2000

The Application Of Object-oriented Techniques To Preliminary Design Problems

Patrick S. Mackessy
Edith Cowan University

Edith Cowan University

Copyright Warning

USE OF THESIS

The Use of Thesis statement is not included in this version of the thesis.

The Application of Object-Oriented Techniques to Preliminary Design Problems

by

Patrick S Mackessy B.Sc., Grad Dip (Comp)

A Thesis Submitted in Partial Fulfillment of the Requirements for the Award of
Master of Science (Computer Science)

At the

Faculty of Communications, Health & Science,
Edith Cowan University, Mount Lawley.

Date of submission: 31st October 2000

I certify that this thesis does not incorporate without acknowledgment any material previously submitted for a degree or diploma in any institution of higher education; and that to the best of my knowledge and belief it does not contain any material previously written by another person except where due reference is made in the text.

Signature

Date May 31st, 2006

ACKNOWLEDGMENTS

I wish to thank my supervisor, Mr. Maurice Danaher for his valuable advice. I am grateful to Mr. Danaher for his unstinting time and enduring patience throughout the past three years. I would also like to thank Ms. Ursula Ladzinski, Ms. Marea Carfax-Foster, Ms. Kathleen Henderson and Mr. Lewis Corner for their proofing and comments and more than anything else for their unstinting encouragement. I am also grateful to Mr. Mark Tait of Intellicorp, who lent me his KEE Manual, which also proved indispensable.

I also wish to acknowledge the help and support provided by the Office of the Auditor General. In particular Mr. Pearson, the Auditor General and Mr. Tuffley, the IS Audit Manager have created an environment, which has inspired the pursuit of excellence in the study of the use and control of IT. The Office also provided an IBM ThinkPad, which proved indispensable.

ABSTRACT

The Application of Object-Oriented Techniques

To Preliminary Design Problems

Preliminary structural design is an early stage in building design during which the engineer formulates and assesses a number of different structural schemes. It is conceptual in nature and involves decision making, which relies on heuristics. Whilst preliminary structural design has not been well supported by PC software, recent research has indicated the potential for knowledge-based, object-oriented systems to assist in the area.

This thesis explores the issues that arise when object-oriented techniques are used to develop knowledge-based software. It reviews certain basic principles of structural design, methods of representing structural design knowledge and earlier approaches to the design of software to support preliminary structural design.

The thesis describes how the writer created a software development methodology to apply object-oriented analysis and design techniques. It then describes the use of this methodology to develop a system for preliminary structural design, including the drafting of requirements, the creation of an object model for these requirements and their implementation in Kappa-PC software.

The thesis proposes an approach to the development of software to support preliminary design in buildings and has demonstrated this approach in a prototype design tool. It has also described some of the difficulties hindering the effective application of the object-oriented methods.

LIST OF CONTENTS

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1. Introduction

1.1 Aim of the Study

This study concerns methods for the application of object-oriented computing techniques to the production of computer systems that can assist with preliminary structural design. The research proposal for this study was drafted in May 1997. The primary objectives included in the proposal were to:

- Adopt and analyse a particular approach to the application of computers in the support of preliminary structural design; and
- Determine whether or not it was practical to implement this approach in software using a PC-based object-oriented knowledge engineering environment to develop a knowledge based design system.

1.2 Problems Addressed

The first problem addressed was to find a suitable approach to the problem of providing support for structural design. After some preliminary reading the writer decided to adopt an approach to the problem, which was first reported by Maher (1984). During the early phases of the study this approach was analysed in depth.

The approach chosen relies upon a formalised model of the design process, which several researchers, including Krishnamoorthy (1996) have described as the decomposition-based model, and which provides computer support by way of an expert system. Maher has described this approach in several papers and used it to produce an expert system, known as HI-RISE, which was designed to assist with the preliminary design of tall buildings. Other researchers have also adopted this approach, including Harty (1987), who also demonstrated

its implementation in an expert system known as DOLMEN, which was designed to extend the range and functionality of HI-RISE. Sause et al. (1992) have extended this decomposition-based approach and proposed the '*multilevel selection-development*' (MSD) model, which is a process generalisation model for structural design.

The writer considered several other approaches including those employing case based reasoning, described by Lim et al (1996), transformation, described by Fenves & Baker (1987), and neural networks, described by Liu & Gan (1990). These approaches were rejected because of the initial level of domain knowledge required, the difficulties involved in obtaining suitable software and the complexity of programming required. The writer chose to adopt the approach described by Maher and Harty because:

- This approach was based on a formalised model of the design process, which other researchers have taken up and incorporated in prototype systems;
- It incorporated basic structural engineering concepts described in the standard textbook, by Lin & Stotesbury (1981);
- The approach was well documented by Maher and Harty; and
- The approach appears well suited for implementation in an object-oriented knowledge based system.

The writer reasoned that it would be feasible to implement this approach in a knowledge-based system by creating a series of prototypes, building on the experience documented by the developers of the HI-RISE and DOLMEN systems. These prototypes would be refined gradually, as the writer became more familiar with the domain knowledge of structural engineering. This reasoning was borne out during the study and the project resulted in the

partial completion of a prototype expert system, which incorporated most of the functionality required for the design of simple, rectangular buildings.

The second problem was to determine a suitable PC-based object-oriented knowledge engineering environment on which to implement the approach chosen. The writer chose the Kappa-PC development application because of its availability, low cost and its ability to run under Windows 95. In addition, Hasan et al (1994), Kiernan et al (1996) and Tsang and Bloor (1994) had indicated that Kappa-PC had been used to produce expert systems quickly and economically. As the writer was intending to work with Kappa-PC, which was an object-oriented development system, it also became necessary for the writer to develop an object-oriented software methodology. The methodology developed for the study is described in chapter 5.

1.3 Significance

Preliminary structural design is an early stage in the design process during which a number of different structural schemes are formulated and assessed. Harty (1987) pointed out that this task involved decision-making, which relied on heuristics and that it was not well defined. Furthermore, commercial programmers had not written software for it.

As already noted in section 1.1, the primary purpose of this study was to adopt a suitable approach to the provision of computing support and to identify the difficulties involved in using this approach to implement a prototype system on a microcomputer using an object-oriented, knowledge engineering toolkit.

The study also set out to assess what other research work was being done in this field and to determine what kinds of approaches had been proposed for representing design knowledge and activities.

Since 1984 several researchers have published research on expert systems intended to support various aspects of preliminary structural design. Maher (1984) is one of the earliest sources in this area, she demonstrated a mainframe system to support structural design, which she referred to as HI-RISE, this was probably the first expert system for the preliminary structural design of high rise buildings.

Harty (1987) extended the approach described by Maher and implemented it in the DOLMEN system, an expert system for use with multi storey buildings. She reported that, although the approach adopted in DOLMEN was similar to that of HI-RISE, DOLMEN was significantly more advanced, using the knowledge of experts and incorporating methods for design evaluation. She contended that systems could be written to support the designer in the exploration of alternatives by performing routine tasks, making intelligent suggestions, and producing, evaluating and ranking designs but leaving overall control with the designer.

DOLMEN was created on the KEE application development environment on a Sperry Explorer minicomputer. In Artificial Intelligence terms KEE is referred to as a hybrid development environment, because it allows the system developer to combine various knowledge representation schemes including, frame-based representation and rule-based reasoning. Kee also provides LISP, which can be used to create functions interactive graphics, active values, which are also referred to as monitors and rule-based reasoning. The Kee system is also an object-oriented programming environment, which provides objects, methods, message passing, encapsulation and inheritance. It is an expensive, specialised development application, which allows the user to combine AI methodologies, functional programming and interactive graphical user interfaces.

In her report, Maher (1984) concluded that the HI-RISE system illustrated that computer aids could be developed for preliminary design, but much more work needed to be done

before such aids could be practical. Harty's subsequent work demonstrated that a more user-friendly prototype system could be developed on a minicomputer. Furthermore, in her report she pointed out that "the ultimate objective was to be able to use systems like DOLMEN on a computer to which most engineers have access". This objective was achieved to a certain extent by Gavin (1988), who used LISP to write a PC program to replicate the major features of the KEE system, known as SPIKEE, a Simple Prototype Implementation of KEE. Gavin then demonstrated that the DOLMEN system could run on SPIKEE and produce the same results as it had on the Explorer minicomputer.

Gavin's suggested that systems like DOLMEN could be:

- Interfaced with other PC software, so that CAD drawings of the design could be produced and design data could be transferred to other PC analysis software including spreadsheets and databases; and
- Incorporated into integrated design software packages.

Since DOLMEN was first developed in 1987 and subsequently implemented on SPIKEE, more user-friendly and efficient PC tools have become available. In particular, flexible integrated development toolkits are now available, which allow programmers to design and modify expert systems. These tools also allow the creation of portable software, which can be integrated, with a wide range of business software including spreadsheets and databases. For example, expert systems can now be embedded in production applications, can be integrated with CAD packages and can read and write data to and from spreadsheets and databases.

In order to produce a knowledge-based system, which incorporated a decomposition based model of the design process; the writer initially reviewed the reports of both Maher and

Harty and prepared an analysis of their approaches. The writer then commenced to model an approach using this analysis in a prototype design system. The writer then installed the Kappa-PC software on a PC and learned how to use it.

The writer than focused on designing a prototype, which operated like the DOLMEN system and had most of the functionality required to provide assistance during the early stages of structural design. The new system was expected to allow the user to generate alternative solutions to a structural design problem and then to rank them against each other using a series of evaluation parameters. The system would then assist in the selection of an optimal structural design from a ranked list of possible schemes.

The structural design domain knowledge to be incorporated in the prototype was located in Lin, (1981), *Structural Concepts and Systems for Architects and Engineers* and from reports on the HI-RISE system by Maher (1984) and the DOLMEN system by Harty (1987). Further knowledge was indicated by Harty, which was published in British Standards and in the Handbook for Steel Construction.

During design of the prototype it was intended to exploit the features of the Microsoft Windows operating system including Dynamic Data Exchange, (DDE) and to allow the system to communicate with other DDE enabled applications such as Microsoft Excel. This would confirm the potential for future work in the Windows environment, which would include the development of an interface to a PC based CAD application and the embedding of the system in other software products available to the structural designer.

This study is significant because it seeks to bring the power of modern desktop computing to design problems that had hitherto been limited to computing environments that were much less accessible and flexible. The use of object oriented technology and the prospect

of linking to other GUI-based software could see considerable interest in a prototype expert system for an economically important area of structural design, which currently is not well supported.

Evidence gathered during the research phase of this study supports assertions made by made by Harty & Danaher (1994) that:

- Preliminary design is rarely described in books on structural design and it is not a well-defined task. It involves decision making based on heuristics, which is a difficult area for conventional programmers; and
- Commercial structural engineering software does not cater for preliminary structural design tasks.

This thesis also presents a new software methodology, which facilitates the object-oriented development of IT-based tools, which support the decision-making activities needed for successful building engineering design. Furthermore, these tools use open-architecture software, which allows them to be integrated with other design software.

Furthermore, if designers in consultant engineering firms are to obtain increased benefits from already existing computer systems, then these systems must be able to integrate both support and design features. If the time consuming manual transfer of information between design applications can be avoided by facilitating integration of software products, then resources will be available for the more challenging issues of design. In addition to the effects of integration, introducing software, which supports design decision-making, as opposed to existing software, which supports the production of design documentation, will enable engineers to work more efficiently, and to obtain more effective use of the available computer hardware.

1.4 Structure of the Thesis

On commencing the project, the writer completed a survey of literature covering design concepts, preliminary structural design, object-oriented analysis and design and the use knowledge based systems to support engineering design.

While reviewing object-oriented analysis and design issues, the writer focused his attention on the selection and application of object-oriented analysis and design methods. This research is described in chapter 5 and it assisted the writer with the creation of a software design methodology, which combined various object-oriented techniques. The methodology commenced with a high-level analysis stage, followed with a requirements definition stage and then completed overlapping object-oriented analysis and design stages.

Chapter 2 provides general definitions and covers the state space search model, of Newell and Simon (1972), and Simon's ideas on the process of design, artificial intelligence and the relationship between problem solving and search techniques.

Whilst chapter 2 focuses on design in the abstract, chapter 3 introduces the practical considerations required to address the field of structural design. It defines structural design, quoting from Ambrose (1967) and includes a description of how building designs are created in practice, which is taken from Merritt's (1985), *Building Design and Construction Handbook.* The chapter also introduces basic principles of structural design as outlined by Lin and Stotesbury in their book *Structural Concepts and Systems for Architects and Engineers* (1981).

Chapter 4 describes the results of research into the use of computer systems to support preliminary structural design and provides an understanding of the issues involved in the application of object-oriented concepts and techniques in the development of knowledge-

based design systems. The chapter summarizes several common features in the various approaches reviewed.

The chapter also traces the evolution of intelligent systems; in particular the study revealed that there are two predominant ways to represent knowledge in computer systems. One of these is rule-based representation and the other method is frame-based, which uses a network of frames or nodes connected by relations and organized into a hierarchical structure. Each node represents a concept that may be described by attributes associated with the node. The topmost nodes represent the general concepts and the lower nodes represent more specific instances of these concepts. The chapter covers Hayes-Roth's (1985) work on production rule systems, and Fikes & Kehler's (1985) work on the frame based representation of domain knowledge. It also includes Kunz, Kehler & Williams's (1984), work on hybrid systems.

Chapter 5 describes research completed, which allowed the writer to draw together a set of development methods, which included object-oriented analysis and design techniques to create a structured software engineering methodology. In particular the writer adopted a simplified six step object-oriented analysis and design process, which was described in Cross (1996) and which used modeling techniques adapted from Rumbaugh et al. (1991) and Embley et al. (1992). This methodology, which consisted of four different stages: high-level analysis, requirements development, object-oriented analysis and design, allowed the writer to exploit appropriate object-oriented techniques to develop a knowledge-based design system. The writer also describes lessons learned during the creation of the methodology.

Chapter 6 describes completion of the first two stages of this methodology in the development of a knowledge-based design tool. The first stage produced a high-level

analysis of the problem situation and an accompanying conceptual model for a new system. Chapter 6 also describes the second stage, which resulted in the preparation of the requirements specification for the new system, which was proposed as a knowledge based design tool, which was similar to the DOLMEN system described by Harty (1987).

Chapter 7 describes the completion of the final two stages object-oriented analysis and design. The chapter describes the use of six-step analysis process, which was introduced in chapter 5. This process was used as a framework, to guide the analysis and to ensure that the problem was fully understood and that the required diagrams were created.

Chapter 7 also describes the object model, which facilitated the design of the new system. The chapter concludes with a discussion of the difficulties encountered during completion of the development process, which include: difficulties in analysing a conceptual design process, difficulties in applying object-oriented techniques to a knowledge based application and problems caused by the overlapping of the analysis and design phases.

Chapter 8 describes the Kappa-PC application development system including the structures it provides to describe objects, the KAL high level language, the development environment, the debugger and Kappa-PC's reasoning mechanisms.

Chapter 9 describes how the key features of the design model of the proposed new system were converted into a working prototype program. It also describes how the writer used the Kappa-PC development tools for entering and editing code and creating the graphic user interface. A simplified overview of the system is provided and the chapter ends with a discussion of some of the difficulties encountered during implementation.

Chapter 10 covers the operation of the design tool prototype. The chapter describes the key design activities simulated in the prototype system. These include: input of the design

specifications of the building, input of system evaluation features, design of the vertical subsystem, initial sizing of components, use of the steel sections database, detailing of vertical subsystem, design of the horizontal subsystem, detailing of horizontal subsystem, evaluation of design alternatives proposed and the selection of the final design. Various system reports are also described.

Chapter 11 concludes the thesis. It discusses the lessons learned during the study and describes the difficulties encountered during the development of the design tool prototype. It concludes with recommendations for future work.

CHAPTER 2. Design

2.1 Design

This chapter introduces some of the basic ideas, which have contributed to the development of design as a science. The writer used these ideas as a framework to assist with the development of a model of the structural design process. This model was used to develop requirements for the knowledge-based design tool developed during the study.

Sriram et al (1989) say design can be viewed as the process of specifying a description of an artifact that satisfies constraints arising from a number of sources, by using diverse sources of knowledge.

The writer took the following comprehensive definition of the word "design" from the Encyclopaedia Britannica; it expands on Sriram's definition:

Design:

> Plan or scheme as the pattern for making a product, indicates primarily an inter relation of parts intended to produce a coherent and effective whole, ordinarily planned with four limiting factors in mind; the capacities of the materials employed, the influence of the methods adapting the material to their work, the impingement of parts within the whole, and the effect of the whole on those who may see it, use it or become involved in it.
>
> Encyclopaedia Britannica (1988)

The key requirements needed for an effective knowledge-based product design system can be drawn from this definition. They include the ability to represent the product, the parts that make up the product and a plan or scheme to put them together into a coherent and

effective whole. In addition the design system must be able to synthesise alternative, feasible plans and be able to test these plans against the limiting factors.

2.2 Basic Ideas on Design

Several researchers in design including Gero, Maher & Zhang (1988), Dasgupta (1992), Sause et al (1992), and Quinn (1993) have traced the roots of knowledge-based design research to Simon's "*The Sciences of the Artificial*". Simon (1969) described design in terms of a process of search through large combinatorial spaces of partial design alternatives and he asserted that the theory of design ***is*** a general theory of search. Simon addressed several approaches to the study of design, which he referred to as "devising artifacts to attain goals". His ideas covered:

- Problem Solving, Artificial Intelligence, the Search Process and Heuristics;
- Complex Systems, Hierarchical Organisation and Decomposition, and
- State Description and Process Description.

These ideas are discussed in the following sections.

2.3 Problem Solving, Artificial Intelligence and the Search Process

Simon (1969, p 123) considered design to be a form of problem solving, which he likened to a form of search for appropriate solutions from a population of possible alternatives. The solution to the problem or final design is a complete operation, which is built-up from a sequence of component operations. This sequence of operations or solution path starts at the initial problem state and consists of all the states that lead from the initial state to the goal state. It is possible that the space of alternative states could grow to an enormous size, because there are innumerable ways in which the component operations could be combined

into sequences. Problem-solving systems, which carry out design procedures, do not merely assemble possible problem solutions from the components available; they must also search for appropriate assemblies.

Gardner (1989) expands on Simon's ideas adding that problem-solving systems usually have three main components:

- A database, which describes the task/domain situation and the goal, which the problem solving system is trying to achieve. This goal is achieved by applying an appropriate sequence of operators to the initial task/domain situation;
- A set of operators, which are used to manipulate the database. She says these could be rules that can be used to generate new assertions from the database or specialised operators, which can create new assertions from the existing ones; and
- A control strategy for deciding what to do next. In particular, which operator to apply and where to apply it.

The term *search* describes the process that the system applies to discover the appropriate sequence of operators. In general, search techniques are used to find a sequence of operator actions that will move the system from the given initial state to the desired goal states. This view is endorsed by Mittal and Araya (1986), who say that many design problems can be formulated as a process of searching a 'well defined' space of alternative artifacts with similar functionality.

The writer's literature survey identified two basic approaches to search in problem solving; these are *state space search* and *problem reduction.* In the state space search approach the search system reasons forward applying the operators to the structures in the database, which describe the task/domain situation. The objective of applying the operators is to

produce a modified situation, which will eventually become the goal situation. These modified situations can be represented as nodes in a search tree. Simon (1969) has described this in terms of generating trees of partial solution possibilities. Successive application of the operators to the modified situation will result in further modified situations, which can be represented as successor nodes in the search tree. A search system, which reasons backwards and; for which each application of an operator to a problem yields exactly one new problem, whose size or difficulty is less than that of the previous problem, may also be said to be using the state space search representation.

In the problem-reduction approach the search system reasons backwards applying an operator not to the current task/domain situation but to the goal. The goal or problem statement is converted to one or more subgoals, whose solutions are sufficient to solve the original problem. These subgoals may in turn be reduced to subgoals, and so on, until each of them is reduced to a trivial problem. In the problem-reduction approach a node in the search tree is a goal (or set of goals) to be satisfied. Successor nodes will be the different subgoals that can be used to satisfy that goal.

Alison (1994) states that most problems could, in principle, be formulated in either state space search or problem-reduction terms. However, usually one way of formulating the problem will be more natural and more efficient. The appropriate technique to use in a particular case will depend on the nature of the solution to the problem and on the most natural way to go about solving it. In general, state space search is best applied when the solution to a problem is naturally expressed in terms of a final state, or a path from an initial state. Problem-reduction may be better if it is easy to decompose a problem into independent subproblems. In either case, state space search or problem-reduction

representation, a solution is obtained by finding appropriate finite sequences of applications of available operators.

Where a tree structure is used to represent the progress of a state space search, the nodes of the tree represent the set of problem states produced by successive operator applications. The root node of the tree represents the initial problem situation or state and each of the new states are represented by the successor nodes, which emanate from the root. A new state can be produced from an initial state by the application of just one operator. Subsequent operator applications produce successors of these nodes, and so on.

Gardner (1989) says that instead of a tree structure, a graph may be used to better represent the search space. In a graphical representation a directed arc represents each application of an operator. The advantage of the graphical representation is that it accommodates more than one path from the root to a node. Gardner then restates the problem of finding a goal situation in terms of searching a graph to find a node whose associated state description satisfied the goal. She also distinguishes the graph to be searched from the tree or graph that is constructed as the search proceeds. She describes the graph, which is created as the search proceeds as the search graph or tree. This is an explicit graph of nodes and arcs, which grows as the search proceeds. She says the search graph is an implicit graph, which represents the state space or search space. This graph may be thought of as having one node for every state that can be described whether or not there is a path to the node from the root. Many problem domains may have an infinite or very large search space. A search, which examined the effects of all possible sequences of n operator actions, may experience combinatorial explosion of the resulting search tree, because the number of effects to examine expands exponentially with n. If the search space is a general graph, then the search graph may be a tree or subgraph containing a path to a search space node, which is

replicated as a search graph node. This concept is illustrated in Figure 2.1, which shows part of a search space and the corresponding search tree, or solution path.

Gardner explains that searching becomes a problem of making just enough of the search space explicit in a search graph to contain a solution of the original goal. This reflects Simon's point that in general AI search-based problem solvers will tend to produce satisfactory solutions but not optimal ones. They will find *a* solution but, because resources are limited, they may not be able to search long enough or far enough to reach *the* optimal solution.

Gardner suggests several strategies for limiting the search process by reducing the search space or the number of nodes to be examined. These strategies include recasting the problem so that the size of the search space is reduced; finding a better way to represent the problem; and the use of heuristic knowledge from the problem domain to guide the search.

Gardner points out that the term *heuristic search* is imprecise and she cites several different definitions, including: Polya (1957, p. 112), Newell, Shaw and Simon, (1963) and Nilsson (1971). According to Nilsson "...a blind search corresponds approximately to the systematic generation and testing of search space elements. Heuristic search can be achieved if additional information from the specific problem domain can be introduced so as to drastically restrict the search space".

In the 1975 Turing Lecture, Simon and Newell (1976, p. 113) introduced the idea of the *symbol structure* to their discussion of the role of search in intelligence. Their 'Heuristic Search Hypothesis' was that intelligent problem solving involves search, which involves generating and progressively modifying symbol structures until a solution structure is produced. Their terminology differs slightly from Gardner's, which was described earlier in

this section. They asserted that a problem has a test for a class of symbol structures (solutions of the problem) and a generator of potential solutions. To solve a problem is to generate a structure that satisfies the test. Thus there is a problem if it is known what is to be done (the test) and if it is not known immediately how to do it, (the generator). Likewise, a system can state and solve problems because it can generate and test. For there to be a generator of solutions for a given problem there must be a problem space and a space of structures in which problem situations, including the initial and goal situations can be represented. Solution generators are processes for changing one situation in the problem space into another.

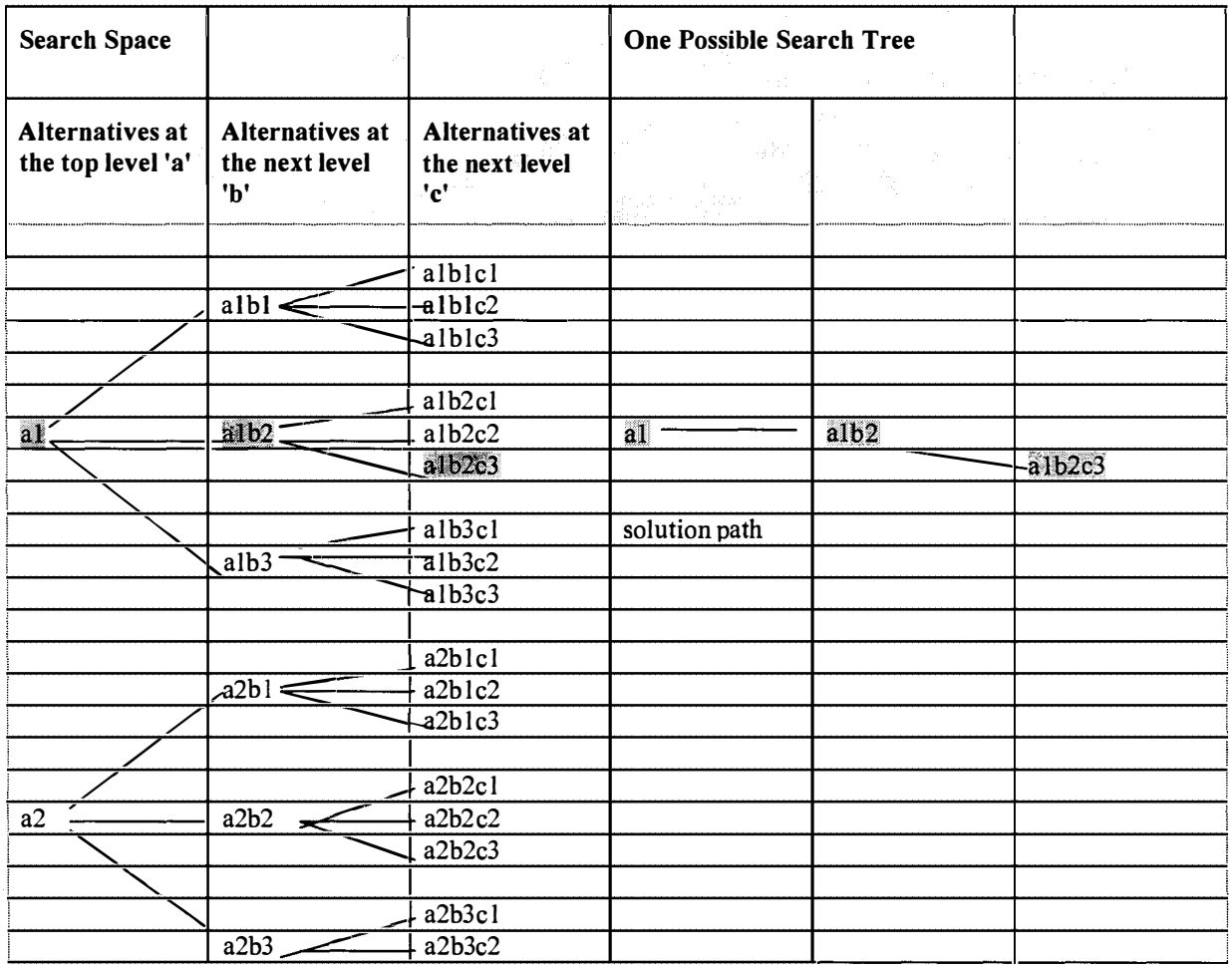


Figure 2.1 Search Trees

Figure 2.1 shows a tree representation of a search. In this case the product can be decomposed into a hierarchy of three levels: a, b, c. There are three possible design options available at each level, these are: at level a: a1,a2,a3, at level b: b1,b2,b3, and at level c: c1,c2,c3. The final product will contain one component from each level.

If all possible combinations at each level were allowed, the search space would appear like the tree on the left of the diagram. The number of possible alternatives at level 'c' could be calculated by applying the multiply rule for combinations. Thus:

Number of alternatives at 'c' = number at 'a' x number at 'b' x number at 'c'

$$= 3 \times 3 \times 3 = 27$$

If some form of heuristics were applied to eliminate infeasible options and if in the example the only acceptable option was 'a1b2c3', then the corresponding search tree would appear like the tree on the right hand side of the diagram.

The test for a successful problem solving system is that when faced with a problem and a problem space it can use limited processing resources to generate possible solutions, one after another, until it finds one that satisfies the problem defining test. The system's generator produces successive solutions, each obtained by modifying the previous one. The modifications in each case are aimed at reducing the difference between the form of the input structure and the form of the final (test) solution, while maintaining the other conditions for a solution.

Simon and Newell (1976) say that in solving combinatorial type problems, tree search can seldom be avoided, and success depends on heuristic search methods. In a heuristic tree search the questions are:

- From what node in the tree should the search resume? and

- What direction should the search take from that node?

Different strategies exist to gather information to assist in answering these questions and in most cases, the information used, may be drawn from the problem domain. For example, the best-first search strategy calls for searching next from the node that appears closest to the goal.

2.4 Complex Systems, Hierarchical Organisation and Decomposition

Simon (1969) noted that complex structures are hierarchical in nature and therefore they may be analysed or decomposed into their constituent components, which correspond to their functional parts. This reflects the fact that the components in any complex system perform particular subfunctions, which in turn contribute to the overall function.

In his book *'Computer Aided Architectural Design'*, Mitchell (1997, p27) explains how the search process might be effected during design; he proposed the 'generative system' as a means for producing potential solutions or paths to the final goal state. He traces the idea of the generative system back to Aristotle (Politics, Section 1290). Aristotle's idea was that an object could be decomposed into a number of components. A list of the components could be maintained and a new object could be conceived of as a combination of the individual components from the list. Furthermore, as many different new objects could be assembled as there are possible combinations of the components. A generative system then, is a system, which decomposes objects into components and then creates a diverse range of new potential objects by recombining these individual components.

Mitchell contended that historically generative systems have played an important role in the development of engineering and architectural design methodology. He also addressed the issue of the combinatorial explosion of potential solutions, which could be created if a

generative system were to generate all the possible combinations of the component parts. He then attributed to Leibnitz, the discovery that the exhaustive generation of possibilities can be practical and useful if the range of alternatives is carefully limited and well defined. This limiting of alternatives opened up the way to the widespread application of generative systems in engineering design.

In theory, a complex structure could be designed by decomposing it into semi-independent components, which correspond to its functional parts. The design of each component could then be carried out independently. This could be done because each component affects the others through its functions and not through the details of its mechanisms for achieving those functions.

Simon also pointed out that there might be more than one way to decompose a complex structure and successive decompositions may produce different alternative collections of subcomponents. Various alternative designs of the top-level object could then be synthesised by recombining the individual subcomponents. Combining alternative options at each level in the hierarchy facilitates the decomposition and subsequent recomposition.

Simon suggested the design process could be viewed as a repetitive cyclic process with two stages; one stage, which generated alternatives and a subsequent stage, which involved testing these alternatives against an array of requirements and constraints. The 'generate' steps may be nested into a whole series of cycles. He says the generators implicitly define the decomposition of the design process, as alternative decompositions correspond to different ways of dividing the responsibilities for the final design between generators and tests. The designer must organise the design process and decide how far the development of possible subsystems will be carried out before the overall coordinating design is developed

in detail. Conversely the designer must also decide how far the overall design might proceed before the components are designed.

Simon's idea of design as a hierarchical decision-making process, dependent on heuristic as well as technical knowledge maps directly to his state space search model of the human problem-solving process, which was introduced in section 2.3.

2.5 State Description and Process Description

Simon (1969, p 211) provided two different descriptions of a given product. A state description, addresses the actual product object; and a process description describes how to create it. Pictures, blueprints and diagrams are state descriptions and recipes, equations and assembly instructions are process descriptions. Process descriptions facilitate the means for generating products.

Simon creates a link between his concept of design as a problem solving exercise and of search and his ideas of state and process descriptions, by saying a problem can be proposed by describing the state description of the solution.

> "the task is to discover a sequence of processes that will produce the goal state from the initial state. Translation from this process description to the state description enables us to recognise when we have succeeded."

He expands on this, saying problem solving requires a continual translation between the state and process descriptions of the same complex structure.

2.6 Design as a Constraint Driven Activity

Harty (1984, p. 14) says design objects are devised with the basic objective of attaining a specific set of goals. This introduces the idea of limiting factors or constraints. She points

out that for the product to be effective, it must attain the specific set of goals for which it was designed. She cites Eastman (1981) who says that design is "the specification of an artifact that achieves both desired performances and is realisable with high degrees of confidence".

Goals, desired performances and realisability are grouped together as constraints. A constraint is a limitation or requirement that restricts or constrains something. Some design goals are determined by the required functionality of the product. However, the product must also be practical and this creates additional design goals. These design goals act as constraints on the solution and that design can be regarded as a constraint driven activity.

Harty (1984, p. 15) cites Mostow (1985) who outlines five categories of constraints, which govern most designs regardless of context. These categories are: functional specification; limitations in the design medium; performance requirements; design criteria on the form of the product; and restrictions on the design process. Harty also cites Maher, (1984), who states that these generic constraints influence the design process in two different ways and can therefore be divided into two categories: "hard" and "soft". Hard constraints are those which must be fully satisfied by the design, while soft constraints need not be fully satisfied. Soft constraints may be represented by numerical variables, whose values will vary. They may be combined and used in algorithms to evaluate and then rank potential design options.

Harty (1984, p. 15) says that Malhotra and Thomas (1980) have shown that new design constraints may be identified during the design process and that design is a cyclic process. An iteration in the cycle occurs when a partial design is evaluated and new constraints are identified, the design goal is subsequently changed and the process is repeated until a satisfactory solution is achieved.

She says that this view leads to another definition of design, which is that it is a type of problem solving in which the "goal, initial conditions and allowable transformations are ill defined", or are perceived to be so by the problem solver. She concludes by saying, "thus both identification and integration of constraints are part of the design process.

2.7 Difficulty of Design

Sriram et al (1989) note that significant design problems are ill defined because they do not have a clearly defined algorithmic solution. There are no clear-cut methods to solve these problems and the engineer deals with them using judgment and experience. They note that artificial intelligence techniques, in particular knowledge based technology, offer a methodology to solve these ill-defined problems. Harty describes two types of difficulty, which may arise in design. The first is related to the complexity of the problem. The second type she describes relates to the degree of novelty of the problem. Harty refers to Brown and Chandrasekaran's (1985) system for classifying design problems. They provide three classes of problems; class 1 problems require pure invention, class 2 problems are close to routine design, but require some innovation when standard methods do not work, and class 3 problems can be solved by routine design.

Routine design is a subset of creative design that does not require any innovation because the problem has been solved before. In a routine design the required functionality is completely specified and can often be described with a number of parameter values for the particular problem. Standard solutions are known, and the resulting design is usually one (or more) of these with the appropriate parameter values determined so that the design satisfies the constraints. Harty says class 3, routine design problems, are the most likely type to elect for automation.

Summary

The chapter has presented an outline of the key ideas concerning design. Succeeding chapters will abstract key design activities from this framework to create a model of the design process.

Simon explained the design process in terms of a hierarchical decision-making search process, dependent on heuristic as well as technical knowledge. Furthermore, the elements of this hierarchical design process could be mapped onto Newell and Simon's state space search model of the human problem solving process. Such a mapping would consist of a state space, rules for traversing the state space and a function that determines whether the element under consideration is an acceptable solution.

The state space if fully expanded will eventually contain all the possible design solutions and a design search process should be able to traverse through the state space in stages. Each stage of the design process corresponds to a level in the decomposition hierarchy produced for the design product object and at each stage the design system generates the next level of the state space and then evaluates all the elements on that level. The system generates the next level by combining the attributes of the partially designed objects with the attributes of each alternative available at the new level. This process is accelerated through the use of heuristics, which reduce the number of alternatives generated at each level.

In this state space search model of the design process the state space can be represented as a set of nodes in the form of a tree or graph. The root node of the tree represents the initial start state of the problem. The search process can be considered to be a tree or graph traversal exercise. In a state space search the search progresses when the search agent or

program applies an operator; initially to the start state and subsequently to the current state, and in the process creates a new state. A successor node of the root node represents each of the new states that can be produced from the initial state by the application of an operator. Subsequent operator applications produce successors of these nodes and a directed arc of the tree represents each operator application. As the search progresses the system examines each node until a goal state is found.

Dasgupta (1992) says that Simon introduced a new paradigm, which he refers to as the artificial intelligence (AI) design paradigm. Dasgupta says that according to the AI design paradigm, the design process begins with a symbolic representation of the problem in a state space; the problem space. The problem representation may include the 'initial' state and a specification of the goal state. The goal state, which is to be achieved, will be a 'data path' consisting of the designs for the components and subsystems of the complete structure and a scheme for fitting them together.

Dasgupta points out that in addition to the goal and initial states, the problem space allows for the representation of partial designs, designs satisfying other requirements, sub-assemblies and components. He says the problem space is a space of possible design states.

Applying a finite sequence of operators effects transitions from one state to another. The result of applying these operations in effect causes a search for the solution through the problem space. Dasgupta adds that since the problem space can be very large, heuristics are needed to control the amount of search so that the goal state can be reached. Dasgupta adds that Simon had recognised the need to distinguish between optimal and satisficing design.

The key ideas presented so far include:

- Design is a special kind of problem solving, which involves the search for or selection of a satisfactory combination of components. It operates by arranging combinations of known components and creating new components by applying design rules.
- Design is a hierarchical process. It starts with an abstract high level or global view and ends with a detailed configuration.
- During the design process an object is decomposed, layer-by-layer and subsequently recomposed by combining the alternative options at each layer or level in the hierarchy. These combinations of subcomponents form the substance of the search space.
- There are various strategies for controlling the search process, including the use of heuristics. These strategies reduce the number of new combinations of components, produced and evaluated during the process.
- Different perspectives of design exist; these include the state descriptions, which are product focused and process descriptions, which focus on design activities.
- Design constraints act to limit the feasibility of possible design alternatives produced.
- Design problems can be classified by the degree of complexity involved. Three classes of design problems have been proposed. Most sources consulted place preliminary structural design in the 'routine design' class. This implies that the possible design solutions produced in the process are predetermined.
- A design system must contain knowledge about how to refine an intermediate design in a stepwise manner. It must be able to represent the initial state of the object, and any intermediate or partial designs and the final design solution created during the process.

- A design system must be able to represent design constraints and must have a mechanism to test the designs it has created against the constraints it has represented.

Chapters 3 and 4 cover structures and structural design, they introduce more product and process related information. Chapter 3 provides an overview of the structural design process and chapter 4 describes how structural design knowledge has been represented in various knowledge based design systems.

CHAPTER 3. Structural Design

3.1 Introduction

This chapter provides an overview of the structural design process. Section 3.2 of the chapter defines structural design and introduces the term *systems thinking* as it relates to structural design; section 3.3 describes what preliminary structural design is, and where it fits in the overall design process. Sections 3.4 through 3.6 explain how the subsystems in a building can be conceptually organised into a hierarchy of vertical and horizontal subsystems. Section 3.7 focuses on how the selection of acceptable designs may be handled, section 3.8 discusses the expertise required to complete successful preliminary structural designs and section 3.9 introduces some of the approximate calculations required in the process.

3.2 Structural Design

Ambrose (1967) describes a building design as 'the projected image of a building which may be presented in any or all of the following forms: a verbal or graphic description, a scaled model, or some other representation'. Ambrose says that the design activity is essentially a synthetic process, which involves the bringing together of many disparate objects into a composite whole. Ambrose points out that design requires:

- Examination of the design problem;
- Establishing of criteria for the design;
- Selective isolation of design alternatives; and
- Evaluation of the completed design.

Each of these tasks is essentially analytical in nature. The purpose of structural design is to provide a plan to build the stable underlying structure for a building, which can safely satisfy the requirements of its owners. Planning requirements include structural stability, functional capability and buildability within predetermined economic constraints. Economic constraints include factors such as the cost of construction and the time taken to complete construction. Structural design consists of the selection, manipulation and association of the form, scale and material of a structure in response to the needs dictated by specific building problems. These specific problems are elaborated when the designer starts to determine how the building must be built so that it fulfils the requirements of its owners.

Lin (1981) provides a comprehensive description of structural design in terms of the provision for a 'need to transmit loads in space to a support or foundation, subject to constraints on costs, geometry, or other criteria'. The process should finish with the production of a detailed specification of a structural configuration capable of transmitting these loads and maintaining system integrity.

Several sources, Maher (1984), Harty (1987), Merritt, D. (1985), Sriram et al. (1989) agree that the following stages will be identified in any structural design process:

- Preliminary structural design or conceptual design, which involves the synthesis of potential feasible configurations, followed by the evaluation and selection of the optimal configuration;
- Analysis, which involves modeling the selected structural configuration and determining its response to external forces;
- Detailed design, which involves the selection and proportioning of the structural components, so that all applicable constraints are satisfied; and

- Evaluation and Optimisation, where the detailed design alternative is evaluated. This may require some backtracking to earlier stages to achieve an optimum design.

'Systems thinking' is a general term to describe a certain way of thinking and working. It has emerged as a powerful and innovative tool for building the necessary frameworks for dealing with complex issues. It allows the user to see whole systems and interrelationships, and to pull out the important data and complex patterns that are at work in a system. Leclerc (1979) has described what this means in terms of engineering. He says it requires thinking in terms of a whole system, describing the components of the system and how they interact, it also requires the use of optimisation techniques, the use of mathematical models to simulate complex data, and it requires data analysis and the comparison of alternatives.

Merritt (1985) provides a useful description of design, which introduces the concepts of systems thinking and the need for information. He defines system design as:

> "systems design is the application of scientific methods to the selection and assembly of components or subsystems to form the optimum system to attain specified goals and objectives while subject to given constraints and restrictions."

According to Merritt, (1985, p 1-9) the simplest building system consists of only two components, these being a floor or flat horizontal surface, and an enclosure that extends over and around the floor to provide shelter. He states that both components must be designed so that they transmit the vertical (gravity) loads, horizontal (lateral) loads and the horizontal and vertical components of inclined loads, to the foundations. Harty (1987) notes that this is done by providing a path for the loads through the structure to the ground below, which provides the ultimate resistance. This path, which is in effect the design, and which

must be documented, is a configuration of walls, columns, beams and floors, which act as a unit to provide overall structural stability.

Both Lin and Merritt refer throughout their accounts to *building systems* and the *systems thinking* paradigm. Merritt (1985) explains how *systems thinking*, which he also refers to as *systems design,* applies to building design. He says that systems design is the process of providing all the information necessary for the construction of a building that will meet its owner's requirements and also satisfy public health, welfare and safety requirements.

He contrasts the systems design approach to building design with traditional building practices. He contends that traditionally buildings were designed by effecting some form of imitation or modifying of designs of existing buildings combined with some form of trial and error. Gordon (1978) also supports this point of view. Merritt asserts that the introduction to building design, of systems thinking, in the guise of operations research or systems design methodologies, has made major advances possible in creativity and innovation. He says that any innovations were rare with traditional building design and were developed fortuitously. In contrast systems design is a more precise procedure that guides creativity towards the best design decisions.

Systems design comprises a rational, orderly series of steps that leads to the best decision for a given set of conditions. These steps include a repetitive series of cycles of the following activities:

- Analysis;
- Synthesis; and
- Appraisal.

The system design process requires that the building be analysed as a system from the outset. This is followed by synthesis or selection of components to form a collection of candidate systems, which meet specific objectives, while at the same time being subject to constraints or variables controllable, by the designer. The final stage includes appraisal of the system's performance, which also includes comparisons with alternative systems. The final stage also allows for feedback to the earlier analysis and synthesis stages, of information obtained in systems evaluations to improve the design. This feedback should result in incremental improvement to the evolving design.

Merritt contends that the major advantage of the *systems design* methodology is that, through comparisons of alternatives and data feedback to the design process, it results in the system's design converging on an optimum system for the given conditions.

Merritt identifies nine separate steps that make up the building design process. These are:

- Analysis of the building as a system commences at step 1, where the definition of purpose and goals is elaborated. The designers obtain a building program and collect any information on existing conditions that will affect building design. The designers define the goals to be met by the building. These goals state the purpose of the building and how it will interact with the environment and with other systems, including heating, ventilation and the utilities. These design goals should be sufficiently specific to guide the generation of initial and alternative designs and to control selection of the best alternatives.
- In step 2, the designers establish the building's objectives and constraints. These objectives are meant to guide design of the building systems at the more detailed levels. The objectives may be numerous and cover such things as minimisation of cost and

construction time, health and welfare considerations and zoning. Merritt says the objectives should contain sufficient information to allow the designer to plan the building's interior spaces and to design specific characteristics of the building and its components including appearance, strength, durability, stiffness, operational efficiency, maintenance and fire resistance. Criteria should be developed for these objectives and the objectives should be weighted to reflect their relative importance to the building owner.

A series of design variables should be developed to represent the values of physical forces calculated or estimated for a particular building system design.

Constraints are restrictions on the values of design variables that represent the properties of the system and are controllable by the designer. Again Merritt adds that standards must be associated with each constraint. A standard is a value or range of values governing an attribute of the system.

- Merritt's third step is the synthesis stage, where the designer puts forward specifications for at least one system that satisfies the objectives and constraints established. It is at this stage that the designers rely on past experience, knowledge and skills. He notes that synthesis often requires input from specialists in several different disciplines including structural engineers, construction experts and materials specialists.

- In step 4 the designers create a model of the system that will allow them to analyse it and evaluate its performance. Merritt describes three classes of models: iconic, symbolic and analog. The most important class of model from the point of view of this study is the symbolic model. Merritt identifies four separate substages in step 4:

  - Select and calibrate a model to represent the system for optimisation and appraisal;

  - Estimate values for the uncontrollable independent design variables;
  - Determine values for the controllable design variables; and
  - Determine the output or performance of the system from the relationship of dependent and independent variables by use of the model.

- In step 5 the designers evaluate the results output from in step 4.
- Step 6 is where value analysis is applied to the whole building system. This may result in either a change to the whole system, thereby producing a new one or in changes to parts of the system.
- In step 7 new models are created for the new systems or at least those designs with good prospects.
- In step 8 these models are further evaluated. During and after step 8 completely different alternatives may be conceived. As a result, steps 4 through 8 should be repeated for the new concepts.
- In the final step the best of the systems are selected. Selection is based on the results of some form of ranking of the scores compiled during the evaluation steps.

Merritt says that systems design processes may be used in all phases of building design. However, he stresses that systems design is most advantageous in the early or preliminary design stages, which corresponds to steps 1, 2 and 3, described in the previous paragraph above. During preliminary design one system may be substituted for another and components may be eliminated or combined in those stages with little or no cost.

The writer obtained a description, which complements Merritt's, from Bedard and Gowri (1990). Their description stresses the idea of decision making in the structural design

process. They regard the process, as a decision-making process that generates the detailed documents, which enable the construction of a product to satisfy a need. The process, which they describe, consists of four interrelated subprocesses, which are shown in Figure 3.1.

The *definition* subprocess involves the identification of a need and the specification of the object to be designed. In this model the *synthesis* and *analysis* subprocesses are interrelated and in the process information cycles back and forth between them until a specification for a feasible alternative has been compiled and has passed the testing and optimising stage of the *analysis* subprocess. During synthesis, components or subsystems are combined, and then tested in the *analysis* subprocess. These two subprocesses are repeated until the assembled product reaches an appropriate standard. The final subprocess, *documentation*, involves the production of drawings and written specifications. These should contain sufficient information to allow the building to be assembled.

These subprocesses can also be viewed as a sequence of time-related events. At the conceptual (*definition)* stage different roughly defined schemes are proposed. Then in the preliminary design *(synthesis* and *analysis*) stage, which follows, the designer selects the best alternative and in the final stages this alternative will be designed in detail.

Bedard and Gowri also draw attention to some of the unique characteristics and inherent difficulties encountered in developing building designs. They say the design of a building is unique because: a single product is designed and is only built once, and; the product will be built in a natural environment.

The design process is inherently difficult because:

- Contributions are required from several disciplines: engineering, architecture, suppliers and contractors;
- Buildings are made of several subsystems, which interact with each other but which operate on different principles;
- Typically the subsystems are designed by different groups;
- The overall design concept is usually imposed at an early stage in the design in the absence of knowledge of how the subsystems will interrelate with each other; and
- The building industry is fragmented into numerous distinct and diverse organisations.

Other factors that the designer must be aware of are: government regulations, standards, building codes and local authority by-laws. All of these contribute to a complex and difficult process. They suggest that the process could be improved if a method could be found to integrate the different viewpoints of the key participants. It could also be improved if there were better ways to consider buildings as complex systems, containing subsystems, whose interactions with each other were key design considerations. They stress that priority should be given to the support of decision-making procedures that incorporate multi disciplinary knowledge and make the resulting information available at the earliest possible stage in the design process.

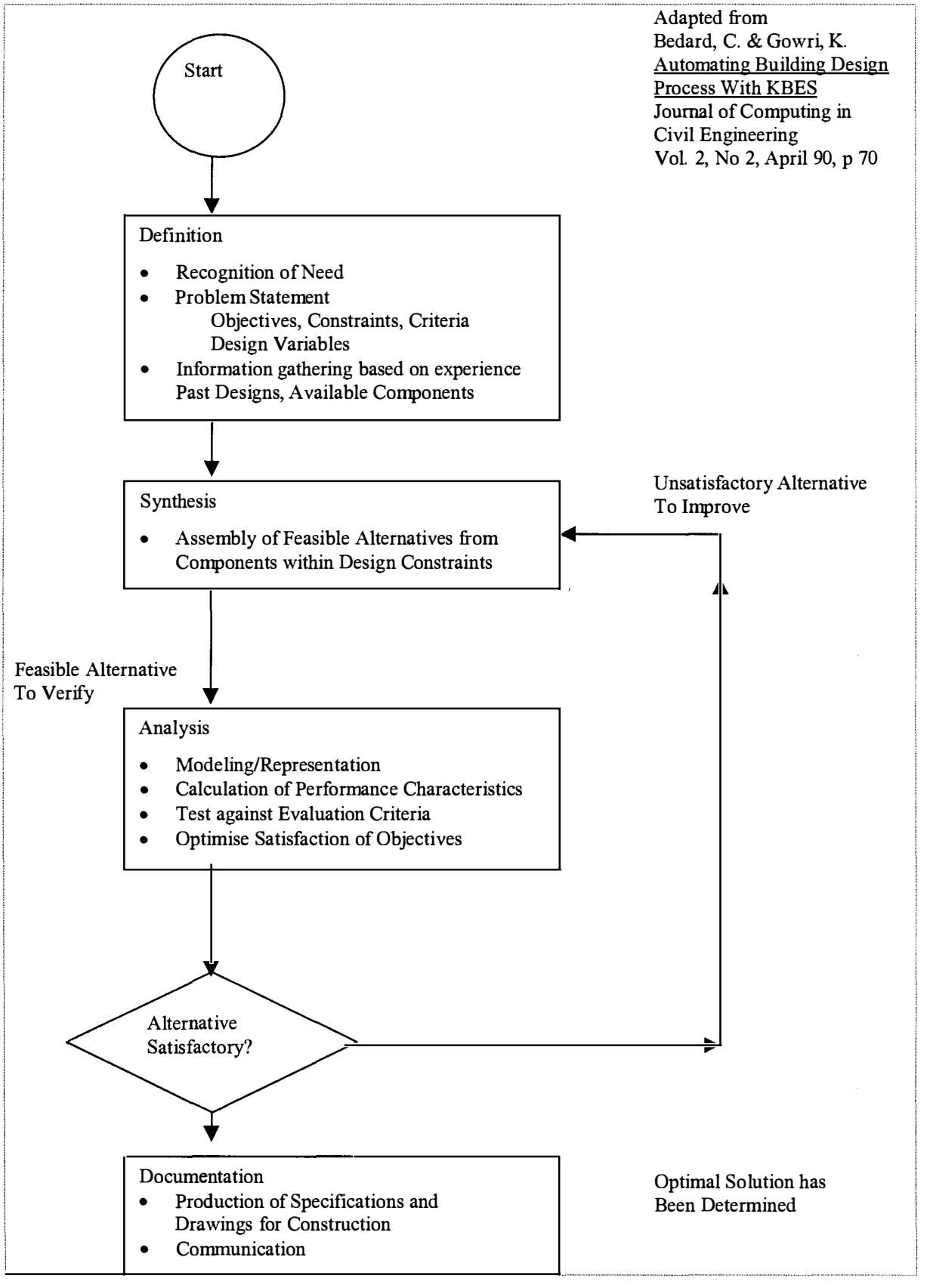


Figure 3.1 The Structural Design Process

3.3 Preliminary Structural Design and the Building Design Process

As noted in the Section 1.3, preliminary structural design is an early stage in the design process during which the engineer formulates and assesses a number of different structural schemes or configurations. The schemes are assessed in order to enable selection of the one, which best satisfies a variety of constraints. This scheme will then be used to produce the detailed design, which determines the overall form of the structure and produces a scheme for constructing it. The results of this process are documented in the initial design specification. According to Harty (1987), the preliminary structural design task is ill defined, it relies on experience and judgment and it involves activities and decisions, which are heuristic in nature.

