sum_x x - \sum_y y = 0$$

is first decomposed into:

$$\sum_x x - \sum_y y$$

and

$$0$$

using the equality operator as the location for the split. The first expression can further decomposed by the same coded instructions into:

$$\sum_x x$$

and

$$\sum_y y$$

This process continues until a single term is isolated which can then be interpreted to provide the appropriate Prolog variable form and associated bindings.

Since the procedure is recursive and relies on pattern matching, and later on unification to match the variables in the equations with their bindings, an implementation in Prolog is most appropriate.

Once a model has been created, and evaluated, options are provided for customising it. Specifications can be altered for comparison with other results. The high level structure of the model can be modified within guidelines which ensure consistency. Individual terms in equations that take different values, or approximating expressions, can be changed to create specific models. These options provide flexibility in model definition, but the "expert" knowledge required to maintain consistency provides a rigid framework for development. As described in Section 5.7, these tools are implemented in a network of extended methods to allow flexible access.

A extended method has been written to provide flexible high level access to the modification procedures which are themselves algorithmic, conforming to the model decomposition discussed in Chapter 5. These therefore, are written as CLAP methods or Prolog goals. Their use is demonstrated in Appendix F

6.3.3 Model Solution

The solution of formulated mathematical models is a well defined procedure with proven algorithms. The realisation of such techniques is most appropriately achieved in a procedural language. For example, Newton's method has been written in C and different versions of the sequential modular program, Espros, are written in FORTRAN ,Fortran8x and C. There is little to be gained by rewriting these solvers, but better access to their constituent parts can be provided for more interactive development of models. For example, the equation based solver has been divided into, among others, tools for degrees of freedom checking, equation linearisation and variable initialisation. These tools can be accessed individually by other solution techniques at suitable points in a solution strategy.

The implemented strategy is defined procedurally in CLAP methods as described in Section 3.4. Constructs such as $if - then - else$ were used for the limited decision making required, e.g. if non-simultaneous solution is possible *then* solve by symbolic manipulation *else* use Newton's method.

6.3.4 Interpretation of Results

As discussed in Section 5.6, designers may wish to view modelling results in different ways. For example, a stream table for displaying flowsheet data, or tables to collate data for individual streams or units. As with modelling tools, these options should be available through menus, which could be achieved with an extended method. This has not been implemented in the work described here.

6.4 Summary of Strategy Representation

This chapter discussed the use of a range of representational techniques for the different aspects of design, as described in Section 2.1. Some problems are well defined with proven algorithms, such as solution of sets of non-linear equations or sequential modular flowsheet simulation. The algorithms need not be changed, but need to provide wider access to their models and, in some cases, other normally internal functions, e.g. degrees of freedom checking. The representation of these techniques should be procedural. In this work, for example, an equation based solver has been written in C.

Access to a wide range of tools which have no predefined order of execution can be achieved by using extended methods, a representational device developed for such situations. Extended methods have also been used for creating a process synthesis tool where high level goals can be defined, but, again, there is no strict procedure for attaining them.

Some aspects of process synthesis require reasoning to achieve a solution, e.g. whether or not to purify process feed streams. This can be represented by rules.

Other techniques utilised include an object oriented description of a hierarchy of flowsheets, the consistency of which can be maintained using object oriented methods and demons. Logic programming, in the form of Prolog, has been used in the translation of "constraint" relations into "specialise" relations.

In conclusion, reasoning for different situations is performed by the appropriate techniques. In the case of overall design, an opportunistic approach within a high level framework can be defined using extended methods. Individual inferences can be made using suitable techniques, e.g. rules for decisions, objects to maintain consistency and Prolog for symbolic manipulation. Strategies can also be represented procedurally using CLAP methods, e.g. for formulation and solution of equation based problems. At the lowest level, algorithms, such as for matrix solution, are most suitably implemented in a procedural language such as C or FORTRAN.

Chapter 7

Conclusions

A prototype system has been developed to provide support for flowsheet modelling throughout design. Issues considered in this work are as follows:

- Existing flowsheeting programs are restricted to the range of problems which can be created from their unit model libraries.
- Detailed modelling of individual unit operations is usually performed by separate programs.
- To maintain data consistency between the different models, it is desirable to have a central representation of a design from which models can be developed. This also allows the development of a single modelling tool which can be used for different applications.

Design has no definite procedure for developing an economically acceptable plant from an initial specification. For this reason, a steady state modelling tool for use throughout design cannot be considered separately from other design tasks. Since no procedure is defined, there is no specific point where a modelling tool is used without reference to other tools. Thus the development of the prototype system had to consider providing access to different design evaluation modules in the opportunistic manner of designers.

7.1 Representation

The prototype system uses model based techniques [43] to represent unit operations, streams, chemical species, modelling equations and generated designs in the object oriented language, CLAP [36]. A central model of the design can be constructed by the user from generic objects and then manipulated with reasoning modules.

An inheritance hierarchy of objects representing process units has been constructed to provide relationships between specialisations. Individual objects have been described in a general manner to cover a range of applications.

Modelling equations have been defined as relationships between attributes of process unit objects. Symbolic representation allows access to the equations and their terms by a designer. Equations have also been related to each other in a hierarchical structure where terms in an equation can be defined by further expressions. This structure, and the ability to express equations symbolically, are useful properties in model development.

Process units and streams are related to each other topographically. A structure of objects has been developed to maintain connected process units and streams as flowsheets. The structure permits hierarchical process development through "refinement" and "part of" relationships.

The hierarchy constitutes the central model described above. The definition of attributes for process unit objects are concerned with steady state calculations and process synthesis. Further attributes will be necessary for implementing additional reasoning modules.

A model based approach has also been used for the representation of high level procedures. Extended methods have an explicit structure and function allowing reasoning about the individual calls in the procedure. The structure of extended methods permits definitions of high level aims which do not necessarily have a fixed procedure for their fulfillment. This has been demonstrated with applications in overall process synthesis and distillation column modelling.

Different tasks in design involve different types of reasoning. Chemical engineering has many algorithms which do not require to be altered. A design system should be able to accommodate algorithms as well as symbolic inference. This has been demonstrated by implementing mathematical solvers in the procedural languages C and FORTRAN which can be accessed by the system. Extended methods have been used for procedures with high level aims but no low level methods for achieving the aims. Prolog has provided symbolic inference using logical rules for decision making applications in process synthesis.

7.2 Modelling

A distinction has been made between the requirements of the modeller who develops models and the designer, who uses them. The equation representation technique provides a useful structure for developing models. The designer, however, requires a flexible way of using these models without having to create new ones.

A mechanism has been developed for generating flowsheet models from high level definitions such as "component balance" and "heat balance". The details of model definition should be determined by an expert modeller along with the rules for their selection for flowsheeting applications.

In this manner, a model is created which the designer can modify for specific applications. Access to the high level selection method enables the designer to select alternative definitions for individual parts of a model rather than forcing acceptance of an automatically generated model. This alteration of the model is constrained by the structure developed by the modeller. At a low level, the designer can select different definitions for equations or individual terms.

Different flowsheet alternatives can be evaluated without having to create an entire flowsheet. This is done by associating different models with a single unit. This reduces the amount of information to be stored and, consequently, to be reviewed by the designer.

Models are associated with the object representing the subject of the model, e.g. a node for a flowsheet or an individual distillation column. A symbolic description of any model is available to the user for display or documentation. Solutions of any previous model can also be reviewed. These modelling facilities should be available throughout design. For this reason, they have been implemented in an extended method.

Since access to equations is required for modification, an equation based solution method has been implemented. A sequential modular technique can be used, but does not provide the required flexibility in model representation and equation access.

7.3 Summary

In conclusion, the representation of equations used in the developed system provides a flexible method for model development. Models have been defined for analysis of systems ranging from block flow diagrams to unit operations. By allowing reasoning about equations, the format also supports interactive model modification as part of a design modelling tool. Thus, the symbolic representation enhances the modelling capabilities of modellers and designers.

The use of a central model provides support for different design tasks while maintaining data consistency. Thus, the modelling tool which has been developed can be used for a range of applications ranging between modelling block flow diagrams in a flowsheeting context, and detailed unit operations in individual design procedures.

Support for different design tasks has been demonstrated with the implementation of steady state modelling and process synthesis tools. The opportunistic manner in which designers access individual tools within a framework of high level aims is achieved with extended methods.

7.4 Future Work

This work has described development of a flexible steady state modelling system. Its position as a design tool has been discussed with respect to other design tasks. To demonstrate the principle of a central model accessing different tools, the modelling system has been supplemented by an implementation of the Douglas synthesis procedure [6]. This work could be extended to include more of the functions illustrated in Figure 1.1, e.g. control and hazard analysis.

Future work could address the implementation of different types of model for these different design tasks, e.g. dynamic and qualitative models. The current library of steady state models could also be extended.

More design strategies are required, for instance, to aid in heat exchange network design, reactor design and separation sequencing. Some tasks can be automated, such as separation sequencing, but reactor design, for example, may require a high level strategy such as that used for overall synthesis. Experts in the specific functions are required to develop these methods.

The graphics package used in this work was quite restrictive. However, in many cases, more efficient communication of information can be achieved with a considered graphical presentation than with text. Further work should be directed to developing an interactive system where relevant information is presented as required. It should also be possible to compare results by displaying solutions simultaneously. The data is structured and accessible, so different options for presentation should be available to the user.

Development of a practical tool from the prototype presented here, requires interfaces to a physical properties package, a robust linearised equation solver and a database. The database should maintain the data generated during each design including models developed and process refinements and alternatives.

Appendix A

The Prolog Programming Language

In order to use CLAP a knowledge of Prolog is not deemed necessary by the author. However, it is necessary to introduce the terminology and principles of the language in order to understand the explanation of the tools developed in this work.

A.1 Facts and Rules

Prolog is a declarative language based on a subset of predicate logic, which means that programs are constructed from "Facts" and "Rules", collectively called "Clauses". The family tree shown in Figure A.1 is represented by the example Prolog program shown in Figure A.2. This example program is used throughout this appendix.

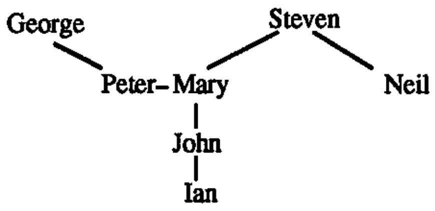


Figure A.1: Family Tree

Each Clause is constructed from a "Predicate", an optional list of arguments, a set of "Goals", and is completed by a full stop. Generally, a Clause with Goals is a Rule, other Clauses being Facts. Figure A.2 will be used throughout this appendix as an example Prolog program. In the figure, the Clauses called `parent` and `grandfather` are Rules containing their requisite conditions, while `father`

```
father(steven,neil).
father(john,ian).
father(peter,john).
father(george,peter).
father(steven,mary).

mother(mary,john).

grandfather(A,B) :-
    father(A,X),
    parent(X,B).

parent(A,X) :-
    father(A,X).
parent(A,X) :-
    mother(A,X).
```

Figure A.2: Example Prolog Program

and `mother` are stated Facts. A further property of Rules is that the argument list contains some Prolog variables, which are identified by the fact that they begin with an upper case letter. For instance, `A` is a Prolog variable, `steven`, however, is a Prolog atom or value.

The labels of the Clauses , e.g. `father`, `mother`, etc. are the Predicates and provide the relationship between the following list of arguments. For instance, in the figure, the statement:

```
father(steven, neil)
```

can be read, "Steven is the father of Neil". The Rule:

```
grandfather(A,B) :-
    father(A,X),
    parent(X,B).
```

can be read "A person, whom we shall call A, is the grandfather of person, B, **if** A is the father of another person, X, **and** X is a parent of B". The definition of the Rule uses the additional symbols:

- "`:-`" the "if" operator, separating the Rule from its Goals,
- "`,`" the "and" operator, separating the Goals.

A third symbol, the "or" operator ";" is not included in the example.

Figure A.2 shows two types of Rules. The **grandfather** Clause has a single conditional definition. However, **parent** has two alternatives. In general, logical Rules with multiple definitions are ordered with exceptions first and general cases last, which reflects the order of execution of Prolog programs. In the case of **parent**, however, neither represents a special case, so the order is unimportant. The two statements could equally well have been written as one using the "or" operator, as follows:

```
parent(A,X) :-
    father(A,X);
    mother(A,X).
```

In summary:

- Prolog programs consist of definitions of *Facts* and *Rules*,
- Clauses consist of a *Predicate*, a list of arguments and a list of *Goals*,
- Facts are *Clauses* which are unconditionally true and are without *Goals*,
- Rules are conditional upon the satisfaction of Goals, and can have several definitions for special and general cases,
- Goals are separated by commas or semi-colons which can be read as the conjunctions "and" and "or" respectively.

A.2 Data Types

There are six basic data structures supported in Prolog. These are:

- variables - uninstantiated terms, e.g. `A`, `Father`, etc.
- numbers - real numbers, e.g. `1.3`.
- integers, e.g. `1, 10, etc.`
- atoms - certain sequences of alphanumeric and special characters, e.g. `john, a, [ ]` (special case of an empty list, see below).
- compound terms - structured data items related by operators, e.g. `P-Q`, `np(john)`

• lists

Lists are compound terms which are particularly important data structures for combining related information. They are written as a sequence of items separated by commas enclosed in square brackets, e.g. `[1,2,3]`, `[a,[A,B,[ ]]]`.

A.3 Program Execution

The example program in Figure A.2 has no apparent beginning or end. This is a property of declarative programs. The method of execution is to question the interpreted program for true statements. For instance, the question:

```
grandfather(george,john).
```

receives the response:

```
yes
```

i.e. the statement is provable with the stated facts. Equally, if the question is:

```
grandfather(george,jock).
```

the response is:

```
no.
```

By replacing the arguments by variables, solutions can be generated, apart from affirmative or negative answers. For instance, the question could be asked:

```
grandfather(george,A).
```

the response is:

```
A = john.
```

When a variable takes a value, in this case `A` takes the value `john`, it has been "instantiated".

Questions can be resatisfied where more than one solution exists. Prolog will present the first solution generated. The user then types a semi-colon which indicates that the goal should be resatisfied. For example, the question:

```
grandfather(A,john).
```

is satisfied by:

```
A = george ;
A = steven
```

A further semi-colon receives the no response.

A.4 Recursion

Prolog allows the specification of recursive Clauses, i.e. Clauses which refer to themselves. For example, the following Clauses can be added to the program in Figure A.2:

```
ancestor(A,X) :-
    parent(A,X).
ancestor(A,X) :-
    parent(A,B),
    ancestor(B,X).
```

Here, the second definition of `ancestor` also calls itself. The first definition is a terminating condition otherwise the recursion would never find a solution. This particular Clause can be used to find all ancestors of a particular person in the database, or to check whether or not one person is an ancestor of another. For instance, the call:

```
ancestor(george, john).
```

first checks the terminating condition that `george` is the parent of `john`. This fails and the second definition is called. In this example, `george` is the parent of `peter` and now the recursive call of `ancestor` is made with the arguments `peter` and `john`. The terminating condition is checked and succeeds, i.e. `peter` is the parent of `john`. Thus the overall Goal is true, that `george` is an ancestor of `john`.

A.5 Variable Unification

When a Prolog variable in a Clause or other Prolog structure is instantiated, all other instances of that variable in the structure are "unified" to the same value.

All variables are local to individual Clauses. For example, the argument list of the `grandfather` Clause contains variables `A` and `B` which also appear in the constituent Goals of the Clause, i.e. `father` and `parent`. If `A` is instantiated to `george` and `B` to `john`, the corresponding variables in the Goals also take these values.

Another example of unification in a complex structure is a representation of equations as compound terms of Prolog variables. For instance, the list of equations may be:

```
[A + B = C, 3*C - D = A, B + D = E]
```

Instantiating `A` to 20 and `C` to 30 produces the revised structure:

```
[20 + B = 30, 3*30 - D = 20, B + D = E]
```

Solution of the first two equations, provides values for `B` and `D`. The structure is now:

```
[20 + 10 = 30, 3*30 - 70 = 20, 10 + 70 = E]
```

A.6 Pattern Matching

Prolog incorporates a mechanism for automatic pattern matching. This allows processing of complex data structures without having to write complex string processing algorithms.

Equation manipulation has been an important part of this work and includes recognisable patterns of symbols. The patterns have been used to locate variables, linearise equations, analytically manipulate and solve equations and generate models form generic patterns.

For example, consider the equation:

```
20 + B = 30
```

The program in Figure A.3 locates the numbers in the expression by decomposing the equation using its operators.

The equation fails against the conditions of the first two Clauses since it is neither a variable or a number. The third Clause matches the equation where `A`

```
find_number(V) :-
    var(V).
find_number(T) :-
    number(T),
    write(T).
find_number(A=B) :-
    find_number(A),
    find_number(B).
find_number(A+B) :-
    find_number(A),
    find_number(B).
```

Figure A.3: Prolog Program Illustrating Pattern Matching

is instantiated to `20 + B` and `B` to `30`. The `find_number` operation is performed recursively on the two terms. `20 + B` next matches the fourth Clause where it is divided into `20` and `B`. `20` is a number so is printed, `B` is not, so it matches the first Clause which does nothing further. Similarly `30` matches the second Clause and is printed.

A.7 Backtracking

When Goals fail or resatisfaction is attempted, Prolog "back tracks" through any choice points which have been identified during the solution. For example, if the question is asked:

```
ancestor(steven,A).
```

the first solution generated is that `steven` is an ancestor of `neil` which is a result of the first definition of `ancestor`, i.e. `steven` is a parent of `neil`.

In the successful completion of the Goal there were three choice points. The first was to investigate the first definition of `ancestor`, the second was to investigate the first definition of `parent` and the third, the first definition of `father`. If the Clause is required to be resatisfied, then the program returns to its most recent choice point, i.e. the selection of the Clause for `father`. The remaining Choices at that point are tested and here one more is successful, i.e. `steven` is also father to `mary`.

If further resatisfaction is attempted, no more `father` Clauses remain unchecked. Execution now returns to the `parent` choice point. The second definition requires `steven` to be a mother and consequently fails. The last remaining choice point is then evaluated by calling the second definition of ancestor. A new choice point is added by the call to `parent`. The first solution generated here, is again that `steven` is the father of `neil`. However, for the Clause to succeed completely, `neil` must be an ancestor of someone. This fails because the database contains no reference to `neil` as a father.

Since this fails, execution returns to the new `parent` choice point where `mary` is generated as a solution. Subsequent checking of `mary` as an ancestor identifies her as the mother of `john`, i.e. `steven` is an ancestor of `john`.

For completeness, the only other solution is `ian`.

Appendix B

Evaluation of Design Variable Specifications

The evaluation of the design variable specifications is based on an incidence matrix which contains rows representing equations and columns representing variables. The algorithm bears a close resemblance to equation ordering algorithms such as P4 [68], in that a forward elimination pass is made, followed by a backward elimination and the evaluation of any irreducible blocks by a variation of a Lee, Christensen and Rudd algorithm [50].

As an example, consider the incidence matrix below containing equations describing a heat exchanger. Sample specifications include U, w_1, w_3, T_1 and T_2.

	Q	U	A	ΔT	w_1	w_2	w_3	w_4	T_1	T_2	T_3	T_4
Eqns 1.	x	x	x	x								
2.					x	x						
3.							x	x				
4.	x					x			x	x		
5.	x						x				x	x
6.				x					x	x	x	x

A forward elimination searches the matrix for equations containing only one unsolved variable. For example, since w_1 and w_2 are specified, and hence eliminated from the matrix, equations 2 and 3 fit this description. When such an equation is located it is removed from the matrix along with the single variable. The search then resumes. Now equation 4 contains only one variable, Q, so can also be eliminated. This procedure continues until no more equations can be eliminated. The order of elimination corresponds to the precedence order for solution, i.e. the order in which the equations are to be solved.

The backward elimination is similar, only it searches for variables appearing in only one equation, for example, variable A, above, i.e. A will be calculated by the equation. The variable and the equation are eliminated form the matrix and the search continues until no more can be removed. If all equations have been eliminated and variables remain, then these variables are valid design specifications (structurally, not necessarily so in practice). The precedence order is the reverse of the order of elimination.

Any equations remaining are irreducible blocks which require the selection of "spikes" or "tears" for elimination to continue. A variation of the algorithm presented by Lee, Christensen and Rudd [50] is used to select the spikes.

1. A list of the occurrences of each variable in the remaining equations is compiled. The entry for each variable is of the form: [Frequency, [List of Equations]]. The list of variables is ordered, those with the lowest Frequency being selected first.

2. The variable with the lowest frequency is selected from the list of variables.

3. A number of equations, one fewer than the frequency, and including all members of the associated list of occurrences, is deleted from the irreducible block.

4. Backward elimination is repeated. If elimination is possible, the variable is a spike. The elimination continues until the procedure terminates, or another irreducible block is found, in which case return to 1.

5. If elimination fails, the procedure is restarted by deleting a different set of equations in 3. If this fails, the next variable with its list of occurrences is selected from the list of variables in 2.

These procedures will find a precedence order for solution, any spikes and any unspecified design variables. However, they need to be able to identify badly specified variables. At any point during the algorithm, if an equation is located which has not been eliminated but contains no undeleted variables, then the specifications have been contradictory.

When such situations are encountered, the elimination terminates and an attempt is made to find the offending specification. This is achieved with a depth first search algorithm shown below.

1. A variable is chosen from the set of specifications.
2. The equations in which it occurs are located.
3. One equation is selected from the set and any specifications are deleted from its list of variables.
4. If no variables remain then this equation has been over specified and one specification must be released.
5. If variables do remain, then the derivation of a solution path for each one is attempted by going to 1. Similarly, if each branch terminates in a specification, or a variable which has already been derived, the equation is over-specified and the original specification is a candidate for being deleted.

This is best illustrated by an example. Consider the equations describing the heat exchanger above. The list of specifications may include U and A. Selecting A, all equations containing that variable are located, the only one being:

$$Q = U.A.\Delta T \tag{B.1}$$

U and A are specifications so they are already "derived" and only Q and ΔT need be investigated. Q appears in two more equations:

$$Q = Cp.w_1.(T_1 - T_2) \tag{B.2}$$

$$Q = Cp.w_3.(T_3 - T_4) \tag{B.3}$$

If Cp is defined, and from the previous example, w_1, T_1 and T_2 are, this implies that Q can be derived from these specifications. Similarly, ΔT appears in one equation involving T_1, T_2, T_3 and T_4. The current specifications do not include T_3 and T_4 and there are no remaining equations containing them. In this case, the specification of A can be considered valid. However, if T_3 and T_4 had been specified, A could be calculated from the other design variables, so is one of a contradictory set of equations.

If all specifications are feasible, the algorithm checks if any spikes have been identified. In which case, the full Newton's method will be suggested for solution. Otherwise, the simple analytical solver will be used. Additions to the procedure determine whether or not the analytical solver could have been used had the specifications been different. In such situations, a feasible set of design variables is presented which will achieve this.

The full algorithm is as follows:

1. Determine the number of degrees of freedom from the equation:

$$N_D = N_v - N_e \tag{B.4}$$

   where $N_D =$ the number of degrees of freedom, $N_v =$ the number of variables in the system, and $N_e =$ the number of equations in the model.

2. If the number of specified design variables is greater than N_D, this indicates that the problem is over-specified. Use backward elimination to suggest a feasible set.

3. Construct an incidence matrix and removed specified design variables.

4. Check that no equations have had all variables specified. If so, go to 11.

5. Forward eliminate.

6. Check that no equations have had all variables specified. If so, go to 11.

7. Backward eliminate.

8. Check that no equations have had all variables specified. If so, go to 11.

9. Evaluate irreducible blocks.

10. If all equations and variables are eliminated (or are spikes) then specifications are feasible. Finish by selecting the solver. If variables remain, these are suitable design variables.

11. Locate contradictory specifications by determining which of them can also be calculated.

Appendix C

Expansion of K Value Expressions

This example is intended to show how the general description of an equation may be expanded. Using the expression for multicomponent vapour-liquid distribution coefficient, K_i, two examples will be described: the ideal Raoult's Law expression and the expansion of the fugacity coefficient term. These illustrate the range of possible expressions derivable for particular situations.

The most general form of the K value expression is given by the equation:

$$K_i = \frac{\gamma_i \times f_i \times \Phi_i \times P_i^*}{\phi_i \times P_T} \tag{C.1}$$

where: γ_i = vapour activity coefficient,

f_i = fugacity coefficient,

Φ_i = Poynting correction factor,

P_i^* = vapour pressure,

ϕ_i = partial fugacity coefficient,

P_T = pressure.

This general form has been implemented as a single CLAP relation which can be adapted to apply under a range of conditions. The above definition has three different types of term:

- A direct reference to a slot in an object. For instance, P_T, is a slot in a stream object.

- A reference to a further expression. P^* above can be further expanded to the Antoine correlation, shown below in equation C.3. If, however, the term can be defined by more than one expression, it can be represented by the third category, below.

- A Prolog variable corresponding to a selection of expressions and values for cases where a single term can take different definitions according to the assumptions made. Terms such as γ_i, f_i, Φ_i and ϕ_i, can be approximated to 1 or can be expanded to large expressions according to the equation of state implemented.

In order to infer the correct model formulation, the types of term are evaluated in strict order.

1. A choice is made for terms with alternative expansions. In the above example, γ_i, f_i, Φ_i and ϕ_i are determined first to allow any further expansions to be carried out in the later stages. This step unifies the Prolog variable in the equation with a numerical value or a reference to a further expression.

2. The term is then checked to establish whether or not it is a relation. If it is, it may be a static relation or a user-defined relation. Static relations may have specified values, similar to slots, or require some inference to determine their values. The value is established by performing a "$check relation".

3. User-defined relations may be expanded to further expressions or may contain specified values. They are evaluated by first checking for specified values for the local instance of the relation, e.g. heat load of stream 4. If no value is found, the generic definition of the corresponding relation is substituted in the place of the term. Rather than repeating the work of expanding the relation, the work space is searched to locate one of the same type, previously evaluated in this way. If this fails, the relation must be expanded as normal.

4. If the term is identified as referring to a slot, it is simply checked to retrieve any value there.

The mechanism for selecting between alternative values and expressions is currently implemented as a set of Prolog clauses which must be provided by the model developer. Future work should address the nature of the interaction required with the end user, since the present version requires some knowledge of Prolog to be able to provide the correct choices.

A "select_eqn_type" call is placed in the "active_code" slot of the generic relation for each term with alternative representations. The call has four parameters, the first being a Prolog variable which is unified with the corresponding term in the equation definition. When a decision is returned, this variable will be instantiated to either a number or an expression which is then unified throughout the relation. The second parameter identifies the term being reasoned about, e.g. "fugacity_coefficient". The remaining two parameters are for providing extra arguments for the inference, intended to correspond to the subject of the relation and, where required, a reference object.

The "select_eqn_type" call is evaluated as a normal Prolog goal. An example of such a clause is given below, which determines whether or not the "fugacity_coefficient" term should be replaced by the value, 1, or an expression depending on an equation of state. The clause has three possible ways of succeeding. The first case returns a reference to an expression for fugacity coefficient if an "eqn_of_state" has been recorded. The second sub-goal of the clause determines whether or not a reduced property correlation is appropriate. The expression returned in variable *Phi* is the reference to either a reduced or non-reduced property expression. If either sub-goal fails, the value of the fugacity coefficient is taken to be 1, from the second major goal.

```
select_eqn_type(Phi $corresponding_to components-C $of Unit,
                fugacity_coefficient, Unit, C) :-
    recorded(eqn_of_state, Unit-_,_),
    reduced_props(Unit, Phi).
select_eqn_type(1,fugacity_coefficient, Unit, C).
```

If ideality is assumed, which is the default case shown above, values of γ_i, f_i, Φ_i and ϕ_i can all be approximated to 1. The identification of ideal conditions is based on the specifications which have been made in the context to which the expression is applied. For instance, in this case the provision of an equation of state implies non-ideality, i.e. the validity of the assumption of ideality depends on information not being provided. The specification of ideal conditions results in the following expression:

$$\frac{1 \times 1 \times 1 \times P_i^*}{1 \times P_T} \tag{C.2}$$

Any term in equation C.2 instantiated to 1 has been through the selection process detailed above.

The vapour pressure term is expanded to the Antoine expression which is represented in a separate relation.

$$lnP_i^* = A - \frac{B}{T+C} \tag{C.3}$$

where: T = temperature,

A, B and C = Antoine coefficients.

If the ideality assumption is invalid, the identification of an appropriate equation of state can provide values for γ_i, f_i and ϕ_i. The same CLAP relation can be used to generate both ideal and non-ideal cases with optional high pressure deviations. The second example is concerned with the expansion of the fugacity coefficient, f_i, only, since the expansion of the other terms is performed similarly. The non-ideal, but low pressure, K value expression has become:

$$\frac{\gamma_i \times f_i \times 1 \times P_i^*}{\phi_i \times P_T} \tag{C.4}$$

The fugacity coefficient relation is in a general format which can be tailored to accommodate different equations of state. The ones implemented are Van der Waals, Redlich-Kwong, Soave and Peng-Robinson. The general expression implemented was obtained from Smith and Van Ness [76], although several other expressions are reported (see Reid et al [77]). The expression is given by:

$$f_i = exp(z_i - 1 - log(z_i \times (1 - \frac{b_i}{v_i})) - \frac{a_i}{b_i \times R \times T^{1.5}} \times log(1 + \frac{b_i}{v_i})) \tag{C.5}$$

where: z_i = compressibility,

v_i = molar volume,

R = universal gas constant,

T = temperature.

The terms a_i and b_i depend on the equation of state chosen. The correct forms are established from the definition of the specified equation. If, as in this example, the Soave correlation is used, the terms are given by:

$$a_i = \frac{0.42748 \times R^2 \times T_c^2}{P_c} \times (1 + f\omega_i(1 - T_r^{0.5}))^2 \qquad \text{(C.6)}$$

$$b_i = \frac{0.8664 \times R \times T_c}{P_c} \qquad \text{(C.7)}$$

where: $f\omega$ = temperature dependence factor,
T_c = critical temperature,
P_c = critical pressure,
T_r = reduced temperature.

The temperature dependence factor is only incorporated in Soave and Peng-Robinson of the implemented models. The Soave expression being:

$$f\omega_i = 0.48 + 1.574\omega_i - 0.176\omega_i^2 \qquad \text{(C.8)}$$

where: ω_i = acentric factor.

The initial statement of the general form of the equation has, therefore, been through several levels of reasoning to reach the final model. The first decision was to implement a non-ideal model and then to ignore high pressure deviations. The terms of the non-ideal expressions were further altered according to the equation of state which had been selected to model the system.

It is worth noting here that the implemented cubic equation of state has been described in a general form which can be adapted to one of four correlations, namely Van der Waals, Redlich-Kwong Soave and Peng-Robinson. Four terms are expressed as variables in order to accommodate the appropriate parts of the equations of state, which is shown below with the relevant terms. The selection of a particular equation determines the set of terms implemented. The general correlation and the definitions of the terms was obtained from Reid et al [77].

$$P = \frac{RT}{V-b} - \frac{a}{V^2 + ubV + wb^2} \tag{C.9}$$

Equation	**a**	**b**	**u**	**w**
Van der Waals	$\frac{27R^2T_c^2}{64P_c}$	$\frac{RT_c}{8P_c}$	0	0
Redlich-Kwong	$\frac{0.42748R^2T_c^2}{P_c}$	$\frac{0.08664RT_c}{P_c}$	1	0
Soave	$\frac{0.42748R^2T_c^2}{P_c}(1 + f\omega(1 - T_r^{0.5}))^2$	$\frac{0.08664RT_c}{P_c}$	1	0
Peng-Robinson	$\frac{0.0.45724^2T_c^2}{P_c}(1 + f\omega(1 - T_r^{0.5}))^2$	$\frac{0.07780RT_c}{P_c}$	2	-1

The value of $f\omega$ for the Soave and Peng-Robinson correlations differs. The Soave expression is given above in equation C.8. The Peng-Robinson expression is:

$$f\omega_i = 0.37464 + 1.54226\omega_i - 0.26992\omega_i^2 \tag{C.10}$$

This appendix has demonstrated the expansion of a single expression which may be part of a larger model. The use of the K-value expression as part of a distillation column model is shown in Appendix D. Two forms of the expression are demonstrated, one using the ideal Raoult's Law expression, the other an expansion of the fugacity coefficient term.

Appendix D

Example Output of Generated Models

It is important for a designer or modeller to be able to evaluate models visually as well as numerically. This provides information allowing modification of models as described in Section 5.5. Presentation of models can be achieved by interpreting their symbolic definitions.

Displaying the mathematical representation of models requires a structured output allowing the designer to locate and study particular sections of a model. The CLAP representation of equations as relations provides such a structure. Individual equations can have terms which can be expanded to further expressions. For example, the Fenske equation is an expression relating K-values of components in the feed, distillate and bottoms streams of a distillation column. The K-value definition can be expanded to an expression relating vapour pressure to total pressure. This decomposition can be continued for vapour pressure.

The decomposition of equations and expressions into the definitions of their terms corresponds to the division of a document into sections, subsections, paragraphs, etc. Thus a model can be displayed, showing this decomposition, which allows the user to assess what modifications are required. Using a document preparation system, such as LaTeX, a table of contents can be generated automatically once the structure of sections and subsections is complete. Not only does this provide the necessary means of locating parts of a model, but it can also be used to document models used throughout a design.

Two types of model can be displayed: generic descriptions and particular instances. Instances are restricted to the current model of a particular unit, while any other models of the unit which have been generated can be displayed in their generic form. When a model is displayed it can be stored in a file, so a library of documented models can be created. The following examples have all been generated by the program from the symbolic definition of the models.

The Fenske equation Example 1 is a generic model, i.e. the definitions of the terms in the equation are not presented. This example contains most of the points discussed above. Only one equation is shown so the decomposition into sections and subsections is not demonstrated. Examples of this type of decomposition are in Example 2 and Example 3. The name of the equation is, however, used as a section heading which is numbered. Numbering follows normal patterns, but sections here are labelled "Example" for clarity. A table of contents is generated for the sections and subsections. This is shown in Example 2 and Example 3.

Equations are presented in two ways. The first is the specialise form of the relation, i.e. all summations and "for all" statements are expanded. The terms of the equation can be constants, specifications or Prolog variables. Prolog variables are replaced by labels constructed of "x" and a unique number. For example, the variable corresponding to the minimum number of plates is replaced by *x3*.

The second form of equations displayed is the generic definition contained in the constraint form of the relation. Thus summations and "for all" statements can be identified. Each term in the return form of the constraint relation is translated into a LaTeX format. For example the two terms in Example 1:

$$min\_number\_of\_plates_{column1}$$
$$k\_value_{light\_key,inlets}$$

are generated from the CLAP representation:

```
min_number_of_plates $of Unit
k_value $corresponding_to components-LK $of Feed.
```

The variables `LK` and `Feed` are instantiated to terms in the list of bindings and `Unit` is instantiated to the subject of the relation by unification.

In addition to the generic description, a reference for the equation is given where one is available. This allows the designer to investigate the models which have been implemented, and maintains a record of their sources for the model developer.

Definitions for the terms of an equation are presented below its general description. Terms with values and those replaced by labels are displayed with the name of the slot or relation to which they are bound. In Example 1, for instance, *x3* is defined as the *min_number_of_plates* slot in object *column1*. Since this is the generic description of the model, all terms are presented in this manner, including ones that are defined by other relations, e.g. *k_value*.

Where the current model instance is being displayed, as in D and Example 3, terms defined by further user-defined relations are presented as lesser sections to the original equation. For instance in Example 2 the definition of the Fenske equation is the same as in Example 1. However, only the definition of *min_number_of_plates* is listed as in the previous example. The definition of the K-value is displayed in a format similar to the Fenske equation. Where the Fenske equation is a "subsection", the K-value is a dependent "subsubsection".

Example 2 represents the default *fenske_gilliland_model* which contains a component balance and the Fenske equation. Example 3 is a modified version of the same model. In this example, the definition of *k_value* has been changed from the ideal vapour pressure model to one incorporating a fugacity coefficient. This was achieved using the model modification tool described in Section 5.5.

The revised model has one of the constants in the original expression replaced by variable *x11* which corresponds to the definition of fugacity coefficient.

Example 1 $fenske\_equation$ of $column1$

The expression is represented by:

$$x3 = log(x2)/log(x1/x0)$$

which is a specific form of the general equation:

$$min\_number\_of\_plates_{column1} = log(separation\_factor)/log(k\_value_{light\_key,inlets}/k\_value_{heavy\_key,inlets})$$

(see literature [1])
$x3 = min\_number\_of\_plates$ of $column1$
$x2 = mf\_separation\_factor$ of $column1$
$x1 = k\_value$ corresponding to components $benzene$ in object $s4$
$x0 = k\_value$ corresponding to components $toluene$ in object $s4$

References

[1] Douglas J.M., *Conceptual Design of Chemical Processes*, McGraw-Hill, New York, 1988.