During preliminary structural design, the designer explores possible alternatives by producing rough designs, which contain approximate measurements, but which still maintain the important features of the design problem. In cases of routine design an experienced designer should be able to easily identify, which features are important.

Eventually the designer will produce a set of feasible designs, which will satisfy the hard design constraints. The designs produced are then examined to determine how well each one satisfies a series of soft constraints, which have been weighted in order of importance. The feasible designs can then be ranked in order of how well they satisfy these soft constraints.

Lin (1981) says that structural design should be approached hierarchically. The architect should think of the design of the building environment as a total system of interacting and space-forming subsystems. This approach will require a hierarchical design process that provides "at least 3 levels of feedback thinking: schematic, preliminary and final."

According to Lin the schematic feedback level generates and evaluates the overall site-plan, scheme of activity, organisation and building configuration options. At the preliminary level, the architect will focus on his or her more promising schematic design options at the same time as addressing the approximate design of specific subsystem options. He or she will consult with the structural designer to identify and design the major subsystems to the extent that their key geometric, component and interactive properties are established. Basic subsystem interaction and design conflicts will be identified and resolved in the context of the total-system objective.

The key objective of preliminary structural design is to select the structural configuration to be used for the final design. During the process a number of rough designs will be produced. When the designer and the architect are satisfied with the feasibility of a design proposal at the preliminary level, then the basic problems of overall design will have been solved and subsequent detailed analysis is not likely to produce major changes. The design process then moves into the final level, which involves in-depth design refinement of all subsystems and components and the preparation of working documents. In summary at the preliminary stage the designer must be able to identify the major subsystem requirements implied by the scheme proposed by the architect and must be able to substantiate their interactive feasibility by approximating key component properties. The designer will at this stage have worked out the properties of the major subsystems in sufficient detail to verify the inherent comparability of their basic form-related and behavioural interactions.

3.4 How Building Subsystems are Organised into a Hierarchy

Lin (1981) stresses the need for designers to understand the overall relationship between the structural and the space form prospects of proposed architectural schemes. To do this,

designers must first conceptualise the schematic options for providing total-system integrity and then compare the likely alternatives for designing the major subsystems. In turn the designers must be able to optimise the interaction of key force and geometric properties of these subsystems at a preliminary design level. This means that the final design should provide the desired size and shape required by the owner, at the same time the underlying structure supporting the building must be able to resist the forces acting on the building to maintain its integrity.

Lin explains the term total-system integrity, saying that a given form option is assumed to behave as a structural whole, which can be analysed as a whole to determine its overall load and the corresponding resistance to that load, which must be designed into the form. This consideration of load will take into account: that the form is fixed to the ground, the form will have a mass, which must be supported, by the ground; and that the form will have to resist horizontal wind and earthquake forces.

When designing a building form, a designer must consider the subsystems that are required to substantiate this assumption of total-system integrity. In turn this means that the designer must be able to understand what subsystems are required and how they will interact to actually achieve this integrity. Furthermore, at the schematic level the designer should be able to identify design options for laying out the interaction of key subsystems. Then, at the preliminary level the designer should work hierarchically to prove the feasibility of such systems by determining their key properties. At this level the designer can use approximate values and the detailed calculations can be postponed until the final stage of design.

Harty (1987) interprets Lin' s view of a hierarchy of structural actions in building forms as follows. She says that this view of the design process envisages 3 stages, schematic, preliminary and final, which represents a hierarchical approach going from the total system

to subsystems to components. First a three-dimensional (3-D) schematic design of the total structure is produced; this is followed by the preliminary design of the major two-dimensional (2-D) subsystems. This, in turn, is followed by the final design of all the individual components, which may be considered one-dimensional.

Harty envisages a repetitive cycle of design stages, which are not usually finalised in one pass through the hierarchy. Each stage provides feedback to the stage before it so that there is considerable iterative refinement before the final design is produced.

It is usually at the 2-D stage that the structural engineer becomes involved in the design, after the 3-D concept has been chosen. The 2-D subsystems examined in preliminary design are the vertical and horizontal structural subsystems. These resist lateral wind and earthquake forces, and the building's gravity loading respectively. The horizontal subsystem is a frame of floors, beams and columns. There is a wide range of combinations: flat plate, slab and beam, slab and main and intermediate beams, waffles and space trusses. Lin describes the horizontal subsystems as *2-D wholes* that act vertically to carry the floor or roof loads in bending, and act horizontally as diaphragms and/or column connectors.

Similarly, the vertical subsystems are visualised as wholes that act to pick up loads from the horizontal subsystems and also act to resist the horizontal, laterally acting forces. The horizontal subsystems must be supported by the vertical subsystems, likewise the vertical subsystems, which are generally slender in nature and unstable, must be held in place by the horizontal subsystems.

There are three primary types of 2-D vertical subsystems found in buildings, these are wall subsystems, vertical shafts and rigid beam-column frames. In contrast, horizontal surfaces can be designed as plate, slab, beam, grid, or truss subsystems, which can be realised in

various materials. As noted in the previous paragraph, the design and construction of horizontal subsystems is related to the arrangement of the supporting vertical subsystem, which consist of regular patterns of columns, frames, bearing walls and/or shafts. The vertical subsystem is usually designed before the horizontal.

Lin notes that during the design process whilst the subsystems are designed separately, they must both be considered more-or-less simultaneously. The designer must keep in mind that they react together to form a 3-D building and that each load bearing subsystem interacts with the other. Thus vertical systems also transmit components of the gravity load to the ground. Likewise the horizontal load bearing systems contribute stiffness resistance to the vertical systems, by acting as diaphragms, which hold the vertical subsystem in place.

Harty (1987) proposed a shortcut to simplify the design process. She says the subsystems can be treated separately by assuming that the vertical subsystem accounts for the entire lateral loading and that the horizontal takes all the gravity loading. This is a conservative approximation, which is adequate for low rise but not suitable for very tall buildings (over 30 floors). She says that this assumption simplifies the design process and she used it in the DOLMEN system.

The height of a building influences much of its design, in particular its horizontal load resisting requirements. While vertical load effects increase linearly with the number of storeys, horizontal load effects vary non-linearly; the overturning moment due to horizontal loading is proportional to the square of the building height, while the horizontal sway is proportional to the fourth power of the building height. Therefore, in a high-rise building, which Lin defines as one, which has at least ten stories, the choice of vertical structural subsystem tends to govern design and must be completed first. In contrast the columns,

walls and stair cores of low and medium rise buildings can usually resist most of the horizontal forces and the choice of the floor system is the predominant design activity.

Lin says this approach reflects the natural hierarchy of an architectural design problem. The structural elements will become architecturally relevant only when it is understood how they can work together, organising, and building subsystems to contribute to the fulfillment of the broader need for enclosing spaces using the 2-D forms. Similarly, these 2-D subsystems become architecturally relevant, when it is understood that they contribute to the overall effectiveness of the 3-D space-form scheme as a total environmental system.

To summarise:

- At the 3-D schematic level, the structure-form relationships are analysed as a total system and the architect aims to provide for overall total-system integrity;
- At the 2-D preliminary level, basic horizontal and vertical subsystems are identified and key component properties and interactions are established; and
- At the 1-D final level all the linear elements, the beams and columns and their connection details are specified in sufficient detail for the preparation of engineering and construction documents.

According to Lin, at the preliminary or conceptual stages, the designer need only keep in mind the four basic structural subsystem interactions that must be accommodated in order to achieve overall integrity in the structural action of a building form. These are:

- Horizontal, gravity-resisting subsystems must pick up and transfer vertical loads to the vertical subsystems and maintain sectional geometry.

- Horizontal subsystems must also pick up any horizontal loads accumulated along the height of a building and distribute them to the vertical shear-resisting subsystems.
- The vertical subsystem must carry the accumulated dead and live loads, and where required, be capable of transferring shear from the upper portions of a building to the foundation.
- The vertical subsystems must resist the bending and/or axial forces due to overturning moments, caused by the lateral wind load. Where possible, these subsystems should be tied together by horizontal subsystems to optimise overall resistance.

3.5 Vertical Structural Subsystems

This section, is taken from Lin (1981), it describes examples of the commonly occurring types of subsystems. There are three main types of vertical structural subsystems, which are in common use:

- Vertical Shaft. This is a tube of walls around a hollow core, in many buildings the shaft would enclose the stair-core or lift-shaft;
- Wall Subsystems. These include solid shear walls of reinforced concrete, masonry, or paneled timber, and trussed walls made of braced steel or timber frame; and
- Frames of Rigid Beam or Columns. These frames are rigidly jointed and may be made out of reinforced concrete or structural steel. The rigid joints make the structure capable of resisting horizontal loading.

According to Harty (1987) a shaft could be considered to be a 3-D unit, which on its own may supply complete stability. The wall and frames subsystems are considered to be 2-Dimensional, as they are only designed to resist the in-plane horizontal loading. In order to

have complete stability, 2-D subsystems must be used in two orthogonal directions. Many combinations of these subsystems may be used, for example, shear walls in one direction and rigid frames in the perpendicular direction, or a shaft together with shear walls.

Harty explains that in addition to selecting which subsystem to use, the designer must determine the location of the subsystem within the building. Usually a grid-like, layout diagram is drawn up for the building. The locations of the walls or frame systems are then mapped on to this layout grid. In general the designer aims to:

- Minimise the number of vertical subsystems, to reduce expense;
- Minimise the area of the walls to maximise the space available in the building; and
- Place the systems in a symmetrical pattern to avoid torsion in the structure.

Usually the architect's plans already indicate the placement of the walls and the engineer needs to incorporate this given information into the design. Having done this the engineer may need to suggest additional walls or structures to complete the design.

3.6 Horizontal Structural Subsystems

Harty explains that any horizontal subsystem will consist of a frame of floors, beams and columns. In general its structural behaviour will be analysed in terms of the overall bending of the frame and in terms of the concentration of shearing forces around the support. Lin divides the overall design methods for horizontal subsystems into two groups, one-way and two-way subsystems. For one-way designs, for example for a slab, any strip or the whole slab can be designed to carry its full tributary load. In a two-way slab the total load will be carried one-half in each of two orthogonal directions and any strip should be designed to carry one-half of its load in simple bending. Furthermore, it is essential to provide

sufficient material or resisting strength around the vertical supporting members to avoid failure in shear transfer of loads to the columns.

There are many combinations of components, which are commonly used in horizontal subsystems. The following descriptions are taken from Lin (1981, pp. 157-200):

- Flat Plate: the floor acts as a plate supported only by columns without beams. Most flat plates are of reinforced or prestressed concrete.

- Slab and Beam: one-way or two-way slabs spanning across beams, which are located along column lines. One-way slabs span between 2 beams, whereas two-way slabs are supported by beams on all 4 sides. A one-way slab may be made from precast, reinforced or prestressed concrete, timber, or concrete topped steel decking. A two-way slab is usually only reinforced or prestressed concrete. The slab transmits the gravity load, horizontally by shear and bending resistance, to the walls; the load then goes directly through the walls and into the foundations.

- Slab, and Main and Intermediate Beams: also referred to as Joist and Girder subsystems. Joists are closely spaced small beams, beams are larger and heavier members and often-span as much as 10 metres apart and girders are deep beams. Girders are designed to pick up heavy loads accumulated from many joists and beams.

  In this type of subsystem, where the slab is not capable of spanning the full bay width, then beams are placed between the column lines to provide support. This may be because of a limitation of the slab material or because the slab depth is required to be very small. When the slab depth is restricted, a type of subsystem known as Ribbed Slab may be used. This type involves the use of steel and

concrete, the steel is corrugated and the concrete is placed on top of it in the form of a slab or in the heavier form is poured into the corrugations. Moulds are used to form the ribbed slabs and intermediate beams, which then span onto main beams.

- Waffles: these are a two-way version of ribbed slab; they are also constructed using moulds, which form the beams and slabs. Usually waffles are used without separate deep beams, but the waffle hollow may be filled in around the columns, or along column lines, forming beam strips of the same depths as the rest of the floor.
- Space Trusses: these are used to cover a very large clear-span area with a flat floor or roof. These consist of large trusses running in one direction and spanning between the columns, which serve as the main carrying members. Smaller trusses span between these large trusses, perpendicular to them. Trusses are similar to the Joist and Girder subsystems, but use trusses instead of joists and girders.

Harty adds that horizontal subsystems also include all those support columns, which are not part of the vertical subsystem. Their sole purpose is the transmission of gravity loads to the ground, and they may be made from concrete, structural steel or timber.

3.7 Selection of Subsystems

From the discussion included in sections 3.5 and 3.6 it can be seen that the two major tasks of preliminary structural design are to select the vertical and horizontal subsystems to make the 3-D concept a reality. Harty says it requires a considerable amount of expertise to select the most appropriate configuration for the project from the wide range of alternatives available. Each building is different, with its own unique set of constraints. The appropriate loadings must be estimated, usually with reference to appropriate Building Codes, and initial sizes chosen for the building components. Furthermore, structural

analysis of the building must be carried out for the particular geometrical constraints, which apply.

3.8 Expertise

To improve the efficiency of the design process it is necessary to capture the expertise required to quickly identify a small number of alternatives, which are worth considering, and to produce outline designs, which are very close to the final detailed designs.

This section explores various aspects of design expertise encountered in preliminary structural design. Experienced engineers use their knowledge of previous projects, together with technical expertise to produce suitable preliminary designs and to advise on the selection of one of them. In practice this expertise is held by experienced designers, who have developed their knowledge by being involved in a large number of projects. This type of knowledge is expressed in the form of either comparisons with previous projects, or in rules of thumb. Harty (1987) stresses the fact that the rules of thumb are all based on generalisations made from a large set of examples, which were produced using sound engineering calculations. In practice, designs effected by rules of thumb are always confirmed by calculation. Sometimes this results in the engineers unearthing cases where the rules of thumb do not work. This adds to their knowledge, which they can apply to subsequent projects.

At the preliminary stage the structural engineer is required to assess values for loading and sizing and to select a structural configuration. In estimating the lateral and gravity loadings for which the building should be designed the engineer is usually guided by the relevant building code, for example Harty (1987) quotes standard number BS6399, British Standards

Institute (1984). This code suggests typical loads and the engineer can start with these values and work out a reasonable figure given the circumstances of the particular project.

After selecting the loading the engineer selects a suitable structural configuration. Then the displacements and forces on structural members are calculated. The beams, columns, slabs and walls are then designed. The engineer uses judgment and experience to reduce the number of possible choices by eliminating options, which can be seen to be unsuitable from the onset.

The next stage involves selecting the initial sizes for the different schemes and analysing them. Rules of thumb may again be used in this stage for example to choose sizes based on span to depth ratios. Harty (1987) says the *'Manual for the Design of Reinforced Concrete Building Structures'*, (*RC Manual*), (Institute of Structural Engineers, 1985), includes many standard rules of thumb, which have been adopted from building code BS8110. They were compiled specifically to assist with the preliminary design of reinforced concrete buildings. They include recommendations for dead loads and the material strengths to be used, maximum slabs and beam spans, and initial sizing of members.

After completing the estimates for loading, selecting configurations and sizing, the engineer is required to evaluate the designs, comparing the alternatives in terms of several different factors. Expertise is required here to establish what factors must be considered and what relative importance should be attached to them. It is at this stage that soft constraints must be addressed. These constraints are of the type, which can be satisfied to a greater or lesser degree, for instance, cost minimisation. Soft constraints differ from hard constraints, which act in an all or nothing manner. Harty says that knowing what factors must be considered and what relative importance should be attached to them requires a great deal of expertise and judgment.

3.9 Approximate Calculations

Harty (1987) cites the recommendations made in the RC Manual of the Institute of Structural Engineers (1985) mentioned above, which says that initial design methods should be simple, quick, conservative and reliable, and that lengthy analytical methods should be avoided. The structural schemes produced should be suitable for the building's function, they should be economical and should allow for the inevitable design modifications.

Typical preliminary structural design includes the design of representative parts of a building, which include estimates for foundations and edge, interior beams, slabs and edge, and interior columns. These estimates need only be compiled every 2 or 3 floors of the building.

Harty has incorporated several recommendations from the *RC Manual* into her model, these are listed below:

- Sizing of beams based on the longest spans,
- Simple formulae for the calculations of moments on slabs and beams (not to be used for rigid frames),
- Stress checks on elements,
- Recommendations for the arrangement of reinforcement in concrete,
- Methods for estimating the total weight of reinforcement for the required areas of main steel, which has been calculated during design.

Harty has also incorporated simplified analysis methods for rigid frames from Lin (1981) and for steel from Joannides (1987).

Summary

This chapter has provided a limited outline of the structural design process and shown how preliminary structural design fits into this process. It has also presented outlines of more formal descriptions provided by Ambrose, Lin, Merritt and Bedard and Gowri. In particular it has relied on material taken from Lin (1981) who has recommended a systems based approach, which perceives the building as a hierarchical organisation of subsystems and which requires the designer to assess the building structure as a whole, to be able to:

- Estimate loadings based on relevant Building Codes,
- Select and assess different combinations of subsystems,
- Size and test members of these subsystems using approximate calculations.

It is clear that the process of creating structural designs is extremely complex and requires input from many disciplines. In particular it requires significant architectural and engineering input. Furthermore, the process must take account of a very wide range of limiting factors including legal requirements and building codes. It is also constrained by economic factors, which include cost of construction and availability of materials.

Although the preliminary structural design process is ill defined several identifiable stages may be recognised. Unfortunately, these stages are also neither precisely defined nor are they clearly distinguishable from each other. The process is further complicated by the fact that these stages are repetitive and the whole process may go through several iterations.

CHAPTER 4. Object-Oriented Design Support Tools

4.1 Objectives

A literature survey was completed to review developments in computer assisted design. It was intended to support the following tasks:

- Determine what progress had been made with the use of computers to support preliminary structural design;
- Gain an understanding of the issues involved in the application of object-oriented concepts and techniques in the development of knowledge-based design systems. In particular what methods had been used to represent preliminary structural design knowledge.

These tasks were intended to support the primary objectives of the study, which were to:

- Adopt and analyse a particular approach to the application of knowledge based computer systems to the task of preliminary structural design; and
- Determine whether or not it was practical to implement this approach in a PC-based object-oriented knowledge engineering environment by developing a prototype, knowledge based design system.

4.2 Object-Oriented Design Support Tools

The survey indicated that research in design theory and methodology and problem solving in Artificial Intelligence has provided a basis for the development of systems based on several different types of models of design processes. These models include decomposition; case-based reasoning and transformation based models. Furthermore, there is a large and growing body of literature related to knowledge based systems in structural design. A

bibliography published by B.H.V Topping et al. (1991) included over 280 papers on the subject, and many more papers have been added since then. Given such a large source of references, it was necessary to restrict the scope of this review to reports dealing with systems, which implemented decomposition based process design models of the type described by Maher (1984) and Harty (1987), which supported preliminary structural design.

The papers presented by Maher (1984) and Harty (1987) identified the potential of knowledge-based expert systems to assist with the task of preliminary structural design. In 1984 Maher built a relatively simple mainframe system, called HI-RISE, using PSRL, a frame-based production system language developed at Carnegie-Mellon University. Harty followed in 1987, with an approach similar to that described by Maher, and built a more complex prototype system, which she referred to as the Dolmen system. She used KEE; a powerful commercial hybrid development environment to build this prototype and it was implemented on a smaller UNIX workstation. Harty (1984, p. 202) predicted that eventually this kind of decision-support software would be implemented on smaller more user-friendly and portable computers, where it would be of most use to designers, whose work required them to move to and fro between office and building site.

Both author's reports focused on systems developed for the preliminary design of regularly shaped buildings; however, the literature survey also located reports describing systems, which supported preliminary design in other structural areas. From these reports the writer selected 18 for further examination. These systems, which are described in section 4.3, were developed using a variety of software packages, which included specialised knowledge engineering development tools, such as ART-IM, KEE and Knowledge Craft. These systems met a wide range of user requirements and were implemented in different

ways. Several types of approach were used including transformations, case-based reasoning, neural networks and genetic algorithms. However, in 11 out of 18 reports examined during the review, the approach used involved a knowledge-based expert system, which implemented some form of hierarchical decomposition product data model. The systems described in the reports had several common features in that they:

- Addressed routine design problems, where all possible solutions were predetermined, without attempting any innovative input to the process;
- Identified separate and clearly identifiable phases in the design process, in Harty's model for example, there are three phases:
    - Specification, which established design context, including user requirements.
    - Formulation, which included the synthesis of feasible alternative structural configurations, using some form of generate-and-test strategy. Included in the strategy was some form of analysis, designed to provide values for various attributes belonging to the alternatives, for use in testing against elimination constraints.
    - Evaluation, where alternatives were assessed and compared. This involved consideration of proposed design solutions in terms of several different features, or soft constraints such as cost, time to build and resistance to sway. Usually designs were ranked and the designer made a selection based on inspection of the list of ranked items.
- Used some form of hierarchical planning, plans being developed at successive levels of abstraction;
- Combined production rules and some form of linked data structure or network of frames or objects;

- Decomposed the design product into a hierarchy of object classes and subclasses, which were represented in a tree like structure;
- Expanded each class in the hierarchy into its various subclasses at each level in the hierarchy, using some form of frame based representation, implemented in a language, which supported object-orientation;
- Used inheritance for each subclass in the hierarchy, allowing the more general attributes of its parent class to be inherited;
- Implemented an optimal search of a predetermined space of design alternatives;
- Used heuristic knowledge about the problem to limit the search space;
- Used knowledge bases, designed to store domain (product and process) knowledge, constraint knowledge, procedural knowledge, analysis algorithms and solution knowledge;
- Implemented domain, constraint, and procedural knowledge in a hierarchical manner;
- Included various methods of knowledge acquisition to obtain the required preconditions, decompositions, constraints, evaluation criteria and functions;
- Implemented user interfaces, usually in the form of multi-window displays with various input images, including menus, dialog boxes, input forms and buttons; and
- In some cases the systems were incorporated into integrated design systems, which catered for other stages of the design process including analysis, and detailed design.

Table 4.1 provides a list of these reports, which address the design of a range of structures including:

- Building foundations;

- Building envelope systems;
- Building facades;
- Building energy systems;
- Fixed offshore jacket structures; and
- Reinforced concrete industrial buildings.

4.3 Examples of Systems, which Support Preliminary Structural Design

The following paragraphs describe key aspects of the more significant systems covered in the survey. Aspects discussed include: sources of the design knowledge used in the system, implementation strategies and knowledge representation.

- **A Building Foundation Design System**

Bravo et al. (1996) discussed the design of a system to assist with the selection of building foundations. They noted that the preliminary design of foundations is done in two stages. In the first stage one or more generic types of foundations are selected and in the second stage the most economical solution is identified and designed.

Selection of the foundation starts with the examination of the results of the soil exploration report and with consideration of the building characteristics. This stage includes the following subtasks:

- Soil classification;
- Determination of soil properties;
- Determination of the minimum depth of the foundation;
- Estimation of allowable bearing pressure;

- Location of possible supporting strata, and;
- Selection of generic type of foundation.

Each type of foundation system has its own specific preliminary design process, which requires information about the vertical subsystems directly supported by the foundation, including the layout of those structural components and the loads transmitted by them. Subsequent stages of design form part of the detailed design of the foundation and are not included in the process of preliminary design.

The knowledge-based system described by Bravo, produced designs for shallow foundations, (footings and mat foundations); semi-deep foundations, (block foundations), and deep foundations (pile foundations). The design solutions were restricted, only square and rectangular footings were used and only mat foundations with uniform thickness.

In describing the model, the authors say the system is designed to address routine design, where all possible solutions are predetermined. The system uses a single solution problem solving process and there is no strategic knowledge used in the application.

They say the application system captures the structure reflected in the models they produced for the design process. The system architecture was established on the basis of the decomposition, which they found at the global problem solving level for the foundation task. They add that the hierarchical decomposition of functional and structural components has been reflected in the data structures in the application and that the system has procedural code to replicate the process activities.

The modeling scheme represents three types of domain knowledge:

- Real-world object classes;

- A framework of conceptual knowledge including physical, behavioural, functional and environmental interactions and state characteristics; and
- Control knowledge to model basic design operations, the relationship between them, their organisation in design tasks and the design strategies that are appropriate.

They used the ART-IM development tool to implement the system. They described it as a hybrid tool that integrates several programming paradigms including, production rules, frames, object-oriented and procedure programming.

Schemes, (ART-IM frames) were used to represent the object classes included in the object model and other concepts identified in the domain knowledge analysis. Production rules and functions were used to effect inferencing applied to relations between object attributes. Functions were also used for the user interface and in attribute or parameter calculations. ART-IM 'facts' were used to store some data elements and intermediate results that did not justify the creation of new object classes.

- **A Building Envelope Design System**

Bedard & Gowri (1990) described this system, which generates feasible combinations of building envelope components. Initially it presents the designer with a list of feasible wall and roof types. The designer can then input data to specify his/her preferences among them and to specify relative levels of preference from a series of performance attributes. The system then ranks the alternatives. The design process model includes the following core preliminary design functions: establish the context; generate feasible design alternatives; and evaluate the feasible alternatives to select the best one.

The modeling device used to develop the system is a frame based knowledge representation scheme. It allows the designers to integrate building code requirements, weather data, and

information on constructional types and building material properties. The model was designed to complete extensive searching through a large 'design space', which comprised a collection of possible envelope design alternatives. The finished system was implemented using the Knowledge Craft development tool, which was running on a DEC VAX 11/785 platform. The knowledge base, which consists of 80 frames, was created using ESCHER, a front-end program developed at the Centre for Building Studies, for encoding engineering knowledge. The user interface, program requirements for calculating thermal performance and energy efficiency and the generation and evaluation of alternatives were developed using Common LISP functions.

The authors report that this tool meets the essential requirements for effective design automation, which are multi disciplinary expertise and availability of results at an early stage in the design process. They add that more significantly, the system provides real preliminary design capability, and that it requires little input data and provides meaningful comparisons between partial design alternatives.

- **A System for Designing Building Facades**

Karhu (1997) describes a prototype system for designing building facades. His paper focuses on research on the development of a product data model to exchange data about the design of precast building facades between architects, structural engineers and precast element manufacturers. The aims of the research were to:

- Define a basic activity model of the building process of precast concrete facades emphasizing the architectural design;
- Analyse the problems occurring in a typical design process;
- Define a product model of a precast concrete facade;

- Draw up a checklist for the data requirements and the information produced;
- Develop software based on the product model;
- Test the prototype software in a real project;
- Assess whether or not the product model based architectural design process enhances the building design process; and
- Propose guidelines for using the product model based approach in architectural design.

The project described by Karhu produced a process data model to reflect the prevailing way of designing facades. Activities were drawn as boxes, with inputs and outputs shown as arrows. Karhu used a formal structural methodology, Structured Analysis and Design Technique (SADT), to create a model of process activities. Activity diagrams were decomposed hierarchically to allow for more detailed information and design activities were described in the lower levels of the diagrams. Then after analysing the data needed in the different stages of the design process, it was possible to develop a product data model of a facade.

The data in the product model was arranged in a systematic way using object-oriented data base principles, which in this case involved the use of schema. The schemas were organised hierarchically and reflected the precast concrete elements, which make up the facade and the type of wall openings, which in turn dictated the shape of the facade. These schemas reflected the decomposition hierarchy of the facade, which includes the following layers: structural system, external subsystem, facade, precast unit and element. These layers are represented by the schema, which use slots to represent attributes, for example, edges, shapes and insulation details.

The schema were developed using the EXPRESS language and its graphical counterpart EXPRESS-G. This language is provided by the International Standards Organisation, (ISO) and is associated with the STEP model. The STEP model, ISO-STEP, (ISO 1992) is the Standard for the Exchange of Product Model Data, (STEP); it is being developed by the ISO-TC184/SC4 Committee of the ISO. STEP is a series of international standards devised to achieve faster design times, better communication of design information and long term archiving of designs. In particular it facilitates the communication of product data to customers and suppliers worldwide.

As part of the project the team developed a prototype computer application to test the product data model. The software platform used was AutoCAD. The AutoCAD AutoLISP programming language was used for the procedural programming. The prototype was tested successfully with data taken from a real design project. Karhu's report describes the difficulties encountered when one tries to use object-oriented analysis and design methods for both process model and product model. Eventually Karhu used the non object-oriented, structured SADT methodology to model the process and he says it was well suited for the provision of an overall description of the activities that occur during the traditional building process. However, an object modeling approach was used to analyse and design the product model.

- **A Design System for the Energy Systems in Buildings**

Doheny and Monaghan (1987) described the development of an expert system, IDABES, which supported the preliminary stages of design for energy systems for buildings. The system's process design model was based on an optimal search of a space of design options, which included all possible solutions, using heuristic knowledge about the problem. Problem knowledge captured in the system comprised five categories: domain knowledge,

constraint knowledge, procedural knowledge and algorithms for analysis and solution knowledge.

The domain, constraint and procedural knowledge were implemented in a hierarchical manner using production rules and a linked data structure of objects. In the model the energy system was decomposed into subsystems and components. The authors refer to the hierarchy produced as an approximation hierarchy. They say a hierarchy is only an approximation to a real world system. Thus a single component may perform more than one function, ie. a pump may be used to heat or cool a space. Likewise, a pump may use components, which are also used in other devices. Nevertheless, they added that the approximation hierarchy should be designed to correspond to the goal nature of the design process.

The key part of the design process is the selection of the subsystem, and at the next level, the selection or addition of components. This selection process may be modularised into goals and subgoals. The design process is modeled as follows:

- Formulation of building specification, including data gathering;
- Determination of the thermal characteristics of the building and its different zones;
- Determination of priority factors (constraints) for the building; and
- Selection of a basic system type.

The selection process starts with selection of a basic system for each zone of the building based on the cooling and ventilation requirements of the building. This is followed by the selection of a system subtype, descended from one of the basic types. A form of evaluation is completed for the subtype and a *Relative Benefits Factor* (RBF) value is computed.

The authors describe a hierarchical testing and elimination scheme, which they have incorporated in the system to prune the size of the search tree. They say that each system class is expanded into its various subtypes at each level in the hierarchy. These subtypes inherit the attributes of the system from which they are descended and, as each subtype is expanded, the knowledge about its attributes becomes more specific.

Each system class is then evaluated and an RBF is computed for each level of the hierarchy. The *total RBF* of each system subtype is evaluated and subtypes with low RBFs are pruned from the search tree. The search proceeds down the tree until the leaf nodes are reached and finally the system with the highest RBF is selected. The procedure is repeated for all zones in the building and a number of different system subtypes may be selected.

The authors found that production rules were ideal for representing surface type knowledge but that the rules became very complex as the level of knowledge becomes deeper. They noted that the frame-based representation they used was more suitable for representing the deep knowledge in their system.

They also recognised the feasibility of using expert systems in simulations, where they could be used both as preprocessors of data and postprocessors of output data.

- **A System for the Design of Fixed-Steel Offshore Jacket Structures**

Soh, C.K., & Soh, A.K., (1988) describe the Interactive Preliminary Design of Fixed-Steel Offshore Jacket Structures System (IPDOJS), which is designed to select an appropriate basic structural configuration for fixed-steel offshore jacket structures. The system does not operate as a standalone system; it is incorporated into the Intelligent Structural Design System (ISTRUDS). ISTRUDS couples a knowledge engineering environment, known as the General Engineering Problem Solving Environment (GEPSE) with a conventional

structural design system, STRUDS, so as to utilise the encoded expert knowledge in the GEPSE system to guide the numerical processing procedures in the latter.

IPDOJS is confined to the routine design of the oil and gas related platforms, which are typically the four-, six-, and eight-legged fixed-steel offshore jacket structures, and which are without skirt-piles for water depths less than 100m.

IPDOJS has a knowledge base containing objects and rules. The objects represent the basic components and geometry of the jacket structures. They are stored in the 'objects' segment of the relational database provided in the GEPSE environment. The portion of active process knowledge in IPDOJS is represented in production rules and stored in the 'rules' segment of the central knowledge base. Algorithms for numerical computations are also embedded within the consequence parts of the production rules, and they generally apply only to the 'if' conditions of the relevant production rules.

The system selects appropriate basic structural configurations, after first solving the following subproblems:

- Select application dimensions for the jacket structures;
- Decide upon the number of jacket legs to use;
- Determine the height of the structure;
- Decide upon a suitable batter for each jacket leg;
- Determine the required number of horizontal framings;
- Compute the dimensions for the jacket bottom; and
- Select appropriate bracing systems for the horizontal framings and the vertical bents.

These tasks must be completed in the sequence shown above, as the output from the first four steps is required as input to step 6. In order to keep control of the processes the system has a set of goal bases and subgoal bases. The goal list serves as a checklist to verify that all the subtasks have been competed before terminating the inference process.

- **A Design System for Reinforced Concrete Buildings**

CIB - Dresden, LAP – Stuttgart (1995), describe PRED, an intelligent prototype, which was developed as part of the Computer Integrated Object-Oriented Model for the Building Industry, (COMBI), project. This project spanned 3 years, from October 1992 to December 1995, and was part of the European Union Esprit Project. COMBI involved 6 separate teams, including software developers, construction engineers and consultants, it was established to develop a prototype of a computer integrated environment for cooperative design and concurrent engineering in the building industry. This prototype was to facilitate the development of intelligent systems for computer assisted engineering.

The scope of the COMBI project system covers the structural foundation and architectural design of reinforced concrete industrial and office buildings in the feasibility, preliminary design, sketch, outline and detailed design phases.

The PRED system provides an interactive, integrated and intelligent tool for assistance in the preliminary design of the bearing structure of reinforced concrete industrial buildings. It has been implemented as an independent submodule of the COMBI system. The system has an object-oriented design and uses artificial intelligence techniques to derive suggestions for appropriate design steps and system solutions in a given design context.

The functionality provided by PRED includes: hierarchical design support, 2D visualisation, integration of analysis tools for verification, an interface to the STEP product (ISO 1992), a

modeling integration framework, cost estimation and automatic re-configuration in the case of changes.

The PRED system helps to derive early conclusions on the structural system of a building and the dimensions of its main members, allowing subsequent evaluation of a proposed solution against present constraints, rules and criteria. Its display screens allow the designer to develop the structural bearing system interactively. It uses the knowledge base to try to interpret the path of design decisions and make suggestions for further steps and solutions.

The engineering knowledge represented in PRED includes structural rules, predimensioning rules, knowledge about critical ranges of dimensioning, minimum dimensions of structural elements and rules for the combination of structural elements to a consistent bearing structure. The design system is intentionally not developed as an expert system. It rather combines methods of artificial intelligence with CAD-technology to form an interactive intelligent tool. CIB-Dresden say that the goal of the system is to overcome the often-insufficient transparency and difficult operation of pure expert systems, while at the same time enhancing computer assisted design with knowledge-based components. The implemented prototype demonstrates the use of advanced CAE-technology to support high-level interaction between the designer and computer based modeling and reasoning in the field of preliminary structural design.

The COMBI prototype includes several design agents, including PRED, which are knowledge based decision support systems. These design agents have application specific data models and COMBI has provided strategies to address communication and interoperability problems to enhance information transfer. COMBI also adheres to the ISO-STEP methodology and its conceptual product-modeling framework is created by using STEP modeling tools.

The scale of investment in the COMBI project reflects the economic significance of computerised design systems and illustrates the increasing sophistication of systems being developed.

4.4 Multiple Selection-Development (MSD)

Maher (1984) and Harty (1987) both described standalone systems designed to assist with preliminary structural design. By contrast Sause et al. (1991) considered integrated systems, which comprised several otherwise standalone systems. In their paper they discuss integrated design computer systems, which provide support for several aspects of building design. They say such systems should "organise, process, manage and communicate the multi-disciplinary information associated with complex design problems." They describe the Integrated Building Design Environment system, which was documented by Sriram and Groleau (1989), and which uses *blackboards* to coordinate and integrate systems, which support architectural design, structural design and construction planning.

Sause et al. proposed a framework of concepts and tools to support an integrated structural engineering design system. Their framework required well-defined models for engineering design that would serve as a theoretical basis for integrated design systems; and suitable approaches for implementing these models. They anticipated two types of models for engineering design; the first type was a product model, which represents the design entities and the relationships among them; and the second type was a process model, which models design activities.

They also described the capabilities of the object-oriented approach, which would facilitate the implementation of these models and which would further act as a unifying concept between the product and process models. They described the Multiple Selection-

Development (MSD), model, which they refer to as a process generalisation model, which represents design knowledge in a manner similar to that of the DOLMEN system. The MSD model organises the design process into:

- Selection subproblems, which involve identifying, ranking, eliminating and selecting from a number of competing alternatives, and
- Development subproblems, which involve design and evaluation of a single alternative.

They noted that, at the time of their report, design product models had not been well defined enough for their MSD model. In their report they list several advantages to be gained by using the object-oriented approach for implementing product and process models, however, they also noted that the approach provided little aid in developing such a model. The advantages they list include:

- Clean mapping. They asserted that entities and activities in a model could correspond directly in the implementation.
- Enforced modularity. They explained that each object is a well defined software module, which is coupled to other objects though, a well-defined message interface. This modularity produces a uniform implementation of entities and activities of different types.
- Data abstraction. They noted that object-orientation provides a mechanism, whereby the internal details of an object can be changed, by changing instance variables, without changing the object's behaviour or message interface.

They also pointed out that there were difficulties in using object-oriented techniques and that the advantages could only be realised if sufficient effort was made in defining the instance variables, methods and message interfaces for each object class. They emphasized

that the objects must be carefully identified and the desired states and behaviours clearly defined and that these tasks could be difficult.

Year	Name of System	Area of Preliminary Structural Design	Authors	Approach Used	Software
1997	No name given	Building facades	Karhu, V.	Development of product model and data exchange with AutoCad	EXPRESS & AutoLisp
1996	ODESSY	Reinforced concrete regularly shaped multi-storey buildings	Suresh, S., & Krishnamoorthy, C.S.	Integrated system, includes genetic algorithms	Not included in report
1996	TALLEX	Tall regularly shaped multi-storey buildings	Sabouni, A.R., & Al-Mourad, O.M.	KBES	EXSYS Professional & QBasic
1996	No name given	Building foundations	Bravo. G., Hernandez, F. & Martin, A.	KBES	ART-IM
1995	C-LIFT	Offshore heavy lift system	Lim, C.K., Choo, Y.S. & Nee, A.Y.C.	Knowledge base and case based reasoning	ART*ENTERPRISE
1994	DIANAS	Reinforced concrete frame walled buildings	Turk, Z., Isakovic, T., & Fischer, M.	KBES	Prolog & Object Store
1993	No name given	Tall regularly shaped multi-storey buildings	Jayachandran, P., Tsapatsaris, N & Goldstein, B.R.	Not described	Prolog, Fortran, C and C++
1991	Hall Layout Module	Reinforced concrete halls	Yehia, N.A.B., et al	Transformational	Prolog
1990	Building Envelope Design System	Building envelope systems	Bedard, C. & Gowri, K.	KBES	KnowledgeCraft & Escher & Common LISP
1990	SPRED-1	Space grid structure	Gan, M., & Liu, X.	Neural network	Not included in report
1988	TREE	Rectangular halls	Borkowski, A., Fleischman, N. & Bletzinger, K.V.	Intelligent access to data base of previous designs	PDC-Prolog & dBaseIV
1988	IPDOJS	Fixed offshore jacket structures	Soh, C.K., & Soh, A.K.	KBES	GEPSE General Engineering Problem Solving Envirment
1987	DOLMEN	Regularly shaped multi-storey buildings	Harty, N.	KBES	KEE
1987	STRUPLE	Structural configurations	Zhao, F.	KBES	OPS83
1987	IDABES	Building energy systems	Doheny, J.G. & Monaghan, P.J.	KBES	OPS5 & Fortran
1986	LOCATOR	Location of lateral load resisting systems within a 3-D grid	Smith, D.F.	KBES	KnowledgeCraft & CRL
1985	FLODER	Floor systems and framings	Karakatsanis, A.		OPS5 & LISP
1984	HI-RISE	Tall regularly shaped multi-storey buildings	Maher, M.L.	KBES	PSRL

Table 4.1 Knowledge based Structural Design Systems

4.5 Issues in the Development of Object-Oriented Design Systems.

4.5.1 Object-oriented Languages – Procedural and Declarative

The second objective of the literature survey was to gain an understanding of the issues involved in the application of object-oriented techniques in the development of knowledge-based systems designed to support preliminary structural design. This understanding was required in order to:

- Form a preliminary assessment of the usefulness of object-oriented techniques in this area; and
- Anticipate problems likely to occur in the implementation of the selected approach to providing computer support in the Kappa-PC object-oriented knowledge engineering environment.

In Computer Science literature, computer languages are divided into two overall classes: procedural and declarative. Procedural languages, such as C and BASIC, require the programmer to specify the procedures or algorithms the computer has to follow to accomplish the task. In contrast declarative languages, such as SQL, remove this requirement from the programmer, who is merely required to describe a set of facts and relationships, usually stored in tables. A user of such a declarative language may then subsequently query the system to get a specific result.

Webster's New World Dictionary of Computer Terms, (1997) describes an object-oriented program language as a

> "non-procedural (declarative) language in which program elements are conceptualised in objects that can pass messages to each other by following established rules"

A procedural language, such as C, may allow the programmer to implement certain object-oriented programming techniques, however, the C++ language, which was originally developed as an extension of C, is designed to support object-oriented programming, as well as procedural programming. In this context C++ is described as a hybrid language. There are also languages like Smalltalk, which are described as pure object-oriented languages, which provide full object-orientation, going beyond the mere support of techniques.

Procedural computer programs contain data structures and algorithms. These data structures are restricted to basic data types such as integer and character and simple abstract data types such as arrays and queues. The programs are organised in a modular fashion, where the modules represent functions or procedures, which are abstracted from the problem domain. In contrast an object-oriented program consists of objects, which contain both procedures and data. Object-oriented programs are organised around hierarchies of objects, which reflect the relationships of real world objects. The basic mechanisms of object-orientation include message passing and the invocation of object methods and inheritance. In general object-oriented languages also provide abstract data types for use in their object methods, however, the dominant type is the class, which allows the abstraction of real world objects.

Object-oriented analysis and design assists the system developer to model problem situations in terms of real world objects and events. It should be pointed out that a given system development, may combine a conventional or structured design process with a subsequent implementation on an object-oriented hybrid object-oriented language. Likewise an object-oriented analysis and design may be subsequently implemented in a non-object-oriented language.

During this study the writer had of necessity to learn how to program the Kappa-PC development application using the KAL language. While Kappa-PC can fully support object-orientation, it is not exclusively object-oriented, having characteristics of a hybrid

environment. This hybrid functionality made it quite difficult for the writer to learn how to use Kappa-PC effectively. A significant part of the finished prototype design system consists of KAL functions, which perform the algorithmic programming required to simulate design synthesis, detailing and testing and design evaluation. While the structural design objects and subsystems could be represented as objects in the object-oriented paradigm, the writer had to resort to various conventional or structured programming techniques, including program flowcharts to design the algorithmic program functions. The writer also had difficulty in deciding how to represent the multitude of intermediate or transitory variables used in the detailing calculations. The writer made some use of the local variable syntax provided by Kappa-PC, but this was difficult to use and the writer eventually had to create classes to hold these variables as class attributes. As the Kappa-PC classes have global scope, this made algorithmic programming even more difficult and made it essential to keep track of variable names. This was also made difficult by the limitation of identity name size to 31 characters.

4.5.2 Frames and Objects - Similarities and Differences

Fikes and Kehler (1985) state that the fundamental observation arising from work in artificial intelligence has been that:

> "Expertise in a task domain requires substantial knowledge about that domain. The effective representation of that domain knowledge is therefore generally considered to be the keystone to the success of artificial intelligence programs."

They add that if a knowledge system is to use domain specific knowledge then it must have a language for representing that knowledge. Historically frame based languages have been very important developments in the application of knowledge representation. These languages have complemented and extended the production rule systems, found in the

original artificial intelligence systems. Furthermore, with the availability of object-oriented languages and integrated development environments developers have used objects (classes and instances) to implement and enhance frame based representation schemes and to combine frames and production rules to create hybrid systems.

Early artificial intelligence programs were mainly based on production rules. Hayes-Roth (1985) stated that rule-based systems, which used pattern/action decision rules, had played an important role in the development of intelligent software; however, these rule-based systems were without several features that would make them more suitable in a general computing approach. In particular they lacked a theory of knowledge organisation, which would facilitate the scaling up of systems without corresponding loss of intelligibility. Furthermore, rule-based systems were difficult to manage and extend. According to Fikes and Kehler (1985) production rules were an effective way, at that time, of representing domain-dependent behavioural knowledge in knowledge systems. They said that production rules could be easily understood by domain experts and had sufficient expressive power to represent a useful range of domain-dependent inference rules and behaviour specifications. However they also noted that, by themselves, production rules did not provide an effective representation facility for most knowledge-system applications. In particular, their expressive power was inadequate for defining terms and for describing domain objects and static relationships among objects. Furthermore rule based systems could become very complex if they were scaled up.

Minsky (1975) introduced the frame concept as a means of representing domain knowledge in a program. Since then frame oriented representation had been used to code knowledge in systems where the attributes of the projects were very complex. In these systems the frames were organized into a taxonomy. Each frame contained a set of slots, representing the attributes. Frames were appropriate for defining terms and for describing objects and

taxonomies of classes and subclasses and their relationships. However, Fikes and Kehler (1985) added that although frames could describe the objects, they could not describe how the objects were to be used. They then described how domain-dependent behaviour could be attached to frames in the form of methods or procedures written in some other programming language such as Lisp. However, they added that further enhancement to the frame representation scheme was needed to provide domain-dependent inferential reasoning, decision-making and control. This enhancement was already available in the form of production rules, which could represent domain-dependent inference rules and object behaviour. In addition, software was becoming available, which would allow a developer to integrate production rules with the frame-based languages.

Fikes and Kehler (1985) and Kunz, Kehler and Williams (1984) both noted that the major inadequacies of production rules were found in the same areas, which were effectively handled by frames; the strengths and weaknesses of rules and frames were complementary to each other. Thus a system designed to integrate the two would benefit from the advantages of both techniques. Fikes and Kehler (1985) explained the advantages of integrating production rules and frames into a single hybrid representational facility and this has since led to the development of hybrid systems that combine the advantages of both component representation techniques. Both sources also assert the ability of object-oriented computing to provide a principle for unifying these representations and reasoning techniques thereby allowing the development of object-oriented, hybrid rules and frames systems, where the frames were represented by objects.

On the negative side Merritt (1998) noted that, whilst there is a synergy between objects and knowledge bases, objects are not frames. He also noted that, some implementations of object-oriented technology are procedural in nature, and do not resemble the logical programming commonly found in expert systems. Logical programming involves the

dynamic matching of patterns. Several other writers also support Merritt's note of caution; thus Moss (1991) and Gailly (1991) separately noted that:

- Compromises must be made for objects to work properly in a knowledge-based environment; and
- Difficulties arise when systems, which require logical programming techniques, are implemented in a procedural-programming environment.

Luger and Stubblefield (1993) support the notion that object-oriented programming improves on the ability of frames to provide a natural way to represent classes, inheritance and default values. However, they also add that frames behave passively in contrast to objects. Object behaviour is implemented as attached procedures, called methods, which are invoked through messages, sent to the object by the user or other objects. This contrasts with the behaviour of frames, which have passive monitoring. In effect, from a programming perspective, frames have global scope, whereas in accordance with the object-oriented principle of encapsulation, object variables are private and have restricted scope and object methods react to specific messages.

Objects have characteristics of both data and programs in that they retain state variables as well as being able to react procedurally in response to appropriate messages. They are active in the sense that the methods are bound to the object itself, rather than existing as separate procedures for the manipulation of a data structure.

Notwithstanding these apparent contradictions, overall this review found enough evidence to conclude that the object-oriented approach promised considerable advantages for organising and representing the knowledge required to design objects and for combining this knowledge with production rules, which simulated design practices.

4.5.3 Object-Oriented Modeling of Design Knowledge

Booch (1994) describes an object model design method as being one, which "lets us map our abstractions of the real world directly to the architecture of our solutions". Such a design method allows the designer to focus on both the objects and the operations in the model of the real world. In Henderson-Sellers (1997) this is referred to as the process of creating a model of the real system to be represented in the computer system. Booch (1994) cites Ledgard's model of this programming task, which is described as follows. In Ledgard's model the system developer models the real world problem, in this case structural design, in terms of a problem space. This problem space has real world objects, each of which has a set of appropriate operations and real world algorithms, or procedures for solving problems. These algorithms operate on the objects and provide transformed objects as results. Ledgard continues saying that, when a computer system is developed, the real world problem is modeled in the software.

Some of the references cited in section 4.2 described the application of object-oriented analysis and design techniques to model domain dependent design knowledge and some went further and described the use of object-oriented programming techniques to implement design systems. In total, these references allowed the writer to understand how object-oriented techniques had been used. Turk et al. (1994) argue that all engineering software operates on models and that the object-oriented paradigm is well suited for the modeling of engineering products and processes. In their paper they demonstrate that object-oriented analysis can be successfully applied to the modeling of an engineering domain, in this case a system for the analysis, design and proportioning of buildings. Their model contains a hierarchy of class objects, which is based on the same criteria for modeling space decomposition as the Standard for the Exchange of Product Model Data (STEP), ISO, (1992).

La Rota (1990) et al also propose a model-based approach as a framework for integrating various aspects of the structural engineering design process. They characterise the design task as an under-constrained and ill-structured problem solving process, which involves a search for solutions in a large space of alternatives. They developed an interactive design assistant to aid design engineers in viewing and analysing an evolving design at various levels of abstraction and from multiple viewpoints. This system was designed and implemented using an object-oriented approach, which provided the necessary mechanism for integrating multiple abstractions and perspectives.

They contend that model-based reasoning when applied to structural design implies the ability to derive system behaviour from a structural description of the system. The design of a structural system can be looked upon as a successive refinement of a functional description in a hierarchical manner, which continues until the functional elements are specific enough to be replaced by specific structural components, such as beams, girders and columns. As well as a functional perspective this iterative process incorporates a physical description of the objects being designed and a behavioural description. Both physical and behavioural descriptions evolve during the design as different components are selected, described, sized, located and finally tested and the structure is analysed and evaluated at successively more detailed levels of abstraction. They describe the advantages of the object-oriented approach, which allowed them to represent the required knowledge base in terms of the interacting components, which include:

- The integration of the functional, behavioural and physical aspects of the system in a hierarchy of structural components;
- Knowledge associated with spatial descriptions in terms of location and connections of substructures; and

- Knowledge associated with tools for analysis, evaluation and selection.

They contend that the object-oriented approach is also suited for the representation of topological and geometrical description in Formex [1] algebra. They also note that a Formex representation can be associated with an object that will respond to messages, perform the appropriate transformations and display itself in an efficient manner. They also reported that they had considered other computer languages and had found that their limited data structures and limited graphics capabilities would make the implementation task difficult.

Löfqvist (1993) has further expanded on this theme; he contends that the next generation of computer programs for use in building design must also be able to exchange information between each other. A significant part of this information relates to objects and includes the relationships between objects and their properties and functions. This generation of programs must also be able to exchange knowledge such as experience and heuristics. Löfqvist makes the point that structural designers traditionally use many different forms to record structural knowledge; he says these include sketches, drawings, flow diagrams, and results from analyses. He adds that the computer programs used for drafting and analyses are designed to process the appropriate data structures for these forms but because these forms are so different it is difficult to transfer data between the programs. Thus an analysis program would find it difficult to extract information from a fragment of a drawing used in a CAD program.

---

[1] Formex algebra provides a formal mathematical approach to the spatial description of the structural system.

Löfqvist (1993) proposed a *product model*, which he says, is a computerised model of a product component as a solution to these problems. He says the product model can be used to reflect all aspects of building design, however, it must be able to describe the overall structure as well as the structural components. Each subcomponent as well as the assembled components will be described as a separate product model. He adds that the product model must not only describe itself but also include information about the relationships between the component and the overall structure. He says these requirements make it necessary to use a hierarchical data structure to build such a product model and that the object-oriented approach is well suited to the modeling of hierarchical organisations of real world objects. He says this is usually done in an object-oriented environment by using a *frame* data structure.