Contents of Example 2

Example 2 *balance* of *column*1

The expression is represented by:

$$x0$$

which is a specific form of the general equation:

$$equations$$

Example 2.1 *fenske_gilliland_model* of *column*1 ($x0$)

The expression is represented by:

$$\begin{array}{c} x1 \\ - \\ - \end{array}$$

which is a specific form of the general equation:

$$\begin{array}{c} fenske\_equation_{column1} \\ \forall_{light\_components}, molar\_flowrate_{inlets} * mole\_fraction_{light\_components,inlets} = \\ molar\_flowrate_{distillate\_stream} * mole\_fraction_{light\_components,distillate\_stream} \\ \forall_{heavy\_components}, molar\_flowrate_{inlets} * mole\_fraction_{heavy\_components,inlets} = \\ molar\_flowrate_{bottoms\_stream} * mole\_fraction_{heavy\_components,bottoms\_stream} \end{array}$$

Example 2.1.1 *fenske_equation* of *column*1 ($x1$)

The expression is represented by:

$$x7 = log(x6)/log(x4/x2)$$

which is a specific form of the general equation:

$$\begin{array}{c} min\_number\_of\_plates_{column1} = \\ log(separation\_factor)/log(k\_value_{light\_key,inlets}/k\_value_{heavy\_key,inlets}) \end{array}$$

(see literature [1])
$x7 = min\_number\_of\_plates$ of *column*1

Example 2.1.1.1 *mf_separation_factor* of *column*1 ($x6$)

The expression is represented by:

$$0.995/0.005 * (0.99/0.01)$$

which is a specific form of the general equation:

$$\begin{array}{c} mole\_fraction_{light\_key,distillate\_stream}/mole\_fraction_{heavy\_key,distillate\_stream} * \\ mole\_fraction_{heavy\_key,bottoms\_stream}/mole\_fraction_{light\_key,bottoms\_stream} \end{array}$$

(see literature [1])
0.995 = $mole\_fraction$ corresponding to components $benzene$ in object $s5$
0.005 = $mole\_fraction$ corresponding to components $toluene$ in object $s5$
0.99 = $mole\_fraction$ corresponding to components $toluene$ in object $s6$
0.01 = $mole\_fraction$ corresponding to components $benzene$ in object $s6$

Example 2.1.1.2 k_value **corresponding to components** $benzene$ **in object** $s4$ ($x4$)
The expression is represented by:

$$1 * 1 * 1 * x5/(1 * 760)$$

which is a specific form of the general equation:

$$\forall_{components}, vapour\_activity\_coefficient * fugacity\_coefficient * poynting\_correction * vapour\_pressure_{components,s4}/partial\_fugacity\_coefficient * pressure_{s4}$$

(see literature [2])

$vapour\_pressure$ **corresponding to components** $benzene$ **in object** $s4$ ($x5$)
The expression is represented by:

$$exp(15.9008 - 2788.51/(366 + (-52.36)))$$

which is a specific form of the general equation:

$$\forall_{components}, exp(antoine\_A_{components} - antoine\_B_{components}/temperature_{s4} + antoine\_C_{components})$$

(see literature [3])
15.9008 = $antoine\_A$ of $benzene$
2788.51 = $antoine\_B$ of $benzene$
−52.36 = $antoine\_C$ of $benzene$
366 = $temperature$ of $s4$
760 = $pressure$ of $s4$

Example 2.1.1.3 k_value **corresponding to components** $toluene$ **in object** $s4$ ($x2$)
The expression is represented by:

$$1 * 1 * 1 * x3/(1 * 760)$$

which is a specific form of the general equation:

$$\forall_{components}, vapour\_activity\_coefficient * fugacity\_coefficient * poynting\_correction * vapour\_pressure_{components,s4}/partial\_fugacity\_coefficient * pressure_{s4}$$

(see literature [2])

vapour_pressure **corresponding to components** *toluene* **in object** *s4* (*x3*)
The expression is represented by:

$$exp(16.0137 - 3096.52/(366 + (-53.67)))$$

which is a specific form of the general equation:

$$\forall_{components}, exp(antoine\_A_{components} - antoine\_B_{components}/temperature_{s4} + antoine\_C_{components})$$

(see literature [3])
16.0137 = *antoine_A* of *toluene*
3096.52 = *antoine_B* of *toluene*
−53.67 = *antoine_C* of *toluene*
366 = *temperature* of *s4*

References

[1] Douglas J.M., *Conceptual Design of Chemical Processes*, McGraw-Hill, New York, 1988.

[2] Reid R.C., Prausnitz J.M. & Poling B.E., *The Properties of Gases and Liquids, 4th Edition*, McGraw-Hill, New York, 1987.

[3] McCabe W.L. & Smith J.C., *Unit Operations of Chemical Engineering, 3rd Ed.*, McGraw-Hill, 1976.

Contents of Example 3

Example 3 *balance* of *column*1

The expression is represented by:

$$x0$$

which is a specific form of the general equation:

$$equations$$

Example 3.1 *fenske_gilliland_model* of *column*1 ($x0$)

The expression is represented by:

$$x1$$
$$-$$
$$-$$

which is a specific form of the general equation:

$$fenske\_equation_{column1}$$
$$\forall_{light\_components}, molar\_flowrate_{inlets} * mole\_fraction_{light\_components,inlets} = molar\_flowrate_{distillate\_stream} * mole\_fraction_{light\_components,distillate\_stream}$$
$$\forall_{heavy\_components}, molar\_flowrate_{inlets} * mole\_fraction_{heavy\_components,inlets} = molar\_flowrate_{bottoms\_stream} * mole\_fraction_{heavy\_components,bottoms\_stream}$$

Example 3.1.1 *fenske_equation* of *column*1 ($x1$)

The expression is represented by:

$$x19 = log(x18)/log(x10/x2)$$

which is a specific form of the general equation:

$$min\_number\_of\_plates_{column1} = log(separation\_factor)/log(k\_value_{light\_key,inlets}/k\_value_{heavy\_key,inlets})$$

(see literature [1])
$x19 = min\_number\_of\_plates$ of $column1$

Example 3.1.1.1 *mf_separation_factor* **of** *column*1 ($x18$)
The expression is represented by:

$$0.995/0.005 * (0.99/0.01)$$

which is a specific form of the general equation:

$$mole\_fraction_{light\_key,distillate\_stream}/mole\_fraction_{heavy\_key,distillate\_stream} * mole\_fraction_{heavy\_key,bottoms\_stream}/mole\_fraction_{light\_key,bottoms\_stream}$$

(see literature [1])
$0.995 = mole\_fraction$ corresponding to components *benzene* in object $s5$
$0.005 = mole\_fraction$ corresponding to components *toluene* in object $s5$
$0.99 = mole\_fraction$ corresponding to components *toluene* in object $s6$
$0.01 = mole\_fraction$ corresponding to components *benzene* in object $s6$

Example 3.1.1.2 ***k_value*** **corresponding to components** ***benzene*** **in object** ***s4*** **(*x10*)**
The expression is represented by:

$$1 * x11 * 1 * x17/(1 * 760)$$

which is a specific form of the general equation:

$$\forall_{components}, vapour\_activity\_coefficient * fugacity\_coefficient * poynting\_correction * vapour\_pressure_{components,s4}/partial\_fugacity\_coefficient * pressure_{s4}$$

(see literature [2])

vapour_pressure **corresponding to components** *benzene* **in object** *s4* **(*x17*)** The expression is represented by:

$$exp(15.9008 - 2788.51/(366 + (-52.36)))$$

which is a specific form of the general equation:

$$\forall_{components}, exp(antoine\_A_{components} - antoine\_B_{components}/temperature_{s4} + antoine\_C_{components})$$

(see literature [3])
$15.9008 = antoine\_A$ of *benzene*
$2788.51 = antoine\_B$ of *benzene*
$-52.36 = antoine\_C$ of *benzene*
$366 = temperature$ of $s4$

fugacity_coefficient **corresponding to components** *benzene* **in object** *s4* **(*x11*)** The expression is represented by:

$$exp(x12 - 1 - log(x12 * (1 - x13/x14)) - x15 * x16/(x13 * 8.314 * 366) * log(1 + x13/x14))$$

which is a specific form of the general equation:

$$\forall_{components}, exp(compressibility_{components} - 1 - log(compressibility_{components} * 1 - state\_eq\_b\_coeff_{components}/molar\_volume_{components}) - state\_eq\_a\_coeff_{components} * r\_k\_soave\_alpha/state\_eq\_b\_coeff_{components} * universal\_gas\_constant * temperature_{s4} * log(1 + state\_eq\_b\_coeff_{components}/molar\_volume_{components}))$$

(see literature [4])
$x12 = compressibility$ of *benzene*

$state\_eq\_b\_coeff$ **of** $benzene$ **(**$x13$**)** The expression is represented by:

$$0.08664 * 8.314 * 562.1/48.9$$

which is a specific form of the general equation:

$$state\_eq\_coefficient\_b * universal\_gas\_constant * critical\_temperature_{benzene}/critical\_pressure_{benzene}$$

(see literature [4])
48.9 = $critical\_pressure$ of $benzene$
562.1 = $critical\_temperature$ of $benzene$
$x14$ = $molar\_volume$ of $benzene$

$state\_eq\_a\_coeff$ **of** $benzene$ **(**$x15$**)** The expression is represented by:

$$0.42748 * 1 * 8.314^2 * 562.1^2/48.9$$

which is a specific form of the general equation:

$$state\_eq\_coefficient\_a * state\_eq\_T\_dependence * (universal\_gas\_constant)^2 * (critical\_temperature_{benzene})^2/critical\_pressure_{benzene}$$

(see literature [4])
$x16$ = $redlich\_kwong\_alpha$ of $benzene$
760 = $pressure$ of $s4$

Example 3.1.1.3 k_value **corresponding to components** $toluene$ **in object** $s4$ **(**$x2$**)**
The expression is represented by:

$$1 * x3 * 1 * x9/(1 * 760)$$

which is a specific form of the general equation:

$$\forall_{components}, vapour\_activity\_coefficient * fugacity\_coefficient * poynting\_correction * vapour\_pressure_{components,s4}/partial\_fugacity\_coefficient * pressure_{s4}$$

(see literature [2])

$vapour\_pressure$ **corresponding to components** $toluene$ **in object** $s4$ **(**$x9$**)**
The expression is represented by:

$$exp(16.0137 - 3096.52/(366 + (-53.67)))$$

which is a specific form of the general equation:

$$\forall_{components}, exp(antoine\_A_{components} - antoine\_B_{components}/temperature_{s4} + antoine\_C_{components})$$

(see literature [3])
16.0137 = $antoine\_A$ of $toluene$
3096.52 = $antoine\_B$ of $toluene$
−53.67 = $antoine\_C$ of $toluene$
366 = $temperature$ of $s4$

$fugacity\_coefficient$ **corresponding to components** $toluene$ **in object** $s4$ $(x3)$ The expression is represented by:

$$exp(x4 - 1 - log(x4 * (1 - x5/x6)) - x7 * x8/(x5 * 8.314 * 366) * log(1 + x5/x6))$$

which is a specific form of the general equation:

$$\forall_{components}, exp(compressibility_{components} - 1 - log(compressibility_{components} * 1 - state\_eq\_b\_coeff_{components}/molar\_volume_{components}) - state\_eq\_a\_coeff_{components} * r\_k\_soave\_alpha/state\_eq\_b\_coeff_{components} * universal\_gas\_constant * temperature_{s4} * log(1 + state\_eq\_b\_coeff_{components}/molar\_volume_{components}))$$

(see literature [4])
$x4 = compressibility$ of $toluene$

$state\_eq\_b\_coeff$ **of** $toluene$ $(x5)$ The expression is represented by:

$$0.08664 * 8.314 * 591.7/41.1$$

which is a specific form of the general equation:

$$state\_eq\_coefficient\_b * universal\_gas\_constant * critical\_temperature_{toluene}/critical\_pressure_{toluene}$$

(see literature [4])
$41.1 = critical\_pressure$ of $toluene$
$591.7 = critical\_temperature$ of $toluene$
$x6 = molar\_volume$ of $toluene$

$state\_eq\_a\_coeff$ **of** $toluene$ $(x7)$ The expression is represented by:

$$0.42748 * 1 * 8.314^2 * 591.7^2/41.1$$

which is a specific form of the general equation:

$$state\_eq\_coefficient\_a * state\_eq\_T\_dependence * (universal\_gas\_constant)^2 * (critical\_temperature_{toluene})^2/critical\_pressure_{toluene}$$

(see literature [4])
$x8 = redlich\_kwong\_alpha$ of $toluene$

References

[1] Douglas J.M., *Conceptual Design of Chemical Processes*, McGraw-Hill, New York, 1988.

[2] Reid R.C., Prausnitz J.M. & Poling B.E., *The Properties of Gases and Liquids, 4th Edition*, McGraw-Hill, New York, 1987.

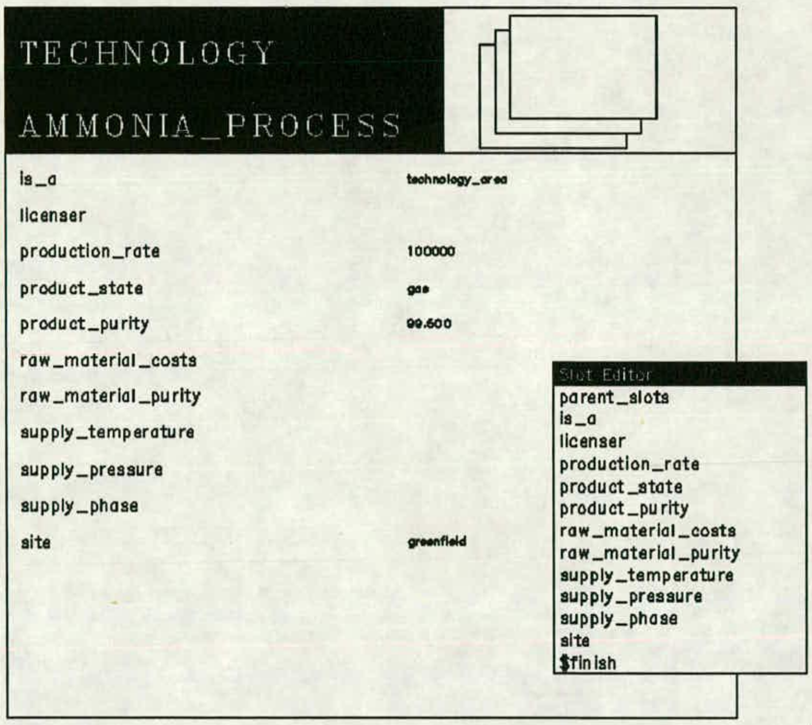
[3] McCabe W.L. & Smith J.C., *Unit Operations of Chemical Engineering, 3rd Ed.*, McGraw-Hill, 1976.

[4] Smith J.M. & Van Ness H.C., *Introduction to Chemical Engineering Thermodynamics, 3rd Edition*, McGraw-Hill, Tokyo, 1975.

Appendix E

Application of Overall Process Synthesis Procedure

This appendix illustrates the application of extended methods to process synthesis. For this example, design of an ammonia synthesis loop is considered. The Douglas hierarchy of decision levels [7] has been implemented as described in Chapter 6. The figures were produced by the system as screendumps.



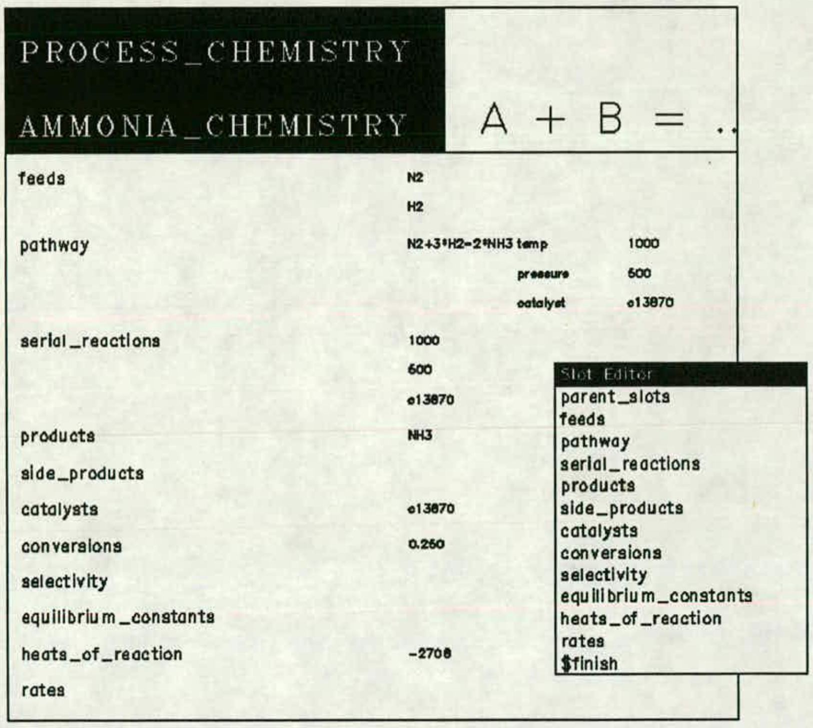
is_a	technology_area
licenser	
production_rate	100000
product_state	gas
product_purity	99.600
raw_material_costs	
raw_material_purity	
supply_temperature	
supply_pressure	
supply_phase	
site	greenfield

Figure E.1: Object Describing Initial Process Specification

Four objects were defined which provide a high level description of the design available for review throughout the synthesis procedure. These objects are intended to provide common data structures for design and other functions, such as preliminary economic analysis as discussed in Section 1.1.1.

The **technology** object corresponds to the initial statement of the requirements of the design. Figure E.1 shows the **technology** object defined for the ammonia synthesis loop. The slots which have values represent high level decisions from which the design is developed. Production rate, state and purity, and the choice of site are business decisions not necessarily made by design engineers.

These slot values may be changed as the design develops. Economic assessment of a design may indicate that these constraints cannot be met profitably unless the specifications are revised. Furthermore, the initial statement is often incomplete. Preliminary design may reveal that a particular process cannot be profitable without having to waste time and effort on an extensive information gathering exercise. However, if the process has potential, the information will be required at the point of development which has been reached. It is undesirable to have to restart the synthesis procedure for individual pieces of information, so access to this object is required throughout synthesis.



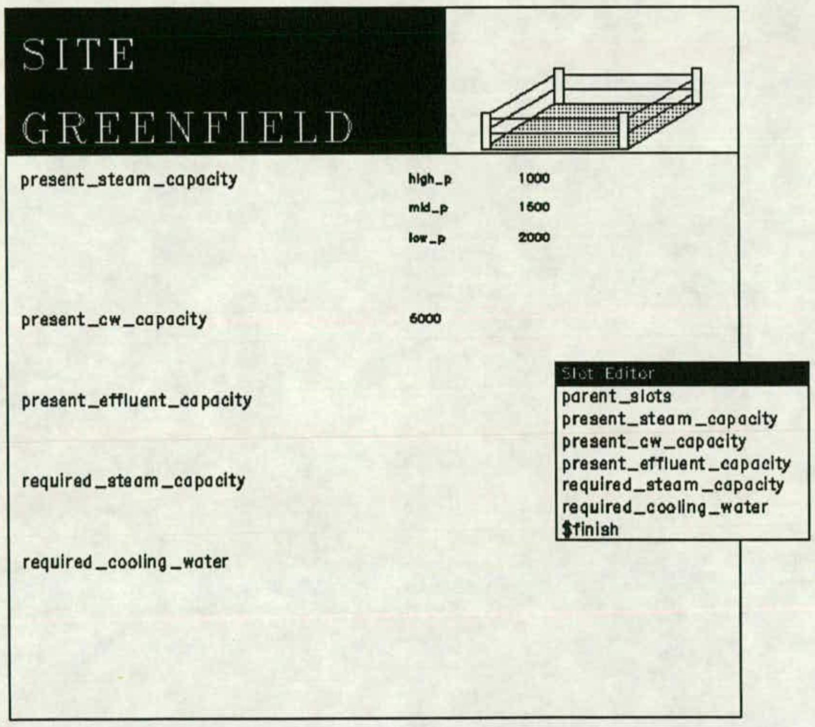
```
PROCESS_CHEMISTRY

AMMONIA_CHEMISTRY

feeds                      N2
                           H2

pathway                    N2+3*H2=2*NH3 temp      1000
                                         pressure  600
                                         catalyst  e13870

serial_reactions           1000
                           600
                           e13870

products                   NH3
side_products
catalysts                  e13870
conversions                0.250
selectivity
equilibrium_constants
heats_of_reaction          -2708
rates
```

Figure E.2: Object Describing Process Chemistry

The second new object represents the process chemistry. The instance for ammonia synthesis is shown in Figure E.2.

This object is distinct from objects representing individual reactions which can be associated with individual reactors. As with the **technology** object, the **process_chemistry** object is intended to provide a link to other functions. The **process_chemistry** object should be prepared by chemists investigating the reaction path and its associated properties, or as a result of a database search.

The object contains a list of reactions occurring in parallel or series. The list of reactions is stored in the **pathway** slot and the conditions are repeated in the **serial reactions** slot. In this example, only one reaction is investigated. Reactions which are carried out separately, i.e. have different conditions, are described in individual **reaction** objects which are associated with specific reactors. A **process_chemistry** object is associated with a particular **technology**. For example, the Haber ammonia synthesis process has an associated process chemistry, while a particular company may have a different process with its own reaction scheme. In this way, different process chemistries can be evaluated as discussed in Section 1.1.1.