Sause et al (1992) have concluded that two types of engineering design models can be determined, in addition to the *product model* they also propose that design activities should be modeled, and this requires the creation of a *process model*. They assert that these models are essential steps toward the development of integrated computer design systems, and that the object-oriented approach is attractive as an implementation tool and as a unifying concept between models.

Summary

This chapter has described the results of the writer's literature survey of systems, which support preliminary structural design. The most common approach noted in the survey was one that involved a knowledge based expert system, which implemented some form of hierarchical decomposition, which decomposed the design product into a hierarchy of object classes and subclasses, which were represented in a tree like structure;

Several researchers described engineering design applications, which applied object-oriented methods and reported that the object-oriented methods allowed easy representation of knowledge in different perspectives, in different levels of abstraction and in providing the appropriate links and relationships between them. They also reported that object-orientation also makes the task of the reasoning system easier in that the transition from one level or perspective to another can be performed easily, while maintaining overall consistency in the evolving structure.

However, other researchers have drawn attention to some of the difficulties, which may arise as developers combine the technologies.

The chapter also provided a brief outline of the evolution of knowledge representation in intelligent systems and introduced the concept of frame based reasoning. The writer also observed the relationship between frames and object-oriented objects

CHAPTER 5. An Object-oriented Software Methodology

5.1 Introduction

This chapter describes how a set of software development methods, which included object-oriented analysis and design techniques, was drawn together to create a structured, software engineering methodology. This methodology allowed the writer to address the primary purpose of the study, which was to investigate whether or not object-oriented analysis and design techniques could assist with the development of a knowledge-based design tool. The chapter also discusses lessons learned during the creation of this software development methodology.

The software methodology was designed to allow a developer to move from high-level abstract design down to low-level component design. Additionally it ensured that work products, their relationships, and the processes applied to produce them were clearly documented. The software methodology assisted the writer to develop:

- Prepare an initial problem scope statement;
- Document high level requirements for the new software;
- Specify detailed system requirements; and
- Apply appropriate object-oriented techniques to assist with the conversion of these requirements into an architectural design.

5.2 Developing the Methodology for the Project

During the initial literature survey, the writer attempted to determine what object-oriented analysis and design techniques were available and whether or not they would be suitable for use with the design tasks, envisaged. The research is summarised in section 5.3. The writer then arranged certain of these object-oriented techniques in an ordered series of steps to

create a development process. Initially the writer had determined that the whole software development project could be completed in two stages, these being object-oriented analysis and object-oriented design. However, it was soon realised that a preliminary high-level analysis stage was required to focus project objectives.

During the initial survey the writer had identified a high-level analysis process, which was part of an approach attributed to Checkland (1981) and which was described by O'Connor (1992). The process relied upon a simplified application of the Soft Systems Methodology (SSM) to provide a definition of the problem to be solved and a model of the system proposed as part of the solution. This model, which included the system tasks and associated procedures, was used as a framework for a high-level system description. The writer adopted O'Connor's approach and used it to complete the high-level analysis stage.

The high-level stage was intended to clarify the scope of the problem and to develop the objectives of the project. It was to identify and separate concerns and determine what areas of the problem were to be analysed. Analysis at this stage would determine what actions were necessary to fulfill the primary objectives of the project and would document these actions in the form of a *root definition* of the problem. This analysis would also establish a *relevant system*, which would form a proposed solution to the problem. At the same time a *conceptual model* would be built up to show the relationships between the activities required and to represent the system processes encapsulated in the root definition, which in this case were the steps required to create a structural design. During the high-level analysis attention was focused upon the information gathered during the literature survey, which covered the basic principles of preliminary structural design and the application of knowledge-based computing to support this phase of design.

After reviewing reports describing the use of several of these object-oriented analysis and design methods, the writer realised that these methods:

- Assumed that a requirement specification already existed; and
- Did not provide any techniques to produce such a specification.

This realisation made it necessary to include another stage in the proposed methodology. Thus after the completion of the high-level analysis a stage was proposed which would help to specify detailed system requirements for the new design tool. This requirement specification was to serve the following purposes:

- Communicate precisely, what the proposed design tool ought to do, from the user's perspective;
- List the functionality required from the relevant system for the design tool;
- Describe overall the approach, which was selected for the provision of computer support; and
- Document an understanding of the key aspects of the domain of structural knowledge, which were to be represented in the relevant system.

The writer then selected a set of methods to use in the requirements stage. These methods are described in section 5.6. The high level analysis and requirements stages were expected to establish a sufficiently detailed system requirement specification to allow subsequent object-oriented analysis to proceed. This would provide an 'object model' for the system and the following design phase would complete the system architecture, which would then be implemented in a fifth and final stage. During the final stage the writer would then complete the coding and implementation of a system prototype.

Figure 5.1 provides an overview of the software engineering methodology arranged for the project. The next section describes the analysis and design techniques selected for the project.

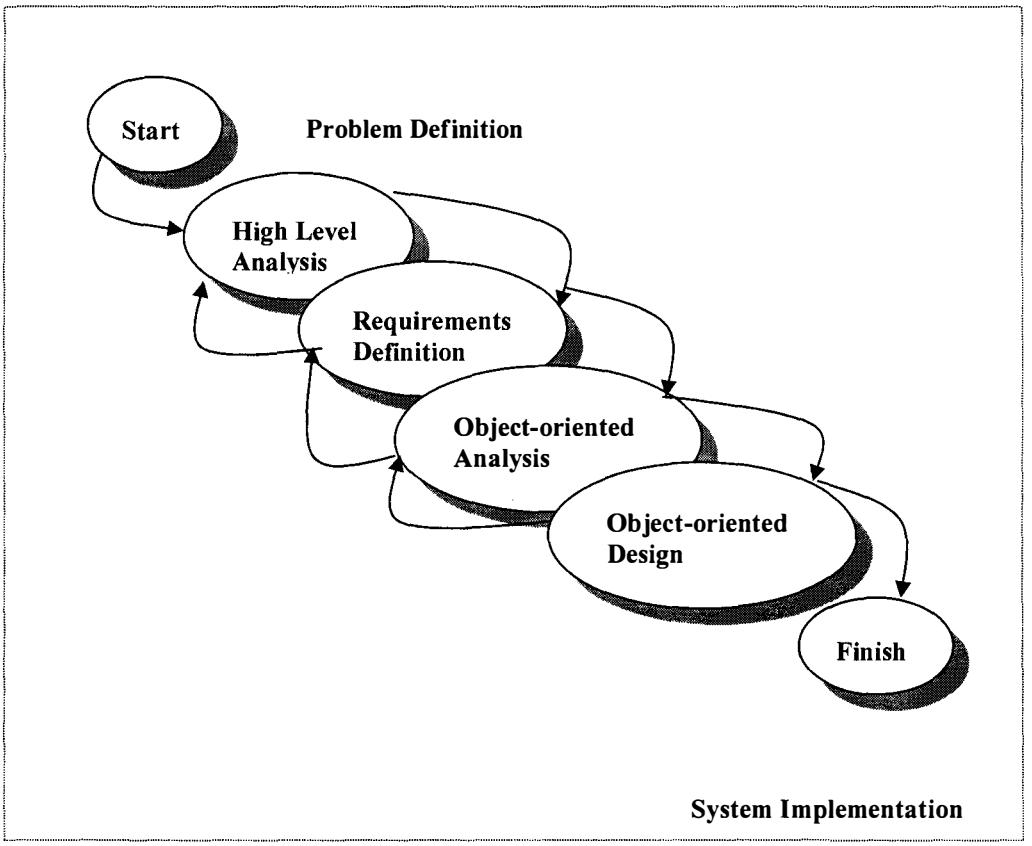


Figure 5.1 **Overview of the Software Methodology Created for the Project.**

5.3 Selecting Appropriate Object-oriented Analysis and Design Techniques

One of the primary objectives of the study was to identify and apply suitable object-oriented analysis and design techniques, rather than conventional ones. Furthermore, the techniques chosen had to accommodate the analysis and subsequent modeling of structural objects and complex design events.

The selection process was a difficult task. Several different approaches to object-oriented analysis and design were identified in the literature survey. In particular, Graham (1994, pp. 196-224) outlines the following object-oriented design methods: Booch's Method, Booch (1986) and Booch (1991), General Object-oriented Design (GOOD), Seidwitz and Stark (1986), Hierarchical Object-oriented Design (HOOD), HOOD Working Group (1989), Object-oriented Structured Design (OOSD), Wasserman, Pircher and Muller (1990),

Jackson Structured Design (JSD), Jackson (1983) and Case Responsibility and Collaboration (CRC), Beck and Cunningham (1989).

To compound the difficulty, Graham (1994, pp. 229-256) also describes a number of separate object-oriented analysis methods, which include the following: OMT, Rumbaugh et al (1991), Ptech, Martin and Odell (1992), OSA, Embley et al (1992), CRC, Wirfs-Brock et al (1990) and Coad/Yourdon, Coad and Yourdon (1990) and Henderson-Sellers, (1992).

Reflecting the trend for integrated methodologies, Page-Jones (2000) describes the Unified Modeling Language (UML), which seeks to provide an integrated approach to object-oriented analysis and design. This writer also noted that the UML has been taken up into various commercial, proprietary, software development methodologies. For example, the Rational Software Corporation's methodology, Rational Software (2000), provides a set of UML tools and techniques, which accommodate the complete software development life cycle.

To facilitate the search for appropriate methods, the writer found it necessary to revert to first principles, he therefore researched how the terms analysis and design, were used in the object-oriented paradigm.

The term 'analysis' was addressed first. The purpose of analysis is to describe a problem, ie. to formulate a model of the problem domain, it is concerned with what happens rather than how it happens, and it focuses on behaviour not form. The writer notes that a model is a complete description of a system from a particular perspective. By way of contrast, the purpose of design is to create an architecture for the evolving implementation and to establish common approaches that must be used with the disparate elements of the system. According to Booch (1991), software architecture encompasses the set of significant decisions about the organisation of a software system. This includes the structural elements

and their interfaces, the behaviour of these elements, as specified in a collaboration among them and the subsequent composition of these elements into larger subsystems. Booch adds that design should begin as soon as a model of the system has been created.

According to Graham (1994) analysis means the decomposition of problems into their component parts. In conventional computing, analysis is understood to include the specification of user requirements and the system's structure and function. Analysis does not cover implementation. Furthermore, the high-level, strategic and business analysis is usually separated from system analysis. Object-oriented system analysis also contains an element of synthesis. It involves abstracting user requirements and identifying key domain objects, which is followed by the assembly of these objects into an object model that will support physical design at some later stage. Graham says that the synthetic aspect arises because the analysis is applied to a system, and this requires the analyst to impose a structure on the domain.

Graham (1994) says that object-oriented analysis must describe 3 key aspects for a proposed system:

- Data; objects and/or concepts and their structure, which are described in analysis in the conventional paradigm by entity relationship (ER) diagrams;
- Process, which is described in conventional analysis by data flow diagrams, (DFDs) or activity diagrams; and
- Control of system behaviour, which is described in conventional analysis by state transition or entity life cycle diagrams.

He adds that object-orientation combines two of these aspects, data and process, by encapsulating local behaviour, in the guise of object methods, with data. However, Graham (1994, p. 228) also says, that the control aspect for a proposed system is more difficult to

integrate and that in several of the approaches he described, control in the form of rules and/or constraints, appears to be accommodated as an afterthought.

The primary purpose of design is to decide how the system will be implemented. During design strategic and tactical decisions are made to meet the functional requirements of the system. The term design implies a form of architectural modeling, which comprises logical and physical design. Design adds detail, precision and implementation dependent features to the model created during analysis. Object-oriented design methods have the following basic design steps:

- Identify the objects and their attribute and method names;
- Establish the visibility of each object in relation to other objects;
- Establish the interface of each object and procedures for exception handling; and
- Implement and test the objects.

According to Henderson-Sellers, (1997, p. 69), one can identify three distinct phases in the traditional software life cycle: analysis, design and implementation. In the analysis phase, the problem is examined in terms of user requirements and it is set in the problem space. Usually, it is agreed that the transition from analysis to design occurs when the project moves into the solution space, to provide the software solution. The design phase is a phase of progressive decomposition, where more and more detail is provided. He adds that the stage after design is implementation, where the program is written, tested and put into use.

The traditional life cycle is a series of steps with gaps between them. The steps are well defined and are associated with clearly identified deliverables. The deliverable, output by one step, becomes part of the input for the next step. Henderson-Sellers contrasts this well ordered life cycle with the object-oriented life cycle and notes several differences. He says that object-orientation supports a seamless transition from phase to phase and this makes it

difficult to pinpoint where one stage ends and another begins, likewise it is difficult to detect the point at which the deliverable should be achieved. In the object-oriented life cycle the project is grounded in the user/real world and the user requirement analysis and design specification stages are highly merged. Focus is placed on classes and not on systems, and clusters of classes may be developed. Furthermore, the project status of individual clusters will not necessarily be synchronized.

Henderson-Sellers provides a rationale for separating object-oriented analysis and design as follows. He says that in the analysis stage, which he refers to as conceptual modeling, the developer is trying to represent an information system design. Thus during analysis the developer is creating a model (of the human perception of) of the real system to be represented in the information system. By way of contrast, design is the process of creating a model of the information system (artifact) to be constructed based upon the model of the real system.

In contrast to this view, Graham (1994, p. 194) says that object-oriented methods include methods for analysis and design, and that the two stages overlap. He says that analysis and design, at least up until the logical design stage, can't be distinguished as separate activities, in the way they are separated in conventional methodologies. This lack of separation is most clearly manifested when systems are prototyped. When prototyping takes place, the development process goes through an iterative cycle of overlapping analysis and design stages.

After considering the discussion above, the writer decided that the analysis stage would require the creation of a model of the problem area, which would represent what the new system should do, whereas during the design stage, appropriate strategies to implement this model would be developed. The information describing what the new system should do was to be provided in the requirements stage.

After distinguishing the terms, analysis and design, the writer went on to select appropriate analysis and design techniques from the wide range available. This task was facilitated by reference to Cross (1996), who has described a simplified object-oriented analysis and design process, which is based on the work of Wirfs-Brock et al. (1990) and Rumbaugh et al. (1991). This process was adopted by the writer, who used it as a framework to guide the object-oriented analysis and design phases of the project. The key steps in this process, as described by Cross (1996), are set out below:

- Understand the problem. Gain sufficient understanding to be able to begin to solve the problem;
- Identify the objects. Group real life objects that exhibit identical behaviour into classes;
- Determine the responsibilities of the objects, which are to be represented by object methods;
- Determine the associations between objects, including the links between them and the messages they send;
- Determine the attributes, slots and methods, contained by the objects; and
- Complete the system design by organising the objects into a hierarchy and establishing the inheritance links required.

Cross (1996) adds that the steps in this process should be facilitated by the use of modeling and diagramming techniques based on those described by Rumbaugh et al. (1991) and Embley et al. (1992). This writer distinguishes diagrams from models, noting that the diagram is a view into a model from a particular perspective and it provides a partial representation of a system. Cross describes three types of diagrams:

- Object model diagrams, which describe the static structure of the objects in a system and their relationships. In this type of diagram, the objects, including classes, subclasses and instances are drawn as rectangles. Different types of straight lines with variously shaped arrow heads are used to represent different types of associations of classes;
- State transition diagrams; these are also known as dynamic models and are used to describe the control aspects of a system. They are used to reflect changes in object states, which are caused by system events, which may be effected by interactive functions, such as monitors and demons. These diagrams use rectangles with rounded edges to represent objects' states and connecting lines to represent events; and
- Functional diagrams or data flow diagrams, which describe computations, processes, non-interactive functions and data flows within a system. They use rectangles to represent objects, ellipses to represent processes and the '==' symbol to represent stored data.

To summarise the writer referred to Graham (1994) to gain an understanding of the various object-oriented analysis and design techniques available before selecting the ones to be used in the project. He then referred to Cross (1996) to clarify their appropriate use. The six-step process, described above and attributed to Cross (1996), was then used as a framework, within which to apply modeling techniques appropriated from Rumbaugh et al. (1991) and Embley et al. (1992). These modeling techniques were applied to develop an object model, state transition and functional diagrams and several informal diagrams.

5.4 The Software Engineering Methodology

The software engineering methodology was finally arranged as follows:

- Stage 1 - High-level, problem/situation analysis; which effected a simplified application of the Soft Systems Methodology and which produced an initial problem scope statement;
- Stage 2 - Requirement specification. Structural design information assembled from Lin (1981), Maher (1984) and Harty (1987) was reexamined and key elements documented. The developer obtained an understanding of the domain and identified requirements for the system, which included a list of design activities;
- Stage 3 - Object-oriented analysis. This stage created a model to incorporate the requirements, previously identified; and
- Stage 4 - Object-oriented design. The writer then completed the software design process by preparing the architectural model for the new system, keeping in mind the requirements of the Kappa-PC application development system.

Figure 5.2 illustrates the software engineering methodology and the individual stages in the project are described in the following sections.

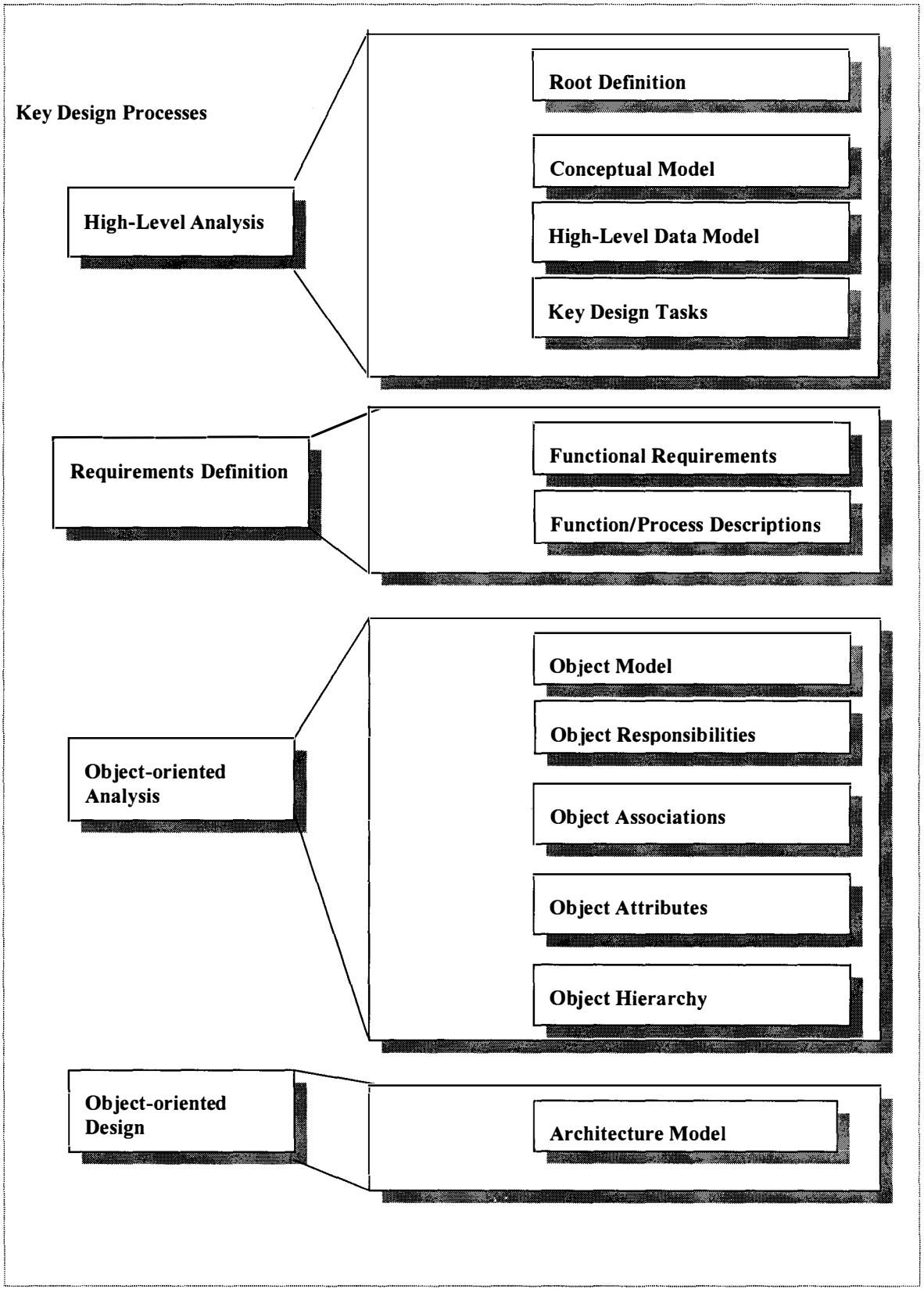


Figure 5.2 **Detailed View of the Software Engineering Methodology**

5.5 High-Level Analysis Stage

The high level analysis was completed to:

- Facilitate an understanding of the problem situation;
- Clarify the scope and objectives of the project;
- Take into account the different perceptions of the problem situation, which were expressed by the various researchers whose reports were consulted; and
- Separate concerns and identify areas where further analysis and development might be required.

The high-level analysis stage was designed to follow the process described in O'Connor (1992). The two main activities completed in this stage were problem analysis and information analysis. The problem analysis activity was designed to develop a relevant system for the problem of the structural design problem solving process and root definition describing the relevant system and a conceptual model to represent the design tasks associated with the root definition. During the information analysis phase, the information requirements associated with the design tasks were identified and a high-level data model developed.

Investigation of the problem situation was initially expected to develop a number of problem situation descriptions reflecting the differing perspectives of the preliminary structural design process of the different researchers referred to. From these descriptions, it was then possible to develop a root definition of the problem.

According to O'Connor (1992) a conceptual model is created as a logical expansion of the root definition to represent the minimum set of activities, which are needed to define what a proposed system was meant to achieve at an overall level. This model was to be the basis

for enquiring into the domain of structural design knowledge in order to produce a framework for the specification of an information system. In effect the conceptual model would create a system model of the preliminary structural design domain. Information flows and processing requirements were to be established to extend the conceptual model.

Relying mainly on references to Lin (1981), Maher (1984) and Harty (1987) the high-level design stage was also expected to establish the key design issues, information requirements and design tasks, which the new design tool system was to support. It would also establish the information model to describe the necessary design information and relationships. From a system design perspective, the major objectives of this phase were to define the:

- Hierarchy of functions for the system, which was to support the preliminary structural design tasks;
- Likely menu and screen layouts, by which the user would operate the new system;
- Strategies for the integration of the system and its external data sources;
- Representation of the structural design product and process models, underlying the system; and
- Output process, whereby design information would be made available to the user.

This phase would create a series of diagrams for the individual design tasks and corresponding information flows between them. Furthermore, the conceptual model together with the information flows and processing requirements, also established, would allow the requirements stage to proceed.

5.6 Requirements Specification Stage

According to Perry (1995), there is no one right way to specify requirements; the area is a difficult one to work in and has plagued the IT industry for decades. Perry provides some

guidelines to assist with the correct specification of requirements; he says requirements should:

- Identify the necessary functions to be executed;
- Identify the information required by these functions;
- Be comprehensive; and
- Be unambiguous.

In addition he says that requirements must state the problem to be addressed and identify any implementation constraints and performance characteristics. Perry describes a seven-stage process for specifying requirements:

- Identify needs:
    1. Determine the problem or objective;
    2. Determine the desired characteristics, success factors and assumptions made.
- Analyse Requirements:
    1. Define the scope;
    2. Identify the rules, processes and data involved;
    3. Determine the task the system is to perform;
    4. Identify all data and processes required; and
    5. Uncover business details, rules and policies inherent in the processes and data described above.

In this project, the requirements stage was designed to expand upon the conceptual model of the problem situation. It was also designed to incorporate material assembled during the literature survey, which included research notes regarding design principles and methods

and notes taken from detailed reports of the knowledge-based design tools developed by Maher and Harty.

The writer completed a detailed analysis of the reports of Maher (1984) and Harty (1987) to enable him to draft a list of functional requirements for the new system. In particular the writer documented the simulation of the major structural design tasks in the two systems. The writer then created a description for each design task, in order to build up the conceptual model for the new system.

5.7 Object-oriented Analysis Stage

Object-oriented analysis was completed to identify and model domain objects and their behaviour, structures and users. The writer ensured that this analysis covered the relevant structural design rules and processes. An overall model and a series of smaller object class diagrams were prepared and these became progressively more detailed as analysis and design proceeded.

The model of the system served as a partial solution for the requirement specification described in the previous section and it was accompanied by a collection of other informal diagrams and notes, which were intended to capture enough information about the relevant system to allow design of the new system to proceed.

During the analysis stage the writer re-read the reports of Maher (1984) and Harty (1987), along with the requirement specification and accompanying notebooks. He then applied the six step object-oriented analysis and design process, described in section 5.3, as a framework, to guide the analysis and to ensure that the problem was fully understood and that the required diagrams were created.

5.8 Object-oriented Design Stage

This stage was concerned with how to implement the object model in the Kappa-PC environment. During the design stage, considerations of various underlying application development system objects, such as the Kappa-PC inferencing mechanism and the user interface components were factored into the analysis model. In addition, components of the object model were reviewed for technical feasibility to ensure that they could actually be implemented on the Kappa-PC platform. The products of design modeling included object diagrams and message passing schemes. The design stage was enhanced through the creation and modification of a series of system prototypes, which were written after some preliminary analysis and outline design. The techniques used in this phase were:

- Identify names for the objects and their attributes and methods;
- Establish the visibility of each object in relation to other objects;
- Establish the interface for each object and its exception handling, where required; and
- Prepare for the implementation and testing of the objects.

It was intended to use only one overall model in the two object-oriented stages and it was expected that the transition from analysis to design would be straightforward.

5.9 Problems Encountered During Development of the Methodology

The writer encountered several difficulties in drawing together appropriate methods and techniques to create an object-oriented software design methodology, which would be suitable for use with knowledge-based systems; this section describes the more significant ones.

- **Selection of Techniques**

It was initially anticipated that it would be straightforward to establish an appropriate object-oriented analysis and design methodology to complete the development project. In

practice the writer found that it was difficult to select a set of suitable object-oriented analysis and design methods because there was such a large range of approaches described in the literature. The writer eventually relied upon a simplified framework and set of techniques, described by Cross (1996), which allowed him to complete the project.

- **Lack of High-Level Analysis and Problem Solving Tools**

Graham (1994) notes that unfortunately, the high-level, strategic and business analysis stage is usually separated from the object-oriented system analysis. This writer encountered the same problem and he was unable to identify any object-oriented techniques, which would have provided assistance with the initial high-level analysis stage. The writer finally used a process based on Checkland (1991) and described in O'Connor (1992),

- **Lack of Object-Oriented Input to the Requirements Document Stage**

During the development the writer discovered that object-oriented analysis and design techniques required the prior preparation of a requirements specification. He therefore added a further preliminary stage, during which, the necessary requirement specification, was prepared. The writer also originally understood that the object-oriented analysis and design stages were intended to produce a model of the "real world" problem situation. In effect the real description of what happened in the real world was produced 'outside of the object-oriented paradigm".

- **Problems with the Object Model**

The writer had planned to use object models, state transition diagrams and functional diagrams to model the system in the object-oriented analysis stage, and had therefore consulted various references, in order to ensure the proper use of these techniques. The writer found that these references had indicated that object-oriented analysis worked well with 'state transition machine'-like objects, which included ATMs and graphical user

interface components, which don't change their physical form. However, during this project the writer observed that it was difficult to create object hierarchies and state transition diagrams for:

- Transient objects, which either did not exist at the start of system operations; or which were created and destroyed during operations; and
- Objects, which changed identity, becoming subsumed into other accumulation type objects during operations.

The writer also had difficulty in incorporating the system's rule-base into the object model. He was also unable to locate any references, during the literature survey, which might have assisted in this area and thus resorted to the use of a 'back box' to represent the rule-base in the object model.

Summary

This chapter has described the creation of a software engineering methodology and several problems, which were overcome during the process.

CHAPTER 6. Development Project – Initial Stages

6.1 Introduction

The four-stage software engineering methodology created for this project was introduced in the previous chapter. This chapter describes the first two stages in its application to develop a knowledge-based design tool. Stage one produced a high-level analysis of the design problem and stage two a list of requirements for the new system.

Section 6.2 describes the high-level analysis stage and section 6.3 describes the requirements stage.

6.2 High-Level Analysis Stage

This section describes the high-level analysis stage of the software design process. The key inputs to this stage came from the following sources of information:

- Principles and recommendations concerning structural design, Lin (1981);
- Maher (1984) and Harty (1987), these reports describe two knowledge-based expert systems, which incorporated Lin's recommendations; and
- Sause and Powell's (1992) description of the Multiple Selection-Development (MSD) process generalisation model proposed for structural design.

The writer completed two main activities during the high-level analysis stage; these were problem analysis and information analysis. Problem analysis produced a root definition, which described the problem situation and a conceptual model, which represented the design processes associated with the root definition. The information analysis phase identified a set of generic information requirements, associated with structural design activities, and a set of specific information requirements, associated with the conceptual model. These information requirements were combined into a high-level data model.

The root definition of the problem was formulated as follows:

> ***Provide*** a knowledge-based system, which would assist an engineer by proposing preliminary designs for buildings in structural steel and reinforced concrete.
>
> ***Analyse*** the domain of structural engineering and a proven approach to the representation of this domain knowledge in knowledge-based systems.
>
> This analysis would enable the writer to ***produce*** the required ***models, data objects*** and ***algorithms*** to represent this knowledge in software. Subsequent implementation of this software would also allow the writer to ***determine*** whether object-oriented computing techniques were suitable for implementing the required system.

The outcome of the high level analysis indicated that assistance could be provided in the form of a knowledge-based expert system.

6.2.1 Conceptual Model and High-Level System Overview

A simplified diagram of the conceptual model is shown in figure 6.1. The conceptual model was created to extend the root definition and to represent the minimum set of activities needed to define what the proposed system was meant to achieve at a high-level. The writer also created a simplified overview diagram, shown in figure 6.2, to illustrate the preliminary structural design tasks and the associated system functions incorporated in the conceptual model. The writer used the same names for the design tasks: specification, formulation and evaluation, as did Harty (1987) Table 6.1 lists the high-level design tasks and their related functions, which must be supported by the system, in order for it to complete the required design tasks. The overview is accompanied by a summary of functions, which presents a high-level description of the function supported by the system.

Subtask	System Function
Specification	Check Design Parameters
	Input user requirements
	Review Evaluation Features
Formulation	Design Vertical Subsystem
	Get Assumed Sizes
	Set Initial Sizes
	Detail Vertical Subsystem
Evaluation	Evaluate Vertical Subsystem
	Evaluation Report
Formulation	Design Horiz Subsystem
	Detail Horizontal Subsystem
Evaluation	Evaluate Horiz. Subsystem
	Evaluation Report
Output	Design Report

Table 6.1 Preliminary structural design functions

6.2.2 High-Level Information Analysis

The writer also completed an information analysis to establish outlines for the information flows, which are required to support the system functions identified. These information flows are summarised in the high-level data model shown in figure 6.3.

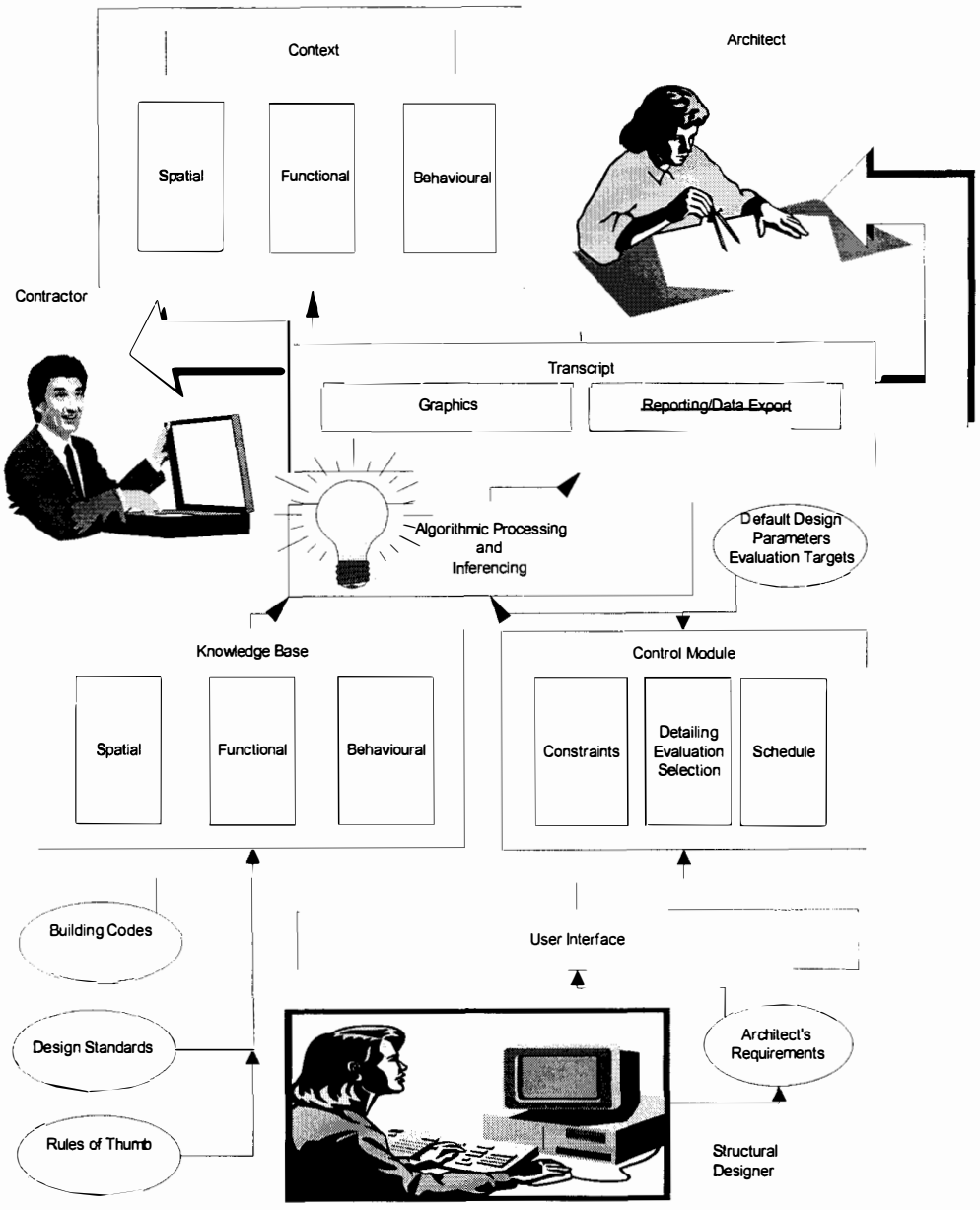


Figure 5.1 Conceptual Model

Figure 6.1 Conceptual model for preliminary structural design

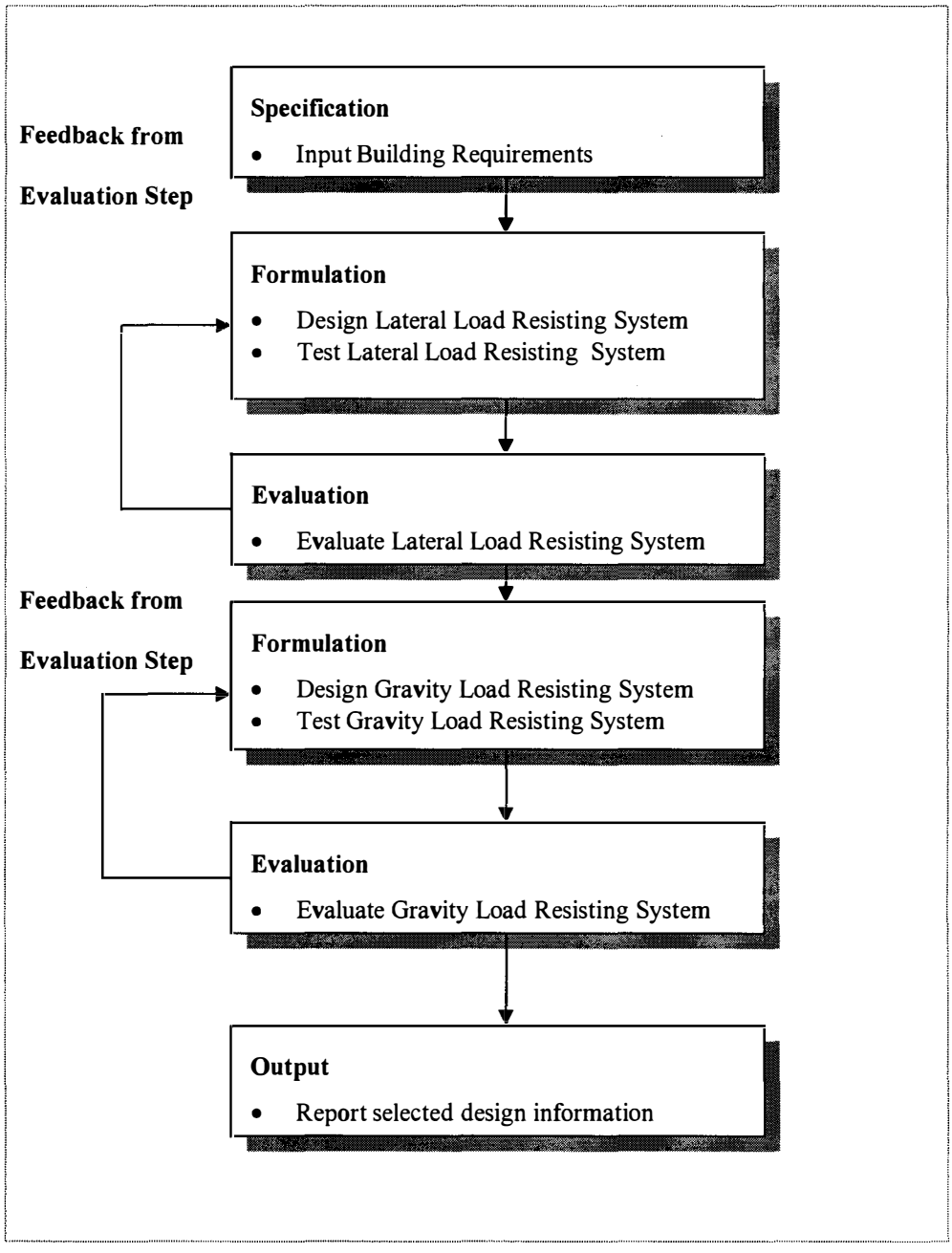


Figure 6.2 Overview of Preliminary Structural Design Processes

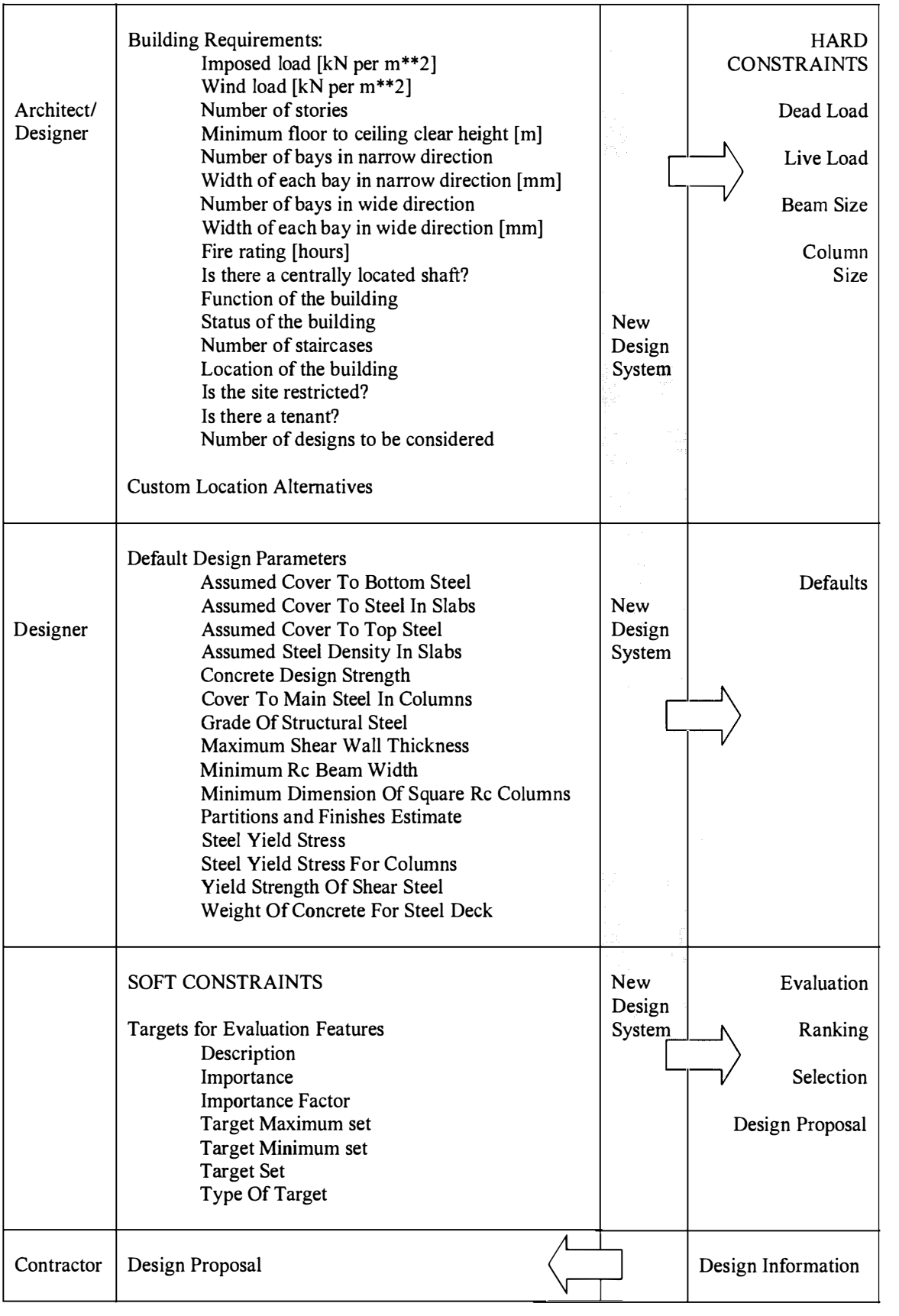


Figure 6.3 High Level Data Model for Preliminary Structural

6.2.3 Summary of Functions

This section covers documentation of the major functions identified for the design system. During the high-level analysis stage the writer identified twelve system functions. By way of an example, one of these functions, *Check_Design_Parameters* is described below. The remaining function descriptions are reproduced in Appendix A.

The requirement specification stage, which is described in section 6.3, followed on from the high-level stage and was designed to provide more information about each function, including the lower level processes within each function.

SUBTASK	SYSTEM FUNCTION
Specification	**Check Design Parameters** The Check Design Parameters function is required to allow the user to input and review the default design parameters in the knowledge base and to ensure that they are appropriate to the type of design envisaged by the user.

6.3 Requirements Specification Stage

In the requirements specification stage the writer analysed the key functions, identified in the high-level stage, which the system was to support. The writer identified and documented the lower-level processes within each function. These processes are listed in Table 6.2.

6.3.1 Summary of Functions and Design Processes

This section concerns the lower-level processes within each function. In order to document these lower-level processes, the writer analysed the reports of Lin (1981), Maher (1984) and Harty (1987), he then determined which design tasks the new system needed to be able to perform and then how a suitable set of system functions could be organised to support these

tasks. Once the functions were identified and organised the writer then determined what lower processes were required and then at a still lower level what activities were needed to ensure that these processes could be completed.

Subtask	System Function	Design Process
Specification	Check Design Parameters	Input/Review Design Parameters
	Review Evaluation Features	Input/Review Evaluation Features
	Input user requirements	Enter Design Specifications for Building
Formulation	Design Vertical Subsystem	Design Vertical-3D-Level
		Design Vertical-2D-Narrow Level
		Design Vertical-2D-Wide Level
		Design Vertical-2D-Material Level
		Design Vertical-2D-Narrow-Location
		Design Vertical-2D-Wide-Location Level
	Get Assumed Sizes	Set Initial Steel Deck Unit, Slab Depth, Beam Spacing, Slab Type and Floor Depth
	Set Initial Sizes	Find Steel Sizes and Wall Thicknesses
	Detail Vertical Subsystem	Detail Braced Frame Narrow Options
		Detail Rigid Frame Narrow Options
		Detail Shear Wall Narrow Options
		Detail Braced Frame Wide Options
		Detail Rigid Frame Wide Options
		Detail Shear Wall Wide Options
Evaluation	Evaluate Vertical Subsystem	Selection of Best *n* Options
	Evaluation Report	
Formulation	Design Horiz Subsystem	Enter Design Specifications for Building
		Design Floor Level
		Design Support Beams Level
		Design Intermediate Beams Level
	Detail Horizontal Subsystem	Detail Floor Level
		Detail Support Beams Level
		Detail Intermediate Beams Level
Evaluation	Evaluate Horiz. Subsystem	Selection of Best *n* Options
	Evaluation Report	Evaluation Report
Output	Design Report	Design Report

Table 6.2 The main design processes identified for the design tool system

During this stage the writer identified twenty-eight different key design processes, which were required to support the major functions of the new system. Due to the limited size of this report, the writer has not included details of these twenty-eight processes in the report.

However, Appendix A contains example descriptions for the processes required to support the key Design Vertical Subsystem function.

Summary

This chapter has described the high-level analysis and requirements stages of a software design project. The literature survey completed at the start of the high-level stage provided an understanding of a systems approach to preliminary structural design from Lin (1981). It also provided examples of the successful application of knowledge-based systems to support this approach to design from Maher (1984) and Harty (1987).

This survey research was used extensively in the high-level analysis described in this chapter. This analysis produced a root definition of the design problem situation and a closely associated conceptual model for a new design system.

The survey stage was also used in the requirements documentation stage, where particular emphasis was placed on the reports of Maher (1984) and Harty (1987). This stage produced a list of functional requirements for the new system. These functions were based on the list of design tasks, which accompanied the conceptual model. These tasks fulfill the design activities associated with the root definition and together they constitute an outline for a new system to address the problem of assisting with preliminary structural design.

CHAPTER 7. Development Project – Final Stages

7 1 Object-oriented Analysis

This chapter describes the final stages of the design project, during which the writer applied object-oriented analysis and design techniques to the functional requirement specification to create an object model to be used in the final design of the new structural design tool. Completion of these two stages involved re-reading the original reports of Maher (1984) and Harty (1987), along with the lists of design processes and user displays drawn up during the requirements specification stage.

During the requirements stage the writer had produced several notebooks with informal diagrams, flowcharts and fragments of pseudo code and these were referred to during the final stages. The six step object-oriented analysis and design process, described in chapter 5, was used as a framework to guide the analysis and to ensure that problems were fully understood and that the required diagrams were created. The following paragraphs describe how this part of the project was completed.

7.1.1 Identify the objects.

In this step, the writer created the object model by abstraction from the requirement specification. The work started with the identification of objects, which exist in the design environment, and with the subsequent grouping of those objects, which exhibited similar behaviour, into a hierarchy of object classes. Several groups of building system objects were identified, during the process. The structural design object classes, which make up the building hierarchy, are shown below in Figure 7.1, which shows one completed design with appropriate alternatives attached at each level.

At each level in the hierarchy, the design tool was required to provide an appropriate set of design options, from which it could generate alternatives for that level. In the model for the new design tool these options were represented by the *Alternatives* class, which was also organised in an object hierarchy, which is shown in Figure F.4, in appendix F.

Design Object Class	**Level of Abstraction**
Building_1	
Orthogonal_2D_Systems	**Vertical 3D Schematic Level**
Rigid_Frame_Narrow	**Vertical Structural Subsystem**
Rigid_Frame_Wide	..
Reinf_Concrete	..
RF_2_Narrow	..
RF_2_Wide	..
Reinf_Concrete_Slab	**Horizontal**
2_Narrow_Beams	**Structural**
Intermediate_None	**Subsystem**

Figure 7.1 Object classes in a completed design, which is displayed hierarchically.

From the research work done earlier in the project, it was realised that structural engineers have a large range of possible layout options, at the vertical 2D level, for their structural schemes. Thus the DOLMEN system, for example, had four possible rigid frame, four braced frame and five shear wall layouts, in both narrow and wide perspectives. In the DOLMEN system, a separate object hierarchy of KEE class units, which was called the *Location Alternatives* was used to represent the various layout options. In the model for the new system, the writer also assigned the location alternatives into a separate class, which resulted in a much simpler object model. This class is shown in Figure F.5, in appendix F.

Separate classes were also required to model the different types of composite physical units. For example, the hierarchy shown in Figure F.6, in Appendix F, was designed to represent the precast concrete unit options. Each lower level class, in that diagram, represents a separate B11, precast unit. The new model also required classes to represent intangible aspects of the structural design domain. These non-physical entities included elements of the plan used to guide the building design process, and the default design parameters. Other

non-physical entities such as the evaluation features were also represented in the object model and some of these are shown in Figure F.9, in appendix F. A separate class was created for each evaluation feature.

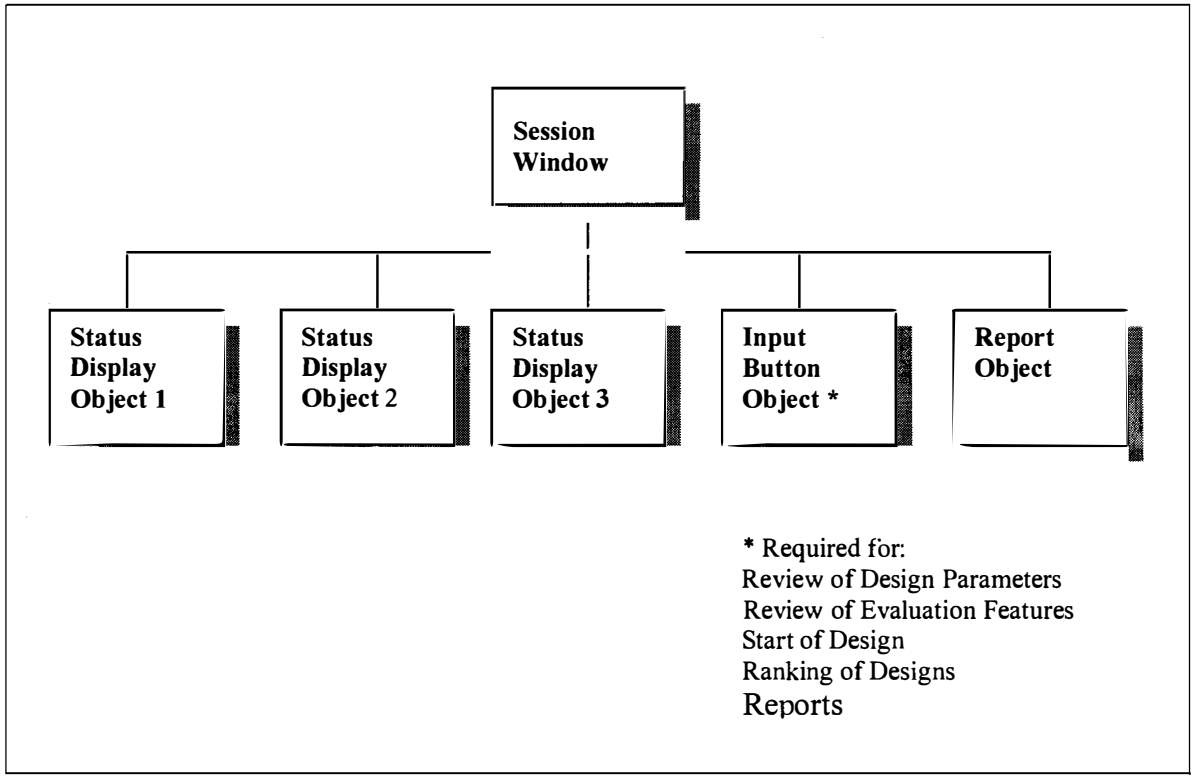


Figure 7.2 Object model of the user interface.

The user interface in the new system was designed to allow the writer to control each phase of the operation of the system. This was done to facilitate debugging during system implementation; it did not resemble the interfaces in the completed HI-RISE and DOLMEN systems. The model of the new user interface included the session windows, input buttons and output displays and reports required to facilitate this interaction. It comprised an association of several object classes, which is shown in Figure 7.2.