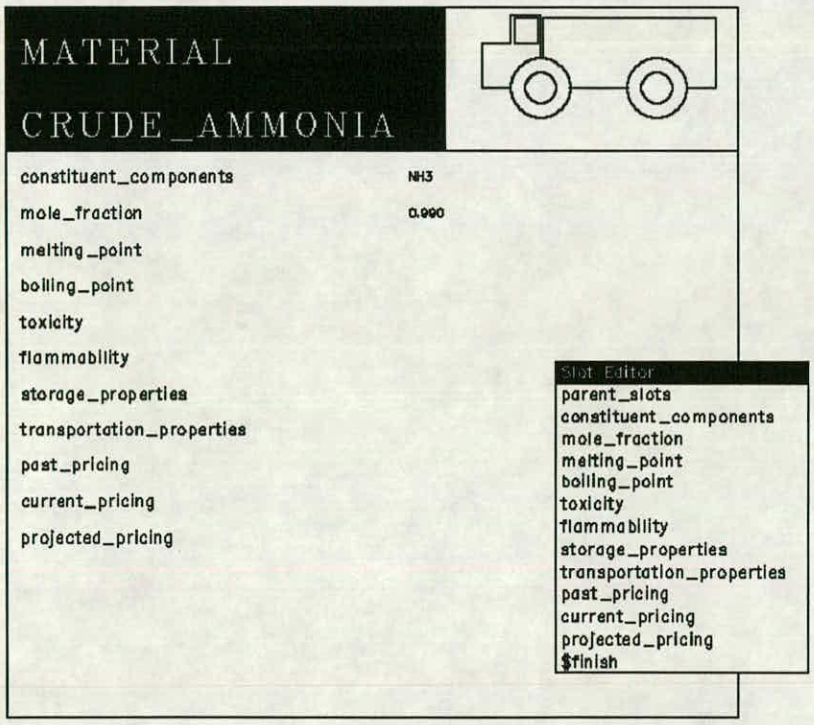


```
SITE
GREENFIELD

present_steam_capacity          high_p    1000
                                mid_p     1600
                                low_p     2000

present_cw_capacity             6000

present_effluent_capacity

required_steam_capacity

required_cooling_water
```

Figure E.3: Object Describing Potential Site

The third class of object represents the site under consideration for the plant, as shown in Figure E.3. This choice has a direct bearing on the economics of a particular design because all plant utilities, such as cooling water, steam and electricity need to be provided. A new site, therefore, requires extra plant at greater expense. It is important to have a model of the utilities available so that any extra requirements are considered in an economic evaluation of the design.

The fourth object is used to represent bulk materials. A `material` object, as shown in Figure E.4, does not describe a stream in the plant, but a commodity which is bought or sold. In this example, `material` objects are used to represent the feed which is either bought and transported to the plant, or is imported from another plant on the same site, the product crude ammonia, and the waste produced.


MATERIAL

CRUDE_AMMONIA

```
constituent_components        NH3
mole_fraction                 0.990
melting_point
boiling_point
toxicity
flammability
storage_properties
transportation_properties
past_pricing
current_pricing
projected_pricing
```

Figure E.4: Object Describing Product Material

As with the previous three objects, the `material` object is intended to provide consistent access to the design by different design functions. The object shown in Figure E.4 includes slots for pricing, transportation and storage properties. These are not necessarily for use by a design engineer except for reference.

E.1 The Implemented Hierarchy of Decision Levels

The extended methods used to implement the hierarchy of decision levels are detailed in Section 6.1. A design is initiated with the specification of a `technology`, a `process_chemistry`, a `site` and the feed and product `materials`. A `design` object is created which refers only to a single design of a specific `technology`, thus individual designs can, in principle, be documented and stored in a database.

The system developed uses a window which is divided into a text section and a graphics section. The user is presented with a menu of design options in the graphics section as described in Section 6.1. The menu is labelled with the name of the current synthesis level and a message is displayed in the text window to highlight this.

The first decision is to determine whether the process should be continuous or operated in batches. The first menu, shown in Figure E.5 is a reduced form of the general menu, in that the `analysis` and `topology` options are not made available.

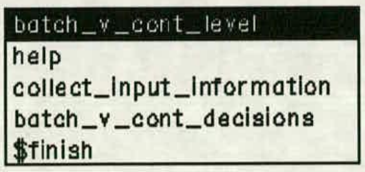
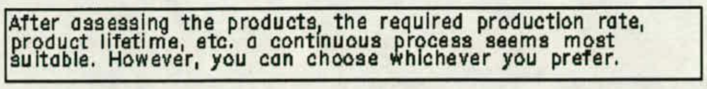


Figure E.5: Menu Presented at the Batch vs Continuous Decision Level

The `help` option, discussed in Section 6.1, details the requirements for completion of a decision level. The `collect_input_information` entry provides access to the four classes of high level object described above, which represent the current design.

If `$finish` is selected, a message appears in the text window explaining that a decision is required before being able to continue. Thus, in order to proceed with the design, `batch_v_cont_decisions` must be selected. This choice invokes a heuristic decision, coded as Prolog rules, assessing the choice of a batch or continuous process. Figure E.6 shows the output from this evaluation. In this example, a continuous process is advised, but the menu allows the selection of either, to allow both alternatives to be assessed should this be required. The entry in the menu refers to a continuous fluid process since the product and feeds are both fluids. If solids are specified in the **technology** object, this choice would be for a continuous solid process.



After assessing the products, the required production rate, product lifetime, etc. a continuous process seems most suitable. However, you can choose whichever you prefer.

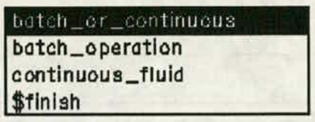


Figure E.6: Result of Batch vs Continuous Heuristic Decision

Once the decision has been made, in this case to design a continuous process, the "batch versus continuous" decision level is complete. The next level is the input-output structure level which is indicated by a message in the text window and a new label for the main menu. The choices in the menu shown in Figure E.5 now include **analysis** and **topology** which were inappropriate before where no flowsheet items were present. The new menu is shown in Figure E.11.

Any option in the main menu can be selected, but without any flowsheet items, only messages are displayed indicating that a flowsheet should be specified first. Selection of **topology** provides access to the flowsheet specification options which have been detailed in Appendix F. The initial step is to create units for the flowsheet. Figure E.7 shows the limited choice of units restricted by the synthesis level.

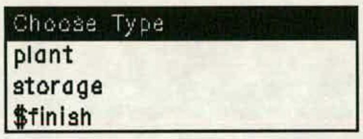
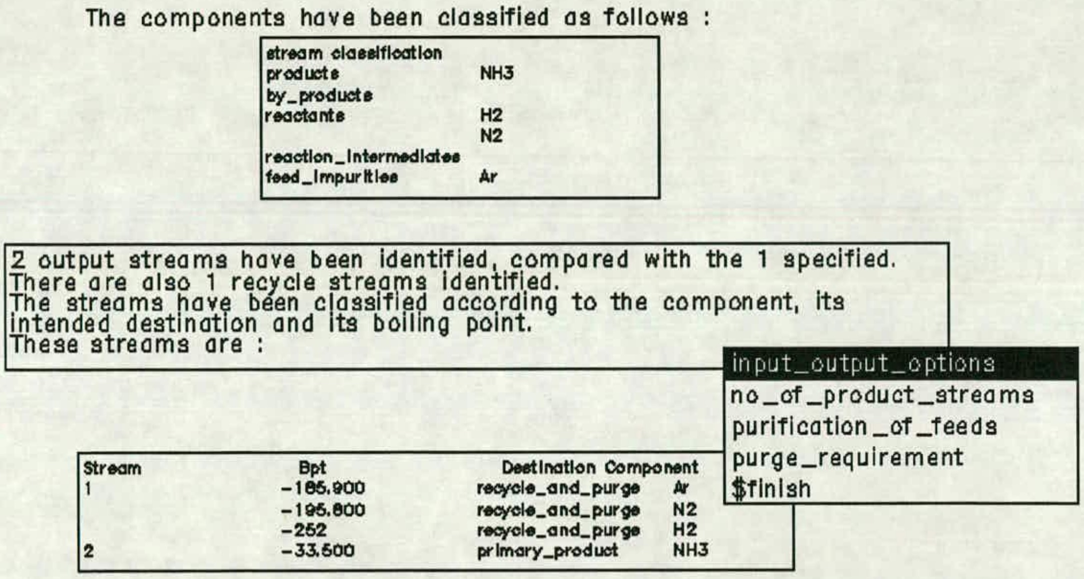


Figure E.7: Menu of Flowsheet Items Available at the Input Output Structure Level

In this example, a **plant** object called **ammonia_plant** is created with one inlet stream and one outlet. The streams are named automatically. Returning to the main menu, the input-output level decisions may be accessed to evaluate the flowsheet. Three decisions are available at this level: calculation of the number of product streams, a decision about whether or not to purify the feed stream, and the assessment of the requirement of a purge. The choices are displayed as a menu as shown in Figure E.8. The diagram also shows the output generated by the `no_of_product_streams` option.



```
The components have been classified as follows :

stream classification
products                NH3
by_products
reactants               H2
                        N2
reaction_intermediates
feed_impurities         Ar

2 output streams have been identified, compared with the 1 specified.
There are also 1 recycle streams identified.
The streams have been classified according to the component, its
intended destination and its boiling point.
These streams are :

Stream        Bpt          Destination Component
1             -185.900     recycle_and_purge   Ar
              -195.800     recycle_and_purge   N2
              -252         recycle_and_purge   H2
2             -33.500      primary_product     NH3
```

Figure E.8: Classification of Output Streams and Input Output Level Decisions

This operation classifies the components present in the process, i.e. N_2, H_2, Ar and NH_3, as products, by-products, reactants, reaction intermediates or feed impurities. These are then ordered by boiling point. Neighbouring components in the same class are lumped together in a single stream and associated with a destination.

Figure E.8 shows two output streams compared with only one so far specified. It also indicates that the reactants should be recycled, and since an impurity, Ar, is present, the recycle should also have a purge.

A purge may not be required if the impurity can be removed before the reactor. In this case, it is likely to be impractical, but it may be considered using the `purification_of_feeds` option shown in the menu in Figure E.8. A short question and answer session provides the qualitative information required for a decision to be made. This is shown in Figure E.9. The recommendation here is that the raw materials should not be purified.

```
The feed impurities are [Ar]
Are all components considered inert?  y.
Are any catalyst poisons? n.
At this stage it is probably best not to purify the raw materials.
```

Figure E.9: Heuristic Judgement of Requirement for Feed Purification

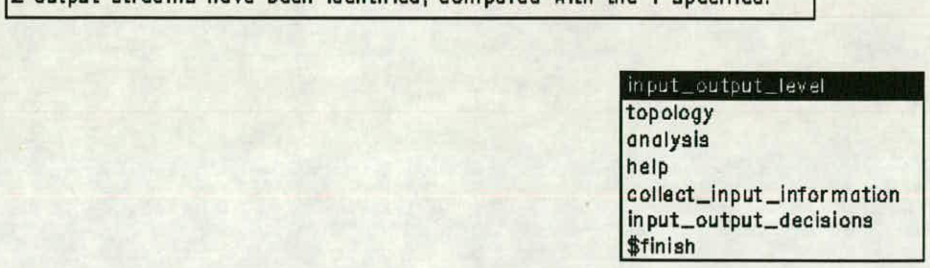
Since the impurity is not to be removed, a gas recycle and purge must be considered. To investigate this, the remaining decision in the menu in Figure E.8 is selected. The output is shown in Figure E.10, confirming that the reactants to be recycled are incondensible and the presence of an impurity implies the use of a purge.

```
A gas recycle is required because some reactants which, presumably,
are to be recycled, boil at sufficiently low temperatures that condensing
them is not possible with cooling water even at high pressure.
A purge may be required as some impurities are also incondensible.
```

Figure E.10: Heuristic Judgement of Requirement for Purge

The decisions which have been considered using the menu have not required any action. The aim is to provide advice and make the designer aware of the decisions that can be made at a particular level of detail. As a result action may be taken but it is not essential.

Certain actions are essential for the completion of a decision level. For example, at the input-output structure level, a **plant** object must be present in the flowsheet with sufficient inlet and outlet streams to transport the specified bulk materials. These essential actions are checked when **$finish** is selected on the main menu.



```
2 output streams have been identified, compared with the 1 specified.
```

Figure E.11: Message Indicating Incomplete Output Stream Specification

In the case of the ammonia synthesis process so far defined, selection of **$finish** at the input-output level invokes a list of checks which identify that only one output stream has been specified where two are required. The message is shown in Figure E.11.

In addition to the statement of the required action, notice is given of the point in the synthesis procedure where the information can be provided. Execution is returned to that point. In this example, progress to the recycle structure level is blocked and the input-output structure level is recalled. The input-output level menu is also shown in Figure E.11.

This message prompts the creation of a new output stream followed by selection of `$finish` once more. The checks this time discover that the inlet and outlet streams do not contain any of the specified materials. Figure E.12 shows the message displayed in the text window.

```
Stream s0 is not associated with a feed material (syn_loop_feed..$null).
More information is required at the input_output_level.
```

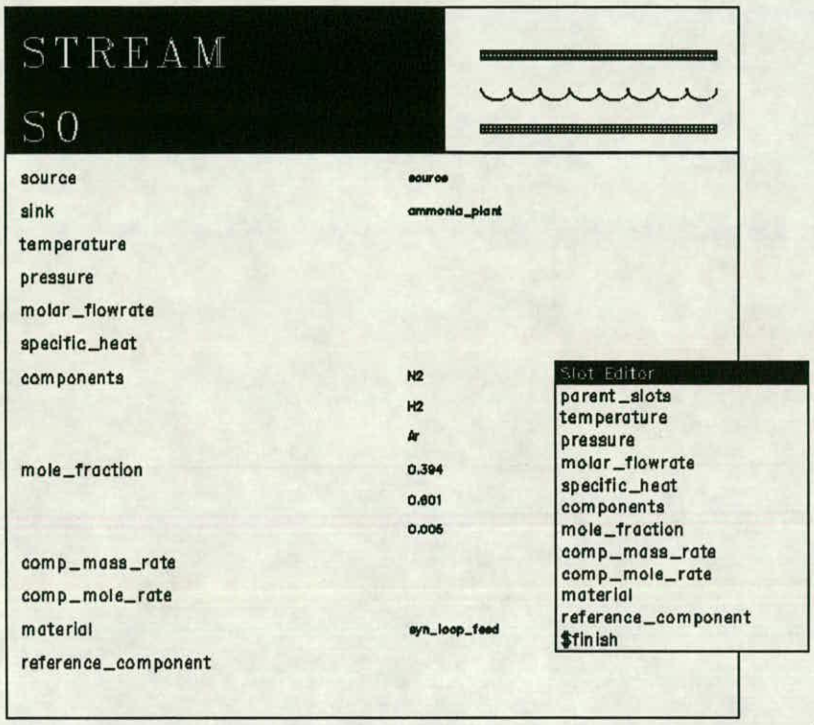
Figure E.12: Input Output Structure Message Indicating Required Action

This message prompts the user to edit stream s0 using the `analysis` menu, associating the stream with a material, in this case `syn_loop_feed`. Figure E.13 shows the result of setting the `material` slot to `syn_loop_feed`. The composition properties of the `syn_loop_feed` object are copied to the `components` and `mole_fraction` slots of the stream object.

The output streams, s1 and s2, are similarly associated with `material` objects, corresponding to `crude_ammonia` and `purge` respectively.

The conditions for completion of the input-output structure level have now been met. Selection of `$finish` in the menu shown in Figure E.11 advances synthesis to the recycle structure level. The structure of the main menu is unaltered. However, its label is changed to indicate the new synthesis level and a new set of decisions is available.

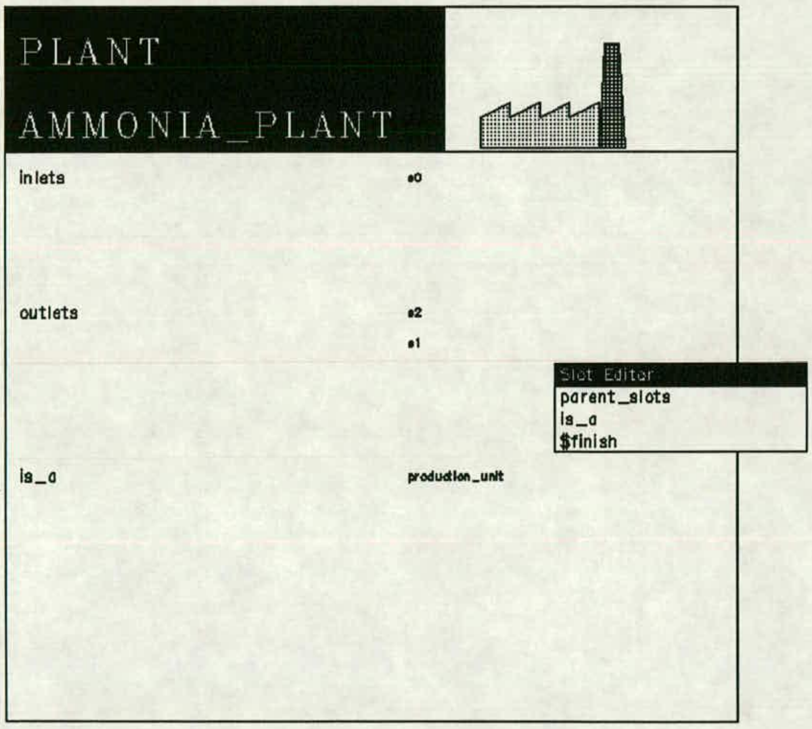
The synthesis level has advanced, but the flowsheet has only the detail provided by the input-output structure level, as shown by the `plant` object in Figure E.14. The first choice at the new level may be to develop the flowsheet. Alternatively, the `help` option can provide a high level statement of the goals to be achieved at this level. Another option is to assess the non-essential decisions.



STREAM
S0

```
source                      source
sink                        ammonia_plant
temperature
pressure
molar_flowrate
specific_heat
components                  N2
                            H2
                            Ar
mole_fraction               0.394
                            0.601
                            0.005
comp_mass_rate
comp_mole_rate
material                    syn_loop_feed
reference_component
```

Figure E.13: Object Describing Stream S0



PLANT
AMMONIA_PLANT

```
inlets                      s0
outlets                     s2
                            s1
is_a                        production_unit
```

Figure E.14: High Level Object Describing an Ammonia Plant

In this case, if **help** is selected, a requirement for objects representing reaction and separation is identified. If **decisions** is chosen, a message is displayed again indicating that at least one reactor is necessary to perform the reactions specified in the **process_chemistry** object. Figure E.15 shows the message generated for the ammonia process.

```
There is no reactor performing reaction:
[(N2+3*H2=2*NH3)-[temp-1000,pressure-500,catalyst-c13870]]
```

Figure E.15: Recycle Structure Message Indicating Required Action

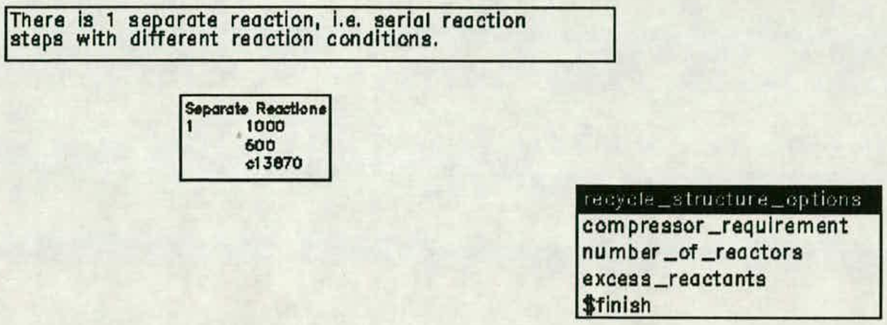
The first action, therefore, is to develop the flowsheet. Using the **topology** option, a new design node is added as a refinement of the initial description. Objects representing **reaction**, **separation** and a **divider** are created, and the connecting streams are defined.