A primary purpose of the design project was to develop the new design tool on an object-oriented knowledge-based application development system, Kappa-PC being the system selected. However Kappa-PC applications will not operate without the Kappa-PC run time environment being in memory, that is unless they are compiled into stand alone, executable

C programs or into a dynamic link library. The run time environment provides a wide range of system objects, including an object browser, inference system, and rule base, which form part of the new design tool. Unfortunately the writer was unable to find references, which described how their inclusion in the model should be diagrammed. These features were therefore shown as a 'black box' in the writer's object model. Production rules were also treated as black boxes; the writer also being unable to find references describing the modeling of production rules. A diagram of the overall object model is shown below, in Figure 7.3. Other classes of objects identified included those used to represent the Default Design Parameters, the Evaluation Features and the Schedule class, which was used to hold information required to control the sequence of design activities. Appendix F contains several diagrams, which collectively comprise the object model.

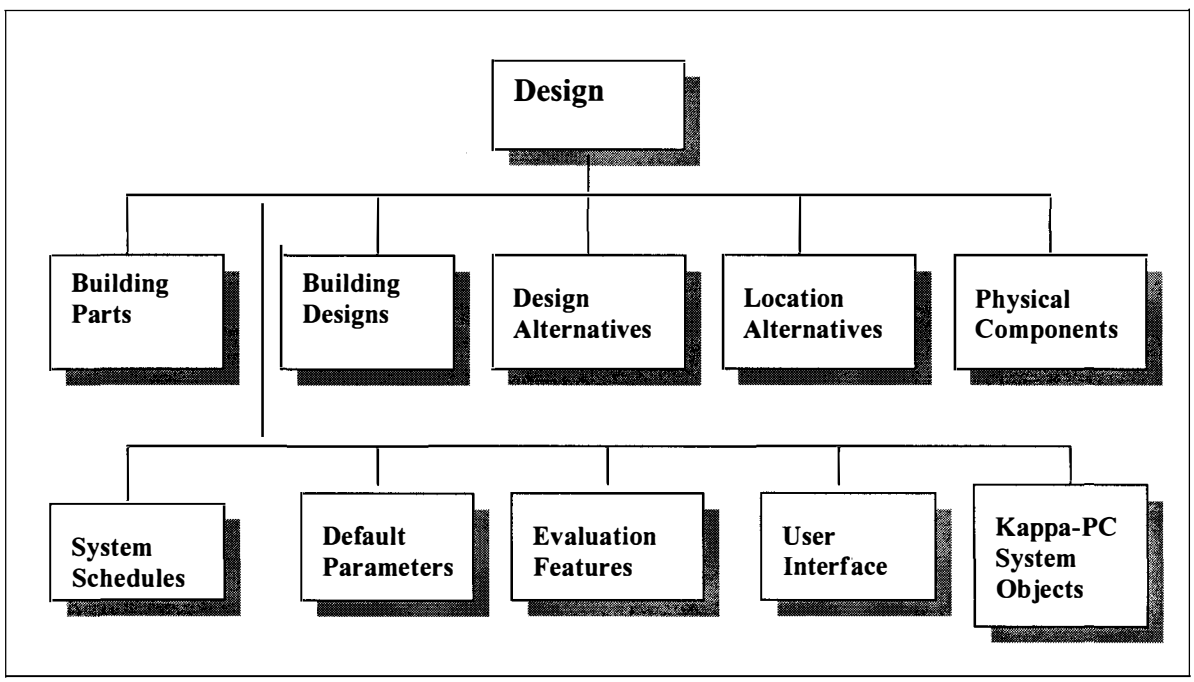


Figure 7.3 Object model for the NOVA design tool.

7.1.2 Determine the Responsibilities of the Objects.

In order to document what each object was supposed to do, the writer followed the process recommended by Rumbaugh et al. (1991). In this process state transition diagrams, also

known as dynamic models, were developed for key objects. According to Henderson-Sellers (1992) the state transition diagram provides the basic mechanism for documenting the behavioural aspects of the object model, showing how a class responds to events. He recommends that diagrams should be produced for all non-trivial classes.

The writer initially encountered difficulties in preparing these dynamic diagrams, finding them to be non-intuitive to implement. However, state transition diagrams were produced for the user interface, a generic partial design class and for the evaluation feature classes.

Because of the size of the system it was necessary to divide it into three arbitrary subsystems, these were the three design stages: specification, formulation and evaluation. The writer used the same names for these three stages of design, as did Harty (1987).

- **Specification**

During the specification task, the system was designed to allow the user to input the

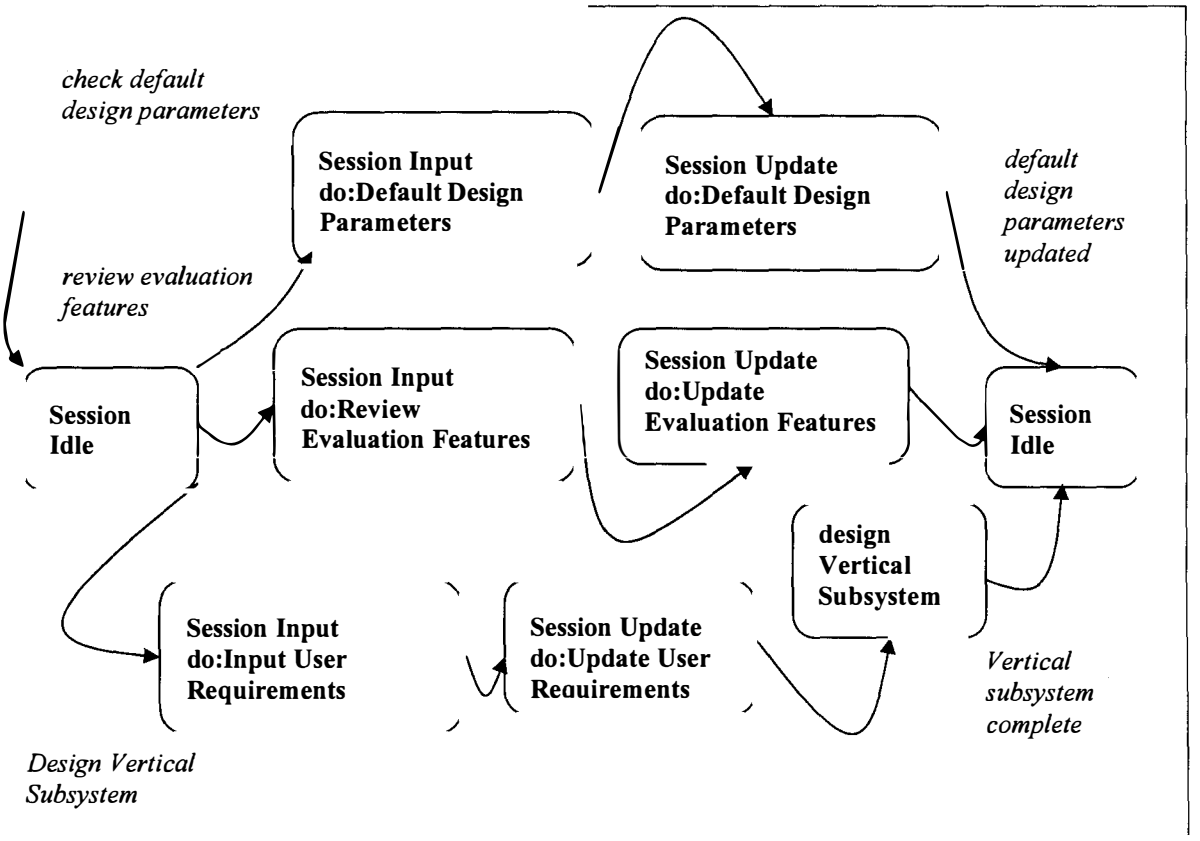


Figure 7.4 **State transition diagram for session, showing specification events.**

requirements for the new building, to confirm the draft design parameters and to review the evaluation features. Figure 7.4 shows the effects of this phase on the user interface.

- **Formulation**

During the formulation stage of the design process envisaged for the new design tool, it was necessary to simulate a process of design synthesis, which was to be followed by the detailing and testing of the physical components making up these designs. Synthesis required that the design tool would initially create and subsequently modify a set of design objects. In the Dolmen system the program code, which effected these changes was invoked through the use of slot monitors or demons, which were located in the appropriate classes or frames. In the prototype, developed for the new design tool, these changes were designed to be initiated by the user. In effect, the user was required to select the appropriate input button and then the system would execute the code, associated with the button. Subsequent processing was effected via a series of design functions, which communicated with each other via a message-passing scheme. This arrangement allowed the writer to control the starting of each separate stage of the design process and it also facilitated debugging and system enhancement. In a finished system, (ie. not a prototype), the input buttons would be removed and the system would proceed automatically.

Several events were identified, which effected the user interface, these included: *review_evaluation_features, design_vertical_subsystem, design_horizontal_subsystem, check_default_design_parameters and evaluate_designs*. These events were modeled in state transition diagrams; Figure 7.6 shows the design, display and evaluation events. The event *design_vertical_subsystem* changes the status display object to *Design_Vertical_Subsystem*; this starts the system design process. The first phase of the series of *Design_Vertical_Subsystem* functions corresponds to the specification phase, during which the user inputs the building requirements. The system was designed to move

automatically into the formulation phase and start to execute a whole series of design functions.

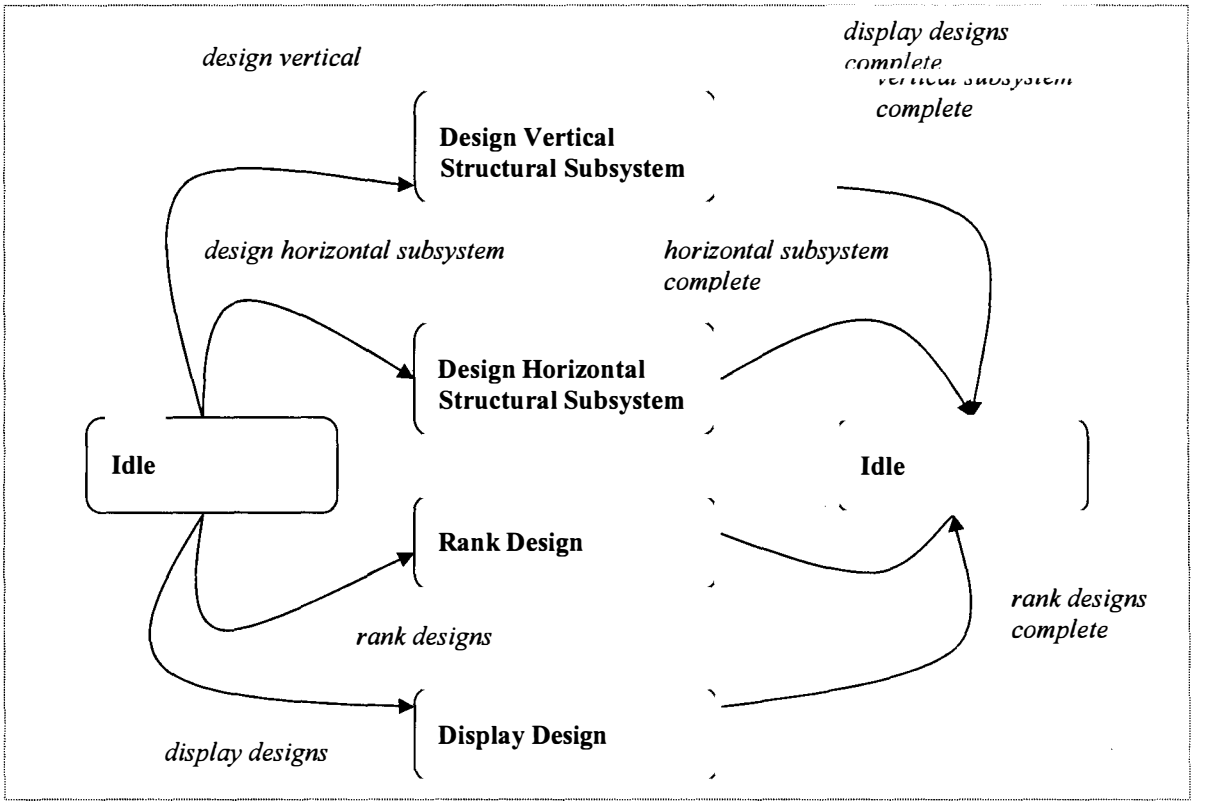


Figure 7.5 State Transition Diagram for the User Interface

DOLMEN commenced design synthesis by then setting up the root of the search tree by creating a subclass of the object *Building*. Then after enquiring whether or not the use wishes to review the *Default_Design_Parameters,* the system initiates the design synthesis process.

- **Evaluation**

The evaluation process represented in the new system tries to reproduce the functions exhibited by DOLMEN. Figure 7.7 shows state diagrams for the evaluation events.

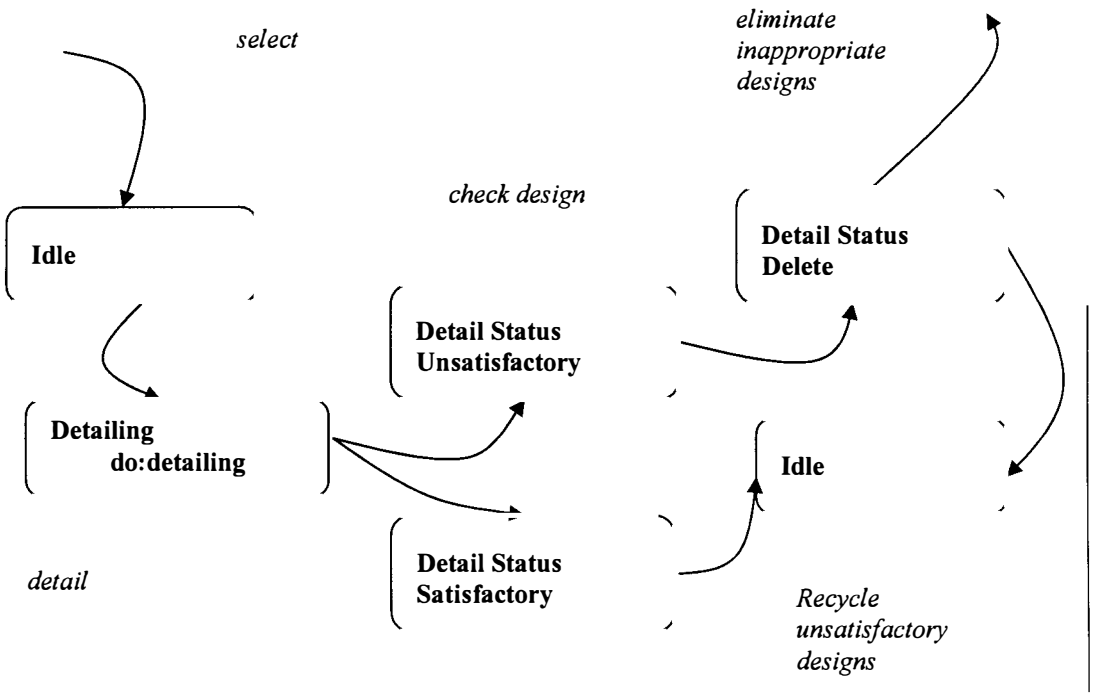


Figure 7.6 **Formulation stage for a generic partial design class.**

and figure 7.8 provides a composite picture of the interplay of the evaluation process objects, superimposed on a layout for the evaluation report. This diagram is intended to show how the evaluation feature objects are intended to operate in the new system, taking advantage of the object-oriented message-passing paradigm.

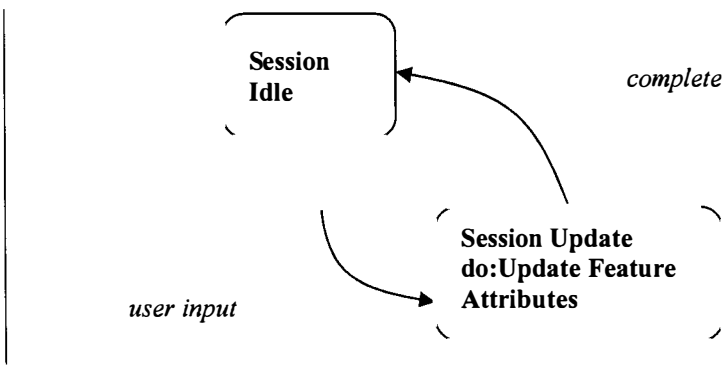


Figure 7.7 **State transition diagram, showing update of feature attributes**

Layout for Evaluation Report							
Evaluation of Alternative							
Feature value	Height 13.89	Sway	Time	Clear space	Column size	Total	
Set?	Yes	X	X	X	X	X	
Max	15.5	X	X	X	X	X	
Min	12.5	X	X	X	X	X	
Objective	minimise	X	X	X	X	X	
Importance	quite	X	X	X	X	X	
		X	X	X	X	X	
1. orth*st*RF-2-N*RF-2-W		X	X	X	X	X	
Percentage optimisation	53.52	X	X	X	X	X	
Total score	107.04	X	X	X	X	X	

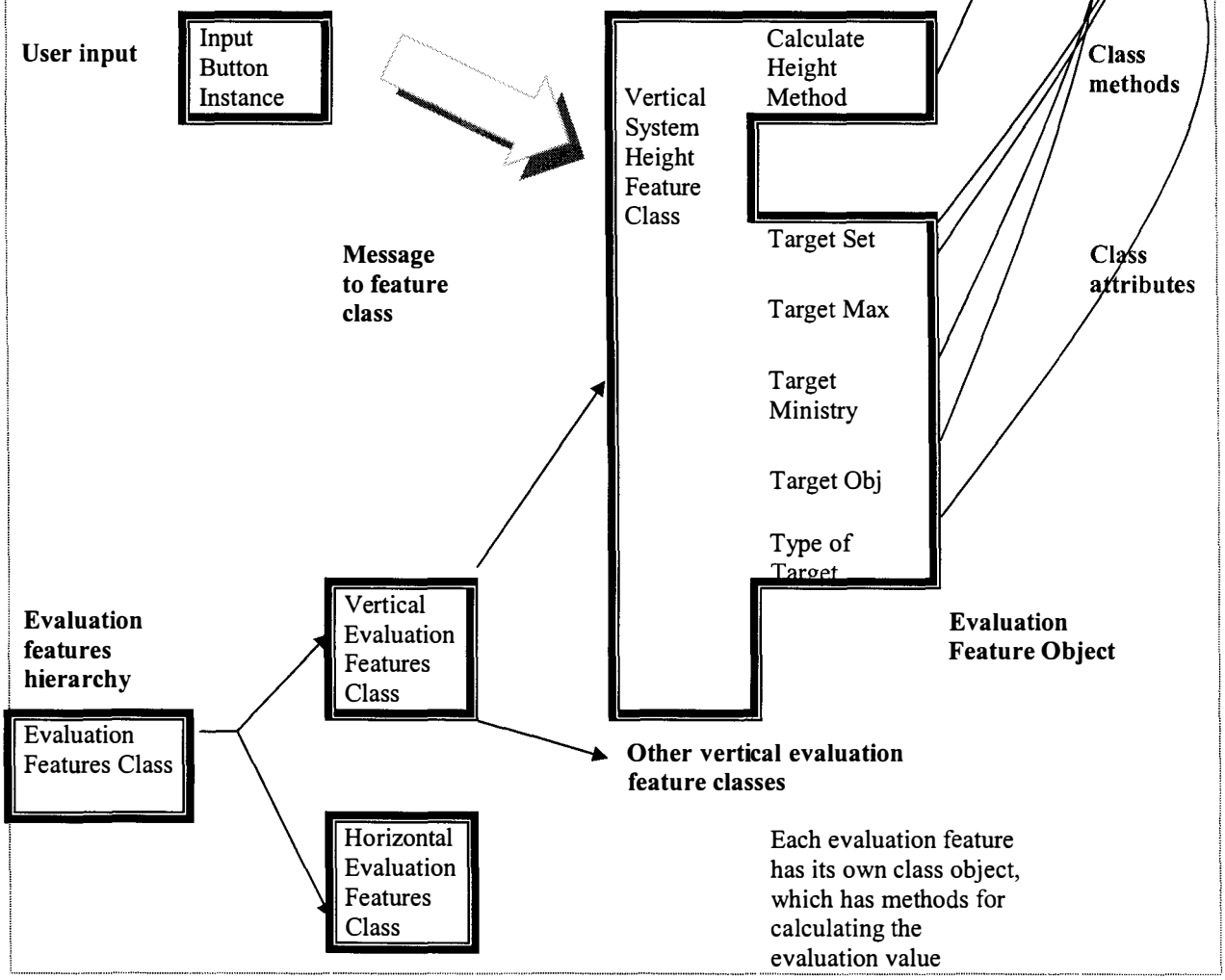


Figure 7.8 **Object Model for the evaluation process**

In the model, each evaluation feature is given the appropriate methods and attributes to allow it to calculate the feature value for a given design option, the identity of which is passed to the feature, as part of the message invoking the class method, when the system requires the feature value.

- **Other System Processes**

Dictionaries in the form of a series of tables were completed to describe the activities and actions of the objects in the object model, most of which were not described individually in state transition diagrams. The main table is shown in Table 7.1.

Class	Activities Associated With The Class	Description
Session		
Input Button	Clear_Hierarchy	Clear the search tree from the object Kappa browser
Input Button	Count	Count designs in the search tree
Input Button	Check_Design_Parameters	Review and update design parameters
Input Button	Review_Evaluation_Features	Check each feature and change if required
Input Button	Design_Vertical_Subsystem	Create the search tree for the vertical 2D subsystem
Input Button	Detail_Vertical_Subsystem	Calculate sizes of physical components, check against rule base
Input Button	Design_Horizontal_Subsystem	Create the search tree for the horizontal 2D subsystem
Input Button	Detail_Horizontal_Subsystem	Calculate sizes of physical components, check against rule base
Input Button	Design_Report	Display selected designs
Input Button	Rank_Design	Display evaluation values for a top given number of selected designs
Input Button	Display_Status_1	Indicate which load resisting system being designed
Input Button	Display_Status_2	Indicate which level in the design hierarchy is being designed
Input Button	Display_Status_3	Indicate which design task is in process
Input Button	Evaluation_Report	Display evaluation values for all designs
Input Button	Display_Final_Design	Display key design details
Input Button	Quit	Save Kappa file and close system

Table 7.1 Table of key design events.

Table 7.2 shows examples of object activities in the form of methods, which represent the behaviour of particular objects.

Class	Activities Associated With The Class	Description
Design Alternatives	Calculate Weight of Floor	
Floor Alternatives	Calculate Depth of Supporting Beam	
	Calculate Depth of Beam Under Floor	
	Calculate Floor Cost	
	Detail	
Location Alternatives	Calculate Number of Frames	Determine how may subsystems will be used
	Calculate Number of Interior Frames	Determine how may interior subsystems will be used
	Calculate Width of Shear Wall	Determine width of wall subsystem
Evaluation Features	Feature Calculation	Calculate feature value, weighted value and optimisation score

Table 7.2 Table of object methods.

Appendix B - System Notes contains rough workings for the functional diagrams, which describe computations and non-interactive functions within the system. These rough functional diagrams were used extensively to model formulation and evaluation events.

7.1.3 Determine the Associations between the Objects.

- **User Interface**

Analysis of the user interface revealed that it was associated with the Building, Design Parameters, Evaluation Features, Status Display and Reports object classes: in the control of the design process, in the execution of input and output events and in the display of the design results to the user. This is shown in Figure 7.9.

The association relationship also formed the basis for the design of the search tree of partial design objects. Analysis revealed that an association, between the classes in the Building hierarchy and the corresponding classes in the Alternatives and Location Alternatives hierarchies, was required to form the search tree. How these classes associate to form the search tree is displayed in Figures 7.12 and 7.13. The graph, which connects the shaded classes or objects forms one instance of a path through the search tree, and represents one partial design solution.

In the new system formulation processing includes design synthesis followed by detailing and testing. The approach to the design of the synthesis activity was similar to that used in the DOLMEN system as described by Harty (1987).

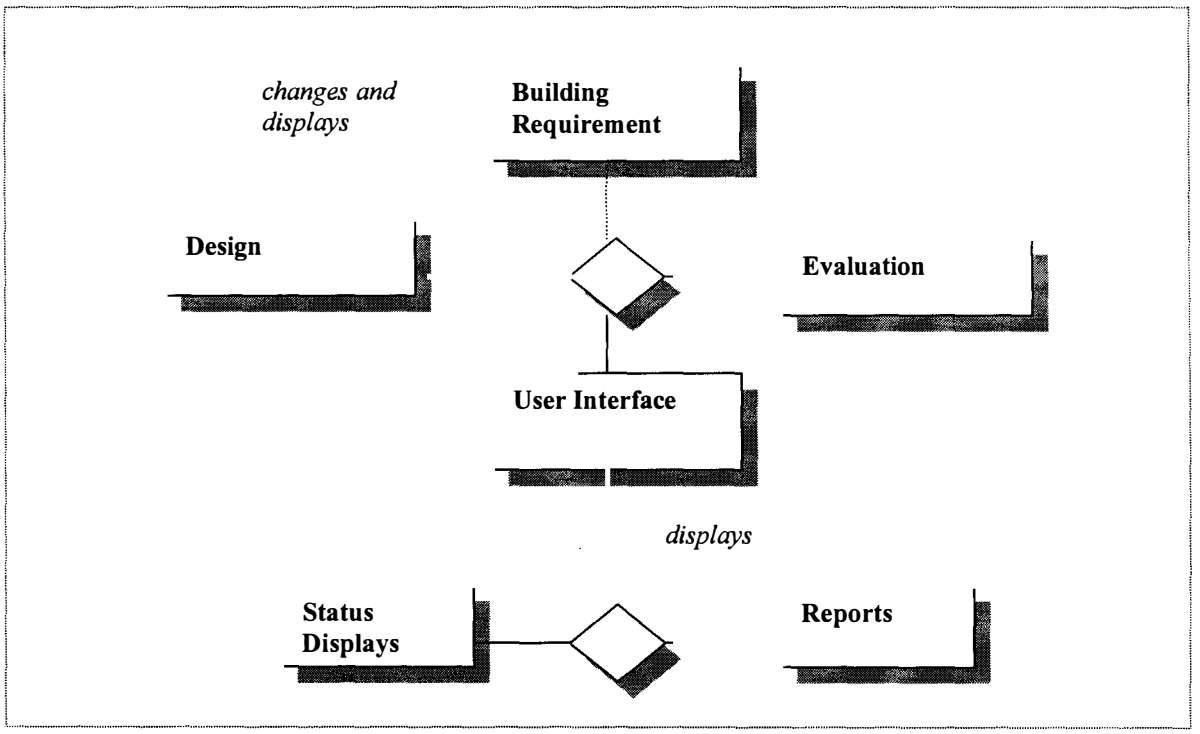


Figure 7.9 **User Interface Associations**

7.1.4 Determine the attributes contained by the objects.

Attributes were determined for each of the objects identified in the system object model. Attributes for the significant design objects are shown in Tables in Appendix G. The attributes were initially identified by reference to the documentation provided in the reports of Maher (1984) and Harty (1987). These attributes were refined and more details were added as the design progressed.

7.1.5 Organise Object Hierarchy and Establish Inheritance Links

In this step the objects were placed into hierarchies of classes and inheritance links were determined between the class members. Simple inheritance links were developed for the

alternative hierarchies, which included the design options and configuration location alternatives, and for those classes representing physical components. More complex inheritance links were required to design the search tree.

- **Formulation**

The designers of HI-RISE and DOLMEN followed the principle of locating generic class attributes as high as possible in the class hierarchy. This allowed them to distribute these attributes, using their respective system's inheritance facilities to the best advantage. Specialised attributes were added to the objects, which appeared lower down in the hierarchy, these attributes had a smaller or non-existent range over which they were inherited. This reflected the fact that these objects were more specialised and were beginning to more closely resemble the real world objects, which they represented.

In the new system, generation of alternatives at the *Vertical_3D* stage involved the creation of subclasses of *Core* and *Orthogonal_2D_Systems* (*Core_1* and *Orthogonal_2D_Systems_1*). These two were attached, using the subclass relationship, to the *Building_1* class, which contained the original specification for the building, which all alternative designs must accommodate and which formed the root node of the hierarchy.

Kappa-PC provides two forms of system objects, classes and instances. Kappa-PC allows a class to form a subclass, which can inherit the parent class's attributes and methods. A Kappa-PC class can also form an instance of itself, which also inherits its attributes and methods; however, no further descent is allowed from the instance, which is not allowed any subtypes. Attachment of the new design classes to the search tree, using the subclass relationship, allows the new objects to inherit their parent class's slot values. They can also pass on these attributes, and any they might have of their own, to their subclasses.

The new system was designed to construct the search tree in a manner similar to that of the DOLMEN system. Analysis of this feature resulted in several sketches and working diagrams, which were drawn to determine how this process had been effected. Two of these diagrams are reproduced in Appendix B in Figures B.3 and B.4, which show some of the workings made for the design of the inheritance links required in the search tree of partial design objects. As in the DOLMEN system, the hierarchy for the new system was designed such that the *Braced_Frame_Narrow_1, Rigid_Frame_Narrow_1*, and *Shear_Wall_ Narrow_1* classes were created and attached to *Core_1* during generation of *Vertical_2D_ Narrow* alternatives. Likewise, the *Braced_Frame_Narrow_2, Rigid_Frame_ Narrow_2,* and *Shear_Wall_Narrow_2* were also attached to *Orthogonal_2D_ Systems_1*. In this way a hierarchy or tree of possible alternatives was built up, with the leaves defining the current partial designs.

This organisation of the classes and subclasses is shown in Figures 7.11 and 7.12. In the new system each node in the tree can inherit all of the slots of its parent, through the subclass *(part_of)* relationship and those of its alternative parent class, through the *(is_alt)* relationship, via a copy function, which copies attributes from the appropriate alternative class. This method of constructing the search tree requires the use of multiple inheritance. Multiple inheritance was provided as a standard feature on the KEE system, which was used to develop the DOLMEN system. However, it is not available on Kappa-PC and the writer had to program a series of copy functions to provide a work-around. Figure 7.10 shows how this was conceived for the new system.

For example, the partial design class at the 2D-Narrow level, *Rigid_Frame_Narrow_1* inherits the slots of *Core_1* and Building_*1*, (simple or direct inheritance) as well as the slots of the *Rigid_Frame_Narrow* class (through multiple inheritance). These slots contain the attributes, which define the characteristics of braced frame structures. As the search for

alternatives proceeds down the hierarchy each new level adds the appropriate functionality required at that level.

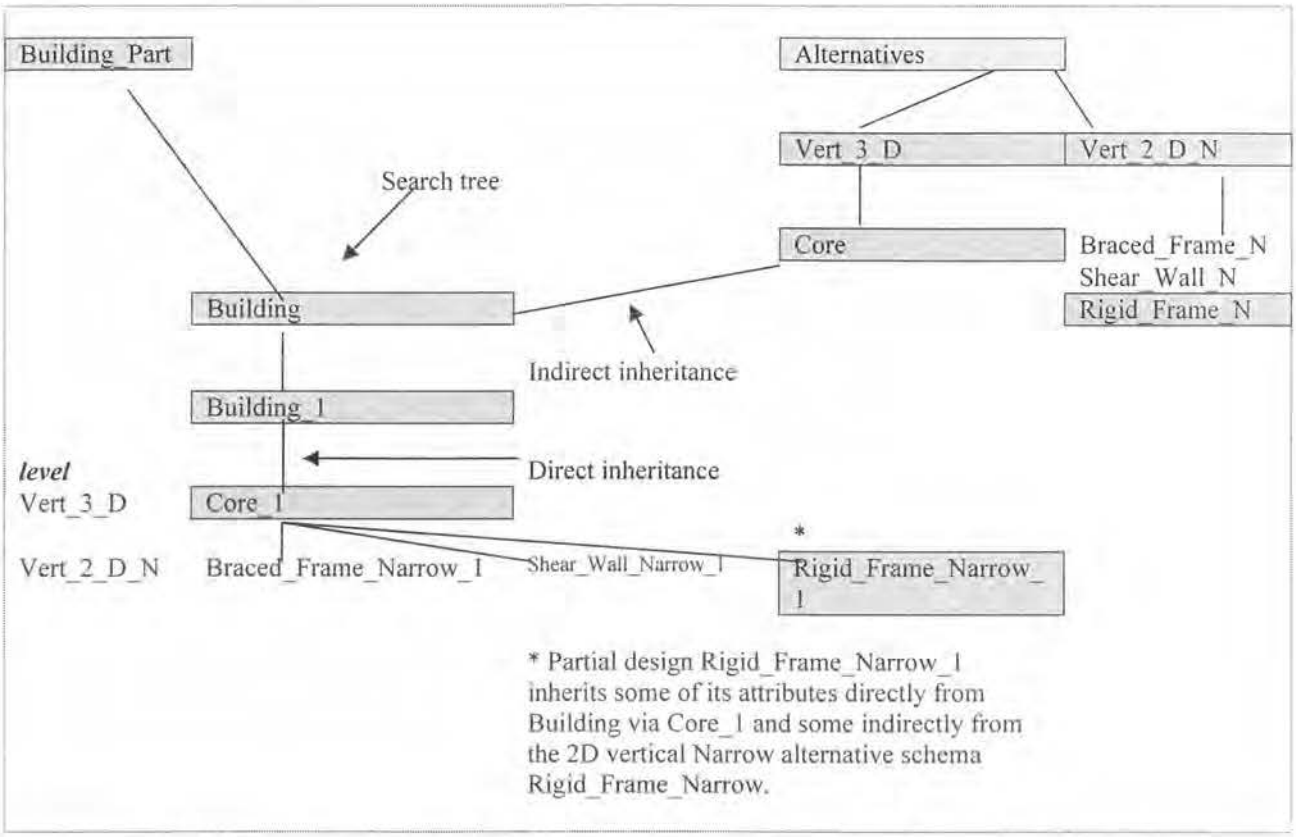


Figure 7.10 Subsection of Design Hierarchy Showing Multiple Inheritance

In the design of the new system it was found to be difficult to determine an appropriate scheme for method placement in the partial design classes. Several different schemes all involving elaborate message-passing schemes were tried before the final scheme was chosen.

The new object model incorporated both product and process models, which represented the design process and the building product being designed. The hierarchy in the object model was intended to support the product model and represents aspects of the structure, function and behaviour of the building subsystems. The process model simulated the activities performed by the structural designer, who creates and works with the product model. To a large extent process activities were simulated by the algorithms in the synthesis component,

which included the '*generate new units*', detailing and testing (elimination) functions and in the subsequent evaluation and ranking components. The functions simulate the synthesis of new design classes, the application of heuristic knowledge to eliminate infeasible designs and the completion of the rough calculations, which are used to size beams and columns. They also eliminate those designs, which cannot accommodate the required stresses. Several functional, data flow diagrams were created to facilitate the modeling of these process flows and subsequently to facilitate the required design methods and functions for these processes. Figures B.1 and B.2 in Appendix B show some of the workings for these diagrams.

- **Evaluation**

In order to support evaluation the new system is required to accommodate hard and soft design constraints. The hard constraints represent building requirements, which must be achieved. Failure to meet these requirements should result in the elimination of the design from further consideration. The soft constraints were to be represented by numerical variables. These were to be set up in the form of design targets and by a set of evaluation criteria for each synthesis mode or subsystem level of abstraction.

The evaluation criteria were represented by a hierarchy of evaluation objects. The generic attributes of this hierarchy were located in the root object of the hierarchy. An Evaluation Feature object at the lowest and most specialised level in the hierarchy represented each soft constraint. Each object at this level has its own method, which allowed it to calculate the required feature value and other values including the optimisation score. These calculations are invoked via a message-passing scheme, which is shown in figure 7.8.

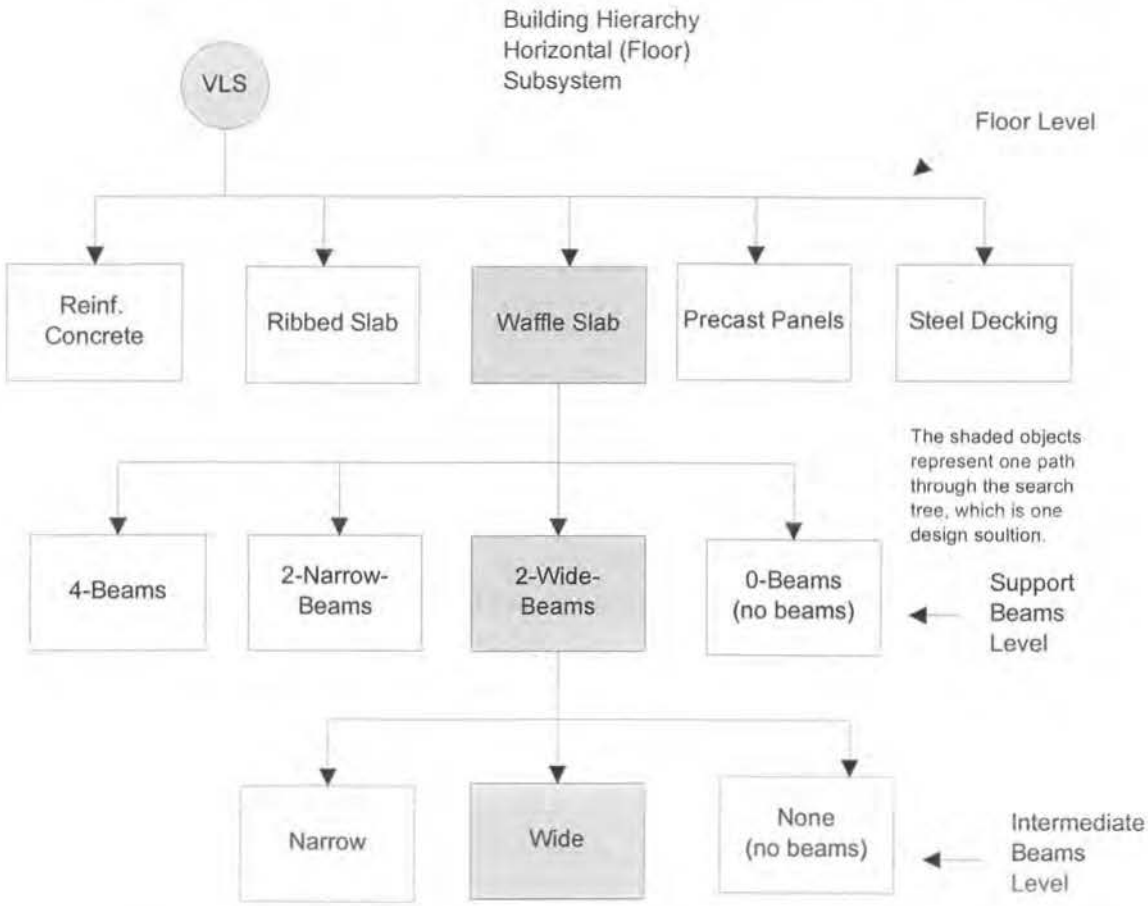


Figure 7.11 **The Building hierarchy – horizontal structural subsystem**

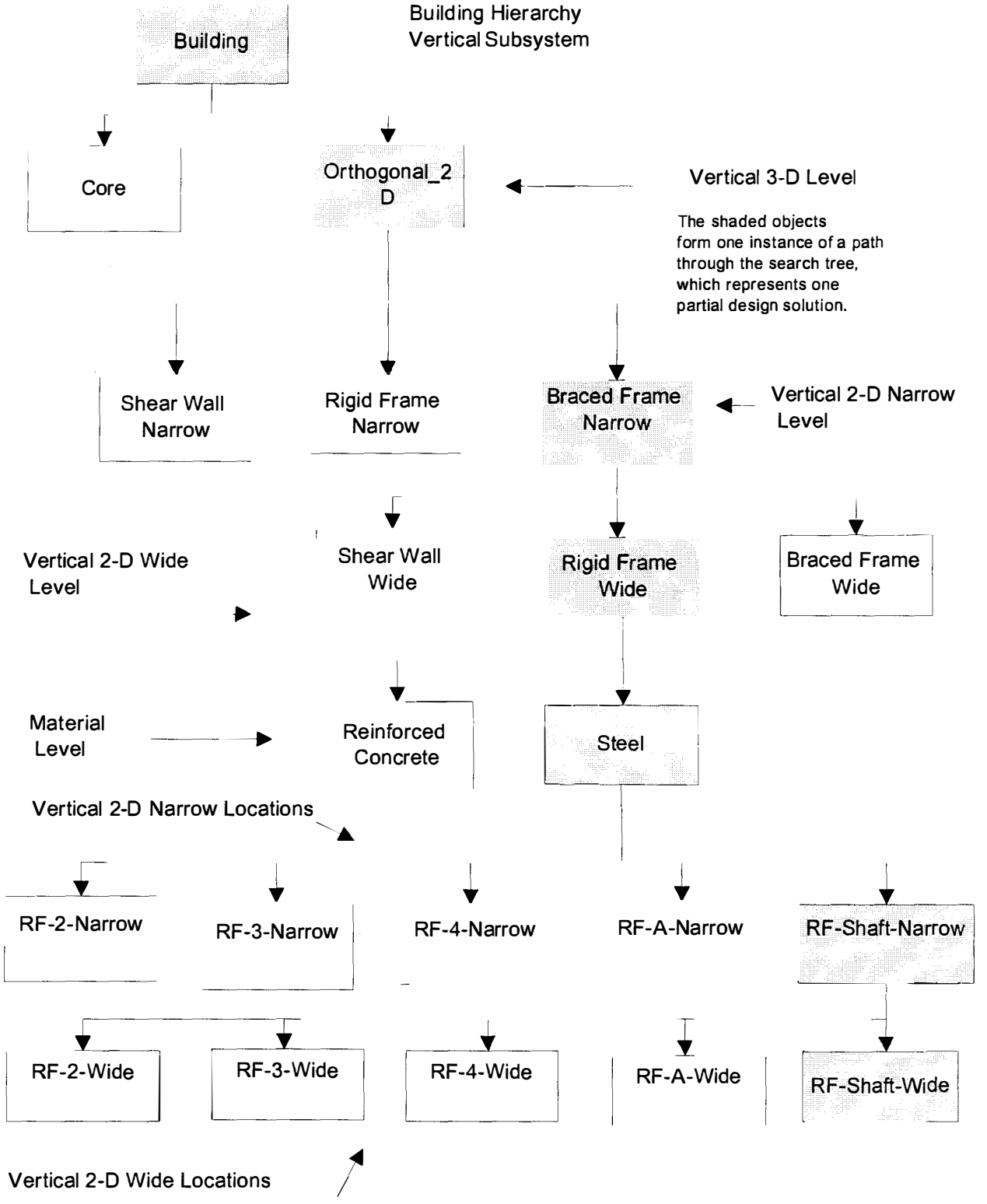


Figure 7.12 The Building hierarchy – (vertical structural subsystem)

7.2 Object-oriented Design

This section describes the fourth stage of the analysis and design project, which is object-oriented design. This stage was concerned with how to model the software required to implement the object model established in the analysis stage.

The design stage was intended to complete the object model, which comprised the models, diagrams, flowcharts and accompanying notes, which were produced in the analysis stage. Other objects, such as the Kappa-PC inferencing system and certain user interface components were also integrated with the model. In addition, the analysis objects were reviewed for technical feasibility to ensure that they could actually be implemented on the Kappa-PC platform.

The main products of the design stage were object diagrams and message passing schemes. During this stage several system prototypes were created. The design process required several iterations, each iteration resulting in a more sophisticated prototype.

7.2.1 Design Details

- **Specification**

The user interface part of the object model was used as the basis for describing the input and status display objects, required to support the specification functions and processes. These allowed the user to input and review Default Design Parameters and Evaluation Features and subsequently to input the specifications from which the new building was to be designed.

The writer relied on the display images provided in the Kappa-PC libraries to design the user interface. These images are accessed via and used in conjunction with the Session Window.

Figure 7.13 Kappa-PC Session Window, showing graphic images

The KAPPA-PC Session Window allows the user; to customize the interface with a choice of graphics and display objects, to create the interface required.

The KAPPA-PC images are able to display the output of the application or to accept input from the user. The Session Window consists of a display area and a menu bar. The display area contains all the images, which can be defined either programmatically or via the graphics ToolBox, or the Select menu, which is shown in figure 7.13.

The Session Window has two modes: Layout and Runtime. Layout Mode is used to manipulate graphic images through the mouse-and-menu interface. Runtime Mode is used when the system is being used to present the application interface to an end user.

The menu bar of the Session Window contains seven pull-down menus: Align, Image, Edit, Control, Options, Window, and Select. Figure 7.14 shows the image edit windows, which include the Instance Editor and the Button Options windows, which the writer used to tailor the input objects used in the system. The final NOVA application user interface is reproduced in figure 10.3.

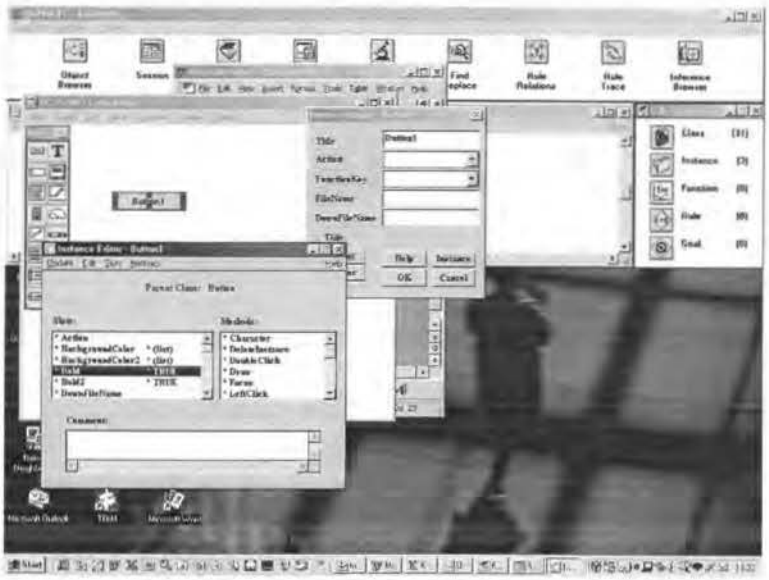


Figure 7.14 The Kappa-PC image editor

- **Formulation**

Preliminary structural design requires the selection of subsystems at both the vertical and horizontal levels, this requires two sets of formulation functions, which are arranged according to the subsystem level. The selection process comprises a series of steps at which alternative design are produced and tested.

This section describes design considerations for the key design processes. Several generations of flowcharts were required to design the formulation component, which simulated the system plan-generate-test activities, which included processes for design synthesise, component detailing and testing.

NOVA incorporates both product and process models. A hierarchy of object classes, which are shown in Figures 7.11 and 7.12, represents the product model, which describes aspects of the structure, function and behaviour of the building hierarchy. The process model has been designed as a series of detailing and testing (elimination) functions, which effect the plan-generate-test activities completed during design synthesis. These functions simulate the application of heuristic knowledge to eliminate infeasible designs. They also simulate

the rough calculations used to determine the sizes required by the beams and columns and to eliminate those designs, which cannot accommodate the required stresses.

NOVA has a knowledge base, which includes decomposition, planning, constraint and evaluation knowledge. The decomposition knowledge is represented in the system as a hierarchy of systems and subsystems, which are implemented as Kappa-PC classes, which are shown in table 7.5. These classes have attributes, which are represented in the slots, which contain descriptive values and have a set of procedures, which are represented by the methods attached to the classes.

The planning knowledge in the system includes a *Schedule* class, which has several slots, which contain lists of sequences of operations for the design process. These sequences are referred to by the program code, which effects the design synthesis.

- **Design Synthesis**

The following section describes synthesis at the vertical structural subsystem. It includes the processes shown in table 7.3.

On start up the user enters a number of details for the new building, for example the number of stories and the various dimensions of bays. The system then builds the search tree by creating a subclass of the class Building, which is called *Building_1*. It then creates further classes called *Core_1* and *Orthogonal_2D_Systems_1*. It continues down the hierarchy creating new subclasses, ie. *Braced_Frame_Narrow_1* and *Braced_Frame_Narrow_2,* checking them against its design rules before adding them to the search tree/object hierarchy.

Each new class added to the search tree inherits attributes:

- from its parent class, through the Kappa-PC single inheritance mechanism, which is provided by the *MakeClass* function; and

- From the appropriate *Alternative* class, through a user function, *Slot_Copy*, which was written to simulate multiple inheritance, which is not provided by Kappa-PC.

Subtask	System Function	Design Process
Formulation	Design Vertical Subsystem	Design Vertical-3D-Level
		Design Vertical-2D-Narrow Level
		Design Vertical-2D-Wide Level
		Design Vertical-2D-Material Level
		Design Vertical-2D-Narrow-Location
		Design Vertical-2D-Wide-Location Level

Table 7.3 Design processes at the vertical structural subsystem level.

The system keeps on adding new classes at each level and deleting inappropriate ones according to its rules. By the time the system reaches the ninth level in the hierarchy, the *Intermediate_Beams* level it has created a search tree/object hierarchy of valid designs.

The following pseudo code describes this design process.