As a result of the heuristic evaluations made at the input-output structure level, attention is focused on a single process. However, if the designer is not satisfied with a particular judgement, an alternative can be investigated. For example, at this point, an alternative design node can be created to investigate the possibility of removing the feed impurity before the reactor, thus avoiding a purge. In this example, this is impractical, but for a different process there is a range of alternatives which could be assessed at this level.

In this example, there is only one reaction, so only one reactor is required. However, where there are multiple reactions, they must be ordered and the reactants and recycles fed appropriately. One of the heuristic assessments at the recycle structure level provides a judgement on the required number of reactors and the different feed arrangements. The output for the current design is shown in Figure E.16.

The other options available for evaluation are also shown in the menu in Figure E.16. One option which has not been shown in this example, is for the assessment of the number of recycle streams. Here, the requirement for a single recycle was identified at the input-output structure level. This was provided when the refined flowsheet was described. Since the single recycle required has been specified, the option is not required, and so is not displayed.

The **excess_reactants** option is to make the designer aware of the possibility of shifting the product distribution in the reactor by feeding an excess of one reactant. This has an effect on the quantity and cost of recycling the excess, so



```
There is 1 separate reaction, i.e. serial reaction
steps with different reaction conditions.
```

```
Separate Reactions
1       1000
        600
        c13870
```

Figure E.16: Recycle Structure Options Menu and Assessment of Reactions

an optimum should be found. This requires an economic analysis of the different arrangements so the menu option is only to bring the possibilities to the designer's attention.

The other heuristic evaluation available at the recycle structure level is of the need for a compressor on the recycle stream. If a gas recycle is to be used, a major process cost will be incurred by the need for a gas compressor. This decision is introduced at this level because of the effect on the total process cost. It is possible that this process alternative can be eliminated with minimum time being expended.

Figure E.17 shows the response to the selection of `compressor_requirement` with the current data. The recycle is correctly identified, but the compressor requirement cannot be evaluated until there is information about the components in s6, one of the streams in the recycle.

```
1 recycle stream has been specified.
s6 does not have any components specified, so no conclusion about the
necessity of a gas compressor can be made.
```

Figure E.17: Message Indicating Incomplete Process Specification

When the `components` slot of s6 has been set to the list of components being recycled, reselection of this option results in the judgement shown in Figure E.18. In this example, the recycle will be a gas stream, i.e. it has a boiling point less than that of propane, which, heuristically, is the limit for condensing under pressure by cooling water.

It is important to model the flowsheet to give values to the flowrates into and out of the process which can then be used in an economic analysis. It is necessary, therefore, to ensure that the flowsheet is complete at this level. This can be done

```
1 recycle stream has been specified.
The chances are that s6 will be a gas recycle stream, because it cannot
be completely condensed by cooling water even at high pressure.
```

Figure E.18: Heuristic Judgement of Compressor Requirement

either by selecting **help** on the main menu, or **$finish** and reacting to any error messages. If the synthesis procedure advances to the next level, the flowsheet is structurally complete.

Here, if **$finish** is selected, the recycle structure checks ensure that a reaction section and separation section have been defined, all components leave the system and purity specifications are maintained. For example, Figure E.19 shows a message indicating that H_2 is not leaving the system.

```
Component H2 is not in any of the output streams.
More information is required at the recycle_structure_level.
```

Figure E.19: Message Indicating Components Not Leaving the System

This is rectified by adding H_2 to the **components** slot of the appropriate outlet stream, in this case, the purge. On reselection of **$finish**, a change in product specification is highlighted, as shown in Figure E.20. The **technology** object has a specification for 99.5% pure ammonia while the product stream has a value of 99%. The inconsistency is resolved by changing the product specification to 99.5%.

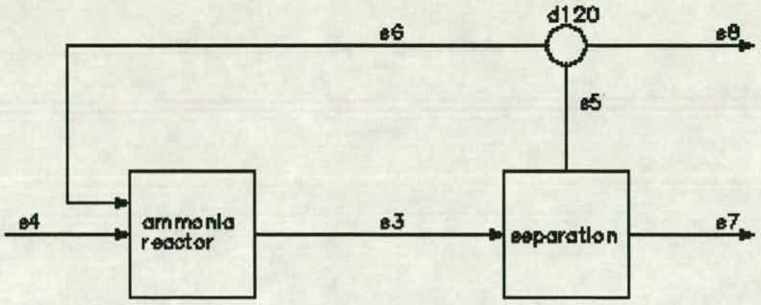
```
The product NH3 is not in any of the outlet streams ([s8,s7]) as pure
as 99.5%, as was specified.
More information is required at the recycle_structure_level.
```

Figure E.20: Message Indicating that the Original Purity Specification is Contradicted

The conditions for progress have now been met, so a model of the flowsheet can be constructed using the **model_flowsheet** option in the **flowsheet analysis** menu. This instigates the model generation methods which select equations for each unit in the flowsheet and each stream. In this example, a reaction balance is used for the **reaction** object, a component balance for the **separation** and a component balance incorporating divider ratios for the **divider**. The equations for the streams relate mole fractions of constituent components, component molar flowrates and the overall molar flowrate.

The specifications placed on the process are 100000tonnes/yr of ammonia to be produced from a given feed and the mole fraction of Ar in the system must not exceed 0.01. The Ar constraint is placed on the reaction section output. The product is assumed to be pure since the separation is not sufficiently detailed to determine which other components would be in that stream.

The model is then evaluated by the degrees of freedom algorithm described in Appendix B. Here, the specifications are acceptable and the equations are solved simultaneously. The results are shown in Figure E.21.



	e3	e4	e5	e6	e7	e8
NH3	700.0	0	0	0.0	700.0	0.0
N2	3750.0	1100	3750.0	3000.0	0	750.0
H2	3150.0	1680.0	3150.0	2520.0	0	630.0
Ar	75	15	75.0	60.0	0	15.0
Total	7675.0	2795.0	6975.0	5580.0	700	1395.0

Figure E.21: Solution of Flowsheet at Recycle Structure Level

The display used here is different from that discussed in Appendix F. The form of the display was determined by the presence of the graphical description which was provided manually. The stream table was generated automatically.

The synthesis procedure has now advanced to consideration of the reactor system. Again, the main menu retains the same structure, only altering the label and the available decisions. As discussed above, the first action could be to refine the flowsheet, or select `help`, or heuristically evaluate the flowsheet with the `decision` option.

If help is selected, the extended method containing the checks for essential action is interpreted to describe the requirements of the current synthesis level. The calling sequence of the extended method is a set of high level statements of the aims of the level. The associated guards are the criteria for meeting these aims. The high level aims are presented as a menu.

In this case the only goal is that the reactors must be "fully specified". Selection of an aim from the menu produces a list of criteria as a rough translation of the code in the guard associated with the chosen aim. Here, the only choice produces the list of statements shown in Figure E.22.

```
reactors_suitable must be satisfied.
know_adiabatic_delta_t must be satisfied.
process_chemistry must be specified
reactions_associated_with_reactors must be satisfied.
```

Figure E.22: Output from Help Option at Reactor System Level

Those ending with "must be satisfied" refer to Prolog goals, the predicate of which is displayed as the condition. The predicates were named to provide as much information as possible, e.g. know_adiabatic_delta_t. Statements ending in "must be specified" refer to slots which must have values, e.g. process_chemistry.

The first statement in Figure E.22 suggests that the specified reactor may not be of sufficient detail for further evaluation. This is confirmed by the heuristic design decisions at this level. The first of these options is to evaluate the reactor heating characteristics, which, on initial selection, displays a message stating the requirement of a more detailed reactor for this operation. This is shown in Figure E.23.

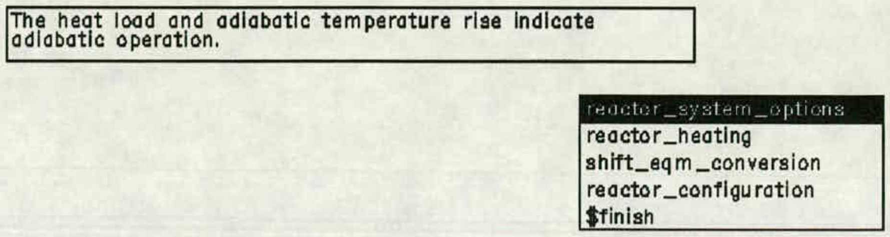
```
The reactors specified so far are not of sufficient detail
to allow an investigation of their heating properties.
More information is required at the reactor_system_level.
```

Figure E.23: Message Indicating Detail of Reactor System is Insufficient

The decisions menu includes an option to propose a reactor configuration based on reaction kinetics. This is detailed by Douglas [7], but has not been fully implemented.

To advance the design, a new node is added to the graph as a refinement of the developed recycle structure. A plug flow reactor, v100, is placed in this new flowsheet, providing the increase in detail required for the evaluation of the reactor heating properties. Further requirements for this judgement are values for reactor heat load and adiabatic temperature rise as indicated in Figure E.22. These values are calculated as part of the design method for the reactor.

The decisions menu is shown in Figure E.24 with the output from the reactor_heating evaluation. The results indicate adiabatic operation.



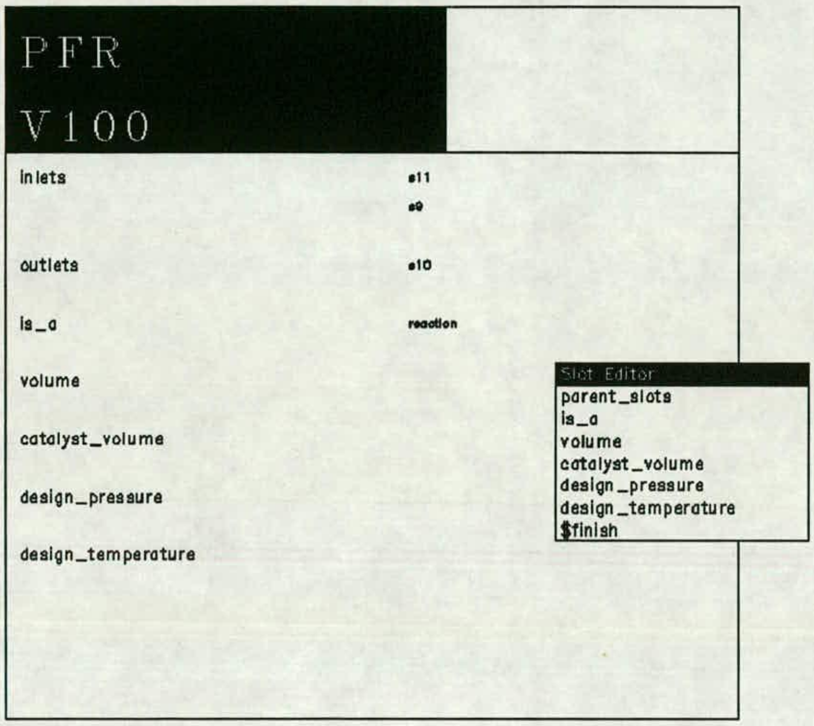
```
The heat load and adiabatic temperature rise indicate
adiabatic operation.
```

Figure E.24: Heuristic Assessment of Reaction Heating Requirements

Consideration of the reactor system is now complete, but as Figure E.25 shows, the information describing the plug flow reactor is not. This requires the development of a plug flow reactor design procedure similar to that described for a distillation column in Appendix F. The use of models specific to such reactors can then be made available.

Figure E.25 shows the slots which distinguish a plug flow reactor from the prototype reaction object. The slots defined for a reaction object are inherited by the plug flow reactor and are accessed by selecting parent_slots in the slot editor menu. Figure E.26 shows these slots with their values many of which have been automatically copied from the more abstract reactor defined in the previous level of the graph. Other values are provided manually after heuristic evaluation using the decisions option on the main synthesis menu.

The synthesis procedure now advances to the separation structure level which is divided into liquid separation and vapour recovery. In this example, the reactor output is a gas, and even with cooling water cannot condense to afford liquid and vapour phases requiring separate treatment. The options for vapour recovery are presented as a table ordered by increasing cost. Normally, the least expensive recovery is condensation, which in this example requires refrigeration. It is a feasible approach since, as Figure E.8 shows, the boiling point of ammonia is significantly higher than the other components. An alternative which could be



PFR
V100

```
inlets                      s11
                            s9

outlets                     s10

is_a                        reaction

volume

catalyst_volume

design_pressure

design_temperature
```

```
Slot Editor
parent_slots
is_a
volume
catalyst_volume
design_pressure
design_temperature
$finish
```

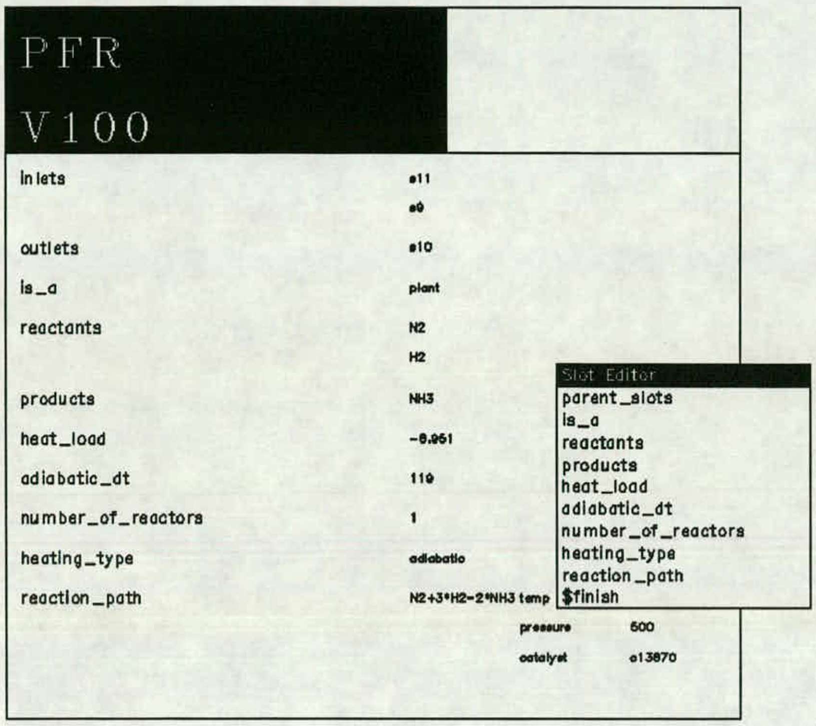
Figure E.25: Object Representing Plug Flow Reactor V100

considered is absorption into water. The two proposals can be distinguished on the basis of which is most expensive, separating ammonia from water or a refrigerated condenser.

As with the reactor described above, consideration of these separation alternatives is complete. Design procedures are required to continue the design of the unit operations.

In summary, this example has illustrated the flexible access to synthesis decisions as well as tools for describing flowsheets and modelling them. The main menu is presented in a consistent format throughout, altering only the label, which informs the designer of the level of synthesis, and the decisions which can be accessed. Action which is essential form the completion of a decision level is ensured by preventing advance to the next level until the information is provided. Options which the designer should be aware of at each level are presented in a separate menu. Heuristic advice is available where appropriate.

The Douglas synthesis procedure is intended to guide the designer to a good base case design. It is used here to aid the reduction of stored information by focusing design effort on the process alternatives which, on the basis of heuris-



```
PFR
V100
inlets                s11
                      s9
outlets               s10
is_a                  plant
reactants             N2
                      H2
products              NH3
heat_load             -8.951
adiabatic_dt          119
number_of_reactors    1
heating_type          adiabatic
reaction_path         N2+3*H2-2*NH3 temp
                                  pressure    500
                                  catalyst    c13870
```

Figure E.26: Slots of V100 Showing Access to Less Detailed Definition

tics, show most potential. However, to complete a design, strategies should be available to support design of individual unit operations.

Appendix F

Worked Example of Opportunistic Modelling in Distillation Column Design

The example in this appendix demonstrates the opportunistic approach to design of a unit operation, in this case, a distillation column. The diagrams were produced as screendumps from the prototype system discussed in the main body of this thesis. The procedure adopted is detailed in Chapter 6. A structure of extended methods provides a framework for high level aims, which, in this example, first allow assessment of the feasibility of distillation, then determination of column conditions and finally calculation of the number of trays. Within that structure, parallel calls use menus to allow the opportunistic selection of tasks. Thus, appropriate options are provided as the need arises, and flexible use of the tools is achieved.

The initial menu shown in Figure F.1, is the general design options menu discussed in Section 6.1. In this example, the menu corresponds to the liquid separation system structure level of the Douglas decision hierarchy.

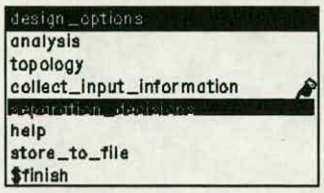


Figure F.1: Menu of Design Options

The options of `help`, `store_to_file` and `collect_input_information` are discussed in Section 6.1. The other choices are demonstrated in this appendix.

The **separation_decisions** method, which is highlighted in Figure F.1 applies to a single separation object. For this reason, when it is selected, the user is required to indicate the separation to which the method is referring. In this example only one separation is defined, that of benzene and toluene shown in Figure F.2. The composition of the inlet stream is known and specifications have been placed on the purities of the outlet streams. The remaining mole fractions have been determined by demons on the mole fraction slot of each stream.

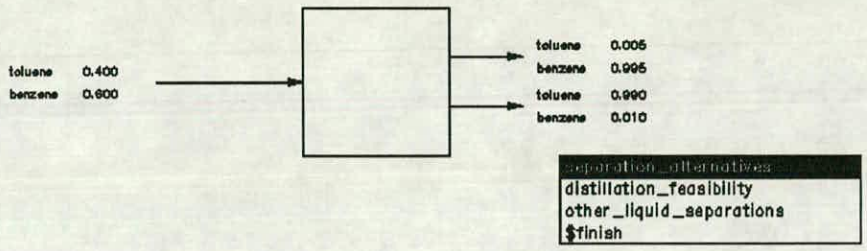
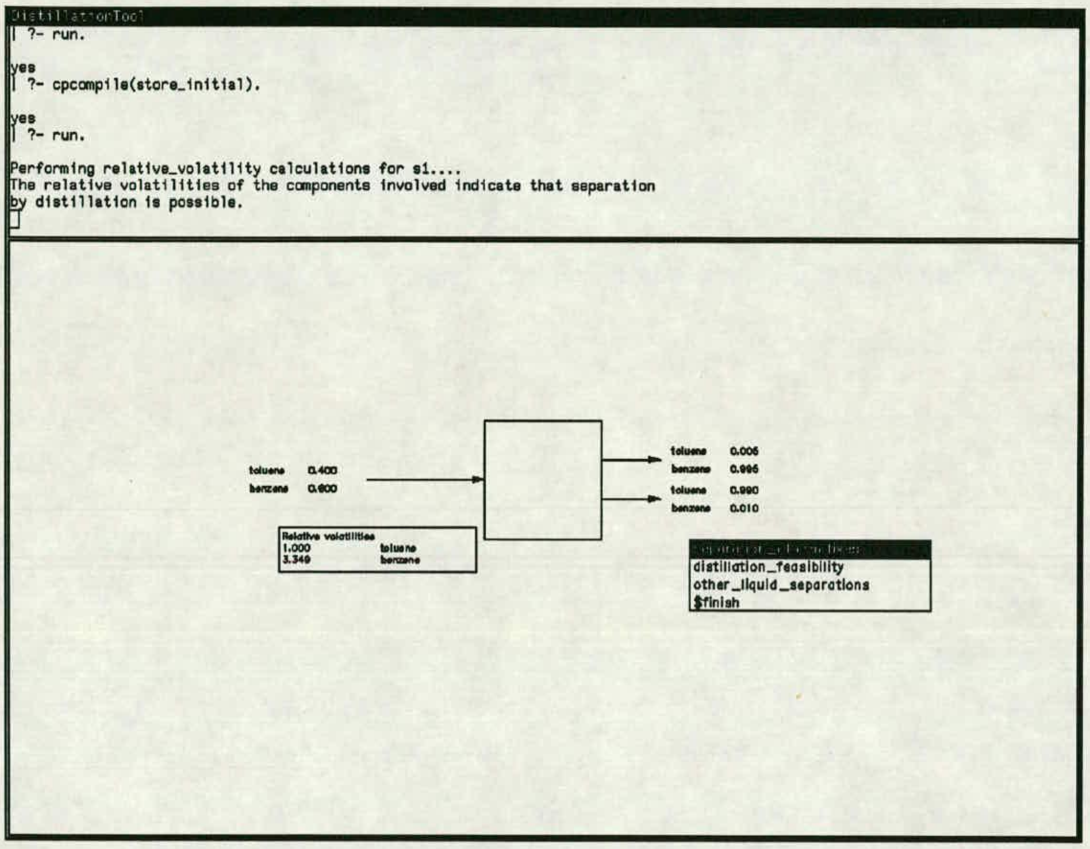


Figure F.2: Separation Design Alternatives

The information that is available at this point allows the choice of evaluating the feasibility of distillation or assessing other liquid separations. These options are supplemented later when more is known.

If `distillation_feasibility` is selected, the relative volatilities of the components in the feed stream are calculated. If more than one stream is fed to the separation, all components are collected and evaluated. Douglas [7] suggests a heuristic method for determining whether or not distillation will be feasible without designing a column. If the relative volatilities of the key split is less than 1.1 then distillation will probably not be economical.

A summary of the results of the relative volatility calculations is shown in Figure F.3. The individual results are presented as they are calculated. The figure shows the split window used for this design tool. The top section is for textual presentation and input. The bottom part is for graphical presentation and menu input. Here, the text window reports that calculations have been performed and the result is that distillation is feasible. The graphical window summarises the calculations with a table of relative volatilities. Here, benzene has a relative volatility of 3.3 compared to toluene .

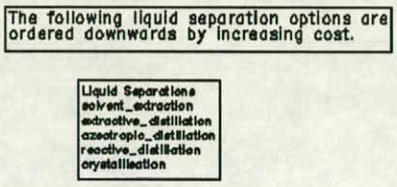


```
| ?- run.

yes
| ?- cpcompile(store_initial).

yes
| ?- run.

Performing relative_volatility calculations for s1....
The relative volatilities of the components involved indicate that separation
by distillation is possible.
```

```
Relative volatilities
1.000          toluene
3.349          benzene
```

Figure F.3: Result of Relative Volatility Calculations

The menu still presents the option of evaluating other liquid separations. In this example, selection of this option results in a table of liquid separations ordered by increasing cost. This is shown in Figure F.4. A full implementation should provide similar help in the design of these alternatives as with the design of distillation.



```
The following liquid separation options are
ordered downwards by increasing cost.
```

```
Liquid Separations
solvent_extraction
extractive_distillation
azeotropic_distillation
reactive_distillation
crystallisation
```

Figure F.4: Selection of Alternative Separations

No further information can be generated from these decision options, so the designer must return to the main menu by selecting `$finish`.

Other evaluation options are available at this stage, so the designer is able to analyse the flowsheet as in Figure F.5. The figure shows the "Analysis" menu which is described in Section 5.7.

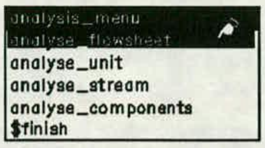
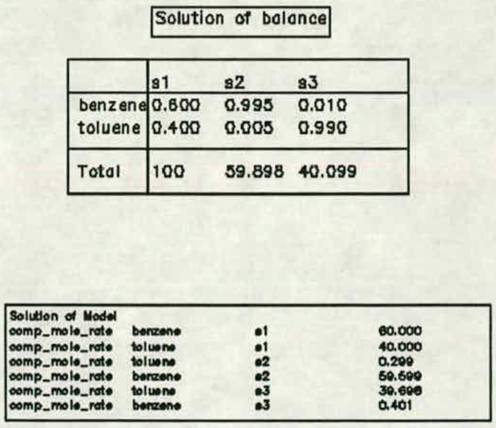


Figure F.5: Analysis Menu Options

The options to analyse a flowsheet include calls to move within the design graph, i.e. select a flowsheet for analysis, and model the selected plant. In this example, the separation section has been modelled, the results being displayed in Figure F.6. The solution has been displayed in two parts:

- A stream table of mole fractions for each component in each stream, and total stream flowrates.
- A table of other values produced during calculation. In this case, this corresponds to the individual component flowrates in each stream.



Solution of balance

	s1	s2	s3
benzene	0.600	0.995	0.010
toluene	0.400	0.005	0.990
Total	100	59.898	40.099

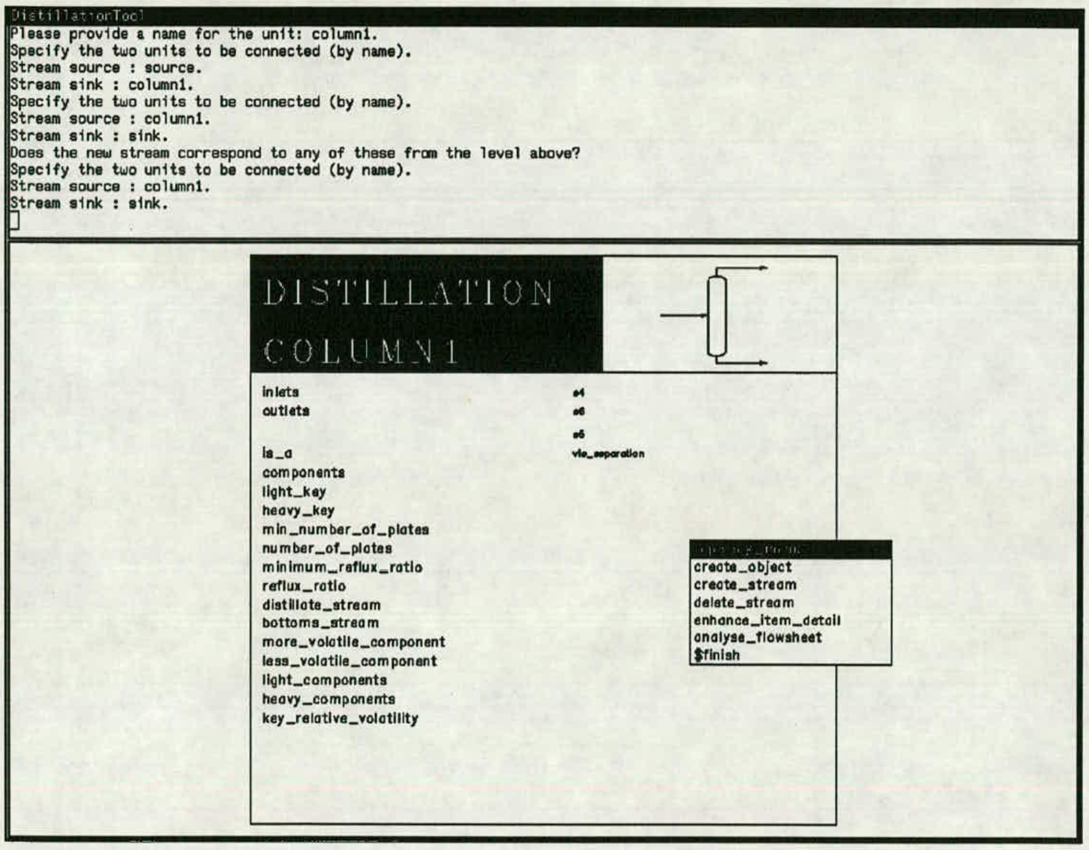
Solution of Model

comp_mole_rate	benzene	s1	60.000
comp_mole_rate	toluene	s1	40.000
comp_mole_rate	toluene	s2	0.299
comp_mole_rate	benzene	s2	59.599
comp_mole_rate	toluene	s3	39.698
comp_mole_rate	benzene	s3	0.401

Figure F.6: Solution of Flowsheet Model

The model generated for this separation has been an overall component balance. The greatest amount of detail that can be provided in a model of a separator is a mass balance including split fractions. Here, no fractions have been specified, so the next best model is either a reaction balance or a component balance. Since no reaction is associated with this unit the component balance is chosen.

So far, a mass balance has been performed around the separation section, and it has been determined that distillation is a feasible method for the separation indicated. To further investigate distillation an additional design node is required. This is achieved *via* the **topology** option on the main menu. The full range of operations is shown in Figure F.7.



```
Please provide a name for the unit: column1.
Specify the two units to be connected (by name).
Stream source : source.
Stream sink : column1.
Specify the two units to be connected (by name).
Stream source : column1.
Stream sink : sink.
Does the new stream correspond to any of these from the level above?
Specify the two units to be connected (by name).
Stream source : column1.
Stream sink : sink.
```

```
DISTILLATION
COLUMN1

inlets                        s4
outlets                       s6

                              s5
is_a                          vle_separation
components
light_key
heavy_key
min_number_of_plates
number_of_plates
minimum_reflux_ratio
reflux_ratio
distillate_stream
bottoms_stream
more_volatile_component
less_volatile_component
light_components
heavy_components
key_relative_volatility
```

Figure F.7: Addition of a Design Node Containing a Distillation Column

Selection of `enhance_item_detail` creates a new design node as a child of the current one and moves the current focus there. Figure F.7 shows the result of creating a new design node. The `create_object` option has been used to create an object representing a distillation column. The text window shows the textual input of the column's name "column1".

Streams are associated with the column using `create_stream`. A message is printed in the text window asking for the specification of the units to be connected. In this example, the first stream is the feed to the column. This is specified as a connection between a **source** and **column1**. The new stream is named automatically as "s4" and is associated with the equivalent stream at the level above, i.e. s1. The specification of the outlet streams is achieved similarly, except that there are two outlets from the separation section at both levels. In this case, the program asks for the association between equivalent streams to be performed by the user. A message appears in the text window indicating that the association is required, as in Figure F.7, and a menu of the relevant streams is displayed in the graphics window. This menu is not shown in the figure.

Figure F.7 shows the resulting column object. The only values specified at this point are the names of the inlet and outlets streams and the generic type of which distillation is a specialisation, i.e. `vle_separation`

Returning to the main menu shown in Figure F.1, selection of `separation_decisions` reveals an additional operation at this level. Since a distillation column is now present in the current design node the option of `distillation_evaluation` is made available, as shown in Figure F.8. The graphical representation of the separation is unaltered.

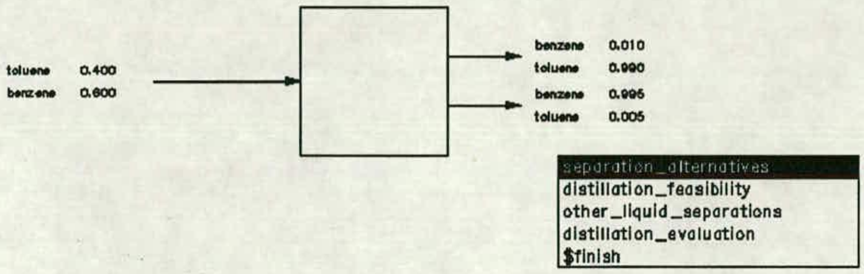
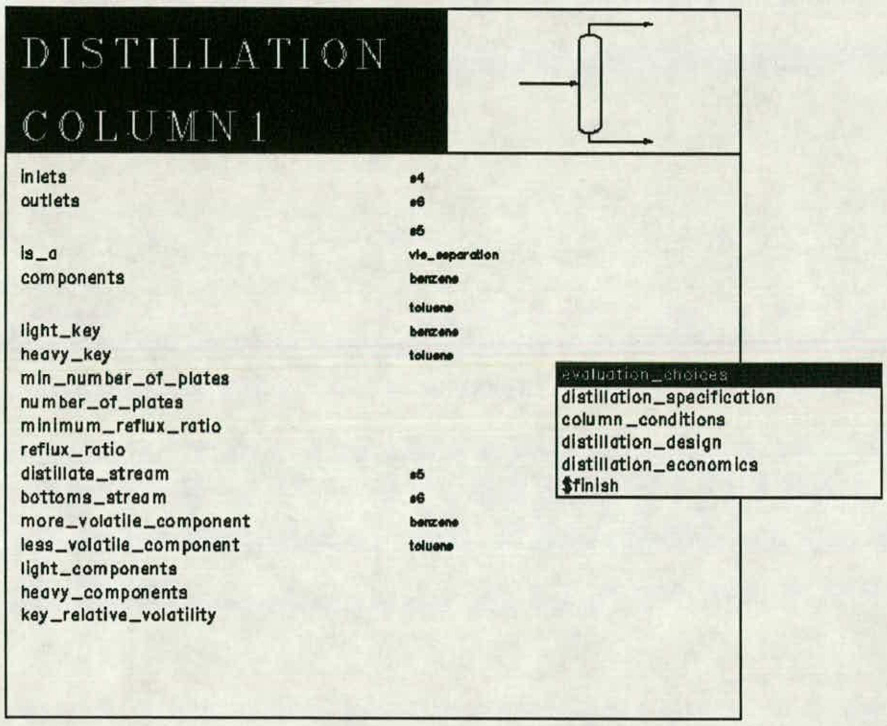


Figure F.8: Additional Call for Separation Evaluation

It is possible for the designer to have changed the inlet specifications of the column so the options for reassessing distillation feasibility and other liquid separations are available.

Selection of `distillation_evaluation` invokes a method which determines details of the distillation operation from the separation at the level above. The components are located and the corresponding slot in `column1` is set. The relative volatility information is used along with the specifications on the product streams to determine the light and heavy keys. The distillate and bottoms streams are deduced by locating which of the output streams has most of the light key. This stream is taken to be the distillate, implying that the other is the bottoms. In the example here, benzene and toluene are the only two components present, so the mixture is binary. Thus the slots for more and less volatile components can be set. The resulting specifications are shown in Figure F.9.



```
DISTILLATION
COLUMN1

inlets                          s4
outlets                         s6

                                s5
is_a                            vle_separation
components                      benzene

                                toluene
light_key                       benzene
heavy_key                       toluene
min_number_of_plates
number_of_plates
minimum_reflux_ratio
reflux_ratio
distillate_stream               s5
bottoms_stream                  s6
more_volatile_component         benzene
less_volatile_component         toluene
light_components
heavy_components
key_relative_volatility
```

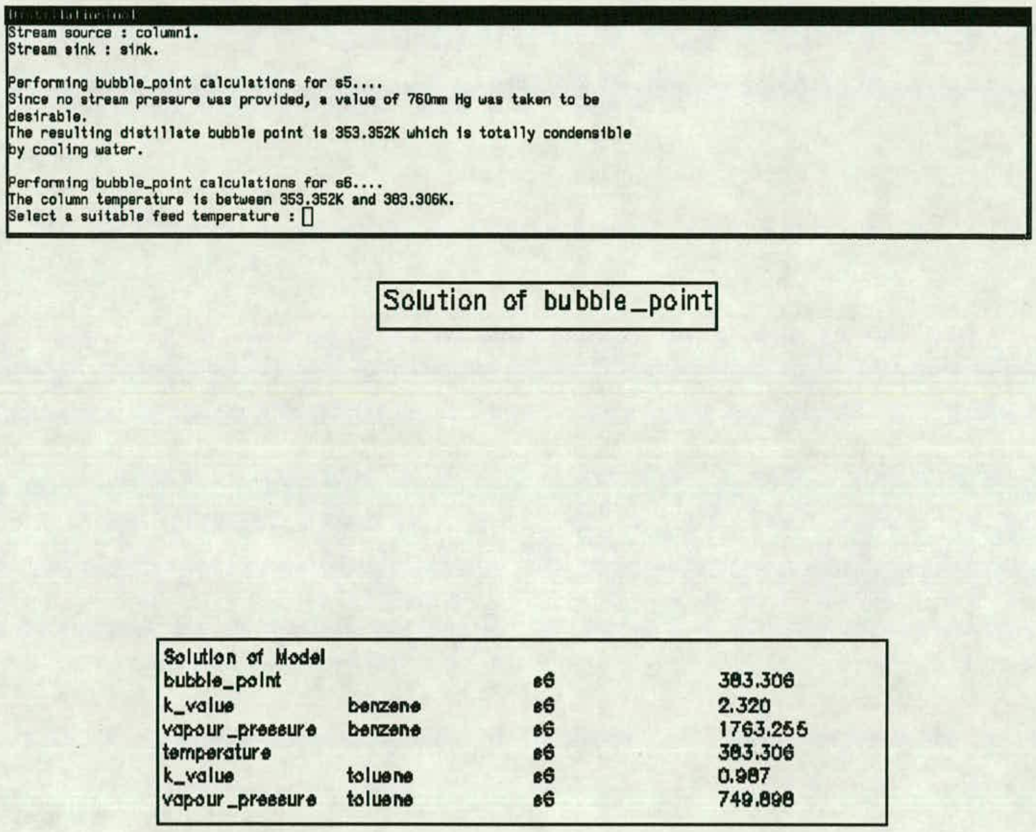
Figure F.9: Distillation Evaluation Choices

The evaluation options open to the designer at this stage are shown in the menu in Figure F.9. `distillation_specification` allows editing of the slots shown in the figure.

If a strict procedure is adhered to, the specification of column conditions should be performed now using the `column_conditions` option. However, they may have been specified before this point, in which case, `column_conditions` can be used to check the specifications against the criteria that it is desirable to have a column pressure near atmospheric and temperatures such that the distillate can be condensed by cooling water and the bottoms boiled by low pressure steam.

The `column_conditions` method calculates the bubble point of the tops and bottoms streams at the specified pressure or 760mmHg if no value is given. These are checked against the temperatures of cooling water and low pressure steam respectively. If the values are acceptable, the specified pressure is accepted, otherwise the pressure required to meet the temperature constraints is calculated. The designer is able to accept the calculated values or select different ones.