- **Design Process Pseudo Code**

The process commences with the function, *Design_Vertical_Subsystem*, which clears out any existing search tree and then loads the sequence of design steps into the appropriate slot in the *Schedule* class. This function then checks whether the user wishes to review the Default Design Parameters and Evaluation Features and then creates the new building object and queries the user for the input of the building requirements.

```
/*************************************************/

FUNCTION: Design_Vertical_Subsystem

CALL THE FUNCTION Ask_About_User_Locations

APPEND CLASSES Vert_3D, Vert_2D_Narrow, Vert_2D_Wide,        Set up the sequence of
Material,Vert_2D_Narrow_Loc, Vert_2D_Wide_Loc, Floor, Support_Beams,  levels to be designed
Intermed_Beams
TO SLOT Sequence_Of_Parts_To_Be_Design IN CLASS Schedule

MAKE A NEW CLASS Building_1, FROM CLASS Building

CALL THE FUNCTION Input_User_Requirements FOR Building_1

CALL THE FUNCTION Set_Defaults
AskValue Global:Review_Defaults_Flag                         Check if defaults to
```

```
                                                                    be reviewed
IF Yes
        THEN CALL THE FUNCTION Check_Design_Parameters

AskValue Global:Review_Evaluation_Features                          Check if evaluation
                                                                    features to be
                                                                    reviewed
IF Yes
        THEN CALL THE FUNCTION Review_ Evaluation_Features

CALL THE FUNCTION Generate_New_Units

/************************************************/
```

The function *Generate_New_Units* starts by looking at the sequence object, the *Sequence_of_parts_to_be_designed* slot in the *Schedule* object to see which level is to be designed next. This part corresponds to the respective level in the building hierarchy, see figure 7.15

Subsystem	Level in Building Hierarchy	Design Alternatives
Schematic Level	Vertical 3D	Core Orthogonal_2Dimension_Systems
Vertical Structural Subsystem	Vertical 2D Narrow Perspective	Braced Frames Shear Walls Rigid Frames
	Vertical 2D Wide Perspective	Braced Frames Shear Walls Rigid Frames
	Material	Reinforced Concrete Steel
	Vertical 2D Narrow Location	Various Configurations
	Vertical 2D Wide Location	Various Configurations

Table 7.4 Levels in the building hierarchy

Having noted the level, the function refers to the *Alternatives* hierarchy and adds a class, bearing the part name to the search tree and then calls one of a series of functions named *Design_xx_Level*, where xx is the appropriate level. This function creates a list of all the alternatives at that level, ie. at the *Vertical_2D_Narrow* level it will have the names *Rigid_Frame_Narrow, Braced_Frame_Narrow* and *Shear_Wall_Narrow*. For each item on the list it creates a subclass, which it attaches to the search tree at the level indicated. These

subclasses are given the appropriate level name with the suffix *_Alt_xx,* where xx is a number, which is incremented for each new class created.

```
/*****************************************************/

FUNCTION: Generate_New_Units


ASSIGN Generate TO SLOT Task IN CLASS Schedule

ASSIGN 0 TO SLOT Number IN INSTANCE Global

COPY THE 1ST ITEM FROM THE FOLLOWING LIST (Schedule:Sequence_Of_Parts_To_Be_Design)
TO SLOT Part_To_Design IN CLASS Schedule

COPY THE 1ST ITEM FROM THE FOLLOWING LIST Schedule:Sequence_Of_Parts_To_Be_Design)
TO SLOT Current_Design_Level IN INSTANCE Global

MAKE A LIST OF ALL THE SUBCLASSES OF Global:Current_Design_Level
AND PUT THE LIST INTO SLOT Current_Design_Level_Subs IN INSTANCE Global

FOR EACH ITEM X ON THE FOLLOWING LIST( Global:Current_Design_Level_Subs,
        DO THE FOLLOWING ACTIONS

            CALL THE FUNCTION Slot_Copy_Levels(x, COPY THE 1ST ITEM FROM THE
            FOLLOWING LIST (Schedule:Sequence_Of_Parts_To_Be_Design))

            CALL THE FUNCTION Design_First_Level_Down(Building_1)

            REMOVE THE 1ST ITEM FROM THE FOLLOWING LIST
             (Schedule:Sequence_Of_Parts_To_Be_Design

            ASSIGN SLOT VALUE New_Designs_In_Creation IN INSTANCE Global TO
            Global:New_Designs_In_Vert_2D_N

            COPY THE 1ST ITEM FROM THE FOLLOWING LIST
            (Schedule:Sequence_Of_Parts_To_Be_Design) TO SLOT Part_To_Design IN CLASS Schedule

            COPY THE 1ST ITEM FROM THE FOLLOWING LIST
            (Schedule:Sequence_Of_Parts_To_Be_Design) TO SLOT Current_Design_Level IN INSTANCE
            Global

            MAKE A LIST OF ALL THE SUBCLASSES OF (Global:Current_Design_Level  COPY THIS
            LIST TO SLOT Current_Design_Level_Subs IN INSTANCE Global

            FOR EACH ITEM X ON THE FOLLOWING LIST( Global:Current_Design_Level_Subs,
            DO THE FOLLOWING ACTIONS

                    Slot_Copy_Levels(x, COPY THE 1ST ITEM FROM THE FOLLOWING LIST
                    (Schedule:Sequence_Of_Parts_To_Be_Design))

FOR EACH ITEM X ON THE FOLLOWING LIST (Global:New_Designs_In_Vert_2D_N,
        DO THE FOLLOWING ACTIONS
        CALL THE FUNCTION Design_Vert_2D_N_Level(x)

        FROM THE FOLLOWING LIST REMOVE X
        (Global:New_Designs_In_Vert_2D_N, x)
```

```
        REMOVE THE 1ST ITEM FROM THE FOLLOWING LIST
        (Schedule:Sequence_Of_Parts_To_Be_Design

        ASSIGN SLOT VALUE New_Designs_In_Vert_2D_W IN INSTANCE Global TO VALUE OF
        SLOT New_Designs_In_Vert_2D_N IN INSTANCE Global

        COPY THE 1ST ITEM FROM THE FOLLOWING LIST
        Schedule:Sequence_Of_Parts_To_Be_Design) TO SLOT Part_To_Design IN CLASS Schedule

        COPY THE 1ST ITEM FROM THE FOLLOWING LIST
        Schedule:Sequence_Of_Parts_To_Be_Design) TO SLOT Current_Design_Level IN INSTANCE
        Global ADD THE STRING _Alts TO THE END OF THE NAME

        COPY A LIST OF ALL THE SUBCLASSES OF Global:Current_Design_Level TO SLOT
        Current_Design_Level_Subs IN INSTANCE Global

FOR EACH ITEM X ON THE FOLLOWING LIST Global:Current_Design_Level_Subs,
DO THE FOLLOWING ACTIONS

        CALL THE FUNCTION Slot_Copy_Levels(x, COPY THE 1ST ITEM FROM THE
        FOLLOWING LIST(Schedule:Sequence_Of_Parts_To_Be_Design))

FOR EACH ITEM X ON THE FOLLOWING LIST (Global:New_Designs_In_Vert_2D_W,
        DO THE FOLLOWING ACTIONS

CALL THE FUNCTION Design_Vert_2D_W_Level(x)
FROM THE FOLLOWING LIST Global:New_Designs_In_Vert_2D_W
REMOVE x


/************************************************/
```

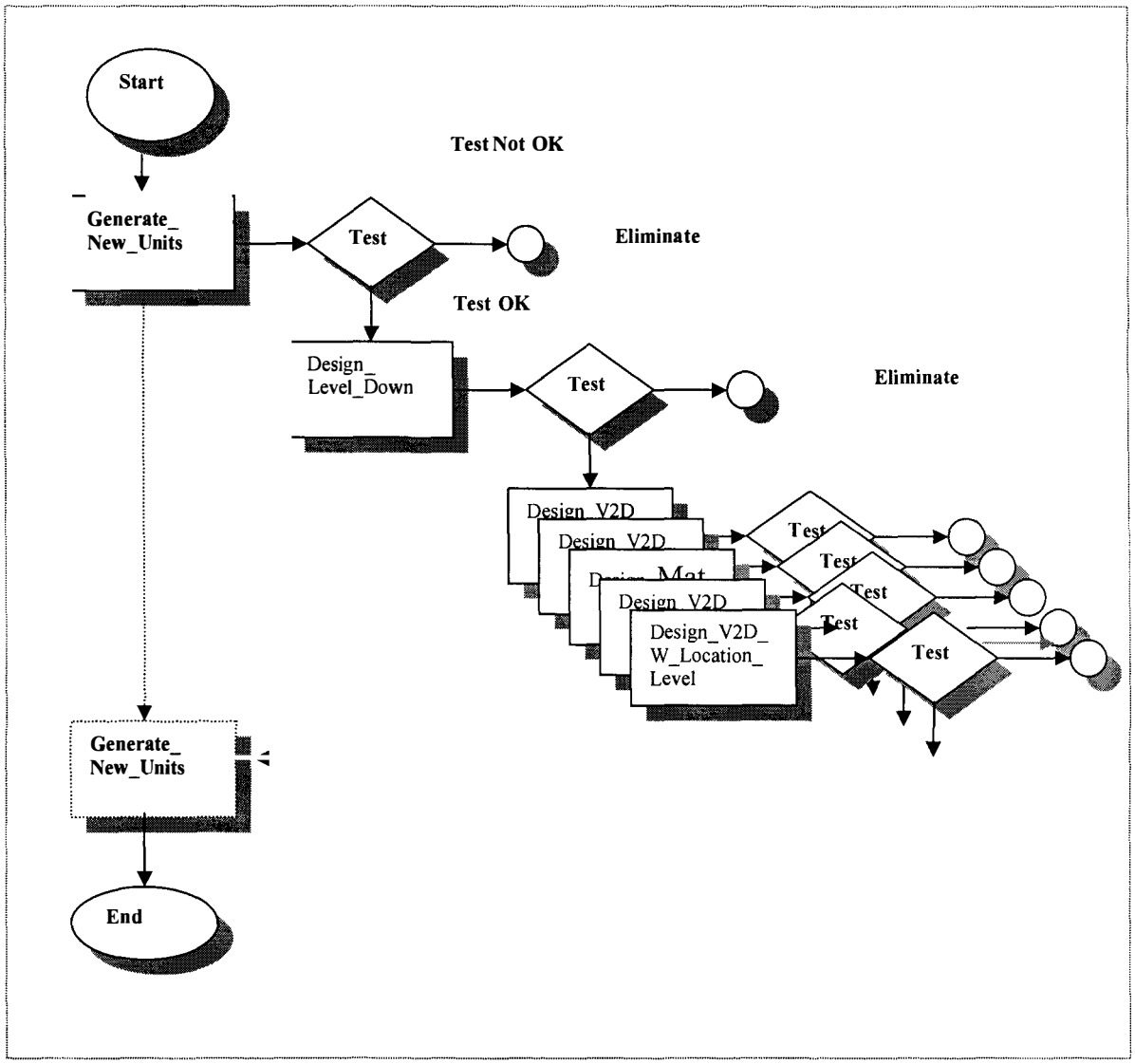


Figure 7.15 Flow chart for formulation

```
/*****************************************************/

FUNCTION: Design_First_Level_Down [First_Level]

ADD TO THE FOLLOWING LIST (Global:New_Designs,First_Level)
EACH OF THE CLASSES CREATED IN THE FOLLOWING STATEMENT

        FOR EACH ITEM X ON THE FOLLOWING LIST (Global:Current_Design_Level_Subs,

                DO THE FOLLOWING ACTIONS
                 Global:Number ←        Global:Number + 1

        FOR EACH ITEM Y ON THE FOLLOWING LIST(Global:New_Designs, y,
          DO THE FOLLOWING ACTIONS

                MAKE A NEW CLASS( x # _ # Global:Number,
                FROM CLASS y)
```

```
CALL THE FUNCTION Slot_Copy_Alts(x)
CALL THE FUNCTION Check_Vert_3D(x # _ # Global:Number
LET
[xx  ← x # _ # Global:Number]
IF ( xx:Eliminated #= Yes)
THEN

If the new class fails the elimination test for the designs
at the vertical 3D level, then it is removed from further consideration.

        DeleteClass(x # _ # Global:Number)
ELSE

If the new class passes the elimination test, then it is added to the list of new
designs.

        ADD TO THE FOLLOWING LIST (Global:New_Designs_In_Creation, x
        # _ # Global:Number)

/*************************************************/
```

- **Detailing and testing.**

Testing of Alternatives - NOVA has a series of 'test and eliminate' functions, with names of the form *Valid-xx-Alt,* where *xx* is the name of the level in the hierarchy. For instance function *Valid-2D-N-Alt* is used to test and eliminate new designs generated for the Vertical_2D_Narrow level. Functions have been written for designs at each level and are used in the first instance to prevent unlikely designs being added to the search tree. They use heuristic knowledge to delete alternatives without further study, however, they do not invoke the inference engine and no production rules are used. The functions are called during the generation of the new designs, which are represented as classes.

A second level of testing is applied to designs, which are not eliminated at the outset. This type of testing requires a more detailed look at the design and invokes the inference engine referring to the material elimination rules, *Global:Rs_For_Material_Elim*, which is a subset of the production rules.

The inference engine is invoked by a checking function, which calls the system's forward chaining inference mechanism,

These rules contain more heuristic knowledge; for example one rule is used to eliminate designs, which have proposed to build more than 20 stories with a rigid frame design.

Detailing - As NOVA proceeds to generate design candidates at each level of the building hierarchy the likely designs are added to the search tree. A second form of testing is now applied to reduce the size of the search tree thereby preventing a combinatorial explosion and at the same time weeding out those designs that are not structurally sound.

This testing requires the partial designs to be quite well defined; therefore it cannot be applied until the designs in the search tree have accumulated sufficient design information. The information required for these tests is created through the process of detailing. This involves calculating estimates for the physical components. Subsequent testing relies on the ability of the system to locate suitably sized steel sections in the steel sections database. If the system is unable to locate a section big enough, then it marks the design to be eliminated.

There are two subsets of detailing functions, those required for the vertical subsystem and those required for the horizontal. Detailing is applied to the vertical subsystem when the locations of the structural alternatives have been selected, ie. at the *Vertical_2D_W_Location_Level.* For the horizontal subsystem or floor system, it is performed when the locations of the support and intermediate beams have been decided and the floor system has been designed. This is at the final level in the building hierarchy, the *Intermediate_Beam_Level.*

There is a subset of detailing functions, which designs the vertical subsystem. This contains functions to detail the three vertical structural subsystem options: braced frame, rigid frame and shear wall. The horizontal subsystem detailing functions perform the design of the flooring systems. These include the following concrete flooring options, flat slabs, Rc

slabs, ribbed slabs and waffle slabs, which each have their respective units comprising precast-floor-units, rib-moulds, ribbed-slabs and waffle-moulds. The system can also design a steel deck floor system, which has a series of possible steel deck units of different sizes.

The detailing functions involve the following steps:

- Select Design Parameters;
- Estimate Initial Sizes;
- Calculate Loadings;
- Select Loadings; and
- Check Design.

The NOVA system has a series of rulesets for checking the validity of roughly designed alternatives. Some checks are concerned with the satisfaction of the most important parts of the structural codes. These rulesets are shown in table 9.1. They also check that designs are of reasonable dimensions, which have been predetermined during the specification stage. These rulesets all have names of the form *Rs_For_Chk_Det_xx_Alts,* where xx is the name of the option to which the ruleset relates. Each detailing function calls the *Check_Design* function to test the designs at various stages in the process. The function is always called with a parameter. For example, when rigid frame checking is required the function call is coded *Check_Design(Bldg, RF);* the parameter RF indicates that the function is to use the ruleset *Rules_For_Checking_Detailed_RF_Alternatives*.

Check_Design uses the appropriate rule from the *Rules_For_Checking_Detailed Alternatives* to check and eliminate any unsatisfactory design. Every time a function needs to check if a steel section has been found, then *Check_Design* is called with the parameter *Element* and it refers to ruleset *Rules_For_Checking_Detailed_ Elements_Alternatives*, which contains one rule *R1_About_Steel Sections*. This check is used with all design option tests to ensure that a section has actually been found.

In the NOVA prototype, the detailing processes are started when the user selects the appropriate input button. However, in a finished version of the system the program would do this automatically. Initially the user requests the system to estimate assumed floor sizes, then the user sets the initial sizes for beams and columns and then selects the *Detail_The_Vertical_Subsystem* input button. There are two common series of functions, which are executed for all design options, and which estimate the floor and beam and column sizes. Then three alternate process flows are used for detailing the rigid frame, braced frame and shear wall partial designs.

When the *Detail_Vertical_System* input button is selected, the detailing function is called and an initial list of items to be designed and analysed, *Global:New_Designs_In_Vert_2D_W_Loc* is created. This list consists of the partial designs on the fringe of the search tree, which have been created at the *Vertical_2D_Wide_Location_Level* of the hierarchy. Each item on the list is detailed in turn; the Kappa messaging facility is used to initiate the appropriate design method. This messaging system is described later in this section.

Details of the design and programming of the Detailing functions have been omitted to restrict the size of this report. However, Appendix C describes the design of the detailing programming for braced frame options. Similar functions are applied to rigid frame and shear wall partial designs; however, their descriptions have not been included in this report.

- **Evaluation**

In the evaluation stage all the likely feasible alternatives are considered in terms of different features such as cost, time to build and overall height. The method used in the design of this system is based upon the one used in the DOLMEN system. It has the following steps:

- Identify relevant features;

- Formulate the features into soft constraints;
- Allocate soft constraint to target, which can be maximised or minimised or accepted at any value;
- Evaluate and rank proposed designs; and
- Allow the user to change, reiterate this step as many times as necessary;

Evaluation criteria were based on approximations of the following features: construction cost, time, clear-space, sway, column size and height. The user could select any or all of these features for a particular design. For each feature selected, a target setting was defined, which could be one of the following: the maximum value, the minimum value, the objective (ie. maximise, minimise, or accept any value), and the importance of the feature (ie. irrelevant, not-so, quite, very, and extremely). For example, the cost of a building might have a maximum value of $ 450,000, a minimum value of $ 300, 000, an objective of minimise, and an importance of extremely. This implies that it is extremely important to minimise the cost of the building, as long as it costs not less than $ 300, 000.

The system used heuristic rules to establish target settings. These rules were invoked at the start of the evaluation stage. For example rule *Rule_For_Prestigious_Building_Cost* stated that if the function of the building (input by the user, at the specifications stage) was such that the building would be considered to be prestigious, then the following target settings were to be established for the cost feature:

- *maximum value* worked out as a function of total floor area.
- *minimum value* worked out as a function of total floor area.
- *objective* minimise
- *importance* very

Once the values had been calculated for each of the evaluation features, they were presented to the user for verification. If a value was unacceptable, the user was able to re-specify the

required criteria, after which the system proceeded to evaluate each alternative. It did this by first computing a percentage optimisation value for each feature. This was, in effect, the degree to which the value of the feature (as calculated for each alternative by a procedure associated with that feature) approached the optimum value, so long as it was within the specified range. For example, if the objective, for an evaluation feature was to minimise the value, then the formula for determining the percentage optimisation was calculated as:

$$\frac{(\text{maximum value}) - (\text{feature value})}{(\text{maximum value}) - (\text{minimum value})} * 100\%$$

If the predetermined maximum cost was $ 450,000, the minimum cost was $ 300,000, and the calculated feature value was $420,000, then the percentage optimisation for the feature was found as follows:

$$\frac{(450{,}000 - 420{,}000)}{450{,}000 - 300{,}000} * 100\% = \frac{30{,}000}{150{,}000} * 100\% = 20\%$$

After the percentage optimisations had been calculated for each feature they were weighted and accumulated to form an evaluation value for the building. The weighting process associated a numerical value with the importance of each feature. For example, *irrelevant* corresponds to 0, *not-so* corresponds to 1, *quite* corresponds to 2, *very* corresponds to 3, and *extremely* corresponds to 4. If the importance target setting for cost was *extremely*, then the weighted percentage optimisation was: 20 * 4 = 80. Similar values were calculated for each feature selected by the user. The values calculated were then accumulated to determine the building's total evaluation value.

When an evaluation score had been determined for each partial design, then the system ranked them and displayed the best '*n*' alternatives, where the number '*n*' had been determined by the user during the specification stage. If he/she did not verify the system's selections, then an option to re-select the number of design alternatives to be considered was

provided. Additionally, the user could revert to altering the target settings, thus invoking the re-evaluation of the alternatives.

The system repeated the evaluation process, for the remaining design alternatives after it had completed the design of the flooring system. This provided the user with a final ranked list of designs. The user was again offered the options to reselect an alternative or modify the target settings. The same evaluation features were used for this evaluation, though the calculation methods differed slightly in some cases, because more accurate information had been made available, when the floors were designed.

In its present state of development the system only performs evaluation after the whole search tree has been completed and it is only coded to evaluate the vertical structural subsystem.

7.3 Difficulties Encountered During the Development Project

- **Difficulties in Analysing Preliminary Structural Design**

Preliminary structural design is a form of conceptual design. The writer noted several references including Maher (1984) and Harty (1987) that referred to conceptual design as being ill defined and which also advised that it was a difficult area for which to provide computer software. It is a problem solving activity, which comprises a series of conceptual decision making tasks interspersed with a series of calculation tasks. It is often difficult to determine in advance, which particular design tasks may be required in a particular project and in what order the tasks are to be applied.

In order to develop computer software to support a given design project, the developer must be able to document precisely what it is that the designer will actually do during the project. This task is difficult because the designer may not proceed in a methodical or structured manner. For example the designer may:

- Switch the way he/she approaches a design task;
- Mix and match design techniques;
- Bypass certain preliminary steps;
- Take risks; and
- Be inspired or use very innovative techniques.

The research completed for this project indicated that system developers had used several different approaches in the provision of intelligent design software, including simulations of decomposition, design transformation and case based reasoning. Regardless of which approach was used, several sources indicated that a promising approach to software support for conceptual design, is one which provides a range of tools, which assist with various phases of the design project and which can be used in a flexible, interactive and iterative manner.

- **Difficulties in applying object-oriented techniques to knowledge based applications**

Graham (1994) says that object-orientation addresses two of the 3 key aspects, required to specify a proposed system, these are data and process. He adds that control of system behaviour is more difficult to integrate into an object model and in several of the approaches he had reviewed, control in the form of rules and/or constraints, appeared to be accommodated as an afterthought.

In this project the writer found it difficult to include the production rules in the object model, other than as a 'black box'. This approach appears to leave something to be desired, but the writer was unable to find a better way to include them, given the object-oriented analysis and design tools selected for the project.

- **Overlapping of the analysis and design phase**

It was difficult to manage the object-oriented stages of the development project. The writer was unable to clearly separate the analysis and design stages and was also unable to precisely distinguish which deliverables were worked upon in each stage. These difficulties resulted in the failure of the writer to produce accurate time estimates for project completion.

The purpose of analysis is to describe a problem, ie. to formulate a model of the problem domain, analysis is concerned with what happens rather than how it happens, and it focuses on behaviour not form. The primary purpose of design is to decide how the new system, which constitutes the solution to the problem, will be implemented. Design creates architecture for the evolving system and establishes common approaches that must be used with the disparate elements of the system. According to Booch (1991), design should begin as soon as a model of the system has been created. However, during this project the writer

produced several models and it was difficult to recognise, which model was the appropriate starting point for the design stage.

The conventional system development life cycle is a series of steps with gaps between them. The steps are well defined and are associated with clearly identified deliverables. The deliverable output by one step then becomes part of the input for the next step. However, as Henderson-Sellers (1992) notes object-orientation supports a seamless transition from phase to phase and this makes it difficult to pinpoint where one stage ends and another begins, likewise it is difficult to detect the point at which a deliverable should be achieved.

Summary

This chapter has described the completion of the analysis and design stages for a knowledge based PC design tool, which was intended to assist the engineer with preliminary structural design tasks. The final two stages were effected via of a simplified object-oriented analysis and design process, which was described in Cross (1996) and which used modeling techniques adapted from Rumbaugh et al. (1991) and Embley et al. (1992).

The object-oriented analysis stage provided a model of the "real world" design problem, by analysing the functional requirements required to support preliminary structural design. On completion of analysis, the design stage developed the systems architecture for the new system. This architecture consisted of notes regarding the structure of the design object, diagrams and flow charts for the algorithmic functions. This chapter has also described several difficulties encountered during completion of the development process. These difficulties are described under the following headings:

- Difficulties in analysing preliminary structural design,
- Difficulties in applying object-oriented techniques to knowledge based application, and
- Overlapping of the analysis and design phases.

CHAPTER 8. The Kappa-PC Application Development Toolkit

8.1 Introduction and Description

Kappa-PC (Intellicorp 1996) is an application development system for PCs. It is designed to provide the following:

- Graphical object-oriented application development in a standard C implementation;
- Integration with existing MS-Windows applications including support for Windows Dynamic Data Exchange (DDE), and Dynamic Link Libraries (DLLs);
- Production of ANSI C program code executables, which allow for the efficient distribution of the finished programs;
- Interfaces to SQL databases, spreadsheet programs and CAD packages; and
- Expert system tools, including an inference system.

In particular it can be used as a domain-independent expert system shell. Hasan et al. (1994), Kiernan et al. (1996) and Tsang and Bloor (1994) have indicated that Kappa-PC has been used to produce expert systems quickly and economically. From research of their work it appeared that Kappa-PC would be a suitable platform on which to develop a system to support preliminary structural design.

The writer therefore installed a copy of Kappa-PC on an IBM 600E Thinkpad laptop and proceeded to explore its system development capabilities. The copy used in the study was Version 2.4 of the Kappa-PC Applications Development system as supplied by the Intellicorp Corporation. This required a 386 type PC or above with a math co-processor, it also needed 4 MB RAM or higher, 4 Mb of hard disk space and the Microsoft Windows 3.1 or one of the Windows 9x series of operating systems. The PC used in the study had an Intel Pentium chip, with 8 MB RAM and the PC also had 4 Gb storage and ran the

Windows 98 operating system, which supports Kappa-PC as a 16 bit Windows application. The Borland Turbo C++ version 4.5 compiler was also installed to facilitate the generation of standard ANSI C code.

Intellicorp (Intellicorp 1996), describe Kappa-PC as a complete development environment, which provides a wide range of edit tools and debuggers for designing and running applications. In the Kappa-PC system, the active components of the application domain are represented by data structures called objects. These objects can be either classes or instances within classes and they may represent concrete things like building subsystems, such as the floors or the walls or components like beams and columns. The objects can also represent intangible concepts like cost or evaluation criteria. A developer can link objects together into an object hierarchy to represent the equivalent relationships among the objects in a model abstracted from a particular domain.

The object-oriented programming tools within Kappa-PC can be used to provide these objects with methods, which contain algorithmic code like that found in the functions in conventional programs. Once the objects and methods have been identified for a knowledge base, then the system can be developed. System development commences with the production of a specification to describe how the objects are to behave and how the system will reason about the objects. Systems built on Kappa-PC usually require a set of pre-written rules, where each rule specifies a set of conditions and a set of conclusions to be made if the conditions are true. The conclusions may represent logical deductions about the objects in the knowledge base and how they might change over time.

In Kappa-PC each rule is a relatively independent module and a reasoning system can be built gradually, rule by rule. Kappa-PC also allows the developer to use object-oriented programming to combine and unify many standard AI methodologies such as, frame-based

representation, production rules, demons or monitors and graphics into a comprehensive hybrid system.

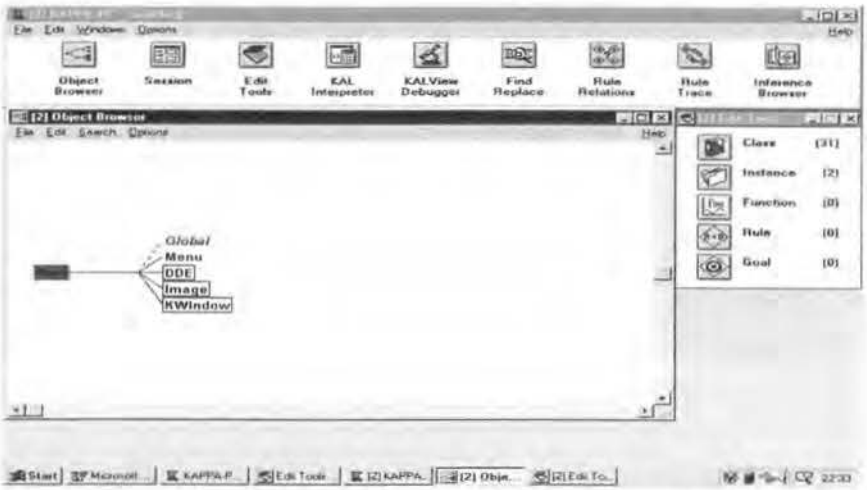


Figure 8.1 Kappa-PC application development input screens

To start Kappa-PC the user can double-click on the Kappa-PC icon. The package will open up with a series of windows, which appear as in Figure 8.1. These windows control the operation of the Kappa system and allow the user to bring into view a number of other windows. Figure 8.1 shows the Object Browser, which allows the user to view graphically and edit the class structure of a program, and to access the EditTools windows, which provides the facility to edit data objects, which consist of classes and instances, rules, goals and functions

User interaction with Kappa-PC proceeds graphically, the system being accesses via a mouse, or by typing into one of the five custom editors or via the Interpreter Window, which allows commands, statements and functions to be input and executed interactively. Graphical input can also be effected via the Object Browser or via one of Kappa-PC's Session Windows.

The following sections briefly outline the specific Kappa-PC facilities used in the study.

8.2 Kappa-PC Structures Used to Describe Objects

- **Objects**

Objects are represented in Kappa-PC as classes and instances of classes. These can be organised into hierarchies or taxonomies using subclass and instance relations. Figure 8.2 reproduces the Object Browser, which displays part of the knowledge base developed during the study. These objects are all classes and the links between them are shown. They represent the R600, Rib Mould class, which represents those moulds with a grid size of 600mm. Within this class of Rib-Moulds there are 4 main types of moulds, based on mould depth in mm, 175, 250, 325 and 400, which are represented by the subclasses R600-175, R600-250, R600-325, R600-400.

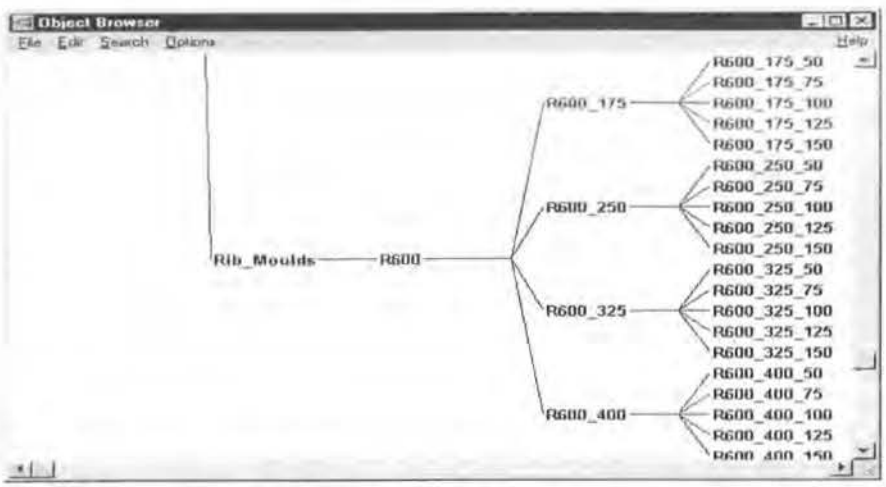


Figure 8.2 The Rib-Moulds Object Hierarchy

The solid lines indicate subclass links, which partition the Rib-Moulds class. These links represent *is_a_subclass* (*is_a_member_of*) relationships. Kappa-PC also provides for the *is_a_kind_of* or instance_of relationship. However, these are the only relationships provided for explicitly in Kappa-PC and other kinds of relationships must be implemented indirectly. For example a developer can use the slots in objects to create links to other objects in order to represent association type relationships. These *is_a_member_of* relationships are used throughout the design tool system created during the study to

construct representations of the design alternatives at different levels and of the hierarchically organised product model.

The links between objects also provide the paths via which objects inherit attributes from other objects higher in the hierarchy. Each class can have any number of slots and Kappa-PC provides two kinds of slots, *member* and *own*. The member slots of a class are inherited by its subclasses while the own slots are not. Furthermore, when a subclass inherits a member slot the slot also acts as a member slot for the subclass, if this subclass is a subclass of the parent. Otherwise it inherits it as an own slot and cannot pass it on to its subclass.

The user can create classes graphically in the object browser window or create them indirectly by using the class edit tool. To use the object browser, the user can click on the class *'root'* and then select *'AddSubClass'* from the edit menu. The user then inputs an appropriate class name.

- **Slots**

Kappa-PC provides a data type, referred to as a slot, which resides in the Kappa object, which may be either a class or instance. The user can update the slots to tailor an object so that it may represent the important properties of a real object. Each slot can be used to describe a characteristic or attribute of the object. To specify the attribute, the user assigns a value to the slot. For example, within the Rib Mould Class noted above, the user has created slots for *average-rib-width, depth-of-topping, gridsize, mould-depth, supplier* and *total-depth*. These attributes complete the description of the 600-mm. size mould and are displayed in the Class Editor window shown in Figure 8.3.

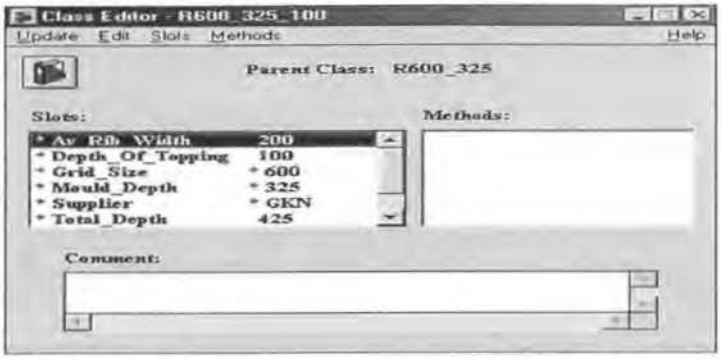


Figure 8.3 Class Editor showing the slots in a Rib Mould.

Slots are inherited down the object hierarchy, and as the hierarchy grows, the classes lower down gradually accumulate inherited slots. As noted above, objects can have their own slots and they can inherit slots from ancestor classes, ie. classes above them in the class hierarchy.

When an object inherits a slot from an ancestor, the object does not have to maintain the inherited slot value; the user can make the slot local to the subclass and then insert a different value from the one inherited by the slot. Kappa-PC also allows slot values to be changed programmatically. This feature is very useful for programming knowledge-based systems. Slot inheritance provides a shortcut to updating attribute values throughout the hierarchy. If a slot value is changed at a point in the hierarchy then the change will be reflected in values of the slots lower down the hierarchy, which have been inherited down through the hierarchy.

Local slots describe features that are private to the object that contains them. If the object is a class, its local slots describe that class itself (as opposed to its members). If the object is an instance, its local slots provide information about that particular instance. The user can input and change slot values using the slot editor, which is shown in Figure 8.4.

Figure 8.4 The Slot Editor

Once a slot is made local and the value of the slot is changed, all classes and instances that subsequently inherit the slot get the new value. This feature was used in the system designed during the study. As the system's search tree of design objects grows, new partial designs are added and at certain levels in the tree detailing calculations are done to estimate and fix the initial sizes of component parts. These calculations result in changes to various slot values in the design objects. These changes are effected programmatically using a variety of assignment functions and the new values are then reflected in the subsequent levels in the hierarchy. This shadowing effect of inheritance is a useful feature of object-oriented programming; all objects below an object with a local slot are affected by the change. The following paragraph describes several types of slot assignments, which are provided by Kappa-PC and which were used in the study.

The Kappa-PC *SetValue* command assigns a value in a single-valued slot or a set of values in a multiple-valued slot. The code fragment shown in item (i) shows how the writer set up a slot in the global instance to act as a loop counter.

(i) SetValue(Global:Loopcounter, 1);

This *SetValue* function sets the slot value at 1. Kappa-PC uses multiple valued slots to hold lists and has several functions, which emulate LISP list processing functions. For example:

(ii) SetValue(Global:List_Of_Designs, Vertical-3D, Vertical-2D-N);

In item (ii) above the function assigns the value of multiple-valued slot *List_Of_Designs*, with the two items, Vertical-3D, andVertical-2D-N, thereby creating a list and returning the

values of the list, ==> Vertical-3D, Vertical-2D-N. The *AppendToList* function adds items to the end of a list and the *GetNthItem* function returns part of the list. For example:

(iii) AppendToList(Global:List, a,b,c);

returns ==> x,y,z,a,b,c and adds items a,b and c to the end of the list.

(iv) GetNthItem(Global:List, 5);

returns ==> b

Kappa-PC provides a set of standard slot options to describe and manipulate object slot values. These slot options describe slots in much the same way that slots describe the attributes of objects. Furthermore, a given slot can have many different options, while at the same time having no value assigned. If a slot does not have a value, at a point in time, then Kappa-PC assigns it the value NULL. Also if a slot value is reset (and it did not have a value before it was assigned one), the new value of the slot will be NULL.

The types of slot options provided by Kappa-PC are:

- Cardinality (single or multiple), this specifies the number of slot values allowed, if multiple is chosen the slot can have multiple values, which are input in the form of a list;
- Allowable Values, this describes the set of allowable slot values, ie. a Boolean slot would have two values; TRUE and FALSE;
- Value Type, this option controls the type of the slot values, ie. text, number, Boolean or object, which can be the name of a class or an instance;
- Slot Inheritance. This option controls the inheritance behaviour of the slots; the values of which can be passed down the hierarchy or stopped at this object using the Slot Inheritance option; and

- Change monitors or demons, these options include the *If Needed, WhenAccessed, Before Change* and *After Cha*nge monitors. These are methods that are activated when object and slot pairs are accessed. They are used extensively in systems, which rely on rule-based reasoning. Monitors may be defined as private functions or functions that change the value of slots elsewhere in the object hierarchy. The If Needed option contains the name of a *method* in this object. The method is automatically executed when the value of the slot is requested and there is no value in the slot ie. when a value is needed. Likewise if the WhenAccessed option is attached to the slot, then the method is executed when the slot is accessed, even if the value of the slot is known.

- **Methods**

Apart from information that describes the object's characteristics each object also contains information that specifies its behaviour. Each action that an object can carry out is represented by a method, which is a procedure, usually written as a KAL program function. Furthermore, Kappa-PC facilitates the characteristic object-oriented process of method activation by programmatically sending and receiving messages. When an object receives a message that corresponds to one of its methods that method is activated and the object carries out whatever procedure is specified by the method. Kappa objects inherit methods in the same way that they inherit slots and this feature has been used during the study to organise the behaviour of the new system.

Kappa-PC methods provide for the object-oriented characteristic of polymorphism. Thus different Kappa objects can have their own individual methods with the same name as the methods in other objects. This then allows the different objects to respond in their own characteristic way, to the same message put out by the application. This facility was used in the new design system to incorporate an element of polymorphism. Thus the new application can issue a single instruction to commence the detailing process of all the partial

design objects in the vertical subsystem. This is done when the design has proceeded down the design hierarchy as far as producing partial designs at the *Vertical-2D-Wide-Location* level. The instruction to commence detailing is then passed round the design hierarchy at that level, using a series of messages and each object reacts according to its type. The user can create object methods via the method editor, which is shown in Figure 8.5.

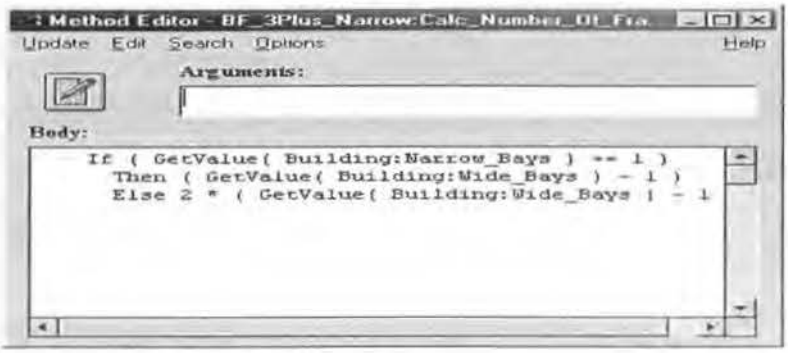


Figure 8.5 The Kappa-PC Method Editor

The method shown in the figure, is a method for calculating the number of frames in the *BF-3Plus-Narrow* location object. Methods can also be created programmatically using the *MakeMethod* function; however, this facility was not used in the study.

A method can be coded to include any KAL function or sequence of functions. Each method has three default arguments: *self, theParent* and *theOwner.* The value of the *self* variable is the object that receives the message and it allows methods to access the values of other slots in the same object. They can also initiate other methods in the same object by sending the message to *self.* Methods can perform several kinds of actions:

- Change the state of the application, generally by changing slot values in an object;
- Send messages, either to the same object or to other objects; and
- Activate other facilities of the Kappa-PC system, such as rule-based reasoning or data access.

If a method causes changes in an application, the changes are typically to slot values in the object that receives the message. If changes need to be made outside of the object that receives the message, then appropriate messages can be sent to the necessary objects.

Method inheritance acts in a similar way to the inheritance of slot values. It can be used efficiently to create and refine the behaviour of objects. Like slots, methods can be inherited, made local and edited at the class or instance level. If the object contains a method, any of its subclasses that do not contain a method of the same name will inherit the method unchanged. If a message is sent to an object to invoke a named method then that method will be invoked in the object, which receives the message, not the other objects in the hierarchy, which may have methods with the same name.

- **Object-oriented Programming**

The Kappa-PC objects, which have been described above, allow the user to describe real world objects and support the main characteristics of object-oriented programming, which are: inheritance, encapsulation and polymorphism.

Inheritance has been used in this study to achieve conceptual clarity via the object model created for the study; thus similar types of objects are grouped into subclasses, which share a common parent. For example, design options, which include *Rib-moulds, Waffle-moulds* and *Steel-decks* are grouped into their own distinct class groupings. Each of these groupings has a common parent class, which has the generic attributes for the whole hierarchy.

In the study the writer also created an *Alternatives* class, to allow the system to refer to the design options, which include the same floor alternatives, collectively. Thus during the generation of alternatives, these objects or at least a subset of their attributes can be included in the subclass of floor alternatives. Thus the floor alternative class contains the *Ribbed-*

slab, Waffle-slabs and *Steel-deck* classes, which in turn include respectively the *Rib-moulds, Waffle-moulds* and *Steel-decks.* This class is shown in the object browser display in Figure 8.6.

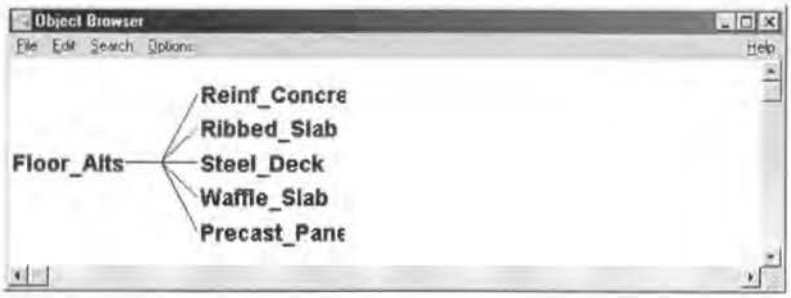


Figure 8.6 The Floor Alternatives Class

Inheritance simplifies object creation. Thus if a new class is to be created, which is similar to an existing one, then it can be created as a subclass of the existing class. The new class automatically inherits its parent slots and the user need add only the new slots, which are required to differentiate it from its parents. This facility is used in the study system during the creation of the design objects, which make up the search tree of design alternatives. The generic class *Building* is placed at the root of this tree and the inheritance mechanism is used to create new subclasses at each design level.

Kappa-PC can only support single inheritance and the system created during the study required additional functional coding to provide for the multiple inheritance required during the generation of the new levels in the tree.

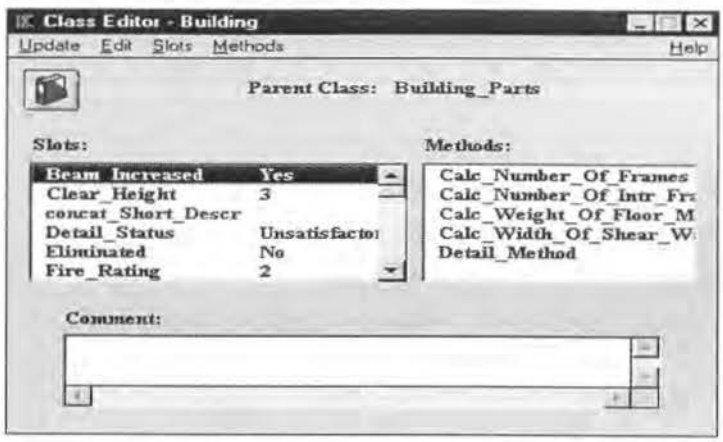


Figure 8.7 The generic class Building, the root of the NOVA search tree.

8.3 The Kappa-PC Application Language

- **KAL**

KAL is a high-level application development language, which allows users to program the functions required to support procedural programming. During the development of the system used in the study procedural programming was used extensively to program the design synthesis and evaluation activities.

KAL can be used to manipulate application objects, mathematical functions, strings, lists, files, control blocks, windows, popup menus, input forms, application graphics, interfaces, and system access. It also allows the user to write functions, methods and rules, create message passing schemes and activate the inference engine, to complete calls to external functions, employ graphics and animation and to facilitate data access.

KAL source code can be compiled to ANSI C. Furthermore, a suitable C compiler can further compile this C code into a dynamic link library (DLL), which runs an average three times faster than the original interpreted KAL code.

As well as object-oriented programming KAL allows the user limited access to non-object local variables, which are used with *Let* and loop constructs and which are settable, ie. they can be used in assignment statements.

- **KAL Source Code Debugger**

The debugger provides the user with a means to debug KAL source code. The user can view functions, methods and the execution stack and can set break points for functions and methods. The user can also set watches on the value of object slots or any other coding entity, by selection. The debugger has two modes; 'step-over' and 'trace-into'. In addition Kappa-PC provides a 'Find/Replace Utility' to allow for local and global find and replace capabilities. Figure 8.8 shows a typical debugger display, this one was created when the

writer was tracing the execution of a function (the *Select_Reinf_Centres* function) during the study project.

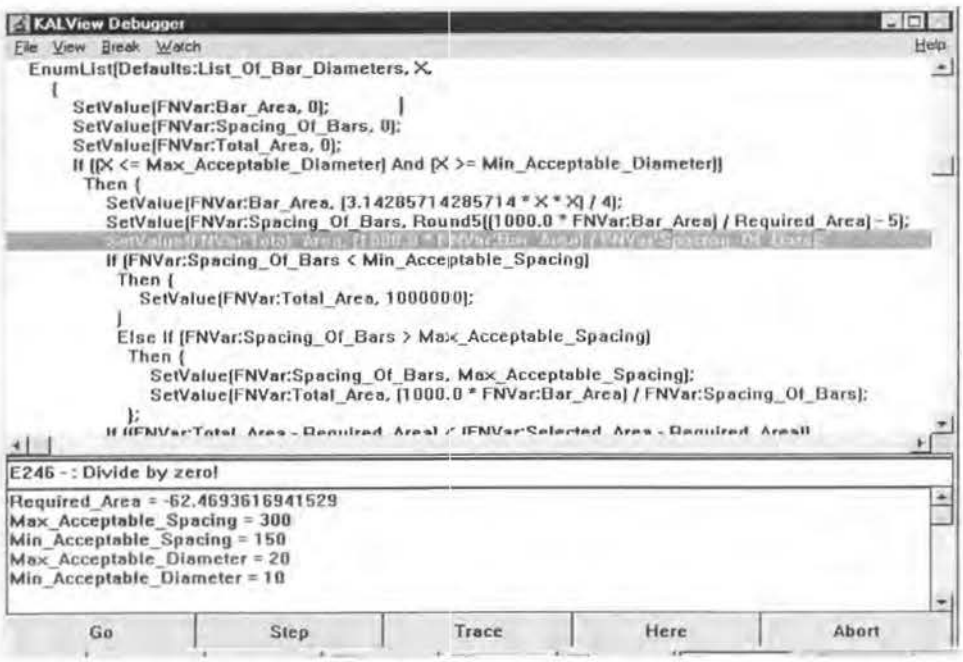


Figure 8.8 Debugger Display During Function Trace.

- **Local Variables**

The KAL language allows local variables, declared in function code with certain key words including *Let, For, ForAll, AreAll? EnumList*, as well as variables, which are used as arguments in functions, methods and rules, to be assigned in the body of KAL code in scope.

For example, the following code is allowed:

Let [x 0] While (x < 10) x == x + 1;

This evaluates an expression with temporary arguments, ie. x1...x10, which are mapped into the expression. However, the Let only maps x1...x10 within the scope of the expression statement.

8.4 Kappa-PC Reasoning Mechanism

Kappa-PC provides facilities for rule-based reasoning, which allows the user to develop rule-based systems. These systems represent knowledge in terms of a set of rules, which determine what the system should do or what conclusions the user should draw in different situations.

In Kappa-PC the rules are represented as "if" (conditions) and "then" (actions) statements, they are associated with a subset of facts, represented as a set of object and slot pairs drawn from the domain knowledge in the system. The Kappa-PC reasoning mechanism consists of a combination of the rules and object slots, which are organized into an inference network and a system interpreter, which controls the application of the rules.

The interpreter has two main modes of reasoning: agenda-controlled forward chaining and goal-driven backward chaining. The study system employs forward chaining through out. In forward chaining the facts in the system are held in working memory, which is continually updated as rules are invoked. The rules represent possible actions to take when predetermined events change these facts in working memory. These actions usually involve adding or deleting items from working memory.

The interpreter controls the application of the rules, given the contents of working memory, and thus controls the actions taken by the system. The interpreter works through the rules in cyclic manner as follows:

- Check to find rules, which have the conditions satisfied;
- Select a rule, based on a predetermined strategy; and
- Perform the action in the action part of the rule, thereby modifying current working memory.

Kappa-PC has several features to enhance its rule-based reasoning, these include four rule-firing schemes: depth-first, breadth-first, best-first, and selective, pattern matching on objects. It also allows priorities to be set for conflict resolution, and provides a flexible explanation facility to explain the conclusions arrived at by the inference mechanism. Kappa-PC also provides features, which allow a developer to debug the inferencing scheme being used. These include, rule trace and break capabilities, slot trace and break capabilities and the ability to "step through the inferencing process".

These tools are accessed through three specialised editor windows in the development environment:

- The Rule Relations Window, which dynamically displays rule networks and interdependent rules. It displays "if" and "then" dependencies for related rules and allows browsing through the compiled rule network and provides interactive editing of rules and their relationships.
- The Rule Trace Window allows the user to specify application components to be examined during the inferencing process. It provides capabilities for active trace, where the user can step through inferencing one step at a time and can momentarily stop inferencing at pre-defined states, change parameters, and then resume the process. The rule trace window displays the active rule list, agenda contents, and trace outputs. The system provides a choice of automatic or active trace, as well as an interactive stepper mechanism.
- The Inference Browser Window facilitates graphical debugging of the rule systems and allows interactive editing of rules. It shows the active path, and the status of slots (known or unknown, which are to be queried from the user, or which are to be deduced

from rules), rules (active or inactive, to be expanded, rules pending, or fired to true or false), and goals (true, false, or unknown). It also provides a step mechanism.

- **Demonstration of the Kappa-PC Inferencing Mechanism**

In the following section the writer describes a simple KAL program, which demonstrates Kappa-PC's inferencing facilities and which also allows the writer to demonstrate Kappa-PC's rule trace facilities, which include the Rule Trace Window and the Inference Browser.. The example shows a trace through the system's rule base as it generates new conclusions. The program was written to operate on a fragment of a rule-base, which was described by Krishnamoorthy and Rajeev (1996).

The program's rule base contains the 9 rules shown below, which allow it to solve a series of structural design problems. On startup the systems prompts the user to input information concerning the number of stories proposed for the new structure and whether or not there are good quality bricks available. Using this information the system then establishes the required load bearing structure. It then requests more information concerning the structural subsystem and eventually it determines the type of floor system.

The rules in the system are shown in Table 8.1. The program has a simple session window, which is shown in Figure 8.9, and which allowed the user to operate the system.

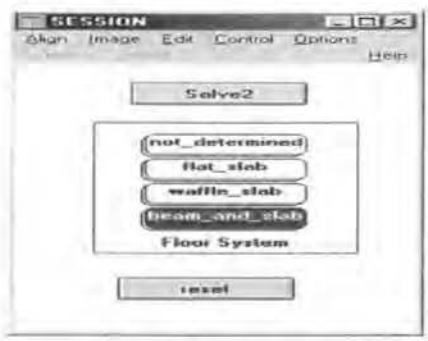


Figure 8.9 **Session window for the rule demonstration program**

****

RULE: 1
IF no_of_stories <= 5 **AND** good_quality_bricks #= available
THEN load_bearing ← masonry_wall

RULE: 2
IF no_of_stories <= 5 **AND** good_quality_bricks #= not_available
THEN load_bearing ← rcc_framed_structure

RULE: 3
IF no_of_stories > 5
THEN load_bearing ← rcc_framed_structure

RULE: 4
IF load_bearing #= rcc_framed_structure **AND** no_of_stories <= 20
THEN structural_system ← rcc_rigid_frame

RULE: 4a
IF load_bearing #= masonry_wall **AND** no_of_stories <= 5
THEN structural_system ← rcc_rigid_frame

RULE: 5
IF load_bearing #= rcc_framed_structure **AND** no_of_stories <= 35 **AND** no_of_stories > 20
THEN structural_system ← rcc_frame_with_shear_wall

RULE: 6
IF structural_system #= rcc_rigid_frame **AND** maximum_span_in_M < 10 **AND** clear_height_in_M < 3 **AND** clear_height_in_M > 2.5
THEN floor_system ← flat_slab

RULE: 7
IF structural_system #= rcc_rigid_frame **AND** maximum_span_in_M > 8 **AND** maximum_span_in_M < 20 **AND** clear_height_in_M > 3
THEN floor_system ← waffle_slab

RULE: 8
IF structural_system #= rcc_rigid_frame **AND** maximum_span_in_M < 8 **AND** clear_height_in_M > 3
THEN floor_system ← beam_and_slab

****

Table 8.1 Rules for the demonstration system

Figure 8.10 shows the program's rules in the Rule Relations Window, which dynamically displays rule networks and rule interdependencies. This window allows the user to query the rule objects in the display, using the right hand side mouse button. Figure 8.11 shows the results obtained when the system is queried to determine, which object slot pairs are related to rule 1.

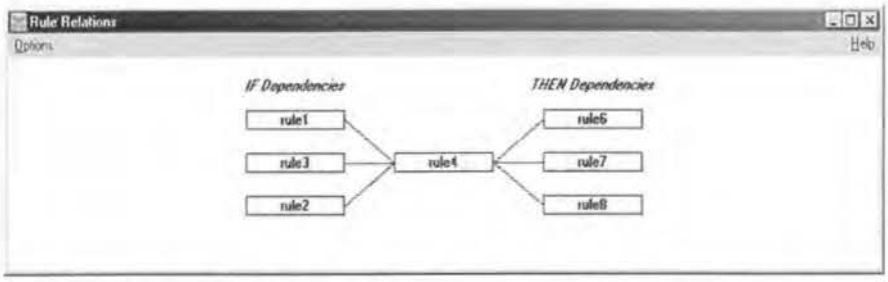


Figure 8.10 **Rule Relations Window for the rule demonstration program**

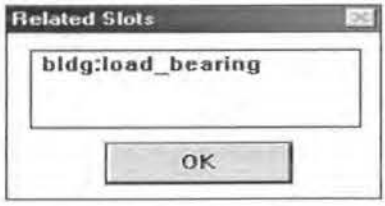


Figure 8.11 **Rule Relations Window query for rule 1**

The Rule Trace Window allows the developer to view the rules that the inference engine invokes in the form of a transcript and to follow the impact of the reasoning process on particular slots in the knowledge base. In a trace the developer can see how the system generates new conclusions, and can trace the source of errors in the application's knowledge base. The Rule Trace window may be used to trace either forward chaining or backward chaining, as the system goes through each particular stage. Figure 8.12 shows the Trace Setup dialog, which must be used to set tracing and breaking on particular rules and/or slots before the reasoning process is initiated. For this demonstration, the writer set up tracing on all 9 rules.

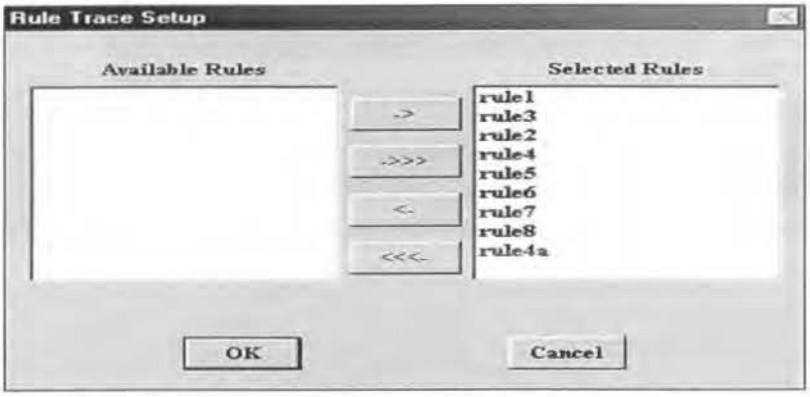


Figure 8.12 **Rule Trace Window set up dialog**

Once the traces had been set up the writer used the Control Menu to begin the reasoning process, he selected the BackwardChain option to begin chaining, this function calls the KAL BackwardChain function. In the demonstration program the rules were organized into a ruleset, named *Global:rules*. This ruleset was represented by a multiple or list slot in the Global instance, which contained the names of the nine rules used. This allowed the user to refer to the rules collectively in the program code. The string *goal2, Global:rules* was input as the argument to the function, see Figure 8.13. Figure 8.14 shows the query window output by the system, which seeks missing information, as it goes through the reasoning process.

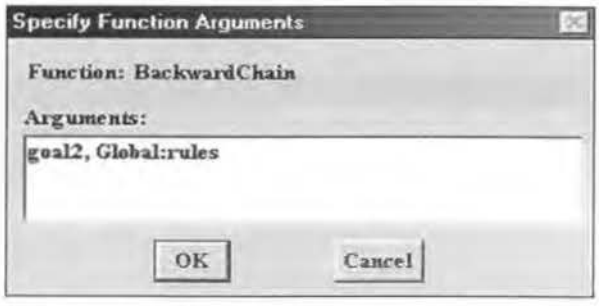


Figure 8.13 Input arguments to BackwardChain rule trace

The Rule Trace Window shows the results of the testing done by the system on the rules selected for the trace, this is shown in Figure 8.15.

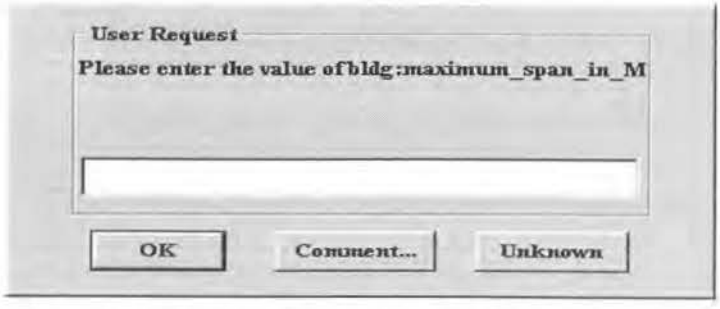


Figure 8.14 System query window output during reasoning

Figure 8.15 **Rule Trace Window showing the results of the testing**

The Inference Browser window allows the developer to view the rules that the inference engine invokes in the form of a graphical network. In the browser the developer sees how the system arrived at its conclusions by examining its lines of reasoning once the reasoning process is complete. The Inference Browser can also be used to trace the source of errors in the application's knowledge base.

Clicking the mouse on the appropriate icon in the Kappa-PC Window starts the Inference Browser. The system then requests the user to select a function, for this demonstration it was necessary to select the BackwardChain function and supply an appropriate argument as shown in Figure 8.16.

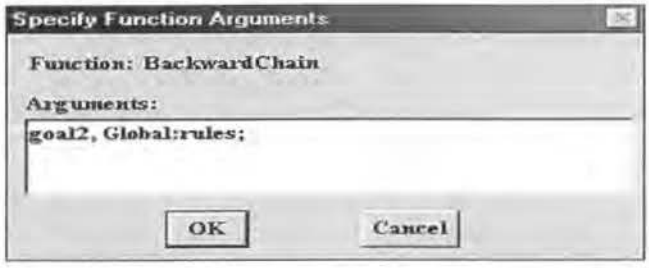


Figure 8.16 **Input arguments to BackwardChain control using Inference Browser**

The system then acknowledges the input argument and sets off to test the rules in the sequence required to satisfy goal2, which in this case was a requirement to determine the floor system.

Figure 8.17 **System announces the start of the inference process.**

The Inference Browser then graphically displays the chaining relations among rules. In this demonstration the writer used it in a stepwise manner, proceeding to test each rule. A series of displays was produced as the Inference Browser worked through the chain of reasoning in the demonstration program. These displays are shown in figure 8.18. As each rule was tested, the system displayed the newly asserted facts, summarized in terms of object:slot pairs and the rules considered. The dashed lines in the displays, link new facts and rules whose conclusions mention the new facts stored in the appropriate object:slot pairs. Among the rules considered, only some apply. Applicable rules have solid lines leading from them toward the facts (pairs), which they mention in their premise. From the window displays it can be seen that the inference browser is a useful tool for analysing the inferencing process and debugging the system once it has been tested.

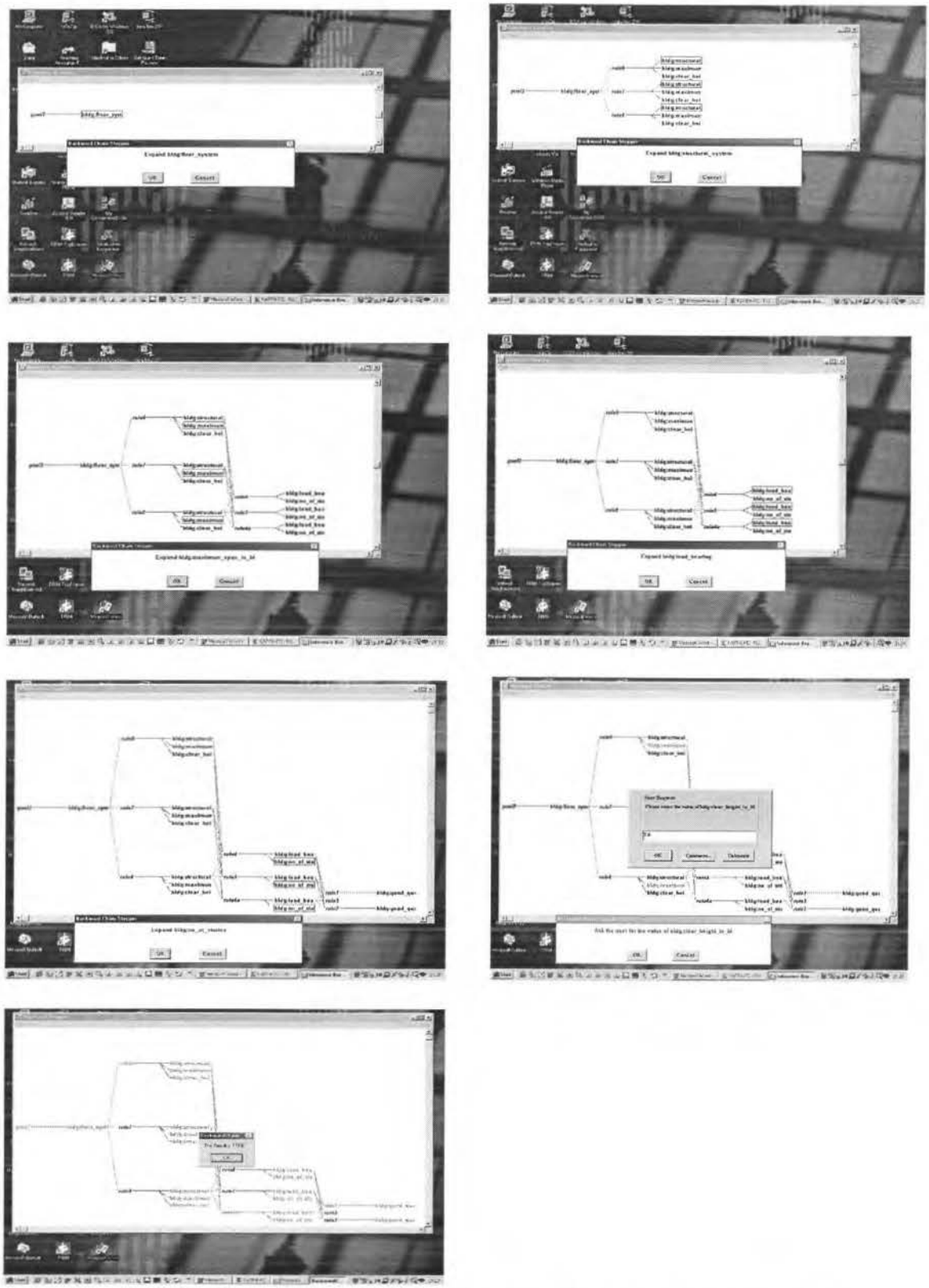


Figure 8.18 **Inference Browser tracing progress of demonstration program**

8.5 Difficulties Encountered in using Kappa-PC

- **Long Learning Curve**

The writer found that there was a steep learning curve to be completed if one was to use Kappa-PC effectively. The system has facilities to support object-oriented programming and at the same time it has the inferencing mechanism necessary for logical programming.

This task is made more difficult due to the large range of specialist debugging and tracing tools both for the KAL language and for the inferencing mechanism.

To use the system properly the user has to understand how to integrate the object hierarchies used to represent domain objects with the production rules needed for inferencing. The writer noted that certain programming tasks might be achieved by using either object-oriented programming or by using the inferencing capability provided by production rules. Unfortunately there are few sources of reference to guide the programmer as to which is suitable in a given case.

Summary

This chapter has described those Kappa-PC facilities used to implement the prototype design tool system.

CHAPTER 9. Implementation of the Object-Oriented Design

9.1 Design Architecture for the NOVA Design Tool on Kappa-PC

This chapter describes how the design for the new system was implemented using Kappa-PC. At the completion of this report the new system, referred to the NOVA design tool system, had reached the stage of working prototype,. A simplified overview of the system is provided in Figure 9.1. The prototype has a Windows based graphical user interface. This interface allows the user to input building specifications changes to design parameters and changes to the evaluation features. During design synthesis, the system displays the current state of the design process. The user can monitor the synthesis process as a tree of design solutions is generated and displayed in the object browser window. The user interface also allows the user to display ranked lists of alternative designs and to display the details of individual designs.

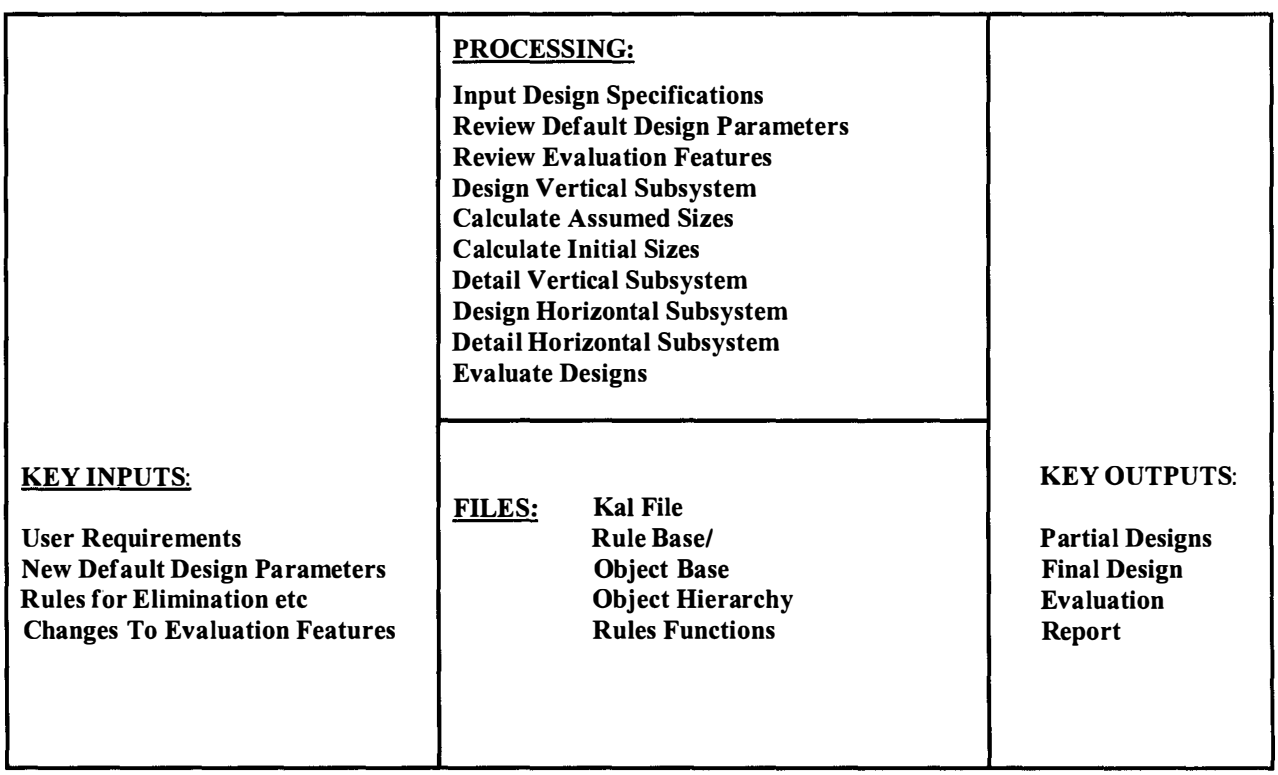


Figure 9.1 Overview diagram of the NOVA preliminary structural design tool

The overall organisation of the NOVA prototype is shown in Figure 9.1.

Heuristic design experience is represented in the NOVA system as a rule-base, which is used in conjunction with the inferencing mechanism. The rule base, which is shown in Table 9.1, includes:

- Rules for finding default design parameters;
- Rules to establish and customise the evaluation functions including target settings for individual features;
- Elimination rules for each level of the design hierarchy, which are used for pruning the search space during design synthesis; and subsequently to test designs after detailing has been completed.

Design Task	**Rule Sets**
Design rules	Rules_For_Finding_Default_Design_Parameters Rules_For_Finding_Default_Target_Settings Rules_For_Estimating_Floor_Depth
Elimination rules for Vertical Subsystem used during Synthesis	Rules_For_Vertical_3D_Elimination Rules_For_Vert_2D_Narrow_Elimination Rules_For_Vert_2D_Wide_Elimination Rules_For_Material_Elimination Rules_For_Vert_2D_Narrow_Location_Elimination Rules_For_Vert_2D_Wide_Location_Elimination
Elimination rules for Horizontal Subsystem used during Synthesis	Rules_For_Floor_Elimination Rules_For_Support_Beams_Elimination Rules_For_Intermed_Beams_Elimination
Elimination rules used during Vertical Subsystem Detailing	Rules_For_Checking_Detailed_Rigid_Frame_Alternatives Rules_For_Checking_Detailed_Braced_Frame_Alternatives Rules_For_Checking_Detailed_Shear_Wall_Alternatives
Elimination rules used during Horizontal Subsystem Detailing	Rules_For_Checking_Detailed_Pre_Panels_Alternatives Rules_For_Checking_Detailed_Rc_Slab_Alternatives Rules_For_Checking_Detailed_Ribbed_Slab_Alternatives Rules_For_Checking_Detailed_Steel_Deck_Alternatives Rules_For_Checking_Detailed_Waffle_Slab_Alternatives Rules_For_Checking_Detailed_Elements_Alternatives
Evaluation and Ranking rules	Rules_For_Ranking_Location_Alternatives

Table 9.1 NOVA System Rule Base

NOVA has a knowledge base, which includes decomposition, planning, constraint and evaluation knowledge. The decomposition knowledge is represented in the system as a hierarchy of systems and subsystems, which are implemented as Kappa-PC classes. These

classes have attributes, which are represented in the slots, which contain descriptive values and have a set of procedures, which are represented by the methods attached to the classes.

The planning knowledge in the system includes a *Schedule* class, which has several slots, which contain lists of sequences of design goals and the sequence in which they are to be satisfied. These sequences, which are referred to by the program code determine the flow of processing, which effects the design synthesis.

The NOVA system accommodates hard and soft design constraints. The hard constraints are implemented via the elimination functions, which are supported by the rule sets in the knowledge base. Each constraint is a combination of design decisions and a corresponding design context that is deemed not feasible. The constraints are used during the synthesis process to eliminate infeasible alternatives.

Soft constraints are represented by numerical variables. They are represented by the design target attributes and their associated criteria, which are attributes of the *Evaluation Features* classes. There is a set of evaluation criteria for the synthesis of both the vertical 2D and horizontal subsystems. These targets, which may be achieved to a greater or lesser extent, are set by the user and are used in a series of evaluation functions, which calculate performance values for the design candidates. The performance values are based on the attributes of the design candidates in the search tree.

9.2. Implementation of the Structural Hierarchy in NOVA

The NOVA system represents structural subsystems as a hierarchy of Kappa classes. The relations between these classes reflect the interactions between their physical equivalents. The hierarchy in the NOVA model is based on that described by Lin (1981) and is similar to that used in both the HI-RISE and Dolmen systems. This hierarchy of classes is shown in Figure 9.2.

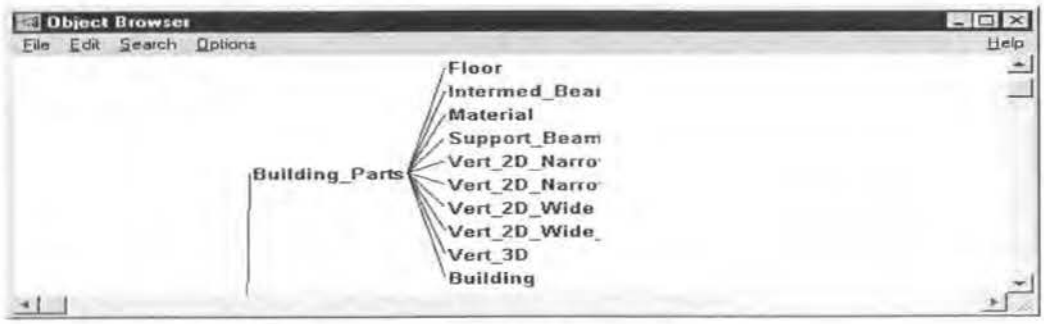


Figure 9.2 **NOVA's hierarchy of structural subsystems.**

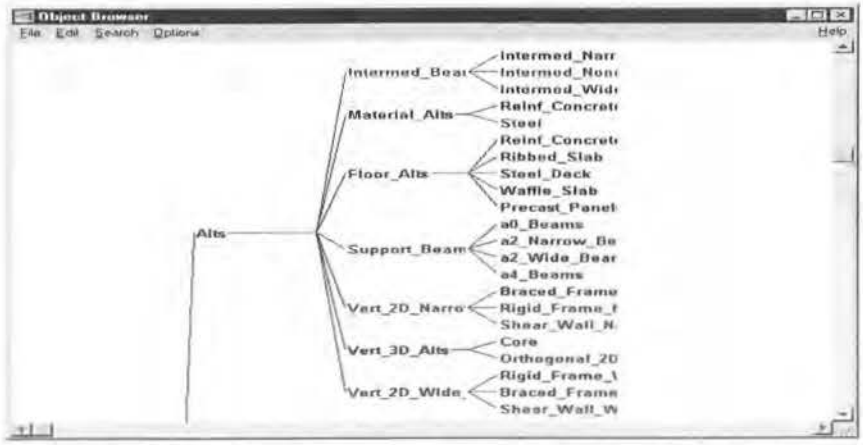


Figure 9.3 **NOVA's hierarchy of alternative subsystems.**

The system provides a set of alternative design options for each level of abstraction in its decomposition of the structural system. Thus at the material level the system can provide steel or reinforced concrete designs. Figure 9.3 shows the Alternatives hierarchy. Figure 9.4 shows the hierarchy of classes, which represent the location alternatives. Figure 9.5 shows part of the hierarchy of classes, making up the search tree of partial design objects, which the NOVA system has produced during the solution of a particular design problem.

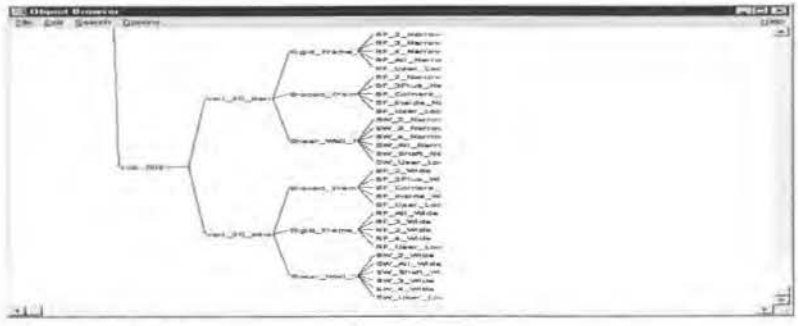


Figure 9.4 **NOVA's hierarchy of location alternatives.**

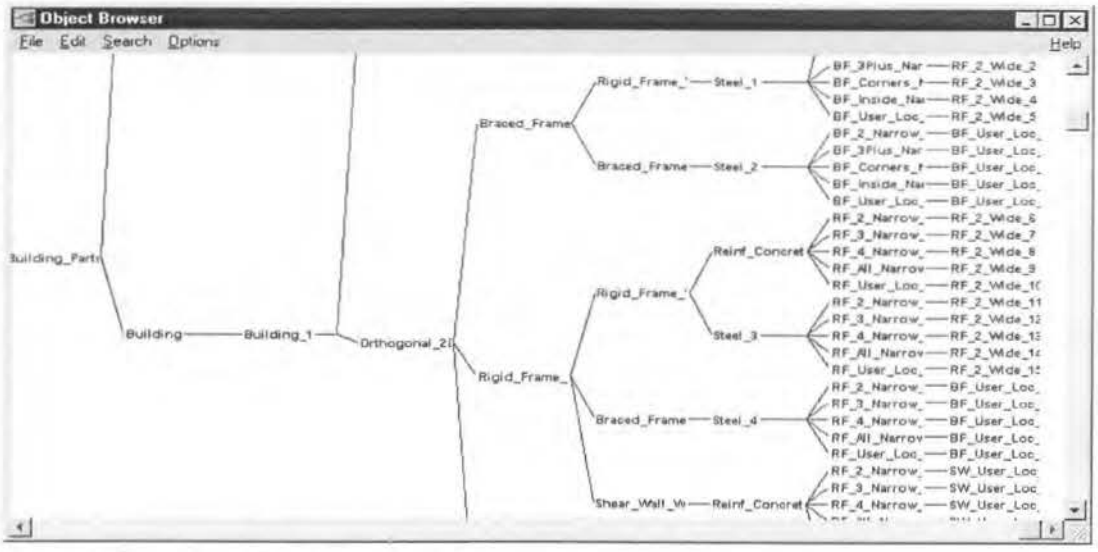


Figure 9.5 The NOVA Search Tree

9.3 Implementation of the Software for the Design Processes

- **Specification**

The software required to implement the Specification functions, consisted of a collection

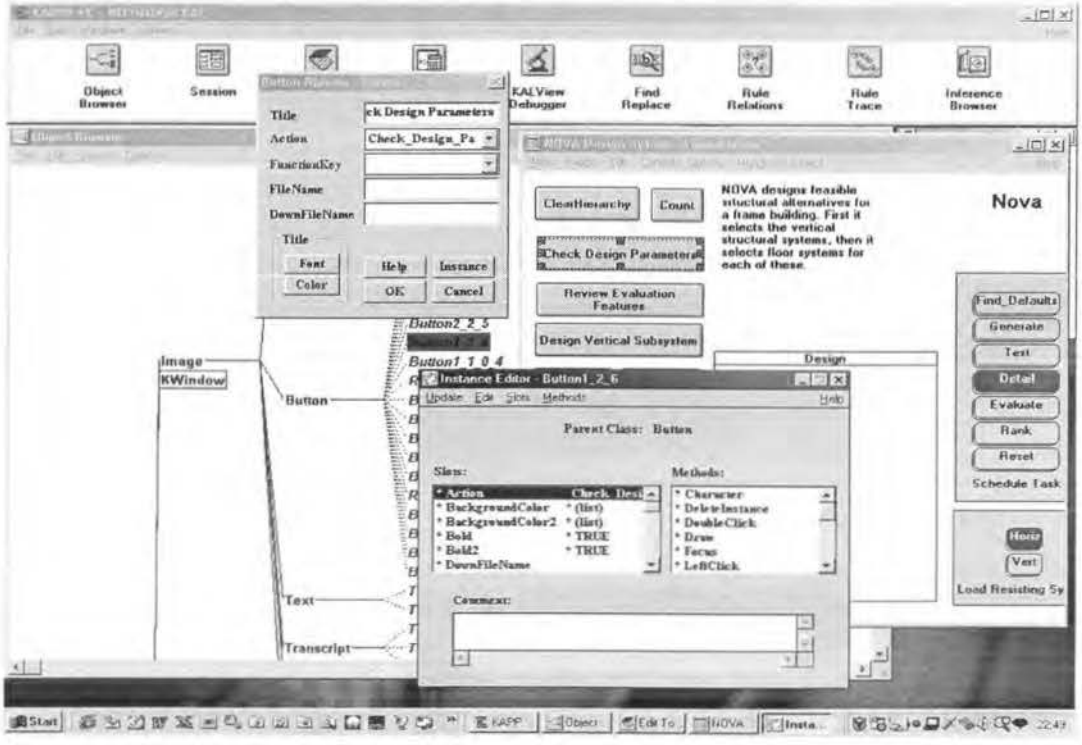


Figure 9.6 The Button image, image editor and session window.

of input and status display objects, which allowed the user to input and review Default Design Parameters and Evaluation Features and to input the specifications for the new building. These objects form part of the system's user interface. The design details for the Specification functions are described in section 7.2.1 at page 130 and in more detail in Appendix A on page 220. Figure F.10 in Appendix F shows the classes used. Figure 9.6 shows the Check_Design_Parameters input button as it appeared in the Instance Editor, during development.

- **Formulation**

The Formulation subtask is completed twice, once during design of the Vertical Subsystem and then again during the design of the Horizontal Subsystem. It consists of the following system functions: Design Vertical Subsystem, Get Assumed Sizes, Set Initial Sizes, Detail Vertical Subsystem, Design Horizontal Subsystem, Detail Horizontal Subsystem. These functions are also referred to collectively as Design Synthesis functions.

The design details for the Formulation functions are described in section 7.2.1 on pages 132 to 142 and in Appendix A on pages 220-222. Figures F.1-F5 in Appendix F show the main design classes used.

- **Design Synthesis**

On start up the user enters a number of details for the new building, for example the number of stories and the various dimensions of bays. The system then builds the search tree.

The system keeps on adding new classes at each level and deleting inappropriate ones according to its rules. By the time the system reaches the ninth level in the hierarchy, the *Intermediate_Beams* level it has created a search tree/object hierarchy of valid designs.

The pseudo code for the key design synthesis components is shown in section 7.2.1. The process commences with the function, *Design_Vertical_Subsystem*, which clears out any

existing search tree and then loads the sequence of design steps into the appropriate slot in the *Schedule* class.

- **Detailing and Testing.**

In order to test proposed designs, the system must calculate certain *details* for each design. The section, which follows, describes how this testing is carried out in the system.

Testing of Alternatives - NOVA has a series of 'test and eliminate' functions, with names of the form *Valid-xx-Alt,* where *xx* is the name of the level in the hierarchy.

The functions are called during the generation of the new designs, which are represented as classes. This is shown in the example below:

```
If Valid_Material_Alt (x # _ # Global:Number, y) Then
        {
         MakeClass( x #_ # Global:Number, y);
```

The functions are of the form:

```
MakeFunction(Valid_Material_Alt, [x y],
{
...
If Drop_IdNum(GetParent(y)) #= Shear_Wall_Narrow And
      Drop_IdNum(x) #= Steel Then
  Not(Valid_Material_Alt)
...
Else
Valid_Material_Alt;
} );
```

This example shows the function written to implement the heuristic knowledge that shear wall subsystems are not allowed in steel buildings. Such a design is not allowed and no further consideration is required.

A second level of testing is applied to designs, which are not eliminated at the outset. This testing function is called as follows:

```
Check_Material(x # _ # Global:Number);
        ...
        If ( xx:Eliminated #= Yes) Then DeleteClass(x # _ # ...
```

The inference engine is invoked by the checking function, which calls the system's forward chaining inference mechanism, as shown below:

```
MakeFunction( Check_Material, [x],
{
.....
ForwardChain([NOASSERT],NULL,
        R1_Abt_Reinf_Concrete_Material,
        R2_Abt_Reinf_Concrete_Material,
        R3_Abt_Reinf_Concrete_Material,
        R1_About_Steel_Material
        );
If .... :Eliminated #= Yes) Then
   .....
   DeleteClass(x);
```

These rules contain more heuristic knowledge; for example this rule is used to eliminate designs, which have proposed to build more than 20 stories with a rigid frame design.

```
/*************************************
     **** RULE: R1_Abt_Reinf_Concrete_Material
 **** A building over, 20 stories high cannot
 be built with a Rc rigid frame
     *************************************/
MakeRule( R1_Abt_Reinf_Concrete_Material, [],
(If) Altbldg:Vert_3D_Level #= Orthogonal_2D_Systems_1 And
 ( Drop_IdNum( Altbldg:Vert_2D_N_Level ) = #= Rigid_Frame_Narrow And
  Altbldg:Stories > 20 And
  Drop_IdNum( Altbldg:Material_Level ) #= Reinf_Concrete,
(Then)   Altbldg:Eliminated = Yes );
```

- **Evaluation**

This section explains how the evaluation components of the object model were converted into Kappa-PC classes and functions. The Kal functions contain algorithms, which deliver

the processing, which was identified in the requirements statements. A set of input buttons allows the user to input evaluation features and subsequently review and adjust these items. An input button is provided to initiate the evaluation process and a dialog box is displayed to allow finer tailoring of design processing. A transcript window allows the user to display the evaluation report and another inset window displays rankings for the top "n" designs.

```
/*****************************************************/
        FUNCTION: Write_Vert_Eval_Report[],
        {
        ..............
        DisplayText(

             SendMessage(Vert_System_Column, Feature_Calculation,x)));

/*****************************************************/


/*****************************************************/
        FUNCTION: Calculate_Column[Bldg],
        {
        IF Vert_2D_N_Level  EQUALS Shear_Wall_Narrow THEN
         {
         Value1 ← 0.0
          }
        ELSE
        IF Vert_2D_N_Level) EQUALS Rigid_Frame_Narrow AND
          Bldg IS concrete THEN
         {
            Value2 ← (Bldg:Width_Of_Column_Narrow)^2
              }

/*****************************************************/


/*****************************************************/
        FUNCTION: Calculate_Percent_Optim [Bldg Feat],
        {
        Feat_Value  ←  SendMessage(Feat, Feature_Calculation, Bldg)

        IF   Feat:Target_Set EQUALS No THEN 0.0
        ELSE
        IF   Feat_Value > Feat:Target_Max OR
            Feat_Value < Feat:Target_Min AND
            Feat:Type_Of_Target EQUALS Any OR
            Feat:Type_Of_Target EQUALS Achieve THEN 0.0
        ELSE
        IF   Feat:Type_Of_Target EQUALS Min AND
            Feat_Value <= Feat:Target_Max AND
            Feat_Value >= Feat:Target_Min THEN
                    Feat:Target_Max - Feat_Value/
                    Feat:Target_Max - Feat:Target_Min*100
        ELSE
```

```
IF Feat:Type_Of_Target EQUALS Min AND
    Feat_Value < Feat:Target_Min THEN 100.0
ELSE
IF   Feat:Type_Of_Target EQUALS Min AND
    Feat_Value > Feat:Target_Max THEN  0.0
ELSE
IF Feat:Type_Of_Target EQUALS Achieve AND
    Feat_Value = Feat:Target_Max THEN  100.0
ELSE
IF Feat:Type_Of_Target EQUALS Any AND
    Feat_Value <= Feat:Target_Max AND
    Feat_Value >= Feat:Target_Min THEN  100.0
ELSE
IF Feat:Type_Of_Target EQUALS Max AND
    Feat_Value <= Feat:Target_Max AND
    Feat_Value >= Feat:Target_Min THEN
            Feat_Value - Feat:Target_Min/
            Feat:Target_Max - Feat:Target_Min*100
ELSE
IF Feat:Type_Of_Target EQUALS Max AND
     Feat_Value < Feat:Target_Min THEN 0.0 ;
```

9.4 Control of the Design Process - the Schedule

The flow of control in NOVA is determined by a plan, part of which is represented by the *Sequence_Of_Parts_To_Be_Designed* slot described above. A body of code is executed for each level (design goal) of the hierarchy, in the order indicated in the sequence. This code determines the order in which synthesis should be implemented.

9.5 Difficulties Encountered During Implementation

- **Trade-off between dynamic rule based programming and sequential procedural programming.**

One of the primary purposes of this study was to explore the issues that arise when one uses object-oriented computing techniques to develop knowledge-based software, which in this case consisted of a new design tool.

To this end, it was intended from initiation to develop this new software on the Kappa-PC application development product, which would provide the required object-oriented language and environment. It was also decided in the design stage of the project, that the system design would use rules, following the same strategy as that used in both the HI-RISE and DOLMEN systems. This would require that a significant component of the system's design knowledge, especially the heuristic knowledge concerning the testing of potential structural design solutions, would be represented in the form of a rule base.

This strategy of using rules was expected to realise several advantages. Thus, when there are a large number of decision points in a piece of software, it easier to understood the effect they will have when they are written in the simple Kappa rule syntax, than when they are written as conditional statements in KAL programming code. Furthermore, the use of rules would allow the writer to take advantage of Kappa-PC 's inference system, which is a systems program for managing rules and applying them dynamically, as appropriate. Dynamic application of knowledge-base rules allowed the writer to use them flexibly during coding of the design synthesis functions. If the writer had taken a conventional approach, by contrast, he would have had to indicate explicitly when any given conditional statements should be applied.

However, during the design stage, the writer had difficulty in incorporating the rule base into the object model for the system and during the subsequent implementation stage he also noticed that:

- The knowledge-base rules were similar in form and effect to conditional, 'IF, THEN, ELSE' statements found in conventional procedural computer programs; and that
- In most cases, an equivalent conditional statement could replace any such a rule.

The writer then had to decide which design decisions would be based on knowledge represented by rules and which design decisions would be simulated in KAL function code, using the appropriate conditional statements. He was able to gain limited guidance on this issue from the Kappa-PC 2.4 Online Help facility, which recommended that:

- Rules are useful if the rule conditions can be broken up into small rules, and if the control structure provided by the inference engine (the forward and backward chaining mechanisms) is appropriate;
- Rules are inappropriate, where the reasoning process requires only a few conditions, but instead calls for a predetermined series of steps; and
- Rules are also inappropriate, where the sequence of events is complicated and needs to be managed. The Online Help facility provides the following example, which it says should be programmed in a conventionally written KAL function:

("First, test this; if X, then do this;
Otherwise, if Y, then do that,
Except in the special case of Z, when you should do something else;
Or if Q, then go back and test whether ...")

The writer also obtained the following limited guidance from a white paper issued by "The Haley Enterprise" (1992), which recommends that as long as the particular conditional situation can be flow-charted then a rule-based system is not required.

The writer found that each particular simulation of a design decision had to be considered separately. Thus where a single condition had to be tested, which was the case during the early stage of design synthesis when the system seeks to ensure that only valid designs are added to the search tree, then the elimination test was coded as conditional KAL function. In the later stages of synthesis, where the designs were detailed, the system needs to test several conditions and these decisions were represented in the form of rules. The associated decision making processes were simulated using functions, which invoked the Kappa inference system.

- **Inconsistency in the system; global scope for object attributes versus principles of encapsulation and information hiding.**

In order for the system's inference system to function properly it needs to be able to react dynamically to changes in appropriate object attribute values, this requires that these attributes are provided with global scope, this requirement is inconsistent with the object-oriented principles of information hiding.

- **Difficulties caused by Kappa-PC's lack of support for multiple inheritance**

Section 7.1.5 describes design synthesis in the new system. During synthesis Class *Building_1* forms the root node of the search tree hierarchy and all the new designs are created under this generic building object. It contains the user's input requirements for the building, which all alternative designs must accommodate. As the design proceeds two subclasses of Building_1; *Core_1* and *Orthogonal_2D_Systems_1* are created, which inherit the attributes from *Building_1.* The system then creates a new level in the tree by creating subclasses from these first two subclasses. These new subclasses also inherit attributes from the appropriate Alternative class.

This form of subclass inheritance is known as multiple inheritance and because Kappa-PC does not explicitly provide support for it, the writer had to design a work around to provide the inheritance links required in the search tree of partial design objects. This work around required the writing of a Slot_Copy function to copy the class attributes from the appropriate Alternative class at each level in the design hierarchy.

- **Difficulties caused by Kappa-PC's limited provisions for local variables**

Kappa-PC supports a full object-oriented programming model, in which all programming entities are viewed as objects. Consequently there is limited programming support for the use of temporary variables, which do not merit the allocation of a permanent system object.

Kappa-PC has a programming language called KAL, which supports settable local variables, which can be declared with reserved words like *Let, For, ForAll.* For example, the following code shows the use of a local variable, x, in the *Let* construct :

```
Let [x 0]
  While (x < 10)
    x = x + 1;
```

In the example the programmer evaluates an expression with the temporary arguments, x. This form of expression is limited, the language only maps x within the expression. Although the programmer is allowed to use {} to include multiple expressions in expression, it is still difficult to use local variables in any extended function. Because, the new NOVA design tool is required to complete lengthy calculations for design variables like sway the writer had to resort to the use of several classes of temporary design variables. This was difficult to program and resulted in much duplication of code.

Summary

This chapter has described the implementation of a design for a knowledge based PC design tool, which was intended to assist with preliminary structural design tasks.

The design was implemented as a graphical user interface, a series of Kappa-PC classes with appropriate methods and accompanying KAL functions, which simulated a version of design synthesis based on a hierarchical planning process and a decomposition based implementation of the plan-generate-test strategy

CHAPTER 10. Operating the Nova Design Tool

10.1 Introduction

Chapters 6, 7 8 and 9 described the development of software for a knowledge-based design tool. This chapter describes the resulting prototype system. Currently the prototype will allows the user to enter building details, review and change default design parameters and evaluation features and produce unfinished designs for a class of regularly shaped buildings. The system is capable of designing, testing, evaluating and ranking the vertical structural subsystems, which form part of a building system. Further work is necessary to finish the KAL functions required to complete the design of the horizontal structural subsystem, which makes up the building's floor system.

10.2 Demonstration of the design tool.

The following sections describe the prototype system, which is referred to as the NOVA system. The writer has described how the system was used to create partial designs for a shopping centre. Figure 10.1 shows the initial Kappa-PC graphical user interface display. This screen has three windows, the first window displays icons for the developer tools, which are along the top; the edit tools window is along the side and below these is the object browser display window.

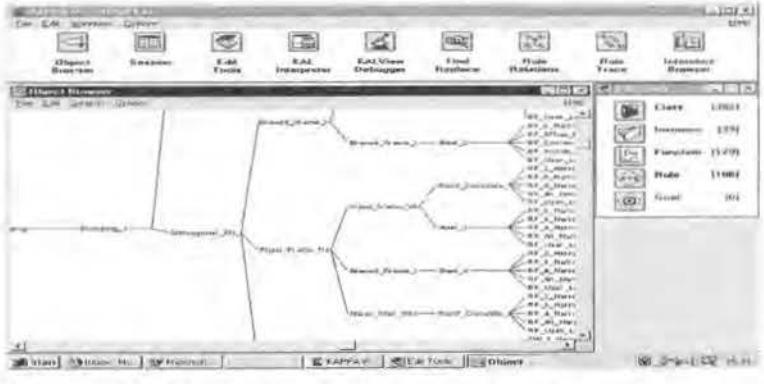


Figure 10.1 The Kappa-PC application development screen

To invoke the NOVA application the user clicks on the file option in the Kappa-PC menu bar, this opens the file selection dialog box and the user then selects Nova.kal. This loads the NOVA application. To commence working the user clicks on the Session window, see Figure 10.2, and selects *Session* from the dialog box.

Figure 10.2 The Session Dialog Box

This opens the NOVA graphical user interface window, see Figure 10.3. This screen allows the user to control the design process. The display consists of a selection of appropriately labeled button images, which allow the user to complete each stage of the design process. As this is a prototype system, several input buttons have been left in place to allow the user to execute parts of the design process manually. These buttons would be removed when the final application is completed. The *Count* button has been left available so that the user can count the number of classes, which represent the partial designs in the search tree at any particular time. The *Clear_Hierarchy* button has also been left so that the user can explicitly clear the object hierarchy of partially completed designs. The *Assumed Sizes* and *Initial Sizes* buttons have also been left in place. These are used during the design of the Vertical Subsystem to set assumed sizes for those components, whose sizes can not be calculated properly until the Horizontal Subsystem is designed and to set the initial sizes for certain Vertical Subsystem components.

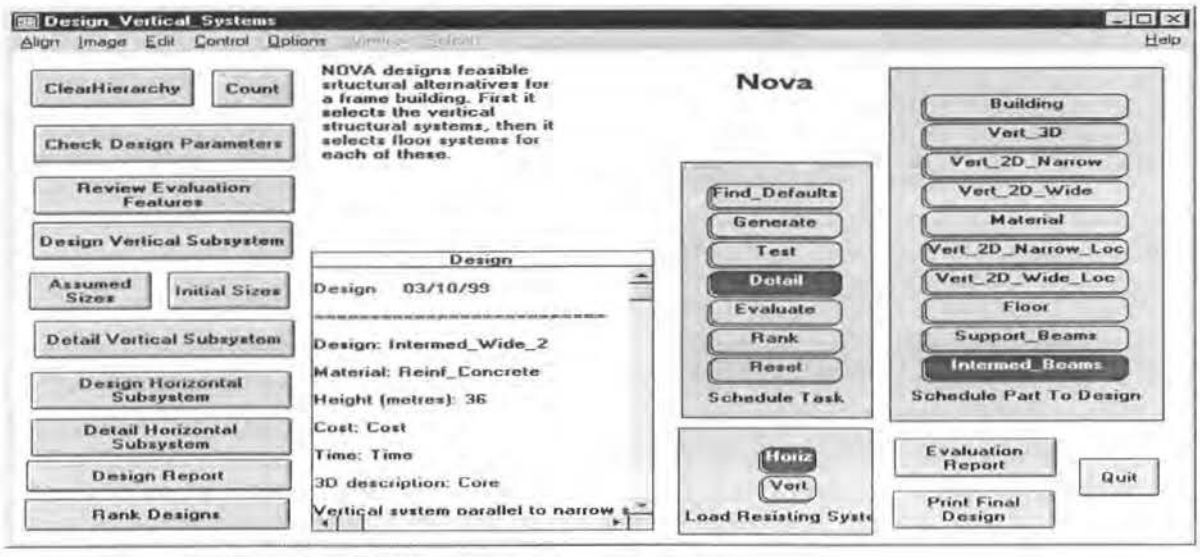


Figure 10.3 The NOVA application user interface

The NOVA application user interface has three output display statebox images, which display the current state of the following slots in the Schedule class, *Task, Part_To_Design* and *Load_Resisting_System.* The images consist of columns of display icons, which allow the user to track the progress of the design process. The *Task* image displays the program's current design task. The *Part_To_Design* image indicates the level in the hierarchy, which is being designed, and the *Load_Resisting_System* image indicates the major subsystem, which is being designed.

The design process starts when the user clears the object hierarchy and starts to design the Vertical Subsystem. After pressing the *Design Vertical Subsystem* button the user is presented with a query form, which asks whether the user wishes to supply a custom location layout scheme. If the user enters *NO,* the system uses its default locations. In the example documented here the user has decided to rely on the locations provided by NOVA and has entered *NO,* see figure 10.4. The corresponding system response, which acknowledges the user's input, is also shown.

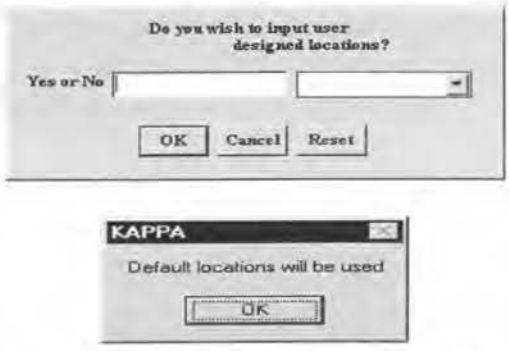


Figure 10.4 System query regarding user designed locations

The user is then presented with a multiple input form, see Figure 10.5, which allows the requirements for the building to be input to the system. These requirements make up the design specification for the building.

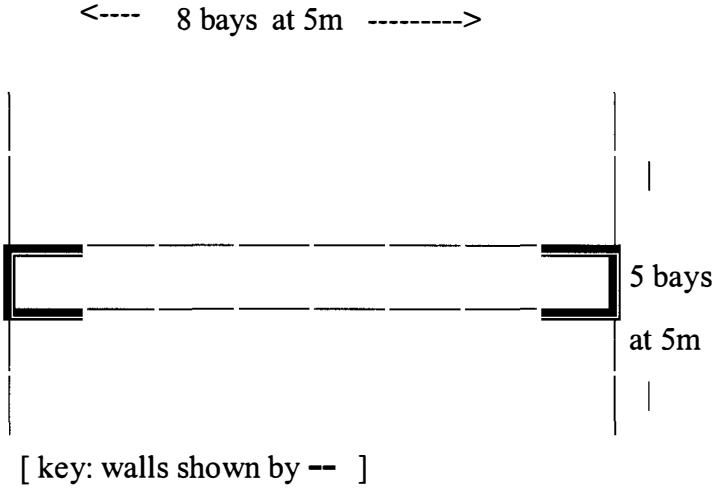
Figure 10.5 Multiple input form for input of building design requirements

In the example described here NOVA was used to design a 4 storey, city centre, shopping centre. The area of the shopping centre was made up of a 5m x 8m grid. The plan of the building was 40m x 64m, with storey heights, floor to ceiling of 3m. The loading chosen was an 8.5 kN/m^2 uniformly distributed imposed load, which included 1.0 kN/m^2 for movable partitions. The dead load included 0.95 kN/m^2 for screed and 0.5 kN/m^2 for services, and loading for block wall partitions along column lines. A value of 1.0 kN/m^2 was used for wind loading. The system produced 207 designs schemes and the writer reviewed the top four design schemes from the ranked list. These included:

i) Two designs with one-way ribbed slabs, which spanned onto reinforced concrete beams along column lines.

ii) Two designs with precast panels, which spanned 4m and which were supported by main composite steel beams.

The system contained these design defaults: concrete of strength 35 kN/m^2, structural steel grade 50 and lightweight concrete was used for the composite steel decks. The fire rating selected for the building was taken as 2 hours. The location for the stairwells and corresponding walls, which would be either shear walls or braced frames, were assumed to be located as shown in Figure 10.6.



5 bays at 5m

[key: walls shown by --]

Figure 10.6 Plan of building

The input form is programmed to prompt the user to provide a value for each of the building requirements. The system maps these requirements to the slots of the Building_1 class, which is the root class in the hierarchy of design objects. The Kappa-PC input form is designed to retain any slot values, which have been used before. Therefore, if the user is happy to accept the existing value of the attribute, displayed in the input form, then he/she ignores the slot in question, otherwise it is necessary to put a mouse click on the input form and enter the new value. For the example used, the following items are input:

Building Requirement	System Input Variable	Example
Imposed load [kN per m**2]	Imposed_Load	8.5
Wind load [kN per m**2]	Wind_Load	1
Number of stories	Stories	4
Minimum floor to ceiling clear height [m]	Clear_Height	3
Number of bays in narrow direction	Narrow_Bays	5
Width of each bay in narrow direction [mm]	Narrow_Dimension	8
Number of bays in wide direction	Wide_Bays	8
Width of each bay in wide direction [mm]	Wide_Dimension	8
Fire rating [hours]	Fire_Rating	2
Is there a centrally located shaft ?	Shaft	No
Function of the building	Function	Shopping Centre
Status of the building	Status	Urban
Number of staircases	Number_Of_Staircases_Per_Floor	2
Location of the building	Location	City Centre
Is the site restricted?	Site_Restricted	No
Is there a tenant?	Tenancy_Known	No
Number of designs to be considered	Number_Of_Designs_To_Be_Considered	4

Table 10.1 Input of building requirements

The initial input to Nova was as shown in Table 10.1. The site was not restricted, as it was a shopping centre, the tenants were not known. Once the design requirements had been input the system prompts the user to review the default design parameters already input to the knowledge base, see Figure 10.7.

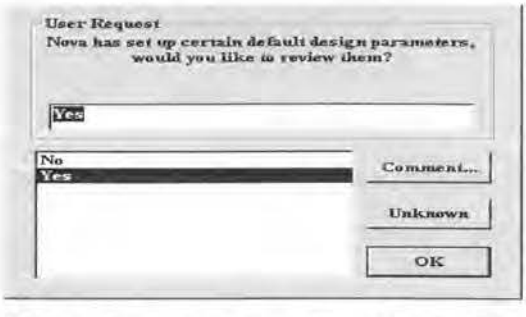


Figure 10.7 User request prompt asking user to review default design parameters

The user can accept this activity or bypass it altogether, thereby leaving the existing values unaltered. These parameters include values for certain commonly occurring variables. They cover the amount of concrete applied to cover the bottom and top steel and steel in slabs and the dimensions for walls and steel beams. They also include constraints to be used in calculations for concrete design strength, grade of structural steel and steel yield stresses. The system then presents the user with a multiple input form, see Figure 10.8, who

is given the option to accept or change the values. Where a variable has been set up with a range of acceptable values the system presents this range for selection. All these variables are represented as slots or attributes of the Defaults class.

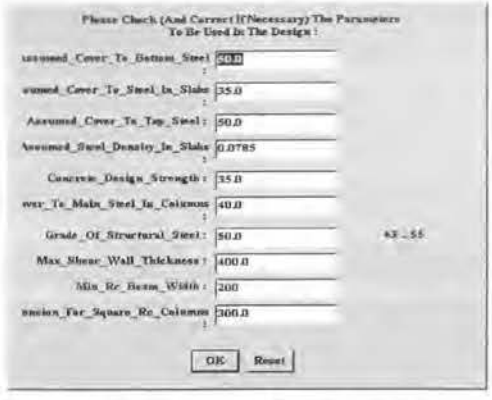


Figure 10.8 Multiple input form for review of default design parameters

The system also has a display facility to allow the user to review the system evaluation features. These features represent 'soft' design constraints. There are two sets, one set for the Vertical Subsystem and one set for the Horizontal Subsystem, there are 12 features in each set. They are represented in the system as subclasses of the 2 respective subsystem evaluation classes Vertical Subsystem Evaluation Features and Horizontal Subsystem Evaluation Features.

Each feature has the following attributes: description, importance, importance factor, target maximum value, minimum, target type or objective and target set flag. If the user decides to use a particular feature in the evaluation process, then the target set attribute must be set to *YES*. The user is then required to make a series of subjective decisions regarding the attributes of that particular feature and then input them using the multiple input form. The values selected for the *Buildability* feature are shown in Figure 10.9.