Figure F.10 shows the result of the bubble point evaluation for the benzene/toluene example. The text window provides a history of the method.



```
Stream source : column1.
Stream sink : sink.

Performing bubble_point calculations for s5....
Since no stream pressure was provided, a value of 760mm Hg was taken to be
desirable.
The resulting distillate bubble point is 353.352K which is totally condensible
by cooling water.

Performing bubble_point calculations for s6....
The column temperature is between 353.352K and 383.306K.
Select a suitable feed temperature : []
```

```
Solution of Model
bubble_point                          s6              383.306
k_value          benzene              s6              2.320
vapour_pressure  benzene              s6              1763.255
temperature                           s6              383.306
k_value          toluene              s6              0.987
vapour_pressure  toluene              s6              749.898
```

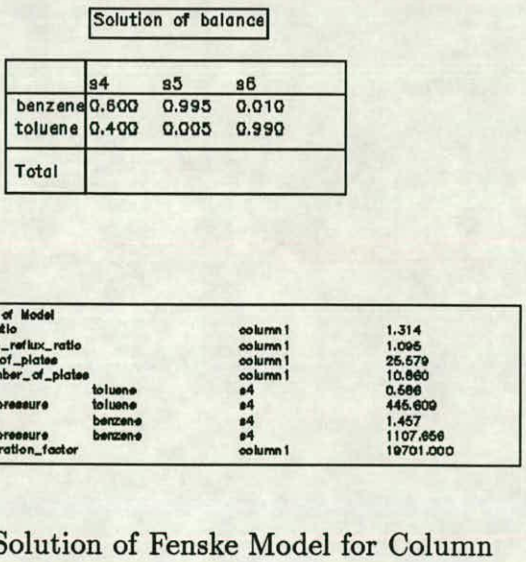
Figure F.10: Verification of Column Feed Temperature

Initially, the bubble point of distillate stream s5 was calculated. Since no pressure was specified, a value of 760mmHg was used. The solution was presented in the graphics window with a summary in the text window. In this case, the bubble point is 353K which, as has been reported, is condensible using cooling water. The same evaluation was performed for bottoms stream s6. The solution of the calculation is shown in the graphics window. The column temperature range is thus identified and a request is made for a suitable feed temperature.

Once column conditions are set, the menu of distillation evaluation choices as shown in Figure F.9, is presented again. Selection of `distillation_design` should invoke a procedure for calculating column dimensions, such as number of trays, tray sizes, column diameter, etc. For this example, only calculation of the number of trays has been implemented. It is sufficient, however, to demonstrate the principles discussed in Section 6.2, i.e. design of unit operations has a procedural structure which depends on the specifications, and full access to modelling facilities is required.

Selection of `distillation_design` invokes the design method which, in this example, only contains a procedure for calculating the number of plates in the column. For the current distillation column, two applicable models are presented: the Fenske equation and a shortcut binary model. If the separation had not been binary, the shortcut binary model would not have been presented.

It is possible to use both models separately. They involve different assumptions so the designer must determine which is most suitable. Here, both models have been evaluated. The results of the Fenske model are shown in Figure F.11. The available stream data is printed in a stream table, and other values in a second table. The missing totals in the stream table indicate the incompleteness of the model for flowsheeting purposes. The aim of the calculation was to determine the number of plates in the column, but other useful information has been generated by the model. This data, including minimum reflux ratio and the minimum number of plates, is displayed in the second table.



Solution of balance

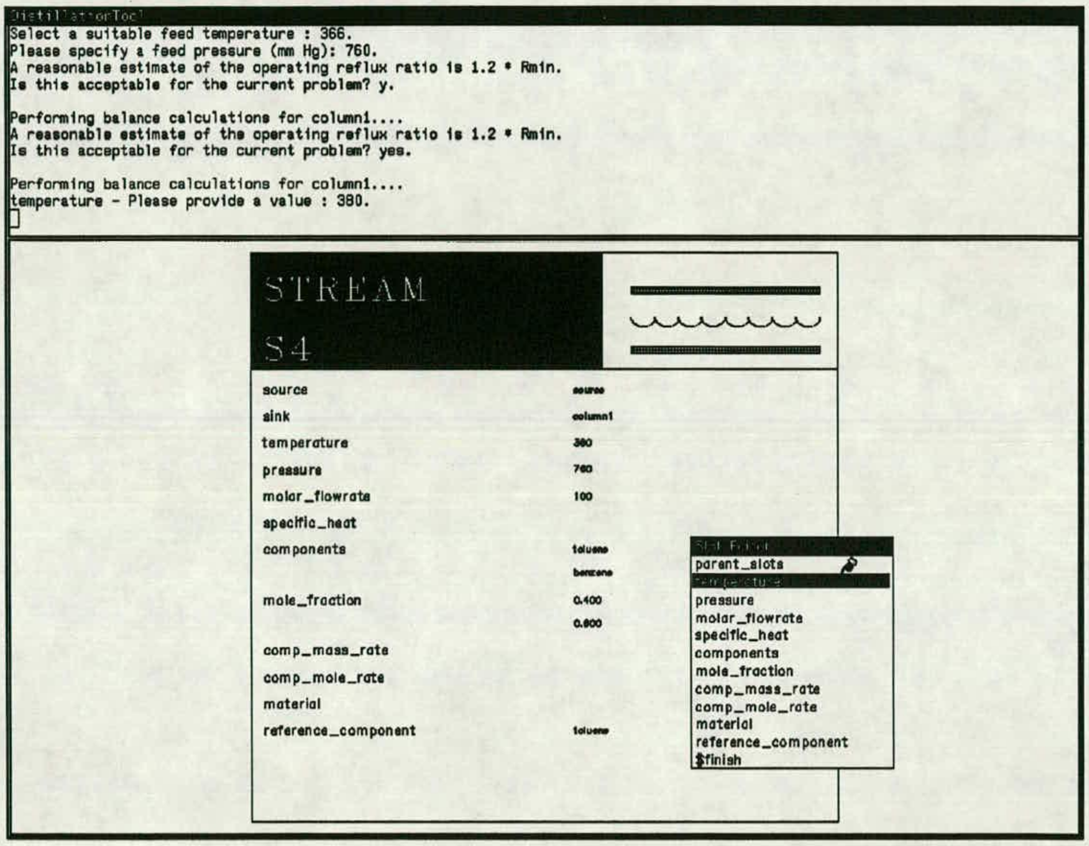
	s4	s5	s6
benzene	0.600	0.995	0.010
toluene	0.400	0.005	0.990
Total			

Solution of Model

reflux_ratio		column1	1.314
minimum_reflux_ratio		column1	1.096
number_of_plates		column1	25.579
min_number_of_plates		column1	10.860
k_value	toluene	s4	0.586
vapour_pressure	toluene	s4	445.609
k_value	benzene	s4	1.457
vapour_pressure	benzene	s4	1107.656
mf_separation_factor		column1	19701.000

Figure F.11: Solution of Fenske Model for Column

The models used to evaluate the distillation column can be modified to assess different specifications and levels of assumption. By returning to the main menu, the analysis options are made available. In order to try different specifications, the feed stream must be edited. Selection of the `analysis` option on the main menu followed by `analyse_stream` (see Figure F.5) provides access to the slots of a particular stream. Here, the feed stream s4 is edited in Figure F.12. `Temperature` is selected from the edit menu and the value, 380, entered in the text window.



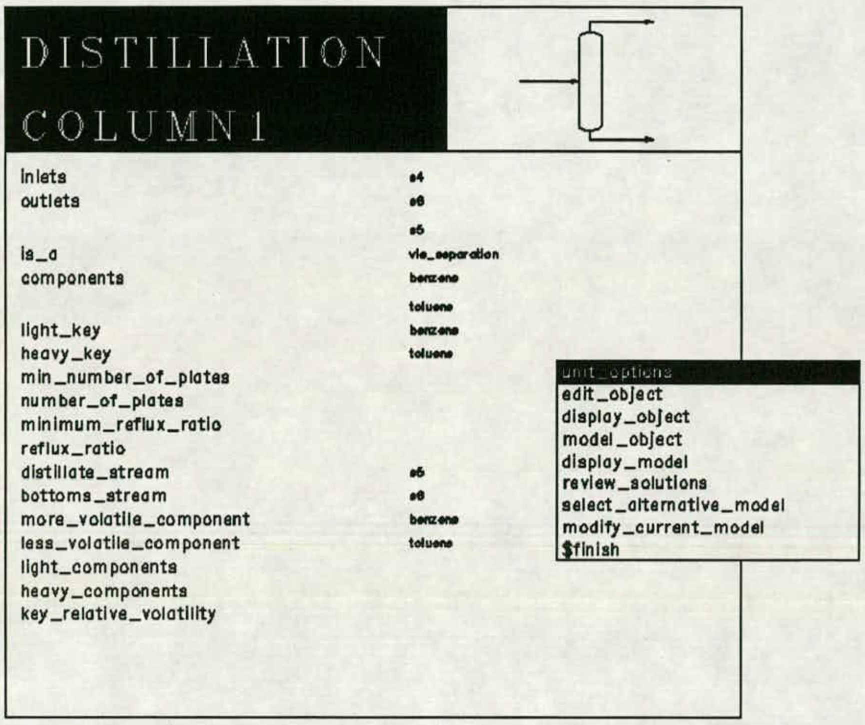
```
Select a suitable feed temperature : 366.
Please specify a feed pressure (mm Hg): 760.
A reasonable estimate of the operating reflux ratio is 1.2 * Rmin.
Is this acceptable for the current problem? y.

Performing balance calculations for column1....
A reasonable estimate of the operating reflux ratio is 1.2 * Rmin.
Is this acceptable for the current problem? yes.

Performing balance calculations for column1....
temperature - Please provide a value : 380.
```

source	source
sink	column1
temperature	380
pressure	760
molar_flowrate	100
specific_heat	
components	toluene
	benzene
mole_fraction	0.400
	0.600
comp_mass_rate	
comp_mole_rate	
material	
reference_component	toluene

Figure F.12: Altering the Feed Temperature Specification

Once the specification has been changed, the models of the distillation column can be re-evaluated. The `analyse_unit` option in the `analysis menu` provides access to the choices shown in Figure F.13, including modelling facilities. The choices are detailed in Section 5.7.

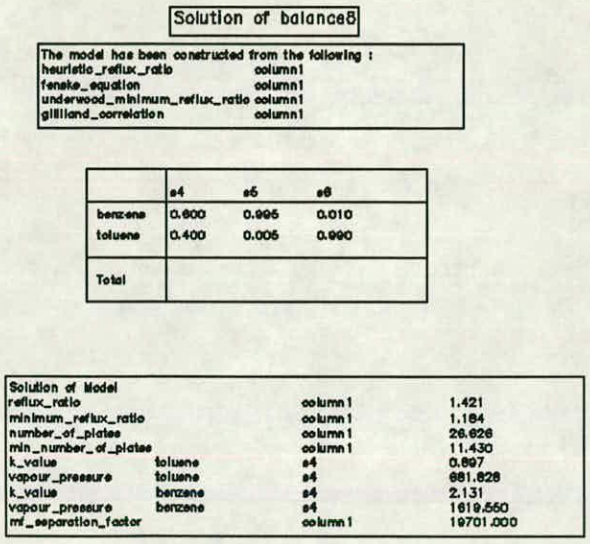
The model with revised specifications is evaluated by selecting `model_object`. When the results are presented as in Figure F.11, the designer may recognise that they are different from those calculated previously. To review all solutions generated for models of this unit, `review_solutions` is selected. The chosen model is displayed as shown in Figure F.14. This is the Fenske model with revised specifications. When compared with Figure F.11 it can be seen that the number of plates and the reflux ratio is different in the two solutions.



DISTILLATION
COLUMN1

inlets	s4
outlets	s6
	s5
is_a	vle_separation
components	benzene
	toluene
light_key	benzene
heavy_key	toluene
min_number_of_plates	
number_of_plates	
minimum_reflux_ratio	
reflux_ratio	
distillate_stream	s5
bottoms_stream	s6
more_volatile_component	benzene
less_volatile_component	toluene
light_components	
heavy_components	
key_relative_volatility	

Figure F.13: Options for Process Unit Analysis



Solution of balance8

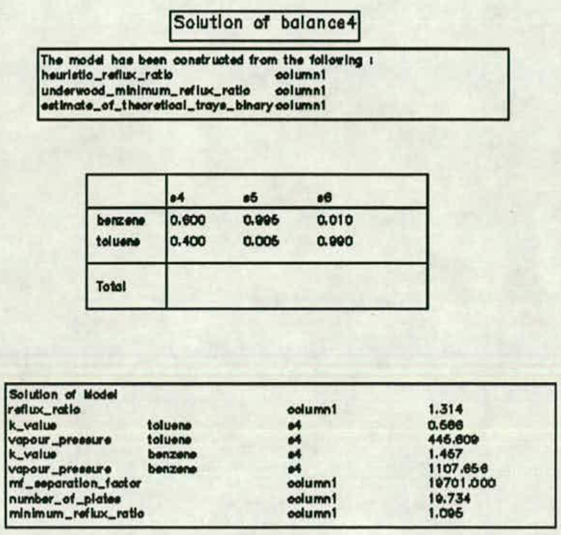
The model has been constructed from the following :	
heuristic_reflux_ratio	column1
fenske_equation	column1
underwood_minimum_reflux_ratio	column1
gilliland_correlation	column1

	s4	s5	s6
benzene	0.600	0.995	0.010
toluene	0.400	0.005	0.990
Total			

Solution of Model			
reflux_ratio		column1	1.421
minimum_reflux_ratio		column1	1.184
number_of_plates		column1	26.626
min_number_of_plates		column1	11.430
k_value	toluene	s4	0.897
vapour_pressure	toluene	s4	681.828
k_value	benzene	s4	2.131
vapour_pressure	benzene	s4	1619.560
mf_separation_factor		column1	19701.000

Figure F.14: Reviewed Fenske Model

The solution in Figure F.14 is supplemented by an additional table detailing the models used. This is useful for comparing results from two different models. For instance, the solution of the shortcut binary model can be reviewed as in Figure F.15. This extra table details the difference in the basis of the models.



```
Solution of balance4

The model has been constructed from the following :
heuristic_reflux_ratio                column1
underwood_minimum_reflux_ratio        column1
estimate_of_theoretical_trays_binary column1
```

	s4	s5	s6
benzene	0.600	0.995	0.010
toluene	0.400	0.005	0.990
Total			

```
Solution of Model
reflux_ratio                             column1     1.314
k_value              toluene             s4          0.586
vapour_pressure      toluene             s4          446.609
k_value              benzene             s4          1.467
vapour_pressure      benzene             s4          1107.656
mf_separation_factor                     column1     19701.000
number_of_plates                         column1     19.734
minimum_reflux_ratio                     column1     1.095
```

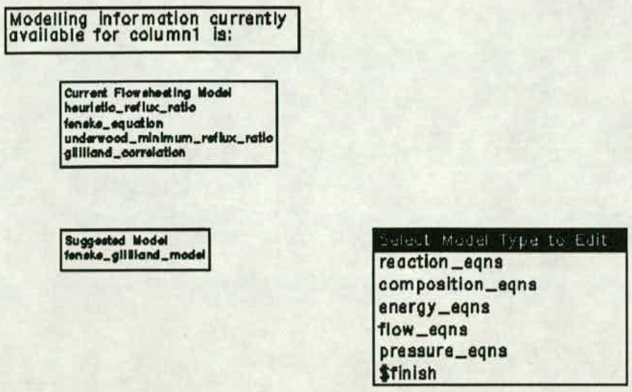
Figure F.15: Reviewed Binary Distillation Model

Other facilities would be useful for displaying models. In particular, simultaneous displaying of different models would allow direct comparison. The graphics system used in the development of CLAP, however, cannot support multiple windows and graphical input together. Another useful feature would be a table displaying the specifications on the model, thus highlighting the differences between models such as the two Fenske models here.

The `analyse_unit` method supports the models used for flowsheet modelling. These are now available to the designer to replace those generated by the distillation design method.

The option `select_alternative_model` informs the user of the current model and the one suggested as most appropriate for the current unit. Figure F.16 shows the model currently used for describing the distillation column and the suggested `fenske_gilliland_model`. The two models here may be very similar, but the old model is not decomposed into the five modelling areas shown in the menu, so if the model is being altered according to this decomposition, one that is known to conform to it must be in place.

It may be decided here to model the distillation column with a simple mass


Modelling information currently available for column1 is:

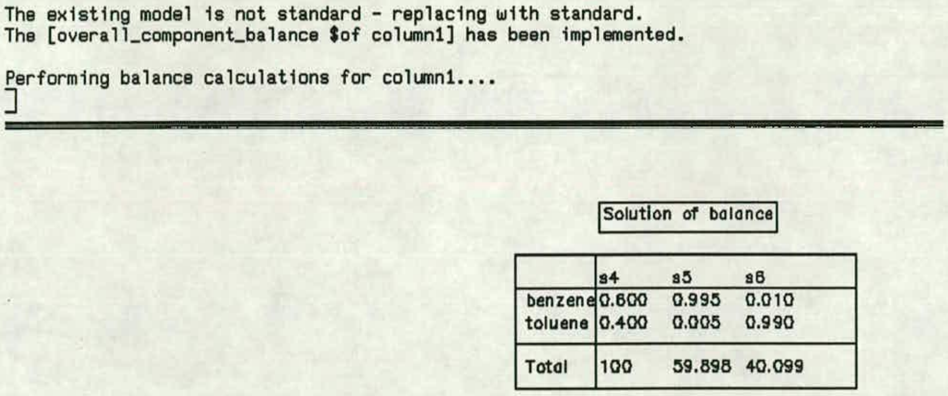
Current Flowsheeting Model
heuristic_reflux_ratio
fenske_equation
underwood_minimum_reflux_ratio
gilliland_correlation

Suggested Model
fenske_gilliland_model

Figure F.16: Selection of an Alternative Distillation Model

balance. The current Fenske model does not incorporate a full mass balance as is shown in Figure F.14, so **composition_eqns** is selected from the menu in Figure F.16. The alternative composition models include the recommended Fenske-Gilliland model, a split fraction model and an overall component balance.

Selection of the overall component balance requires the replacement of the old model which does not conform to the five point decomposition. This is reported to the designer in the text window as shown in Figure F.17. The subsequent selection of **model_object** in the **analyse_unit** produces the stream table also in Figure F.17.


```
The existing model is not standard - replacing with standard.
The [overall_component_balance $of column1] has been implemented.

Performing balance calculations for column1....
]
```

Solution of balance

	s4	s5	s6
benzene	0.600	0.995	0.010
toluene	0.400	0.005	0.990
Total	100	59.898	40.099

Figure F.17: Solution of Overall Component Balance

Once the component balance is completed it can be reviewed with other models. The designer is able to change the model again, this time using the recommended **fenske_gilliland_model** of Figure F.16.

Models can be evaluated visually as opposed to numerically by using the `display_model` option of the `analyse_unit` menu. Up to this point many models have been created to describe the distillation column. These are presented to the user as shown in Figure F.18.

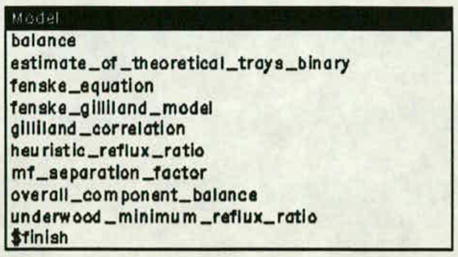


Figure F.18: Selection of Models for Display

Two types of model are represented in the menu in Figure F.18. The first choice, `balance` is the expanded version of the current model of the item. The other choices are the generic descriptions of the individual models which have been used so far, in their unexpanded forms. For example, the Fenske equation model does not include the definition of K-value which is one of its constituent parts. The current model which includes the Fenske equation does, however, incorporate the expansion of the K-value.

Examples of both types of model are presented in Appendix D. The visual presentation is achieved using LaTeX [75]. The symbolic description of the model is translated into a formatted LaTeX file. Once processed, the document is displayed using a LaTeX previewer program.

As a result of viewing the expanded version of the current model the designer might decide that the model could be modified to provide more accuracy. In particular the definition of the distribution coefficient, K, could be revised. This requires selection of `modify_current_model` from the `analyse_unit` menu.