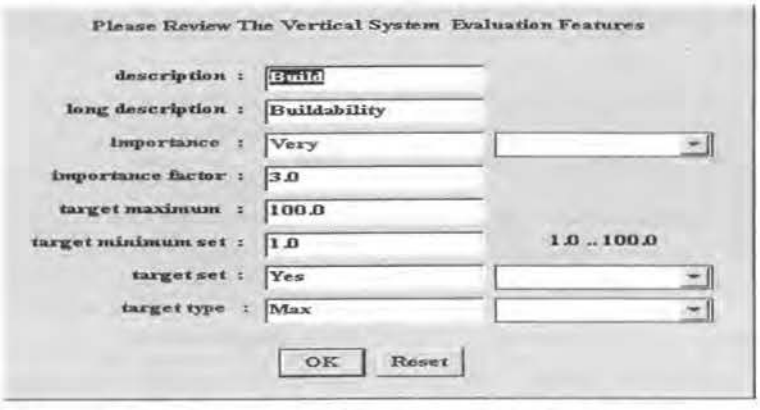


Figure 10.9 Multiple input form for system evaluation features

In the example documented the user has decided to use the *Buildability* attribute in the evaluation process. The *target set* flag has been set to *yes* and then the *importance* value has been selected from the pull down selection dialog box. In this example the importance factor has been given a value of 3 and the *maximum* and *minimum* attribute value targets and the *type* of the target have been set. In the example the user is interested in maximising the value of the buildability feature. On completion of the input of the design requirements the system has all the information it needs to design the Vertical Subsystem.

The system has two major design tasks. These are to select the vertical and horizontal subsystems, which constitute the 3-D whole. The system selects the most suitable configurations for the building, from the alternatives available at each level in the building hierarchy. These alternatives are stored in the knowledge base as members of the *Alternatives* class.

NOVA estimates loadings, chooses initial sizes and completes a limited range of structural analyses for each partial design. NOVA designs in three stages, specification, formulation and evaluation.

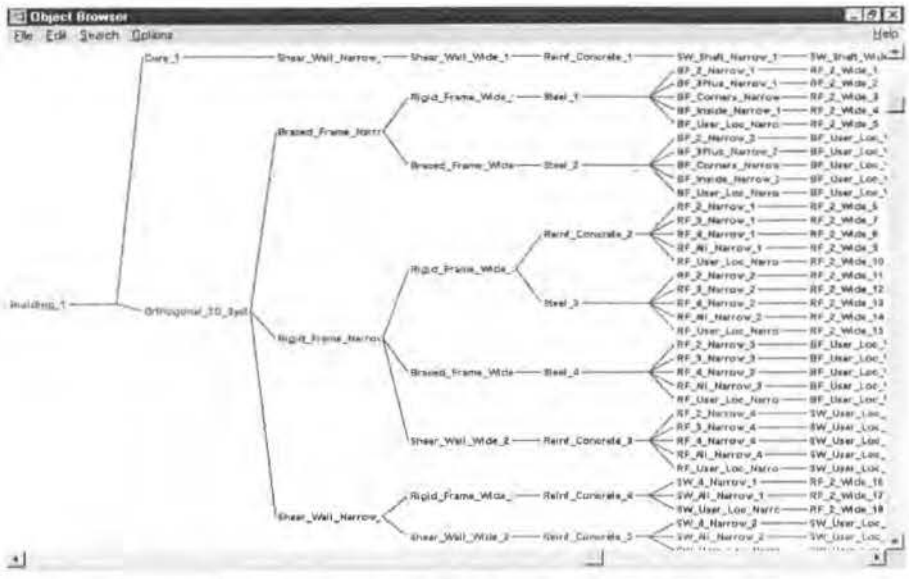
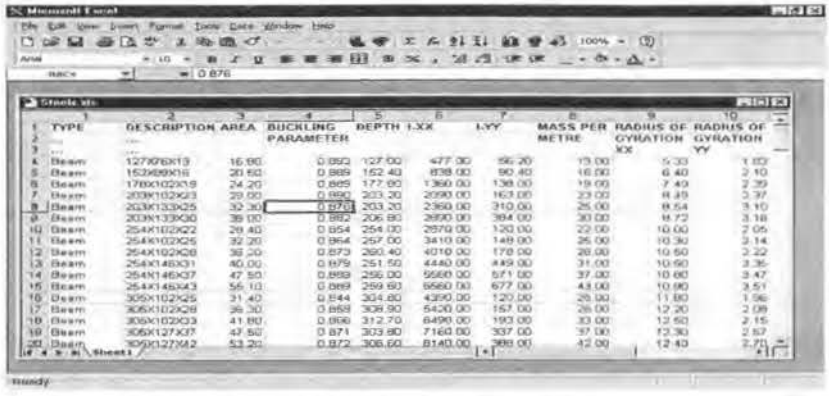


Figure 10.10 Object Browser Display showing search tree for vertical subsystem

As explained in section 7.1.5, the system builds a search tree of partial designs. Each succeeding level in the tree stores the design knowledge accumulated down to that level. The design classes at any particular level inherit the design attributes of their intermediate parent class PLUS the design attributes from their respective alternative class. The system constructs the tree by successively creating subclasses of the immediate superclass in the hierarchy starting with Building_1. Figure 10.10 shows the search tree for a particular design project, which is displayed by the Kappa-PC Object Browser. The KAL function for this task reads through the alternative design options available at each level, which are to be found in the *Alternatives* hierarchy and copies the slot values from these options, into the newly created partial design. This process is described in section 7.1.5. As the synthesis process continues system message are output, which indicate that assumed floor sizes have been established for the vertical subsystem and that approximate sizes have been calculated for the physical components of the structural subsystems. Figure 10.11 shows the spreadsheet from which the system extracts the steel section parameters, which it requires.



	TYPE	DESCRIPTION	AREA	BUCKLING PARAMETER	DEPTH	I.XX	I.YY	MASS PER METRE	RADIUS OF GYRATION XX	RADIUS OF GYRATION YY
4	Beam	127X76X13	16.90	0.892	127.00	477.00	56.20	13.00	5.33	1.82
5	Beam	152X89X16	20.50	0.889	152.40	838.00	90.40	16.00	6.40	2.10
6	Beam	178X102X19	24.20	0.885	177.80	1360.00	138.00	19.00	7.49	2.39
7	Beam	203X102X23	29.00	0.880	203.20	2090.00	163.00	23.00	8.49	2.37
8	Beam	203X133X25	32.30	0.870	203.20	2350.00	310.00	25.00	8.54	3.10
9	Beam	203X133X30	38.00	0.882	206.80	2690.00	384.00	30.00	8.72	3.18
10	Beam	254X102X22	28.40	0.854	254.00	2570.00	120.00	22.00	10.00	2.05
11	Beam	254X102X25	32.20	0.864	257.00	3410.00	149.00	25.00	10.30	2.14
12	Beam	254X102X28	36.20	0.873	260.40	4010.00	179.00	28.00	10.60	2.22
13	Beam	254X146X31	40.00	0.879	251.50	4440.00	449.00	31.00	10.60	3.35
14	Beam	254X146X37	47.50	0.889	256.00	5560.00	571.00	37.00	10.80	3.47
15	Beam	254X146X43	55.10	0.899	259.60	6560.00	677.00	43.00	10.90	3.51
16	Beam	305X102X25	31.40	0.844	304.80	4390.00	120.00	25.00	11.80	1.96
17	Beam	305X102X28	36.30	0.858	308.90	5430.00	157.00	28.00	12.20	2.08
18	Beam	305X102X33	41.80	0.866	312.70	6490.00	193.00	33.00	12.50	2.15
19	Beam	305X127X37	47.50	0.871	303.80	7160.00	337.00	37.00	12.30	2.67
20	Beam	305X127X42	53.20	0.872	308.60	8140.00	388.00	42.00	12.40	2.70

Figure 10.11 Display from the steel sections spreadsheet

After the design of the Vertical Subsystem, which results in the construction of a search tree of partial designs, the user is given the option to start the next phase of design, which involves the detailing the Vertical Subsystem. To continue the user selects YES, in the User Request dialog box, he is then prompted to choose whether to proceed to detail either: all the designs in the search tree; all the reinforced concrete designs; all the steel designs, or a selection of designs chosen by the user. The system then applies its detailing functions to the chosen designs. At the Vertical Subsystem level, the system can offer designs for 3 different types of structures, braced-frame, rigid-frame or shear-wall. The KAL functions, which perform the processing for these tasks are organised into three corresponding groups.

As a result of the detailing process the system will remove several partial design alternatives from the search tree. These are partial designs, which have been detailed and which have subsequently failed one of a series of elimination tests designed for that particular structural subsystem type.

For example when the system details a braced frame option it will proceed as follows. First it will calculate the dead load estimate for the design, which requires it to find the weight of the floor per square millimetre, which in turn requires the calculation of the volume of concrete on the steel deck. The system then applies the specific weight of concrete retrieved as an attribute or slot value from the Default class. The system continues and calculates the

load on the frames due to the beams. To compute these load figures the system needs access to the information stored in the Steel Sections Excel spreadsheet, shown in Figure 10.11.

The program then calls the *Check Design* function to test the design. *Check Design* uses the forward chaining facility of the inference processor to apply a series of elimination tests. These tests are represented by the rules in the rule class *Rules_For_Checking_ Braced_Frame_Alternatives*. The tests are applied to all the braced frame designs, that is those designs at the Vertical_2D_Wide_Location level, which have used the braced frame option at the Vertical_2D_Narrow or Vertical_2D_Wide levels. An example of one of these rules is shown below, this rule eliminates those designs where the calculated uplift is greater than the dead load on the columns. Such designs would be expected to fail against a lateral force and would tip over.

```
*******************************
**** RULE: R1_For_Braced_Frame
**** Uplift greater than dead load
*************************************/
R1_For_Braced_Frame, [Altbldg|Test_Class],
IF Altbldg:Uplift_Wide > Altbldg:Deadload_On_Column_Wide,
THEN SetValue( Altbldg:Detail_Status, Unsatisfactory ) );
```

Figure 10.12 Rule for uplift

In this example, if a design fails the test, then the program will repeat the test for a predetermined maximum number of times, each time selecting a larger steel section, until it has either found a suitable section or the maximum number of tests has been exceeded. If the program exceeds the maximum number of tests or if it fails to fins a suitable section it will eliminate the design.

Figure 10.13 Completion of detailing message at the vertical subsystem level

To continue the design process the user will respond to the system's prompts, by using the computer's mouse to click on the appropriate input buttons. The system will then complete design and detailing for the horizontal structural subsystem. On completion of the design process the system will output a completion message. The user may then instruct the system to rank either all the designs produced or input a number, thereby instructing the system to rank that number of designs. See Figure 10.19.

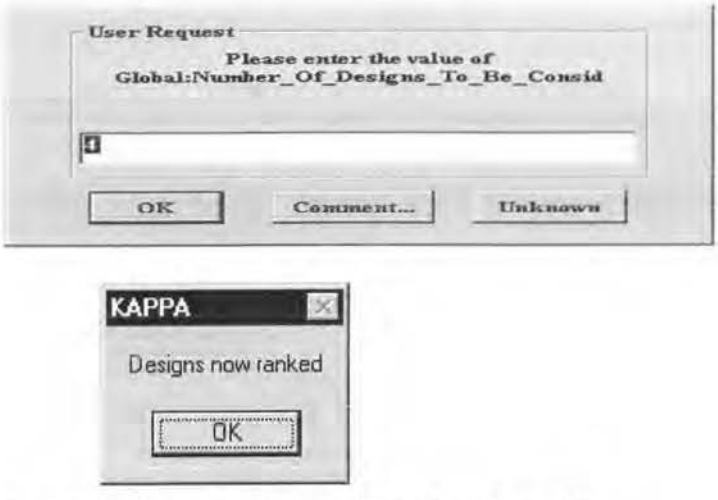


Figure 10.14 System prompt and corresponding acknowledgment

The system's transcript window, which is shown in Figure 10.15, displays the list of items in their respective ranking. Figure 10.16 shows the details listed in the transcript window, in response to the user clicking on the design report input button and supplying the name of the design, for which details are required.

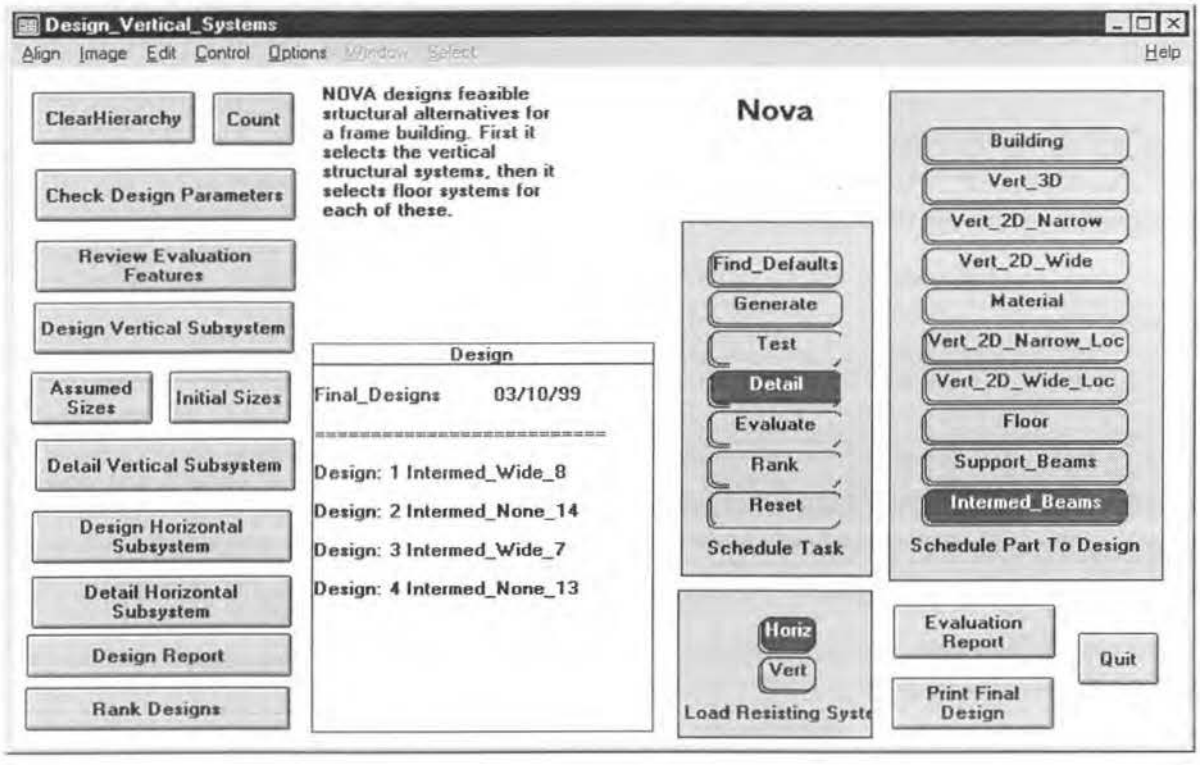


Figure 10.15 System transcript window shows top 4 designs produced

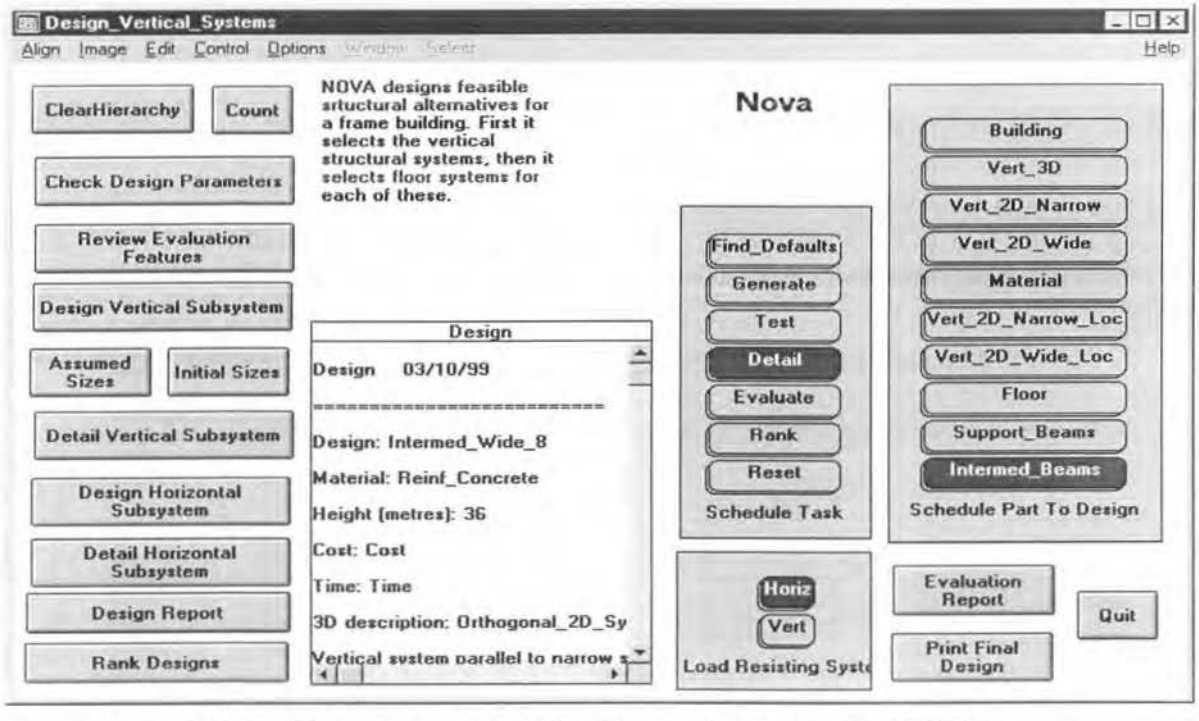


Figure 10.16 System transcript window showing details for selected design

The user also has the option to produce a report showing the results of the evaluation process. The report, which appears in its own transcript window is shown in Figure 10.17.

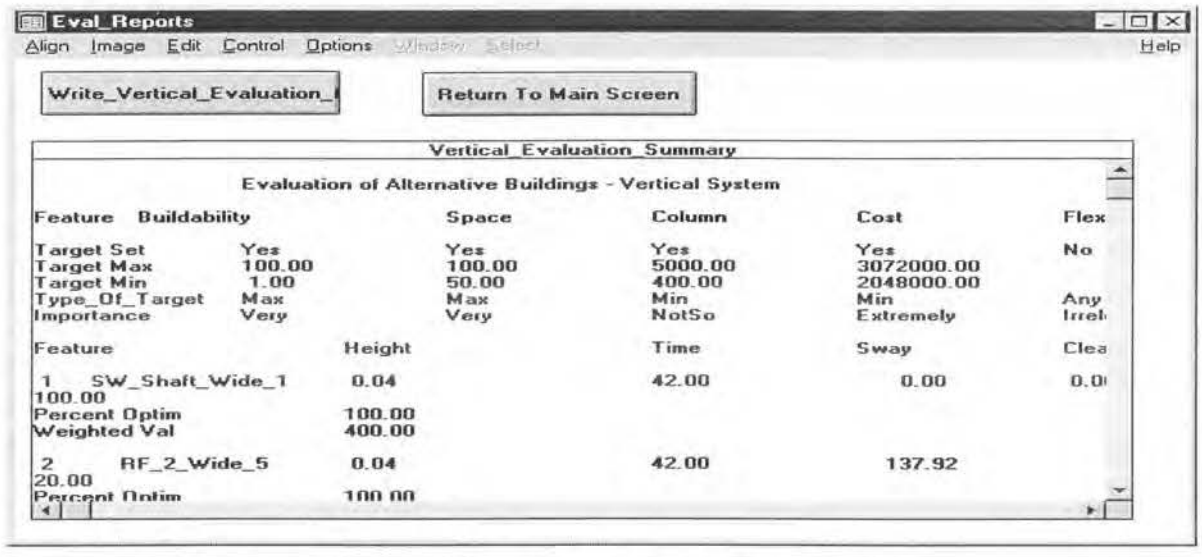


Figure 10.17 System transcript window showing evaluation results

The NOVA design prototype takes advantage of Microsoft's Dynamic Data Exchange (DDE) feature, which can be used with many Windows applications. By exchanging data dynamically NOVA can send and receive data from other applications either through a request or by establishing hot links with them. It can also execute commands in another applications via a DDE message. Thus as described earlier NOVA extracts the information about steel sections from an Excel spreadsheet. The user may also export the design details created by the system to Excel. This facility allows the user to use the NOVA system with other PC tools, thereby achieving a degree of integration of the software tools.

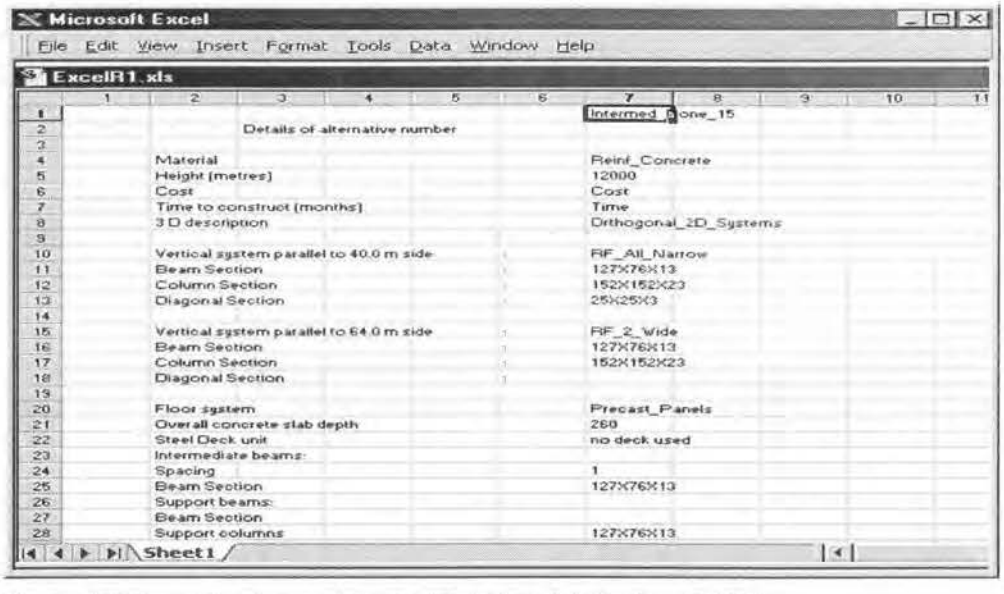


Figure 10.18 Excel spreadsheet with design details from NOVA

Figure 10.18 shows the Excel spreadsheet designed to collect finished design details from NOVA, which is acting as a DDE server.

Summary

This chapter has described what has been achieved regarding the implementation of the software design, which was documented in chapters 6, 7 and 8. A knowledge-based, PC prototype, design tool has been partially implemented on the Kappa-PC application development system. The design tool provides assistance to the structural engineer during the early stages of structural design. It aims to remove some of the tedium involved in preparing design schemes and it provides information to assist the user with design problem solving. The system outputs design information and its open architecture allows it to transfer this information to other PC packages including Excel.

This implementation has demonstrated that object-oriented computing techniques can be used successfully to create a model of a software approach, which supports intelligent design problem solving and which may be translated into a software design for implementation in an application system.

CHAPTER 11. Conclusion

This report describes a research project concerning effective methods for the application of object-oriented computing techniques to support preliminary structural design. The report focuses on a software development process created by the writer, which facilitated the use of object-oriented analysis and design techniques. The process enabled the writer to design a knowledge-based prototype for preliminary structural design, which was implemented on the Kappa-PC application development system.

The software development process comprised four stages: high-level analysis, requirements development and object-oriented analysis and design. The high-level analysis stage allowed the writer to adopt and analyse a particular approach to developing systems for preliminary structural design. During the requirements development stage, the writer produced a list of functional requirements for a design tool. The object-oriented analysis and design stages then allowed the writer to continue and produce a system architecture for the proposed design tool software.

The writer then completed a final implementation stage, which confirmed that it was practical to implement the selected approach in software using Kappa-PC.

The software development process allowed the writer to ensure that:

- The analysis and design activities were properly organised and coordinated; and
- A set of functional requirements was produced to communicate what the proposed system ought to do from the user's perspective.

The object-oriented analysis techniques allowed the writer to produce a series of models of the design process, which specified:

- The objects in the design environment;
- The relationship between the objects;
- The design processes that create, maintain and use the design objects; and
- The rules for the management and use of these design objects.

The research project was organised in a logical pattern, which comprised preliminary, development and reporting stages. The preliminary stage commenced with a literature survey, which facilitated an examination of various aspects of preliminary structural design, after which the principles of object-oriented analysis and design were reviewed. The preliminary stage was brought to a close with the completion of a review of literature covering the use of knowledge-based systems to support engineering design.

The first problem addressed in the preliminary stage of the project was to find a suitable approach to the problem of providing support for structural design. After the literature survey the writer decided to adopt an approach to the problem, which was first reported by Maher (1984). This approach was analysed in depth during the subsequent development stage.

The writer chose to adopt the approach described by Maher because:

- This approach was based on a formalised model of the design process, which other researchers have taken up and incorporated in prototype systems;
- It incorporated basic structural engineering concepts described in the standard textbook, by Lin & Stotesbury (1981);
- The approach was well documented by Maher (1984) and Harty (1987); and

- The approach appears well suited for implementation in an object-oriented, knowledge based system.

The approach chosen relies upon a formalised model of the design process, which several researchers, including Krishnamoorthy (1996) have described as the decomposition based model, and which provides computer support by way of an expert system. Maher has described this approach in several papers and used it to produce an expert system, known as HI-RISE, which was designed to assist with the preliminary design of tall buildings. Other researchers have also adopted this approach, including Harty (1987), who implemented it in an expert system known as DOLMEN, which was designed to extend the range and functionality of HI-RISE. Furthermore, Sause et al. (1992) have extended this decomposition-based approach and proposed the '*multilevel selection-development*' (MSD) model, which is a process generalisation model for structural design.

The second problem in the preliminary stage was to determine a suitable PC-based, object-oriented, knowledge engineering environment on which to implement the selected approach. The writer chose the Kappa-PC development application because of its availability, low cost and its ability to run under Windows 95 and because research revealed that it had been used to produce expert systems quickly and economically. The writer subsequently installed Kappa-PC and proceeded to learn how to use it.

Having solved two key problems; the adoption of a proven approach to the problem and the selection of an application development system the study progressed to the development stage.

In the project development stage, the writer initially organised the software development process, which comprised the various analysis tools and methods required to complete the study. The writer then applied this process to create a software design for the proposed

design tool, which was then implemented. Activities completed in the development stage included analysis of the problem solving approach adopted and production of a conceptual model for a new system. The writer then prepared a list of functional requirements for the new system, which was based on the list of design tasks accompanying the conceptual model, which were similar to functions in the DOLMEN system described by Harty (1987).

The object-oriented analysis and design stages used a six-step analysis process, as a framework, to guide the analysis and to ensure that the problem was fully understood and that the required diagrams were created. The object-oriented analysis stage provided an object model of the "real world" design problem, by analysing the functional requirements required to support preliminary structural design.

During the design stage the writer developed the systems architecture for the new system and developed strategies to implement the object model in the Kappa-PC environment. This required consideration of the various Kappa-PC system objects, such as the inferencing mechanism and the user interface components, which were to be factored into the analysis model. The writer produced several notebooks containing notes regarding the structure of the design objects, diagrams and flow charts for the algorithmic functions and object diagrams and message passing schemes. The design stage was enhanced through the creation and modification of a series of system prototypes, which were written after some preliminary analysis and outline design.

As in any design project, the writer encountered several difficulties and has described them under the following headings:

- Difficulties in analysing preliminary structural design,
- Difficulties in applying object-oriented techniques to knowledge based application,
- Overlapping of the analysis and design phases, and

- Difficulties encountered during implementation.

The following paragraphs summarise these difficulties and describe how they were addressed. Chapter 5 described the creation of a software engineering methodology and several problems, which were overcome during the process.

The writer's initial problem was to select a set of suitable object-oriented analysis and design methods, this was found to be difficult due to the large range of approaches described in the literature. The writer eventually relied upon a simplified framework and set of techniques, described by Cross (1996), which allowed him to complete the project.

The writer also failed to identify suitable object-oriented techniques, which would have provided assistance with the initial high-level analysis stage. The writer eventually used a process based on Checkland (1991) and described in O'Connor (1992) to complete the high-level analysis stage. Again, the writer had to look outside of the object-oriented paradigm to find techniques to allow him to complete the requirements specification process, which was found to be a necessary precursor to the object-oriented analysis and design stages.

During the object modeling stage the writer observed that it was difficult to create object hierarchies and state transition diagrams for several objects. These included transient objects, which either did not exist at the start of system operations; or which were created and destroyed during operations; objects, which changed identity, becoming subsumed into other accumulation type objects during operations; and the system's rule-base into the object model. The writer solved these problems by resorting to the use of 'back box' type diagrams to represent these objects.

Chapter 7 described how the analysis and design stages for the software were completed and how the problems, which were encountered in these stages, were solved. Three types of problem were encountered and are discussed below:

- **Difficulties in analysing the preliminary structural design process.**

During research into the analysis of preliminary structural design, the writer noted that several sources including Maher (1984) and Harty (1987) maintained that this type of design was ill defined and also advised that it was a difficult area for which to provide computer software. These sources also noted that the activity required the completion of problem solving techniques, which in turn comprised a series of conceptual decision making tasks, which were interspersed with a series of calculation tasks. They added that it was often difficult to determine in advance, which particular design tasks may be required in a particular project and in what order the tasks are to be applied.

In attempting to complete the necessary analysis the writer documented what he had identified as the most significant decision making tasks and created a system model, which reflected these tasks and the associated calculations, which were interspersed these tasks. This resulted in a system, which simulated a central hierarchical product decomposition, and which also attempted to facilitate the provision of a range of tools, which assist with various phases of the design project.

- **Difficulties in applying object-oriented techniques to knowledge based application.**

The writer also found that it was conceptually difficult to applying object-oriented techniques to this particular knowledge based application. The writer's difficulty in this area, is best explained with reference to Graham (1994) who said that object-orientation addressed two of the 3 key aspects, required to specify a proposed system, these were data and process. He added that control of system behaviour is more difficult to integrate into an object model and in several of the approaches he had reviewed, control in the form of rules and/or constraints, appeared to be accommodated as an afterthought. Thus, in this project the writer found it difficult to include the production rules in the object model, other than as

a 'black box'. This approach appears to leave something to be desired, but the writer was unable to find a better way to include them, given the object-oriented analysis and design tools selected for the project.

- **Overlapping of the analysis and design phases.**

The writer found that it was difficult to manage the object-oriented stages of the development project. He was unable to clearly separate the analysis and design stages and was therefore also unable to precisely distinguish which deliverables were worked upon in each stage. These difficulties resulted in the failure of the writer to produce accurate time estimates for project completion.

The writer was aware that the purpose of the analysis stage was to describe the structural design process, ie. the problem and to formulate a model of the problem domain. He was also aware that the purpose of the design stage was to decide how the new system, which constitutes the solution to the problem, would be implemented and how the architecture for the new system would be created. However, during this project the writer produced several models and it was difficult to recognise, which model was the appropriate starting point for the design stage.

The writer's was also aware that the conventional system development life cycle could be represented as a series of steps with gaps between them and where the steps are well defined and are associated with clearly identified deliverables. The deliverable output by one step then becomes part of the input for the next step. However, the writer's experience in this project, has lead him to agree with Henderson-Sellers (1992). This source had noted that object-orientation supports a seamless transition from phase to phase and that this made it difficult to pinpoint where one stage ends and another begins, likewise it is difficult to detect the point at which a deliverable should be achieved.

Chapter 9 describes the implementation of the software designed during the project, it included several difficulties encountered during the process. These are summarised below:

- **Trade-off between dynamic rule based programming and sequential procedural programming.**

One of the primary purposes of this study was to explore the issues that arise when one uses object-oriented computing techniques to develop knowledge-based software, which would necessarily use rules to represent a significant component of the knowledge base.

Using heuristics represented as rules was expected to realise several advantages including simplification of design. Thus, when there were a large number of decision points in a piece of the software, it should have been easier to understood the effect that they would have, when they were coded as rules, than when they were written as conditional statements in the system's procedural code. Furthermore, the use of rules should have allowed the writer to take advantage of the inference system. This systems program would be expected to apply rules dynamically, as appropriate, thereby eliminating the need for the programmer to indicate explicitly when any given conditional statements should be applied.

However, during the design stage, the writer had difficulty in incorporating the rule base into the object model for the system. Furthermore, during the subsequent implementation stage he also noticed that in most cases, an equivalent conditional statement could replace any rule. He then had to decide which design decisions would be based on knowledge represented by rules and which design decisions would be simulated in procedural code, using the appropriate conditional statements. The writer found only limited guidance on this issue. Thus according to a white paper issued by "The Haley Enterprise" (1992), as long as the particular conditional situation can be flow-charted then a rule-based system is not required.

In practice the writer found that each particular simulation of a design decision had to be considered separately. Thus where a single condition had to be tested, which was the case during the early stage of design synthesis when the system seeks to ensure that only valid designs are added to the search tree, then the elimination test was best coded as a conditional KAL function. In the later stages of synthesis, where the designs were detailed and the system needed to test several conditions, then these decisions were best represented in the form of rules.

- **Conflict between the requirement for global scope for object attributes versus the object-oriented principles of encapsulation and information hiding.**

The writer understood that in order for the system's inference system to function properly it needs to be able to react dynamically to changes in appropriate object attribute values, this requires that these attributes are provided with global scope. However, the writer was also aware, that this requirement is inconsistent with the object-oriented principle of information hiding. The writer was unable to resolve this conflict himself and was also unable to locate any other sources, which had satisfactorily addressed the issue.

- **Difficulties caused by Kappa-PC's lack of support for multiple inheritance**

The final system design required multiple inheritance, which is not explicitly provided by Kappa-PC. The writer had to design a 'work around' to provide the inheritance links required in the system's search tree of partial design objects.

- **Difficulties caused by Kappa-PC's limited provisions for local variables**

Kappa-PC supports a full object-oriented programming model, in which all programming entities are viewed as objects. Consequently there is limited programming support for the use of temporary variables, which do not merit the allocation of a permanent system object. However, because, the new NOVA design tool was required to complete lengthy

calculations for design variables, the writer had to resort to the use of several classes of temporary design variables. This was difficult to program and resulted in much duplication of code.

The writer implemented the key features of the design model of the proposed new system as a working prototype program. The prototype included a graphical user interface, a series of Kappa-PC classes with appropriate methods and accompanying KAL functions, which simulated a version of design synthesis based on a hierarchical planning process and a decomposition based implementation of the plan-generate-test strategy.

The writer found that there was a steep learning curve to be completed if one was to use Kappa-PC effectively. The system has facilities to support object-oriented programming and at the same time it has the inferencing mechanism necessary for logical programming. Learning was made more difficult due to the large range of specialist debugging and tracing tools both for the KAL language and for the inferencing mechanism.

To use the system properly the user has to understand how to integrate the object hierarchies used to represent domain objects with the production rules needed for inferencing. The writer noted that certain programming tasks might be achieved by using either object-oriented programming or by using the inferencing capability provided by production rules. Unfortunately the writer was unable to find sources of reference to guide the programmer as to which is suitable in a given case.

During the reporting stage the writer documented the key design activities simulated in the prototype system. These include: input of the design specifications of the building, input of system evaluation features, design of the vertical subsystem, initial sizing of components, use of the steel sections database, detailing of vertical subsystem, design of the horizontal

subsystem, detailing of horizontal subsystem, evaluation of design alternatives proposed and the selection and output of the final design.

In summary a design tool to assist the structural engineer during the early stages of structural design has been partially implemented on the Kappa-PC application development system. It aims to remove some of the tedium involved in preparing design schemes and it provides information to assist the user with design problem solving. The system outputs design information and its open architecture allows it to transfer this information to other PC packages including Excel.

This implementation has demonstrated that object-oriented computing techniques can be used successfully to create a model of a software approach, which supports intelligent design problem solving and which may be translated into a software design for implementation in an application system.

LIST OF REFERENCES

Alison, J., (1994), State Space Search vs. Problem Reduction Available WWW http:/www.cee.hw.ac.uk/~alison/ai3notes/Subsection2_6_3_1.html.

Ambrose, J.E. (1967) Building Structures Primer. John Wiley & Sons, Inc. pp. 7&8.

Beck, K., & Cunningham, W., (1989), A laboratory for teaching object-oriented thinking. In OOPSLA'89 ACM Conference on Object-Oriented Programming Systems, Languages and Applications (Meyrowitz, N., Ed) Reading, MA: Addision-Wesley, 1989.

Bedard, C., & Gowri, K., (1990). Automating Building Design Process with KBES. Journal of Computing in Civil Engineering 4,2 pp. 69-82.

Bobrow, D.G., & Stefik, M.J., (1986) Perspectives on Artificial Intelligence. Programming Science 231(8), pp. 23-35.

Bobrow, D.G., Mittal, S., & Stefik, M.J., (1986), Expert Systems: Perils and Promise. Communications of the ACM, September, Vol. 29, no. 9, pp. 880 - 894.

Booch, G (1986), Object-Oriented Development IEEE Transactions on Software Engineering, 1986, Vol. SE-12(2), PP. 211-21.

Booch, G (1991), Object-Oriented Design with Applications. Redwood City, California: Benjamin Cummings, 1991.

Booch, G and Bryan, D., (1994), Software Engineering with ADA. 3rd edition, The Benjamin/Cummings Publishing Company Inc., California.

Bravo, G., Hernandez, F., & Martin, A., (1996) Knowledge-Based Design System for Preliminary Design of Foundation Systems. In B.H.V. Topping (Ed.), Advances in Computation Structures Technology Civil_Comp Press, UK.

British Standards Institution, BS6399 Design Loadings for Buildings, Part 1: Code of Practice for Dead and Imposed Loads BSI, London, 1984.

British Standards Institution, Structural Use of Concrete, BS8110: Part 1:1985, BSI, London, 1985.

Brown, B. C., & Chandrasekaran, B., (1985) Expert Systems for a Class of Mechanical Design Activity, in Knowledge Engineering in Computer Aided Design (Gero, J.S., Ed), North Holland, pp. 259 - 290, 1985.

CIB - Dresden, CIB – Dresden, LAP – Stuttgart, 1995, http://wwwcib.bau.tu-dresden.design/combi/wpd2.htm.

Coad, P., & Yourdon, E. (1990), Object-Oriented Analysis. Englewood Cliffs, New Jersey: Yourdon Press/Prentice Hall, 1990.

Cross, R. (1996) A Search Tool to Enhance the Selection and Utilisation of Reusable Software Modules within the Object-Oriented Paradigm. Faculty of Science, Technology and Engineering, Edith Cowan University, Perth, Western Australia, Master of Science. Thesis, 1996.

Dasgupta, S., (1992) Herbert Simon's "Science of Design" Two Decades Later Keynote paper, Proc. 1st Int. Conf. on Intelligent Systems Eng., Aug. 1992, Edinburgh.

Doheny, J.G., & Monaghan, P.F., (1987) An Expert System for the Preliminary Stages of Conceptual Design of Building Energy Systems. Artificial Intelligence in Engineering. 1987, Vol. 2, No. 2.

Eastman, C.M., (1981) Recent Developments in Representation in the Science of Design, Proceedings of the 18h Design Automation Conference. Nashville, Tennessee, U.S.A., pp. 13 - 21.

Embley, D.W., Kurtz, B.D., & Woodfield, S.N., (1992), Object-Oriented Systems Analysis: A Model Driven Approach. Englewood Cliffs, New Jersey: Yourdon Press, 1992.

Fenves, S., & Baker, N., (1987). Spatial and Functional Representation Language for Structural Design. In Gero, J., (Ed.). (1987). Expert Systems in Computer-Aided Design. Amsterdam: Elsvier, 511, 1987.

Fikes, R., & Kehler, T. (1985). The Role of Frame-Based Representation in Reasoning. Communications of the ACM, 28(9), p. 904.

Fraser, D.J., (1981) Conceptual Design and Preliminary Analysis of Structures. Pitman Publishing Inc., Massachusetts.

Gailly, P.J. & Binot, J.L., (1991) Workshop on Merging Object-oriented and Logic Programming International Conference on Logic Programming, Paris, June 24, 1991.

Gardner, A.v.dL., (1989), Search An Overview., in The Handbook of Artificial Intelligence (Barr, A., Cohen, P.R., & Feigenbaum, E.A., (eds.), William Kaufmann, 1989.

Gavin, T.D., (1988) Design and Implementation of SPIKEE (Simple Prototype of an Intelligent Knowledge Engineering Environment. Submitted for MSc Degree, University of Dublin, Trinity College, October 1988.

Gero, J.S., Maher, M.L., & Zhang, W. (1988) Chunking Structural Design Knowledge as Prototypes. In Artificial Intelligence in Engineering: Design (Ed. J.S. Gero) pp. 3-21. Computational Mechanics Publications, Southampton, UK.

Ginsberg, M., (1993) Essentials of Artificial Intelligence. p. 121 Morgan & Kaufman.

Gordon, J.E., (1978), Structures or Why Things Don't Fall Down. Penguin.

Graham, I.M., (1994), Object-Oriented Methods. Second Edition, Addision-Wesley, 1994.

Harty, N. & Danaher, M. (1994) A Knowledge-Based Approach to Preliminary Design of Buildings. Paper presented in the Proceedings of the Institution of Civil Engineers Structures and Buildings, 1994, 104, May, pp. 135-144. Structural Board Paper 10312.

Harty, N. (1987) An Aid to Preliminary Design. In Sriram, D. & Adey, R.A. (Eds.). (1987). Knowledge Based Expert Systems in Engineering: Planning and Design. Computational Mechanics Publications.

Harty, N. (1987) An Aid to Preliminary Design. Department of Civil Engineering, Trinity College, Dublin, 1987, Ph.D. thesis.

Hasan, K., Ramsay, B., Ranade, S., & Ozveren, C.S., (1994) An Object-oriented Expert System for Power System Alarm Processing and Fault Identification. Proceeding 7th IEEE Electrotechnical Conference.

Hayes-Roth, F., (1985) Rule-Based Systems. Communications of the ACM, September 1985, 28(9), p. 921.

Henderson-Sellers, B., (1992), A Book of Object-Oriented Knowledge. Brookvale, NSW: Prentice Hall

Henderson-Sellers, B., (1997), A Book of Object-Oriented Knowledge – An Introduction to Object-Oriented Software Engineering. Upper Saddle River, New Jersey: Prentice Hall PTR, 1997.

HOOD Working Group (1989), HOOD Reference Manual. Issue 3.0. European Space Agency, Noordwijk, Netherlands, 1989.

Hopgood, A.A., (1993) Knowledge Based Systems for Engineers and Scientists. CRC Press.

Institution of Structural Engineers and The Institution of Civil Engineers, (1985) Manual for the Design of Reinforced Concrete Building Structures The Institution of Structural Engineers, UK 1985.

Intellicorp, (1996) Kappa-PC - System Description and Data Sheet. http://www.intellicorpcom/ kappa-pc /kappa-pc-24. html .

Intellicorp, (1996), Intellicorp Inc 1990-1997, Kappa-PC 2.4 Online Help – Kappa-PC 2.4 User Guide, Mountain View: California, Intellicorp Inc

ISO, 1992, Draft International Standard, DIS 10303 (STEP), Part1: Overview and Fundamental Principles, TC184SC4.

Jackson, M.A., (1983), System Development. Englewood Cliffs, New Jersey: Prentice Hall, 1983.

Joannides, F., & Weller, A. (1987) Structural Steelwork (Elementary Design to BS5950) Dublin, Ireland: Parthenon Press 1987.

Karhu, V. (1997) Product Model Based Design of Precast Facades. Itcon, Vol. 2 (1997), p. 1.

Kiernan, M.J., & Brown, K.E. (1996) The Application of Knowledge Based Techniques To Subsea Acoustic Data Interpretation. IEEE ??

Krishnamoorthy, C.S. & Rajeev, S., (1996), Artificial Intelligence and Expert Systems for Engineers (New Directions in Civil Engineering. CRC Press

Kunz, J.C., Kehler, T.P., & Williams, M.D., (1984) Applications Development Using Hybrid AI Development System. AI Magazine, Vol. 5, No. 3, pp. 41-54, Fall 1984.

La Rota, J.L., Biswas, G. & Basu, P.K. (1990), A Model-Based Approach to Structural Design. Proceedings of fifth Intl. Conference Applications of Artificial Intelligence in Engineering, 1990, p 3-22 Vol. 1.

Leclerc, A., (1979) Systems Engineering and the Civil Engineering Programme at Ecole Polytechnique De Montreal. Paper presented in the Proceedings of the Ist Canadian Seminar on Systems Theory for the Civil Engineer. Calgary, Alberta, 18-19 October, 1979. In Wirasinghe, S.C., & Jordaan, I.J. (Ed), pp. 9-17, University of Calgary, Alberta, Canada.

Lim, C. K, Choo, Y.S., & Nee. A.Y.C. (1996). Integrating Experienced-Based Knowledge in Design for Lift Installation of Offshore Structures. Paper presented in the First International Conference on Computing & Information Technology for Architecture, Engineering & Construction, 1996, 16-17 May 1996, Singapore .

Lin, Y.T, & Stotesbury, S.D. (1981). Structural Concepts and Systems for Architects and Engineers. New York: John Wiley and Sons, 1981.

Liu, X., & Gan, M., (1990). Neural Networks in Structural Preliminary Design. Paper presented at Conference 1990 .

Löfqvist, P. (1993). Preliminary Design of Bridges Used Knowledge-Based Systems. IABSE Colloquium "Knowledge-Based Systems in Civil Engineering", (pp. 223 - 232). Beijing, China.

Luger, G.F. & Stubblefield, W.A., (1993) Artificial Intelligence: Structures and Strategies for Complex Problem Solving 2nd Ed, 1993, Redwood City: Benjamin Cummings.

Maher, M.L., Fenves, S.J. & Garrett, J.H., (1988) Expert Systems for Structural Design In H Adeli (Ed.) Expert Systems in Construction and Structural Engineering 1988 pp. 87-121, London: Chapman and Hall Ltd.

Maher, M.L., Fenves, S.J. & Sriram, D. (1984) Tools and Techniques for Knowledge Based Expert Systems for Engineering Design Advances in Software Engineering, 1984, Vol. 6, No. 4, pp. 178-188.

Maher, ML. (1984) HI-RISE: A Knowledge Based Expert System for the Preliminary Structural Design of High Rise Buildings. Department of Civil Engineering, Carnegie-Mellon University, Pittsburg, 1984, Ph.D. thesis.

Maher, ML. (1987) Expert Systems for Structural Design. Journal of Computing in Civil Engineering, Vol. 1, No. 4, October, 1987, pp. 270- 283.

Malhotra, A., Thomas, J.C., Caroll, J., M. & Miller, L. A., (1980) Cognitive Processes in Design. International Journal of Man-Machine Studies, Vol. 12, pp. 119 – 140.

Martin, J., & O'Dell, J., (1992), Object-Oriented Analysis and Design. Englewood Cliffs, New Jersey: Prentice Hall, 1992.

Means (1982), 1982 Building Systems Cost Guide, Robert Snow Means Company Inc, 1982.

Merritt, D. (1998). Integrating Objects with Knowledge Bases. Object Magazine, March - April 1998, New York, SIGS Publications.

Merritt, F. S. (1985), Building Design and Construction Handbook. 4th Edition McGraw - Hill Inc.

Minsky, M. (1975), A Framework for Representing Knowledge In P. Winston (Ed.), The Psychology of Computer Vision, New York: McGraw-Hill, 1975.

Mitchell, N.J., (1997) Computer Aided Architectural Design. New York, Van Nostrand Rheingold Company, Inc, p.27.

Mittal, S & Agustin, A, (1986) A Knowledge-based Framework for Design Engineering Applications. International Artificial Intelligence Conference Proceedings pp. 856-865.

Mittal, S., & Araya, A., (1986) A Knowledge-Based Framework for Design. Proceedings of the Fifth National Conference on Artificial Intelligence., AAAI-86, pp. 856-865, August 1986.

Moss, C., (1991), Object-oriented Logic Programming. International Conference on Logic Programming, Paris, June 24th 1991.

Mostow, J., (1985) Towards Better Models of the Design Process AI Magazine, Vol. 6, no. 1, pp. 44 - 57.

Newell, A., Shaw, J.C., & Simon, H.A., (1963) Empirical Explorations with the Logic Theory Machine: A Case History in Heuristics. In Computers and Thought (Eds., E.A. Fegenbaum and J. Feldman), New York: McGraw-Hill, 1963, pp. 109-133.

Newell, A., and Simon, H.A., (1972), Human Problem Solving. Englewood Cliffs, N.J.: Prentice-Hall, 1972.

Newell, A., and Simon, H.A., (1976) Computer Science as Empirical Inquiry: Symbols and Search . 1975 ACM Turing Award Lecture, Communications of the ACM, March 1976, Vol. 19, no. 3 pp. 113 – 126.

Nilsson, N.J. (1971), Problem-Solving Methods in Artificial Intelligence. New York: McGraw-Hill, 1971.

O'Connor, A.D., (1992) Soft Systems Methodology – A Case Study of its use within an Australian Organization The Australian Computer Journal, Vol. 24, No 3, November 1992.

Page-Jones, M., (2000) Fundamentals of Object-oriented Design in UML New York, New York: Adison-Wesley, 2000.

Perry, W.E., (1995), A Standard for Auditing Computer Applications – Auditing System Requirements, New York: Auerbach Publications, Warren, Gorham & Lamont, 1995.

Pfaffenberger, P., (1997), Webster's New World Dictionary of Computer Terms Sixth Edition, New York: Macmillan, 1997.

Polya, G., (1957) How to Solve It. 2nd edition, New York: Doubleday Anchor, 1957 p. 112.

Quinn, M., (1993) Knowledge Based Expert Systems and Structural Engineering - Towards Expert Structural Computing. Department of Civil Engineering, University College, Galway, 1993, unpublished dissertation submitted for the requirements of the degree of Master of Engineering Science.

Rich. (1983) Artificial Intelligence (International Student Edition). Auckland, McGraw-Hill, p307.

Rumbaugh, J., Blaha, M., Premerlani, W., et al. (1991), Object-Oriented Modeling and Design. Englewood Cliffs, New Jersey: Prentice Hall, 1991.

Sabouni, A.R., & Al-Mourad, O.M., (1996). Quantitative Knowledge-based Approach for Preliminary Design of Tall Buildings. Artificial Intelligence in Engineering UK, Elsevier Science Ltd., pp. 143 - 154.

Sause, R., Martini, K. & Powell, G.H., (1992), Object-Oriented Approach for Integrated Engineering Design System. Journal of Computing in Civil Engineering, Vol. 6, No 3, July 1992, pp. 248-265.

Seidwitz, E., & Stark, M., (1986), General Object-Oriented Software Development. Software Engineering Letters, 86-002, 1986.

Simon, H.A., (1969) The Sciences of the Artificial. 2nd Edition, Cambridge, Massachusetts, The MIT Press, Massachusetts Institute of Technology.

Simon, H.A., (1996) The Sciences of the Artificial. 3rd Edition, Cambridge, Massachusetts, The MIT Press, Massachusetts Institute of Technology.

Soh, C.K., & Soh, A.K., (1988), IPDOJS - Example of Intelligent Structural Design System Journal of Computing in Civil Engineering October, Vol. 2, no. 4, pp. 329 - 345.

Sriram, D., Stephanopoulos, G., Logcher, R. Gossard, D. Groleau, N. Serrano, D., & Dundee, N. (1989) Knowledge based Application in Engineering Design: Research at MIT AI Magazine, Fall 1989, pp. 79-95.

The New Encyclopaedia Britannica, Volume 4, Macropaedia 1fifth Edition, 1988, p33.

The Haley Enterprise, (1992), Answers to Common Questions About AI. Available WWW http://www.haley.com/1974757313845250/PDF/AnswersToCommonQuestionsAboutAI.pdf

Topping, B.H.V., Jadid, M., & Kumar, B. (1991) Artificial Intelligence and Structural Engineering: A Bibliography. In B.H.V. Topping (Ed.), Artificial Intelligence and Structural Engineering, Civil_Comp Press, Edinburgh:UK, 1991, pp295-320.

Tsang, C.H.K., & Bloor, C., (1994), A Medical Expert System Using Object-oriented Framework. Proceeding 7th Annual IEEE Symposium on Computer Based Medical Systems.

Turk, Z. Isakovic, T. and Fischinger, M., (1994), Object-Oriented Modeling of Design Systems For Rc Building. Journal of Computing in Civil Engineering, Vol. 8, No 4, October 1994, p. 479.