The method presents the model to the user, separating terms which can be expanded from those which can be edited. For instance, Figure F.19 shows the terms of the Fenske equation which can be expanded to be the definition of K and the separation factor. The separation factor is also a term which can be edited. This implementation of the Fenske equation is valid for three definitions of the separation factor: one based on mole fractions, which is used in this instance, one based on molar flowrates and one on fractional recoveries.

Here, the model is further expanded by selecting expansion of the K-value

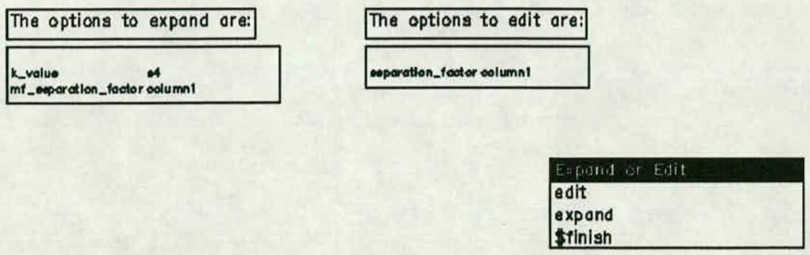

The options to expand are:
k_value e4
mf_separation_factor column1
The options to edit are:
separation_factor column1
Expand or Edit
edit
expand
$finish

Figure F.19: Options for Investigating Fenske Model Modifications

expression. Similarly to the decomposition of the Fenske equation, K can be further expanded to investigate the vapour pressure expression. However, as shown in Figure F.20 the terms which can be edited include fugacity coefficient.

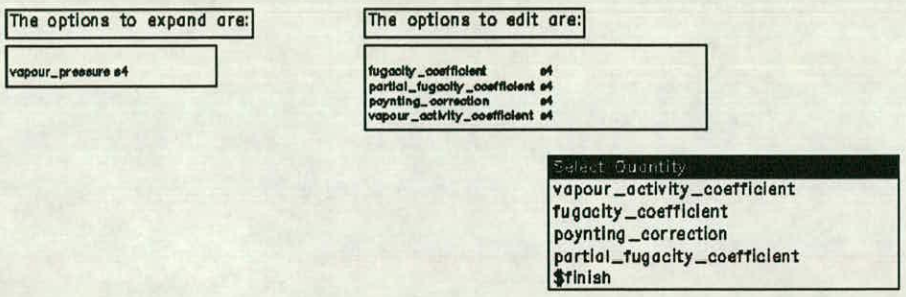


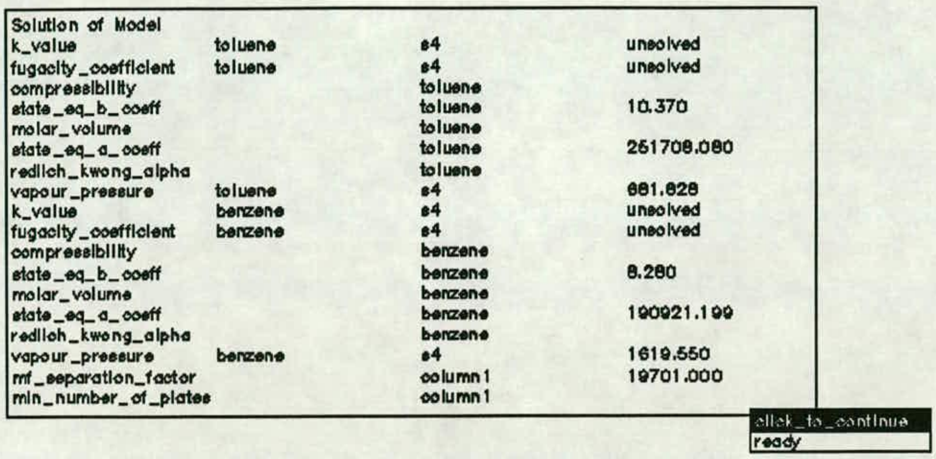
Figure F.20: Selection of Fugacity Coefficient as the Term to Modify

Choosing to edit this term instigates the inference discussed in Section 5.5. The conditions required for the inclusion of an alternative definition are presented to the user. Here, the current value of the fugacity coefficient, 1, is displayed along with the suggestion that selection of an equation of state will allow the replacement of the approximation with a full definition.

There are four equations of state available: Van der Waals, Redlich-Kwong, Soave and Peng-Robinson. Here, Redlich-Kwong was chosen. Once the model is completed, either by selection of an equation of state, or acknowledging acceptance of the original model, it is then evaluated.

Figure F.21 shows part of the solution of the new model. The table includes values which could not be solved which are marked "unsolved". The blank terms in the table correspond to the unspecified variables which are required before the model can be solved completely.

In summary, this example has demonstrated the opportunistic approach to



```
Solution of Model
k_value                 toluene          s4              unsolved
fugacity_coefficient    toluene          s4              unsolved
compressibility                          toluene
state_eq_b_coeff                         toluene         10.370
molar_volume                             toluene
state_eq_a_coeff                         toluene         251708.080
redlich_kwong_alpha                      toluene
vapour_pressure         toluene          s4              681.828
k_value                 benzene          s4              unsolved
fugacity_coefficient    benzene          s4              unsolved
compressibility                          benzene
state_eq_b_coeff                         benzene         8.280
molar_volume                             benzene
state_eq_a_coeff                         benzene         190921.199
redlich_kwong_alpha                      benzene
vapour_pressure         benzene          s4              1619.550
mf_separation_factor                     column1         19701.000
min_number_of_plates                     column1
```

Figure F.21: Solution of Modified Fenske Model

design of a unit operation. Strategies can be developed to guide the decision making procedure. However, these strategies change according to the information available at particular choice points. It has been shown that a structure of extended methods can be used to implement such strategies and make available the tools appropriate to the specifications.

Full access to modelling facilities was required throughout the example, showing the interaction required between strategies and other tools. The use of a range of modelling tools has also been demonstrated.

References

[1] Westerberg A.W., Piela P.C., Subrahmanian E. & Elm W., "A Future Computer Environment for Preliminary Design," *Conference Proceedings: Foundations of Computer-Aided Process Design - Snowmass Village, Colorado, July 1989*, Austin, Texas.

[2] Subrahmanian E., Podnar G.W., Elm W. & Westerberg A.W., "Towards a Shared Computational Support Environment for Engineering Design," *EDRC report 05-41-89*, Pittsburgh (1989).

[3] Preston M.L., "Integrated Process Plant Design," *Conference Proceedings: Foundations of Computer-Aided Process Design - Snowmass Village, Colorado, July 1989*, Austin, Texas.

[4] Beltramini L. & Motard R.L., "KNOD - A Knowledge Based Approach for Process Design," *Comp. & Chem. Eng.* 12 (1988), 939–958.

[5] Myers D.R., Davis J.F. & Herman D.J., "A Task-Oriented approach to Knowledge-Based Systems for Process Engineering Design," *Comp. & Chem. Eng.* 12 (1988), 959–971.

[6] Douglas J.M., "A Hierarchical Decision Procedure for Process Synthesis," *A.I.Ch.E. Journal* 31 (1985), 353–362.

[7] Douglas J.M., *Conceptual Design of Chemical Processes*, McGraw-Hill, New York, 1988.

[8] Talukdar S. & Westerberg A.W., "A View of Next Generation Tools for Design," *Conference Proceedings: AIChE Spring National Meeting 1988,*, Pittsburgh (1988).

[9] Lien K., Suzuki G. & Westerberg A.W., "The Role of Expert Systems Technology in Design," *EDRC report 06-13-86*, Pittsburgh (1986).

[10] Cherry D.H., Grogan J.C., Knapp G.L. & Perris F.A., "Use of Data Bases in Engineering Design," *Chem. Engng. Prog.* 78 (1982), 59.

[11] Benayoune M. & Preece P.E., "Review of Information Management in Computer-Aided Engineering," *Comp. & Chem. Eng.* 11 (1987), 1–6.

[12] Branch J., "Towards Integrated Process Design," *Processing* (March 1985).

[13] Britt H.I., Smith J.A. & Wareck J.S., "A Computer-Aided Synthesis and Analysis Environment," *Conference Proceedings: Foundations of Computer-Aided Process Design - Snowmass Village, Colorado, July 1989*, Austin, Texas.

[14] Stephanopoulos G., Johnston J., Kriticos T., Lakshmanan R., Mavrovouniotis M. & Siletti C., "DESIGN-KIT: An Object-Oriented Environment for Process Engineering," *Comp. & Chem. Eng.* 11 (1987), 655–674.

[15] Nishida N., Stephanopoulos G. & Westerberg A.W., "A Review of Process Synthesis," *A.I.Ch.E. Journal* 27 (1981), 321–351.

[16] Johns W.R. & Romero D., "The Automated Generation and Evaluation of Process Flowsheets," *Comp. & Chem. Eng.* 3 (1979), 251.

[17] Kirkwood R.L., Locke M.H. & Douglas J.M., "A Prototype Expert System for Synthesising Chemical Process Flowsheets," *Comp. & Chem. Eng.* 12 (1988), 329–343.

[18] Lott D.H., "Simulation Software as an Aid to Process Synthesis," *IChemE Symposium Series* 109, 1–22.

[19] Stephanopoulos G., "Artificial Intelligence and Symbolic Computing in Process Engineering Design," *Conference Proceedings: Foundations of Computer-Aided Process Design - Snowmass Village, Colorado, July 1989*, Austin, Texas.

[20] Siirola J.J. & Rudd D.F., "Computer-Aided Synthesis of Chemical Process Designs," *Ind. Eng. Chem. Fundamentals* 10 (1971), 353–362.

[21] Siirola J.J., Powers G.J. & Rudd D.F., "Synthesis of System Designs, III: Towards a Process Concept Generator," *AIChE J.* 17 (1971), 677.

[22] Mahalec V. & Motard R.L., "Procedures for the Initial Design of Chemical Processing Systems," *Comp. & Chem. Eng.* 1 (1977), 57.

[23] Mahalec V. & Motard R.L., "Evolutionary Search for an Optimal Limiting Process Flowsheet," *Comp. & Chem. Eng.* 1 (1977).

[24] Lu M.D. & Motard R.L., "Computer-Aided Total Flowsheet Synthesis," *Comp. & Chem. Eng.* 9 (1985), 431–445.

[25] Davis J.F., "A Task-Oriented Framework for Diagnostic and Design Expert Systems," *Conference Proceedings: Foundations of Computer-Aided Process Design* (1987).

[26] Preece P.E., *PROCEDE - The Process Engineering Design Environment*, Paper presented at CHISA '90 conference, Prague, August 26-31, 1990.

[27] Daniell J. & Director S.W., *An Object Oriented Approach to CAD Tool Control Within a Design Framework*, SRC-CMU Research Centre for Computer-Aided Design, Carnegie Mellon University, Pittsburgh, March 1989.

[28] Henning G., Leone H. & Stephanopoulos G., *MODEL.LA. A Modelling Language for Process Engineering Part I: The Formal Framework*, Laboratory for Intelligent Systems in Process Engineering, Dept. of Chem. Eng., MIT, Cambridge, MA, 1989.

[29] Henning G., Leone H. & Stephanopoulos G., *MODEL.LA. A Modelling Language for Process Engineering Part II: Multifaceted Modelling of Processing Systems*, Laboratory for Intelligent Systems in Process Engineering, Dept. of Chem. Eng., MIT, Cambridge, MA, 1989.

[30] Smithers T., Conkie A., Doheny J., Logan B., Millington K. & Ming Xi Tang, "Design as Intelligent Behaviour: An AI in Design Research Programme," *DAI Research Paper No. 426*, Edinburgh (1989).

[31] Bañares-Alcántara R., "Proposed Working Areas for the Development of a Design Environment System II," *Internal Report* (1990).

[32] Piela P.C., "ASCEND: An Object-Oriented Computer Environment for Modeling and Analysis," Carnegie-Mellon University, Ph.D. Thesis, Pittsburgh, Pennsylvania, April 1989.

[33] Lien K.M., Wahl P.E., Sorlie C. & Hertzberg T., *Integration of Expert Systems and Quantitative Computation in Chemical Engineering Design*, Paper presented at CHISA '90 Conference, Prague, August 26-31, 1990.

[34] Jackson P., *Introduction to Expert Systems*, Addison-Wesley, Wokingham, England, 1986.

[35] Rich E., *Artificial Intelligence*, McGraw-Hill, Singapore, 1983.

[36] Struthers A., "A Knowledge Based Approach to Process Engineering Design," University of Edinburgh, Ph.D. Thesis, Scotland, 1990.

[37] Bañares-Alcántara R. & Westerberg A.W., "Development of an Expert System for Physical Property Predictions," *Comp. & Chem. Eng.* 9 (1985), 127.

[38] Sriram D., Bañares-Alcántara R., Venkatasubramanian V., Westerberg A.W. & Rychener M., "Knowledge-Based Expert Systems: An Emerging Technology for CAD in Chemical Engineering," *EDRC report 06-22-86*, Pittsburgh (1986).

[39] Bañares-Alcántara R., Westerberg A.W., Ko E.I. & Rychener M.D., "Decade - A Hybrid Expert System for Catalyst Selection - 1. Expert System Considerations," *Comp. & Chem. Eng.* 11 (1987), 265.

[40] Lien K.M., "A Framework for Opportunistic Problem Solving," *Comp. & Chem. Eng.* 13 (1989), 331–342.

[41] Levesque H.J. & Brachman R.J., "A Fundamental Tradeoff in Knowledge Representation and Reasoning," in *Readings in Knowledge Representation*, Levesque H.J. & Brachman R.J., eds., Morgan Kaufmann, Los Altos, Calif., 1985.

[42] Clocksin W.F. & Mellish C.S., *Programming in Prolog*, Springer-Verlag, Berlin, 1981.

[43] Kunz J.C., "Model Based Reasoning in CIM," *Intelligent Manufacturing: Expert Systems and the Leading Edge in Production Planning and Control* (1988).

[44] Carnegie Group, Inc., *Knowledge Craft Reference Manual*, Pittsburgh, PA, 1988.

[45] Struthers A., "CLAP Reference Manual," *Internal Report* (1988).

[46] McCabe W.L. & Smith J.C., *Unit Operations of Chemical Engineering, 3rd Ed.*, McGraw-Hill, 1976.

[47] Westerberg A.W., Hutchison H.P., Motard R.L. & Winter P., *Process Flowsheeting*, Cambridge University Press, Cambridge, 1979.

[48] Flower J.R. & Whitehead B.D., "Computer-Aided Design: A Survey of Flowsheeting Programs - Part I," *Chem. Engng.* 272 (1973), 208.

[49] Flower J.R. & Whitehead B.D., "Computer-Aided Design: A Survey of Flowsheeting Programs - Part II," *Chem. Engng.* 273 (1973), 271.

[50] Lee W., Christensen J.H. & Rudd D.F., "Design Selection to Simplify Process Calculations," *AIChE J.* 12 (1966), 1104.

[51] Berger F. & Perris F.A., "FLOWPACK II - A New Generation of System for Steady State Process Flowsheeting," *Comp. & Chem. Eng.* 3 (1979), 309.

[52] Brannock N.F., Verneuil V.S. & Wang Y.L., "PROCESS Simulation Program - A Comprehensive Flowsheeting Tool for Chemical Engineers," *Comp. & Chem. Eng.* 3 (1979), 329.

[53] Rosen E.M. & Pauls A.C., "Computer Aided Chemical Process Design: The FLOWTRAN System," *Comp. & Chem. Eng.* 1 (1977).

[54] Evans L.B., Boston J.F., Britt H.I., Gallier P.W., Gupta P.K., Joseph B., Mahalec V., Ng E., Seider W.D. & Yagi H., "ASPEN: An Advanced System for Process Engineering," *Comp. & Chem. Eng.* 3 (1979), 319.

[55] Rosen E.M., "A Machine Computation Method for Performing Material Balances," *Chem. Engng Progr.* 58 (1962), 69.

[56] Sargent R.W.H. & Westerberg A.W., "SPEED-UP in Chemical Engineering Design," *Trans. Inst. Chem. Engng.* 42 (1964), 190.

[57] Locke M.H. & Westerberg A.W., "The ASCEND II System - A Flowsheeting Application of Successive Quadratic Programming Methodology," *Comp. & Chem. Eng.* 7 (1983), 615.

[58] Hutchison H.P., Jackson D.J. & Morton W., "The Development of an Equation-Oriented Flowsheet Simulation and Optimization Package-I. The Quasilin Program," *Comp. & Chem. Eng.* 10 (1986), 19–29.

[59] Gorczynski E.W., Hutchison H.P. & Wajih A.R.M., "Development of a Modularly Organised Equation-Oriented Process Simulator," *Comp & Chem Eng.* 3 (1979), 353–356.

[60] Shacham M., Macchietto S., Stutzman L.F. & Babcock P., "Equation Oriented Approach to Process Flowsheeting," *Comp & Chem Eng.* 6 (1982), 79–93, Review paper.

[61] Perkins J.D., "Efficient Solution of Design Problems Using a Sequential-Modular Flowsheeting Programme," *Comp. & Chem. Eng.* 3 (1979), 375.

[62] Mahalec V., Kluzik H. & Evans L.B., "Simultaneous Modular Algorithm for Steady-State Flowsheet Simulation and Design," *Comp. & Chem. Eng.* 3 (1979), 373.

[63] Johns W.R. & Vadhwana V., "A Dual Level Flowsheeting System," *PSE '85: The Use of Computers in Chemical Engineering* (1985).

[64] Johns W.R. & Vadhwana V., "Convergence Studies in Dual-Level Flowsheeting," *Chem. Eng. Res. Des.* 64 (1986), 332.

[65] Department of Chemical Engineering, University of Edinburgh, *Esspros User Manual*, 1985-1989.

[66] Norman R.L., "A Matrix Method for Location of Cycles of a Directed Graph," *A.I.Ch.E.J.* 11 (May 1965), 450–452.

[67] Rudd D.F. & Watson C.C., *Strategy of Process Engineering*, John Wiley & Sons, 1968.

[68] Stadtherr M.A. & Wood E.S., "Sparse Matrix Methods for Equation-Based Chemical Process Flowsheeting-I: Reordering Phase," *Comp. & Chem. Eng.* 8 (1984), 9–18.

[69] Stadtherr M.A. & Wood E.S., "Sparse Matrix Methods for Equation-Based Chemical Process Flowsheeting - II - Numerical Phase," *Comp & Chem Eng.* 8 (1984), 19–33.

[70] National Engineering Laboratory, *Physical Property Data Service: Programmers Manual*, East Kilbride.

[71] Kunz J.C., "Concurrent Engineering in Building Knowledge Systems," *Model Based Reasoning in Engineering Summer School*, Edinburgh (1989).

[72] Ponton J.W., Hutton D., Jones G. & Skilling N., "A Demonstration "Intelligent" Physical Properties Data System," *Proceedings of Chempor 89*, Lisbon (1989).

[73] Bañares-Alcántara R., "Proposed Working Areas for the Development of a Design Environment System," *Internal Report* (1990).

[74] Westerberg A.W. & Benjamin D.R., "Thoughts on a Future Equation-Oriented Flowsheeting System," *Comp & Chem Eng.* 9 (1985), 517–526.

[75] Lamport L., LaTeX- *A Document Preparation System*, Addison-Wesley, Reading, MA, 1985.

[76] Smith J.M. & Van Ness H.C., *Introduction to Chemical Engineering Thermodynamics, 3rd Edition*, McGraw-Hill, Tokyo, 1975.

[77] Reid R.C., Prausnitz J.M. & Poling B.E., *The Properties of Gases and Liquids, 4th Edition*, McGraw-Hill, New York, 1987.