Wasserman, A.I., Pircher, P.A., & Muller, R.J., (1990), The Object-Oriented Structured Design Notation for Software Design Representation. IEEE Computer, March 1990, pp. 50-62.

Wirfs-Brock, R., Wilkerson, B., & Wiener, L., (1990), Designing Object-Oriented Software. Englewood Cliffs, New Jersey: Prentice Hall.

APPENDIX A Functional Requirements

The following table lists the functional requirements drafted for the new design tool, referred to as the NOVA prototype system.

SUBTASK	SYSTEM FUNCTION
Specification	**Check Design Parameters** The Check Design Parameters function is required to allow the user to input and review the default design parameters in the knowledge base and to ensure that they are appropriate to the type of design envisaged by the user.
	Review Evaluation Features The Review Evaluation Features system function is required to allow the user to input and review a series of evaluation features also referred to as soft constraints for each structural subsystem. Each feature has an associated set of numerical variables, which are set up in the form of design targets. The user is required to review all variables and determine for each one whether or not it will be used for the current design, ie. 'set' and if so whether the parameter to be minimised, maximised or optimised or whether a set figure is to be achieved.
	Input User Requirements The Input user requirements system function is required to allow the user to input and review the specifications from which the new building is to be designed. The specifications include a list of the owner's requirements, which includes the type of building, location and dimensions. The user is also required to enter values for the loadings imposed on the floor and the wind loading acting on the sides of the building.
Formulation	**Design Vertical Subsystem** The Design Vertical Subsystem system function is required to address the first of the two major tasks of preliminary structural design, which is to select the vertical structural subsystem. This subsystem must be designed to resist lateral wind and earthquake forces. The system is required to provide three potential types of 2D vertical subsystems; these are wall subsystems and rigid and braced beam and column frames. In the design process they are conceived as 2D, wholes that act to pick up loads from the horizontal subsystems and also act to resist the horizontal, laterally acting forces.

Formulation (continued)	The horizontal subsystems must be supported by the vertical subsystems, likewise the vertical subsystems, which are generally slender in nature and unstable, must be held in place by the horizontal subsystems. The system is required to select all possible combinations of the subsystems available in the knowledge-based and subsequently to eliminate any infeasible design proposals. The system simulates the designer's decision-making process, which uses heuristic knowledge. The design process is a series of steps at which alternative designs are produced repeatedly with greater levels of detail.
	Get Assumed Sizes The Get Assumed Sizes function is required to estimate values for the overall floor depth, including beam and slab values. The function is based upon the one in the DOLMEN system and it performs a similar action. The function simulates the design processes for reinforced concrete structures as recommended in the *'Manual for the Design of Reinforced Concrete Building Structures'*, Institute of Structural Engineers (1985), except in the case of rigid frame designs. These processes were written to effect designs in accordance with the British Standards Structural Codes of Practice, BS1 (1985, 1), BS8110: Part 1:1985 and BS1 (1985, 2), BS5950:Part 1:1985. For reinforced concrete buildings the system is required set the floor depth, slab depth and initial beam depth and for steel buildings to set the slab type, floor depth, slab depth, intermediate beam spacing and steel deck unit.
	Set Initial Sizes The Set Initial Sizes system function is required to set the initial sizes for the beam and column sections.
	Detail Vertical Subsystem The Detail Vertical Subsystem system function is required to assess proposed structural configurations for loading and sizing, with reference to relevant building codes and to select or eliminate them. Detailing is required after completion of the final level of the vertical structural subsystem, which is the *Vertical_2D_Wide_Location* level. The system simulates the engineer's design process, completing approximate analysis, detailing and subsequent checking. The system initially selects an appropriate loading then applies it to the structural configuration. Then it calculates the displacements and forces on the structural members and then it selects initial sizes for the beams, columns, slabs and walls, which make up the different configurations and then analyses them, using heuristic rules.

Formulation (continued	**Design Horizontal Subsystem** The Design Horizontal Subsystem system function is required to address the second of the two major tasks of preliminary structural design, which is to select the horizontal structural subsystem. This is a frame of floors, beams and columns, which must be designed to resist the building's gravity loading. It is described in Lin (1981) as a *2D whole* that acts vertically to carry the floor or roof loads in bending mode, and that acts horizontally as a diaphragm and/or column connector. The system is required to form horizontal subsystems from a wide range of combinations of flat plates, ribbed slabs, reinforced concrete, slabs and beams, waffle moulds, precast units and composite steel decking.
	Detail Horizontal Subsystem The Detail Horizontal Subsystem system function is required to assess proposed floor systems for loading and sizing, with reference to relevant building codes and to select or eliminate them. Detailing is required after completion of the final level of the building hierarchy, which is the *Intermediate_Beams* level. As with the vertical subsystem, the system simulates the engineer's design process, completing approximate analysis, detailing and subsequent checking. Several floor system options are available and different detailing processes are required for them. In the case of reinforced concrete slabs, the system initially sets values for the slab depth, floor depth and the maximum moments on the slab in the X and Y directions. It then calculates the area of steel in the slab and in the corners of the slab in the X and Y directions and the mass of steel in the slab per cubic metre.
Evaluation	**Evaluate Vertical/Horizontal Subsystem** The Evaluate Vertical Subsystem and Horizontal subsystem system functions are required to calculate the values of the different evaluation features. The appropriate features are identified based on the initial selection made by the user and the values are calculated using the appropriate methods attached to the feature objects in the system.
	Produce Evaluation Report The Evaluation Report system function produces a report in columnar form, which displays for each whether or not the feature has been selected for use in the evaluation, the minimum and maximum values, the objective of the target and its importance value. The report then displays the feature values for each proposed design, including the feature value, the percentage optimisation achieved for the value and the weighted value of the feature's evaluation score.

Evaluation (continued)	The also ranks the proposed designs and displays the top *n* designs, where *n* is a value preselected by the user. The display is presented in a transcript window in the centre of the main session window.
User Interface	**Produce Design Reports** The Design Report system function also produces a transcript window report in the centre of the main session window. The function displays key design details for designs, which the user can request via a window dialog. The display overwrites anything previously displayed in the window.

Table A.1 List of functional requirements.

During the high-level analysis stage the writer identified twelve system functions. The requirement specification stage, which is described in section 6.3, followed on from the high-level stage and was designed to provide more information about each function, including the lower level processes within each function. During the requirements stage the writer identified twenty-eight different key design processes, which were required to support the twelve major functions of the new system.

Due to the limited size of this report, the writer has not included details of all twenty-eight processes in the report. However, *Detail Braced Frame Narrow Options*, one of the key processes required to support the *Detail Vertical Subsystem* function is described below as an example of how the requirements were documented.

SUBTASK **Formulation**

SYSTEM FUNCTION **Detail Vertical Subsystem**

Definition

- The design tool has an initial series of 'test and eliminate' functions, which have been written for designs at each level in the building hierarchy and which are used in the first instance to prevent unlikely designs being added to the search tree. They use heuristic knowledge to delete alternatives without further study, however, they do not invoke the inference engine and no production rules are used.

  The detailing process is a secondary level of testing, which weeds out those designs that are not structurally sound and which is applied to designs, which are not eliminated at the outset. This type of testing requires a more detailed look at the design and invokes the inference engine

referring to a set of rules designed to check design alternatives. The inference engine is invoked by a checking function, which calls the system's forward chaining mechanism.

Detail Vertical Subsystem Process Flow (see figure A.1)

- The Detail Vertical Subsystem function consists of processes for six particular designs :
    - Detail Braced Frame Narrow Options;
    - Detail Rigid Frame Narrow Options;
    - Detail Shear Wall Narrow Options;
    - Detail Braced Frame Wide Options;
    - Detail Rigid Frame Wide Options;
    - Detail Shear Wall Wide Options
- Listed below are the major steps in the *Detail Vertical Subsystem* process for a particular design:
    - Select Design Parameters;
    - Estimate Initial Sizes;
    - Calculate Loadings;
    - Select Loadings;
    - Check Design.

Detailing involves calculating estimates for the physical components. Subsequent testing then relies on the ability of the system to locate suitably sized steel sections in the steel sections database. If the system is unable to locate a section big enough, then it marks the design to be eliminated.

There are two subsets of detailing functions, those required for the vertical subsystem and those required for the horizontal. Detailing is applied to the vertical subsystem when the locations of the structural alternatives have been selected, ie. at the *Vertical_2D_W_Location_Level*. For the horizontal subsystem or floor system, it is performed when the locations of the support and intermediate beams have been decided and the floor system has been designed.

The vertical subsystem detailing functions contains functions to detail the three vertical structural subsystem options: braced frame, rigid frame and shear wall.

The system requires a series of rulesets for checking the validity of roughly designed alternatives. Some checks are concerned with the satisfaction of the most important parts of the structural codes. These rulesets are shown in table 9.1. They also check that designs are of reasonable dimensions, which have been predetermined during the specification stage.

The rulesets all have names of the form *Rs_For_Chk_Det_xx_Alts,* where xx is the name of the option to which the ruleset relates. Each detailing function calls the *Check_Design* function to test the designs at various stages in the process. The function is always called with a parameter. For example, when rigid frame checking is required the function call is coded *Check_Design(Bldg, RF);* the parameter RF indicates that the function is to use the ruleset *Rules_For_Checking_Detailed_RF_Alternatives*.

Check_Design uses the appropriate rule from the *Rules_For_Checking_Detailed Alternatives* to check and eliminate any unsatisfactory design. Every time a function needs to check if a steel section has been found, then *Check_Design* is called with the parameter *Element* and it refers to ruleset *Rules_For_Checking_Detailed_ Elements_Alternatives*, which contains one rule *R1_About_Steel Sections.* This check is used with all design option tests to ensure that a section has actually been found.

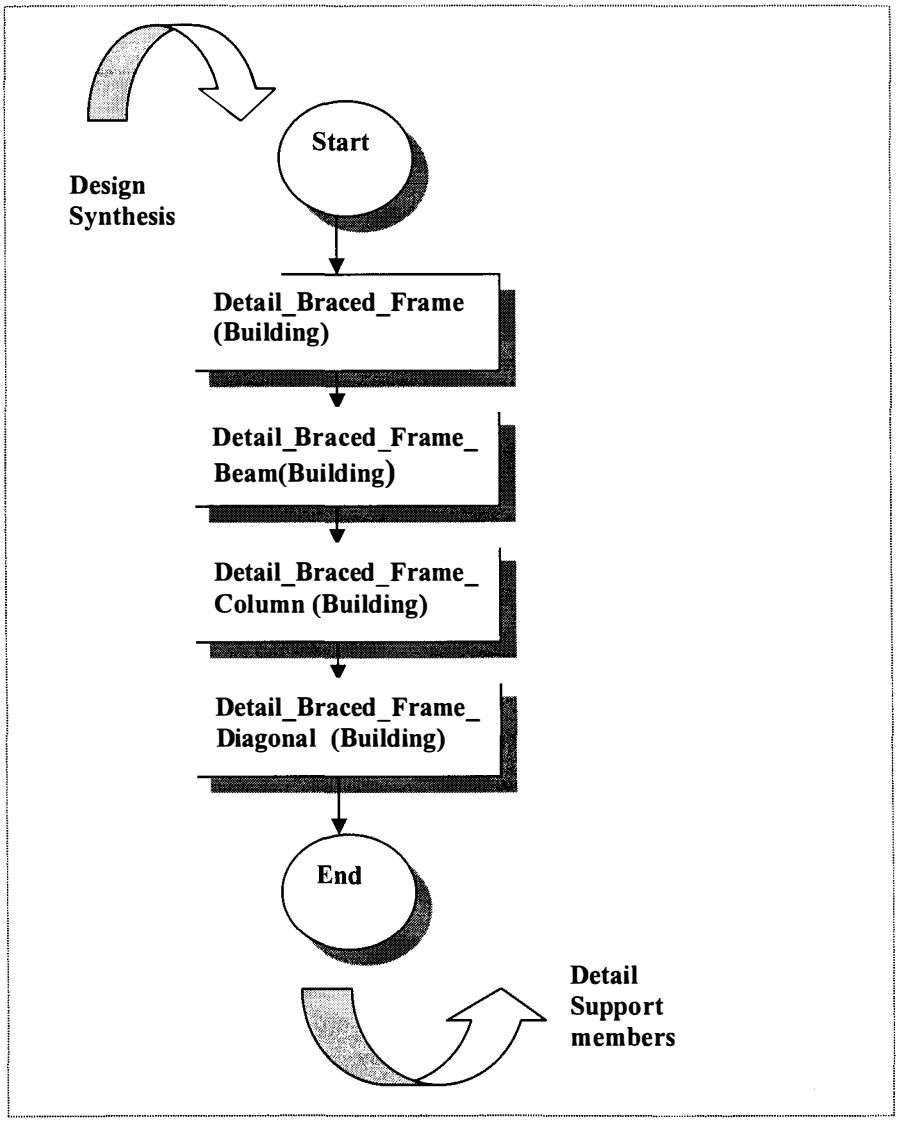


Figure A.1 Detail Braced Frame options

In the *Detail_The_Vertical_Subsystem* process flow, there are two common series of functions, which are executed for all design options, and which estimate the floor and beam and column sizes. Then three alternate process flows are used for detailing the rigid frame, braced frame and shear wall partial designs.

When the *Detail_Vertical_System* input button is selected, the detailing function is called and an initial list of items to be designed and analysed, *Global:New_Designs_In_Vert_2D_W_Loc* is created. This list consists of the partial designs on the fringe of the search tree, which have been created at the *Vertical_2D_Wide_Location_Level* of the hierarchy. Each item on the list is detailed in turn; the Kappa messaging facility is used to initiate the appropriate design method. This messaging system is described later in this section.

Details of the design and programming of the detailing function for Braced Frame options are reproduced below. Similar functions are applied to rigid frame and shear wall partial designs; however, their descriptions have not been included in this report.

The system should be support the following major steps associated with processing detailing for braced frame options.

Detailing of Braced Frame Design Options

For the purposes of this model the braced frame was treated as a vertical truss, where the columns act as chords and the bracing acts as diagonals in a K shape. The arrangement of uprights, horizontal members and diagonals was deemed to be as shown in Figure A.2.

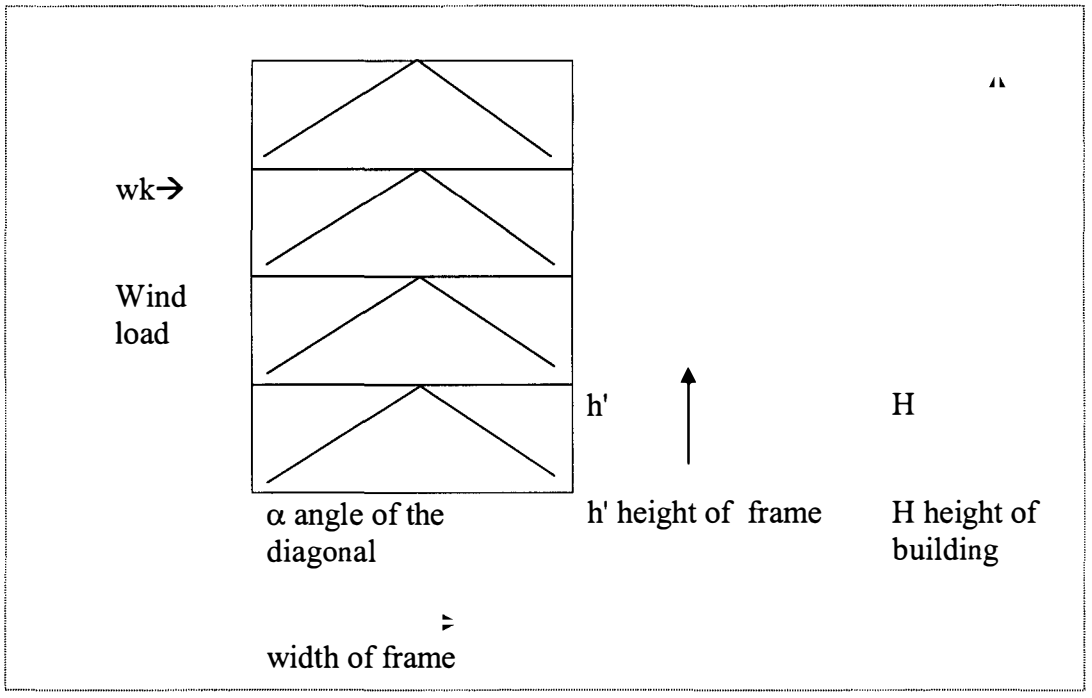


Figure A.2 Braced Frame Construction

Joists were not designed and the beams and columns were Universal Beam Sections and Universal Column Sections; the diagonals were formed by using two equal angle sections. The overturning moment due to the wind load was assumed to be equally distributed to all the braced frames. Only the most heavily loaded frame was designed, furthermore, only columns for the bottom storey were designed. This is in contrast to the design strategy used in HI-RISE, which designs one column every 'n' floors.

Sizing. Uplift; the wind load acting on the side of the building causes an overturning moment, which must be resisted by the reaction R, shown in Figure B.10, to provide a stable structure.

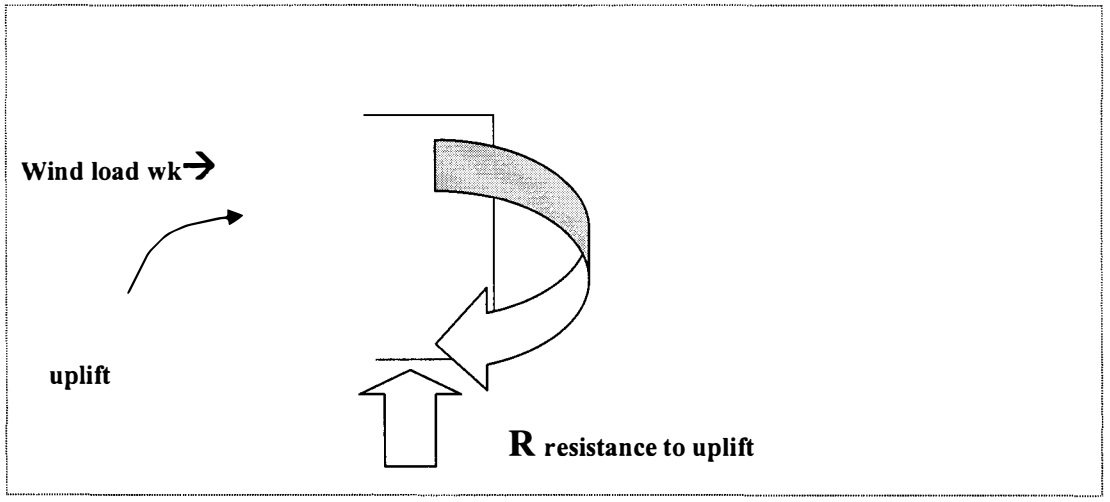


Figure A.3 Resistance to Wind Load

The download reaction must be provided by the dead load, and thus the dead load in the column must be greater than the uplift to prevent overturning. The uplift was calculated as:

$$\text{Uplift} = \frac{\text{wind load x (height of building)}^2 \text{ x width of building}}{\text{2 x number of frames x width of frame}} = \frac{wkH^2B}{2nfL}$$

An overview of the process, used to detail braced frame options, is shown in Figure C.3.

After the initial sizing had been completed, the first detailing function called, which was specific to detailing braced frame options, was the function *Detail_Braced_Frame.* This function first checked to see if the aspect was narrow or wide. Then it calculated or set the following variables:

- width of building perpendicular to frames,
- width of frame,
- weight of column,
- type and location.

It then started a looping process, which continued until the slot, used to flag the function's termination condition; *Detail_Status* was either set to *Satisfactory* or to *Deleted.* Inside the loop the program calculated the following variables:

- dead load estimate,
- number of frames,

Then using the dead load estimate the function calculated the

- dead load on columns,
- uplift.

The function then called the *Check_Design* function with the *Braced Frame* parameter (BF) to determine the *Detail_Status.* The *Check_Design* function referred to the 2 rules, which were written for checking detailed braced frames. These rules are used to check whether or not uplift was greater than dead load. If the uplift was greater then the *Detail_Status* was set to *Unsatisfactory* and a function was called to increase the sizes of beams and columns. The new sizes were then fed back into the looping process.

With regard to the wind load, the uplift had to be less than the dead load calculated using the appropriate steel sections, and if it was not; larger, heavier sections were to be selected, until the dead load was greater than the uplift. If that proved to be impossible, the design alternative in question was deleted. When a suitable section had been chosen, the function passed control to the next function in the sequence, *Detail_Braced_Frame_Beam,* which initially calculated the

- live load estimate as follows:

```
Live_Load_Estimate ← Imposed_Load * 1.6 *
                     Width_Of_Bay_Perp_To_Frame *
                     Largest_Number_Of_Area_Units * 0.5 *
                     Width_Of_Frame
```

- moment in the beam,

Moment_In_Beam ← Live_Load_Estimate + Deadload_Estimate
* 1.4 * Width_Of_Frame
/ 8.0

and

- design strength of steel, and then
- the SXX parameter

It then chose a steel beam section using the SXX parameter. Again the *Check_Design* function was called to ensure that a section had been selected, and if so, the function continued and then re-calculated a new live load estimate, then it re-calculated the dead load estimate and the depth of the beam, for which it called the function *Get_Steel_Section_Value,* with the parameter set to 'depth'. If the detail status, as determined by the *Check_Design* function, was *Satisfactory,* the function then called the next function in the sequence, *Detail_Braced_Frame_Column.* If the detail status had been set to *Delete* then the design was eliminated from further consideration.

The next function in the sequence, *Detail_Braced_Frame_Column,* initially calculated

- dead load on column,
- imposed load on the column,
- axial load on the column as follows:

Axial_Load_Due_Wind_Load_Column ← Wind_Load *
Width_Of__Perp_To_Frames
* Height ^{2} / Width_Of_Frame
* 2 * Number_Of_Frames_Narrow

and then,

- force in the column.

Force_In_Column ← Max(Deadload_On_Column* 1.4
+ Imposed_Load_On_Column * 1.6,
Deadload_On_Column * 1.2
+ Imposed_Load_On_Column * 1.2,
Axial_Load_Due_Wind_Load_Column * 1.2)

Next it calculated the:

- slenderness ratio,
- compressive strength of the steel in the section, and then
- assumed radius of gyration of the column, and
- compressive strength of the steel in the column,

It then calculated the area of steel required in the column using the ratio of the force in the column divided by the compressive strength of steel in the section, then it chose a new section, using the area parameter and finally it checked the design. If the detail status of the design was *Satisfactory,* it then called the next function in the series, which was *Detail_Braced_Frame_Diagonal.*

This function selects the diagonal sections. This selection is based on a calculation of the force in the diagonals. First the function calculates the base shear, V, which is found from the following relationship:

$$V = \frac{wk * H * B}{nf}$$

where:

wk = wind load

H = height

B = width of the building perpendicular to the frame

Nf = the number of frames

α = the angle shown in Figure B.9

The force is then found from the relationship:

$$F_{diag} = \frac{\text{base shear on the frame}}{2 \sin \alpha}$$

$$F_{diag} = \frac{V * 1}{2 \sin \alpha} = \frac{wk * H * B}{nf \, 2 \sin(\tan^{-1}(2h'/L))}$$

This value is then calculated in the following steps:

```
Width_Of_Frame                  ←  Narrow_Dim
Width_Of_Bldg_Perp_To_Frames    ←  Wide_Dim * Number_Of_Wide_Bays
                                       * Width_Of_Frame
Height                          ←  Total_Height
Sine_Alpha                      ←  Sin( Atan( Height
                                   / ( (Number_Of_Stories )
                                   * Width_Of_Frame  *  0.5 ) ) ) )

Force_In_Diagonal_Narrow        ←  ( Wind_Load )
                                   * 1.4 *  Width_Of_Bldg_Perp_To_Frames  *
                                   Height )
                                   / ( 2 * Sine_Alpha *  Number_Of_Frames_Narrow
                                   )))
```

Having calculated the force in the diagonal the function then:

- sets the slenderness ratio to 400 to 1,
- computes the area of reinforcement required in the section using the relationship

```
Area  ←  Force_In_Diagonal_Narrow
```

/ (100.0 * 100.0 * 2.0))

then it sets the compressive strength of the section to 1 and then the function enters a looping process, whereby it resets the slenderness ratio as the maximum of

1) 0.85 times the length, divided by the radius of gyration about the x axis, and

2) the sum of 0.7 times the length, divided by the radius of gyration about the x axis plus 30.00

3)

The designer of DOLMEN attributes this relationship to clause 4.7.10.3 of BS5950. The following pseudo code reflects these computations:

```
Length          ←   Sqrt(( Storey_Height ) 2.0 )
                    + ((0.5 * Width_Of_Frame ) 2.0 ) )

Diag_RXX        ←   Get_Steel_Section_Value( Diagonal,
                    Diagonal_Section_Narrow,
                    Radius_Of_Gyration_XX ) )

Slenderness_Ratio ← Max( (Length1 * 0.85 ) / ( Diag_RXX * 10.0 ),
                        ( ( Length1 * 0.7 ) / ( Diag_RXX * 10.0 ) )
                    + 30.0 ) )
```

The function then chooses a section with the required area, which is calculated after making the assumption that the design stress is 100 N/mm^2. If this yields a column with slenderness greater than 180, then a section with slenderness as close as possible to and under 180 is chosen. When a section has been chose, the detailing of the braced frame design option is complete and the function calls the *Approximate_Supports_Detail* function to detail the supporting beams. If a section cannot be chosen design is eliminated from further consideration.

The process may be summarised as follows:

/**/

Detail_Braced_Frame for selected Building

/**/

- calculate Width_Of_Building_Perp_To_Frames using Wide_Dim * Building:Wide_Bays
- calculate Weight_Of_Column using Get_Steel_Section_Value(Column, Column_Section_Narrow , Mass_Per_Metre) * 9.81 * 0.001 * Storey_Height
- repeat until Design_Status is equal to Satisfactory or Design_Status is equal to Deleted
- calculate Dead_Load_Estimate using the Calculate_ Dead_ Load_ Estimate_ For_ Braced_ Frame function (which calculates the largest dead load due to floor and beams on an area unit in knewtons)
- calculate Height using Building:Total_Height
  - calculate Uplift using

Wind_Load * 1.4 * Width_Of_Bldg_Perp_To_Frames

* (Height)^2 / (2 * Width_Of_Frame * Number_Of_Frames

- calculate Deadload_On_Column using

(Deadload_Estimate_Narrow * 0.5) + Weight_Of_Column)* (Stories)

- reduce the value of Deadload_On_Column by 10.0
- set the value of Design_Status using function Check_Design Building using parameter Braced_Frame

Check_Design uses the following rule to check the design for uplift.

```
/************************************
    **** RULE: R2_For_Braced_Frame
    **** Uplift greater than dead load
    ************************************/
MakeRule( R2_For_Braced_Frame, [],
   Altbldg:Uplift_Narrow > Altbldg:Deadload_On_Column_Narrow,
   SetValue( Altbldg:Detail_Status, Unsatisfactory ) );
SetRuleComment( R2_For_Braced_Frame, "Uplift greater than dead load" );
```

- if Design_Status is equal to Unsatisfactory
  then call function Increase_BF_Sizes(Bldg)
- if (Design_Status is equal to Satisfactory)
  then call function Detail_Braced_Frame_Beam(Bldg)
  else eliminate the design

/***/

Detail_Braced_Frame_Beam for selected Building (for the Narrow perspective)

/***/

- assign Width_Of_Bay_Perp_To_Frames the value Wide_Dim
- assign Width_Of_Frame the value Narrow_Dim
- assign Dead_Load_Estimate the value Dead_Load_Estimate_Narrow
- calculate Live_Load_Estimate using

Imposed_Load
* 1.6 * Width_Of_Bay_Perp_To_Frame
* Largest_Number_Of_Area_Units
* 0.5 * Width_Of_Frame

- assign New_Dead_Load_Estimate the value Dead_Load_Estimate
- repeat until the Design_Status is equal to Deleted or until Design_Status is equal to Satisfactory
- increment the loop counter
- if loop counter >= 1 then assign Beam_Section the value of New_Beam_Section
- calculate Moment_In_Beam using

Live_Load_Estimate

+ (New_Deadload_Estimate * 1.4) * Width_Of_Frame / 8.0

- set the value of Design_Strength_Of_Steel using the function Calc_Design_Strength_Of_Steel(Beam_Section_Narrow, Beam)
- set the value of SXXParm to Moment_In_Beam / Design_Strength_Of_Steel
- find the New_Beam_Section_Narrow using the function Choose_Steel_Beam_Section_SXX(SXXParm)
- set the value of Design_Status using function Check_Design Building using parameter Elements

The function uses the following rule to determine whether or not a steel section has been chosen.

```
/************************************
     **** RULE: R1_About_Steel_Sections
     ************************************/
MakeRule( R1_About_Steel_Sections, [],
   Altbldg:Test_Section #= Nil,
   SetValue( Altbldg:Detail_Status, Deleted ) );
```

- if Design_Status is equal to Satisfactory

  then calculate New_Dead_Load_Estimate using function Calculate_Dead_Load_Estimate_For_Braced_Frame Building
- if New_Beam_Section is equal to Beam_Section or

  loop counter >= 10 and New_Dead_Load_Estimate < Dead_Load_Estimate

  then assign Design_Status the value Unsatisfactory

  increment loop counter
- if Design_Status is equal to Satisfactory

  then assign the value of Dead_Load_Estimate_Slot New_Dead_Load_Estimate
- assign the value of Beam_Depth with the value Get_Steel_Section_Value (Beam New_Beam_Section, Depth)
- call function Change_Steel_Beam Building (New_Beam_Section) to input the new beam section.

```
/*****************************************************************************/
```

Detail_Braced_Frame_Column for selected Building

```
/*****************************************************************************/
```

- assign the value of Width_Of_Building_Perp_To_Frames to Wide_Dim * Wide_Bays
- assign the value of Width_Of_Frame to Narrow_Dim
- assign the value of Width_Of_Perp_Bay to Wide_Dim
- assign the value of Dead_Load_Estimate to Dead_Load_Estimate_Narrow
- assign the value of Dead_Load_On_Column to Deadload_On_Column_Narrow
- assign the value of Height to Total_Height of the Building
- calculate Imposed_Load_On_Column using

  Imposed_Load * Width_Of_Perp_Bay

  * 0.5 * Width_Of_Frame * Stories
- calculate Axial_Load_Due_Wind_Load_Column using

Wind_Load * Width_Of_Bldg_Perp_To_Frames

* (Height)^ 2 / Width_Of_Frame *

2 * Number_Of_Frames_Narrow

- calculate Force_In_Column using

  Max(Deadload_On_Column

  * 1.4 + Imposed_Load_On_Column * 1.6 * Deadload_On_Column * 1.2

  + Imposed_Load_On_Column * 1.2 Axial_Load_Due_Wind_Load_Column * 1.2

- select Column_Section using function Choose_Steel_Column_Section_Area(Force_In_Column / 100.0 * 100.0

- calculate Slenderness_Ratio using

  Clear_Height * 1

  / Get_Steel_Section_Value(Column, Column_Section, Radius_Of_Gyration_YY)

  / 10.0

- if Slenderness_Ratio >= 180.0)

  then calculate Assumed_Column_Radius_Of_Gyration using

  (Round5 Building:Clear_Height * 1.0 / 180.0)

- select Column_Section using function

  Choose_Steel_Column_Section_With_Radius_Of_Gyration(
  Assumed_Column_Radius_Of_Gyration / 10)

- assign the value of Building Column_Section_Slot Column_Section)
- calculate Design_Status (Check_Design Building Elements))
- if Design_Status is equal to Satisfactory
  - then increment the loop counter
  - repeat until the Design_Status is equal to Deleted or until Design_Status is equal to Satisfactory
- assign Compressive_Strength_Of_Section the value 1
- If New_Column_Section is equal to Column_Section or

  loop counter >= 10 and

  Get_Steel_Section_Value(Column,Column_Section, Area) <

  (Force_In_Column / Compressive_Strength_Of_Section / 100.0)))

- if loop counter >= 1

  then assign Column_Section the value New_Column_Section

- calculate Slenderness using

  Clear_Height * 1.0

  / Get_Steel_Section_Value (Column, Column_Section, Radius_Of_Gyration_YY)

  * 10.0

- calculate Compressive_Strength_Of_Section using

  function Calc.Compr.Str.Of.Steel.In.Column(Slenderness, Column_Section, Column)

- calculate New_Column_Section using

Get_Steel_Section_Value(Column, Column_Section, Area, Force_In_Column / Compressive_Strength_Of_Section /100.0)

- assign the value of Building Column_Section_Slot the value Column_Section
- Detail_Braced_Frame_Diagonal Building

/**********/

Detail_Braced_Frame_Diagonal for selected Building

/**********/

- calculate Width_Of_Building_Perp_To_Frames using Wide_Dim* Wide_Bays
- calculate Width_Of_Frame using Building:Narrow_Dim
- calculate Height Building:Total_Height))
- calculate Sine_Alpha using Sin(Atan(Height / Stories * Width_Of_Frame * 0.5))
- calculate Force_In_Diagonal_Narrow using

  Wind_Load * 1.4 * Width_Of_Bldg_Perp_To_Frames* Height) / (2 * Sine_Alpha * Number_Of_Frames_Narrow)));
- assign Slenderness_Ratio the value 400.0
- calculate Areq using BFVar:Force_In_Diagonal_Narrow / (100.0 * 100.0 * 2.0)
- repeat until Slenderness_Ratio >180.0
- assign Diagonal_Section using the function Choose_Steel_Diagonal_Section_With_Area(Areq)
- calculate Length1 using Sqrt(Storey_Height) ^ 2.0) + ((0.5 * Width_Of_Frame) ^ 2.0)
- calculate Diag_RXX using

  Get_Steel_Section_Value(Diagonal,Diagonal_Section_Narrow, Radius_Of_Gyration_XX)
- calculate Slenderness_Ratio using

  Max((Length1 * 0.85) / (Diag_RXX * 10.0),

  ((Length1 * 0.7) / (Diag_RXX * 10.0)) + 30.0));
- assign the value of Areq to Areq + 1
- calculate Design_Status using function Check_Design Building Elements
- if Design_Status is equal to Satisfactory
  - then increment the loop counter
  - repeat until the Design_Status is equal to Deleted or until Design_Status is equal to Satisfactory
- repeat until the New_Diagonal_Section_Narrow is equal to Diagonal_Section_Narrow and

  loop counter < 10 and

  Get_Steel_Section_Value(Diagonal, Diagonal_Section_Narrow, Area) >

  (Force_In_Diagonal_Narrow / (Compressive_Strength_Of_Section * 100.0 * 2.0

  if loop counter >= 1

  then assign the value of Diagonal_Section to New_Diagonal_Section
  - calculate Length1 using Sqrt(Storey_Height) ^ 2.0) + ((0.5 * Width_Of_Frame) ^ 2.0)
  - calculate Diag_RXX using

    Get_Steel_Section_Value(Diagonal,Diagonal_Section_Narrow,Radius_Of_Gyration_XX));

- calculate Slenderness_Ratio using

  Max((Length1 * 0.85) / (Diag_RXX* 10.0),

  ((Length1 * 0.7) / (Diag_RXX * 10.0)) + 30.0));

- calculate Compressive_Strength_Of_Section

  using function Calc_Compr_Str_Of_Steel_In_Column(Slenderness, Diagonal_Section, Diagonal))

- calculate New_Diagonal_Section using

  function Choose_Steel_Diagonal_Section_With_Area

  (Force_In_Diagonal / Compressive_Strength_Of_Section) * 100.0 * 2)

- assign the value of Building Diagonal_Section_Slot Diagonal_Section))
- check if the system is detailing at the Vertical_Subsystem level and if it is call the function Approximate_Supports_Detail to start sizing and checking the Building' supports.

APPENDIX B System Notes

The following section contains rough workings and informal diagrams

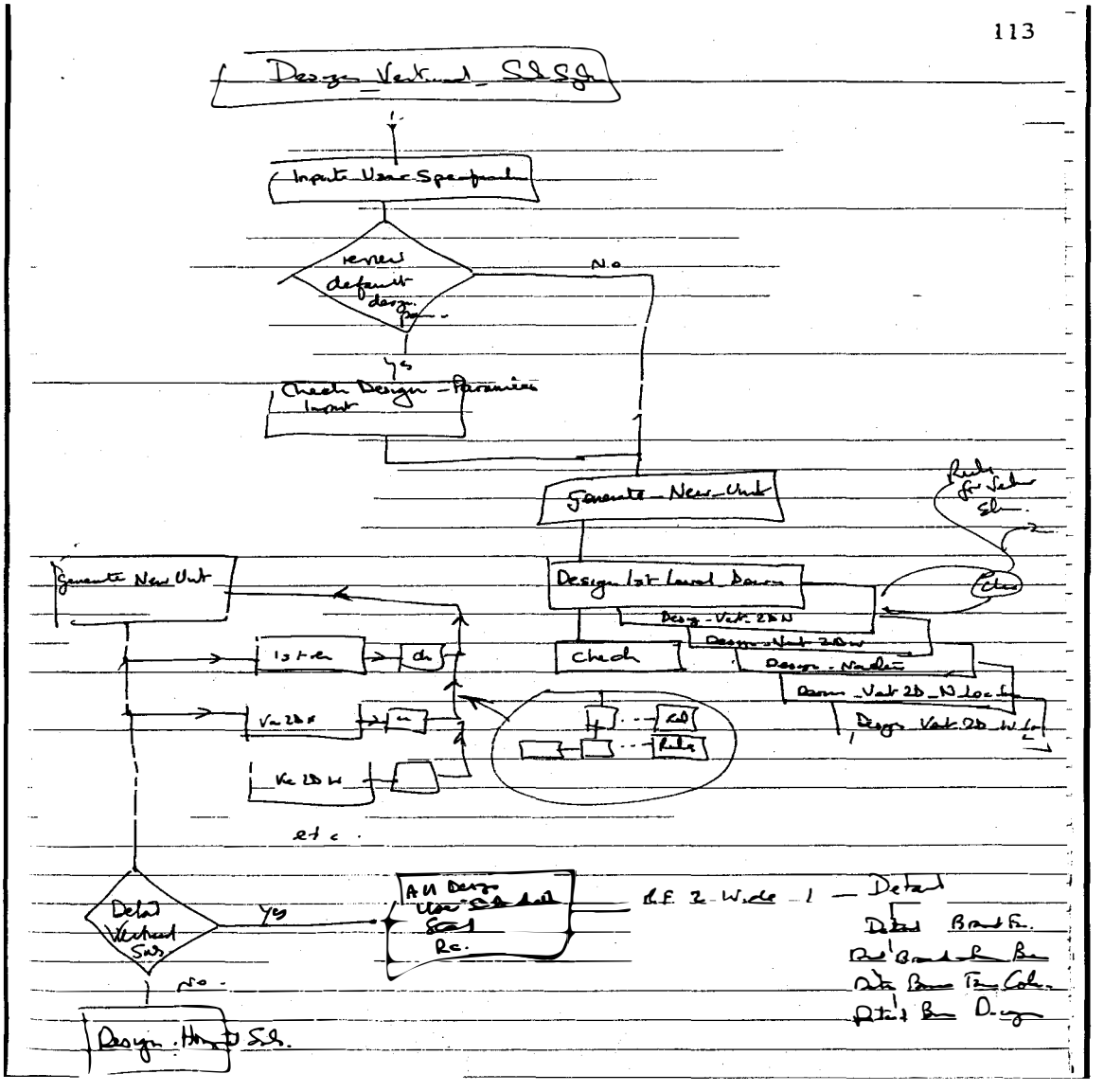


Figure B.1 Top level functional model for Design Vertical Subsystem process

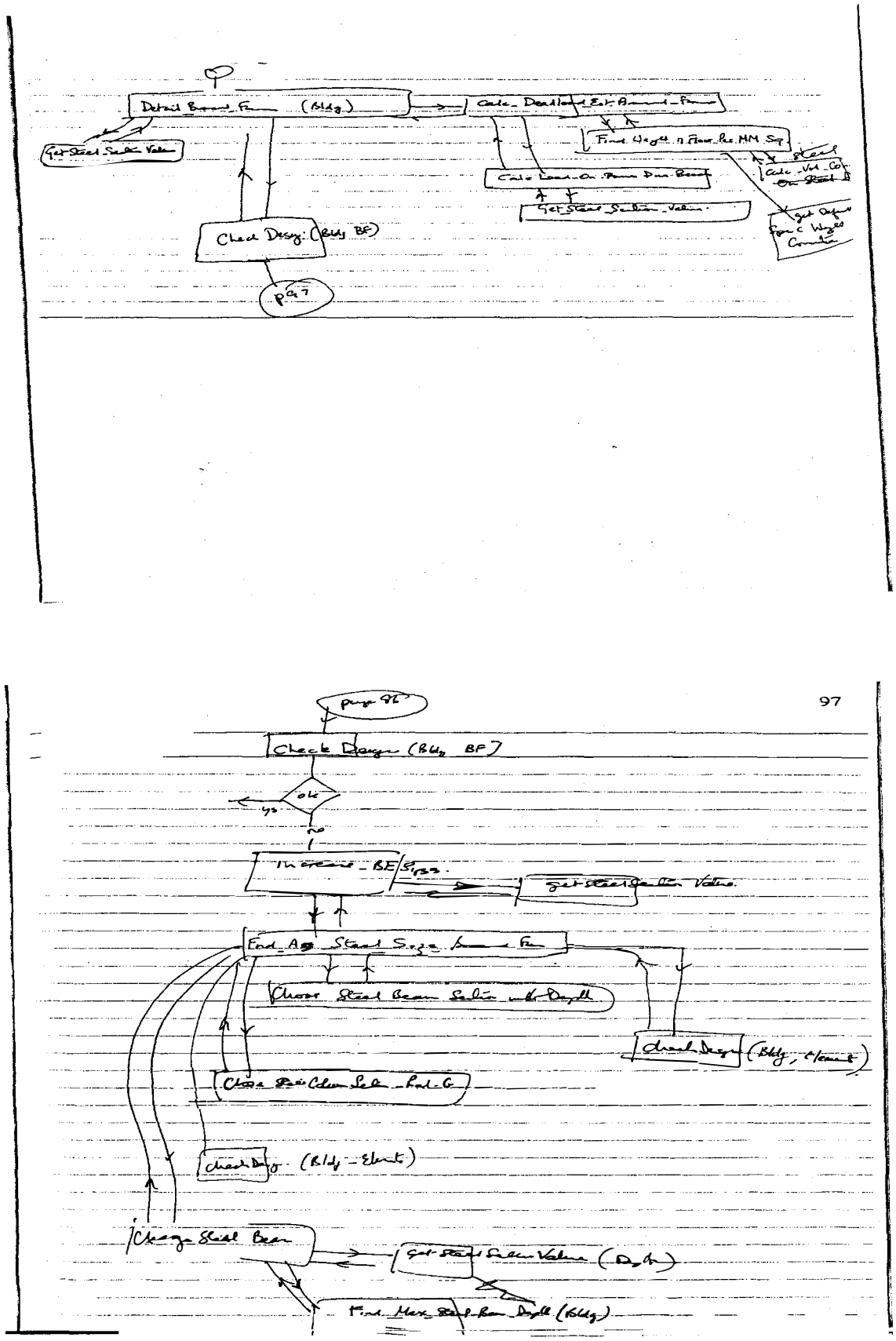


Figure B.2 **Functional Model for the Detail Braced Frame process**

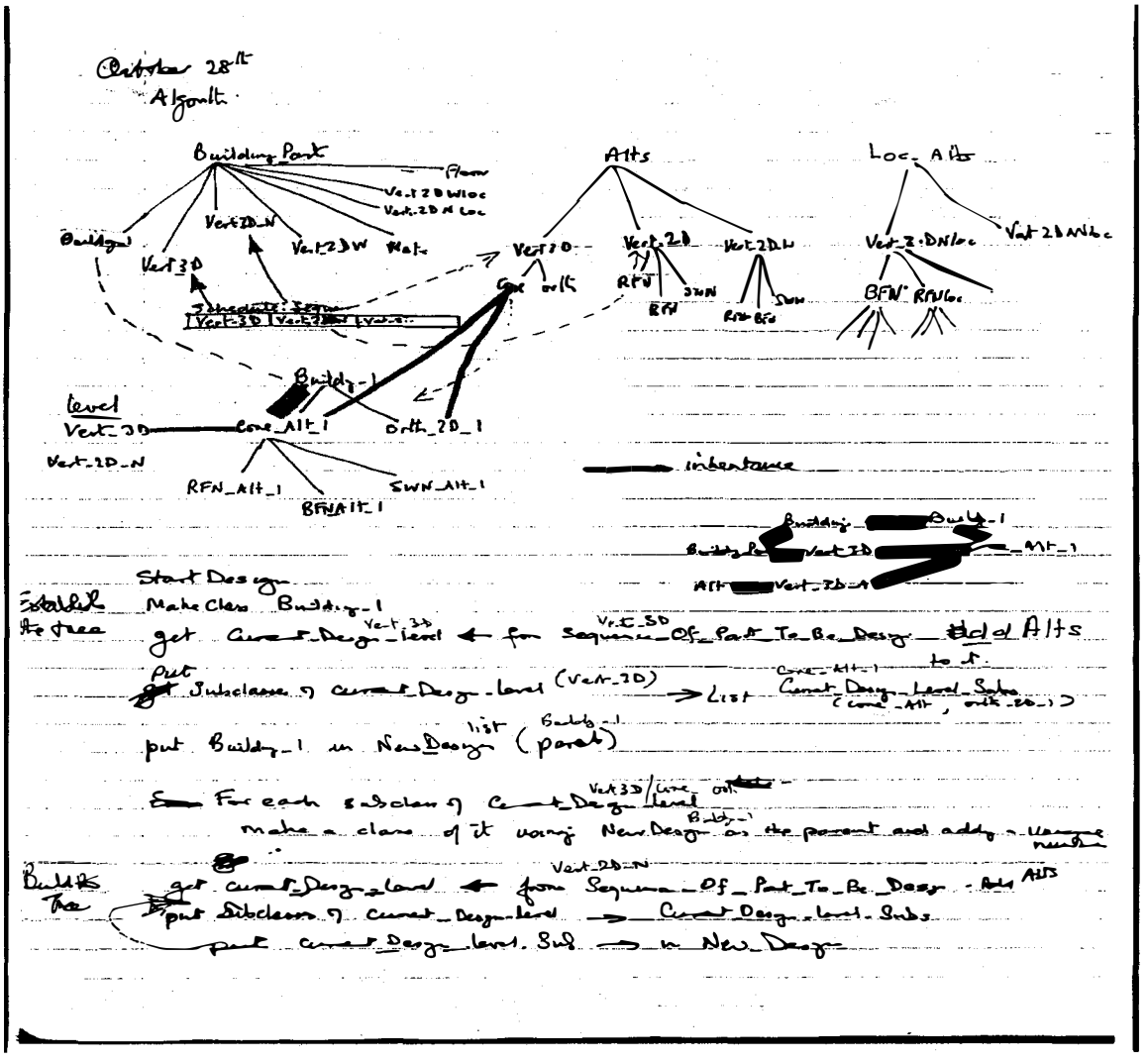


Figure B.3 Workings for inheritance relationships

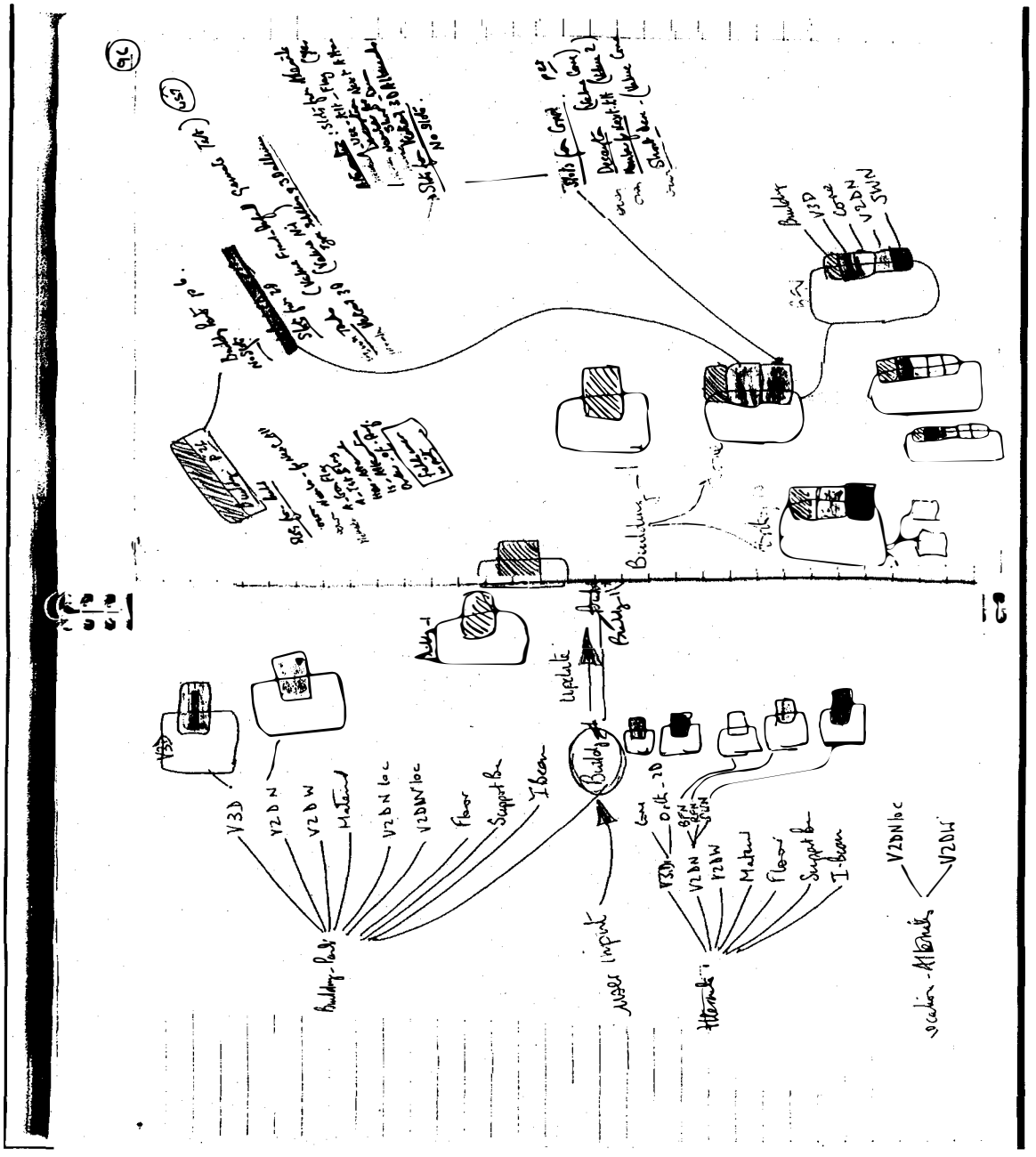


Figure B.4 **Workings made to establish the inheritance links between objects.**

APPENDIX C Detailed Requirements

Initial Sizing in Reinforced Concrete Buildings

The following details were sourced from Harty (1987) In order to make initial estimates of floor depth and beam and column sizes in reinforced concrete buildings the floor was assumed to be a reinforced concrete slab supported by beams on column lines. If the larger bay dimension was more than 1.5 times the smaller, then the slab was treated as spanning one-way, with a span equal to the smaller bay size. Otherwise it was treated as a two-way spanning slab. The depth of the slab was estimated as follows:

IF the slab is a flat slab and the imposed load is less than 0.0051
THEN the effective depth is found by the ratio Span /36.0.
IF the load is greater than or equal to 0.0051
THEN effective depth is Span /33.0.
IF slab is one-way and imposed load is less than 0.0051
THEN the effective depth is Span / 31.0.
IF the load is greater than or equal to 0.0051
THEN effective depth is Span / 28.0.
IF the slab is two-way
THEN a separate function is called to find the effective depth. This function uses interpolation to find the correct ratio.

Beams. The effective depth of a reinforced concrete beam was initially estimated using the ratio of span to effective depth as 15 to 1. The cover to main steel was then added to the effective depth and the sum was rounded up to the nearest 25 mm to give the beam depth and the overall floor depth.

The width of a reinforced concrete beam was estimated by limiting the shear stress in it to 2 N/mm^2, which gave a width of $(1000V/2d)$ mm where V was the maximum shear force in the beam in kN, and d was the effective depth in mm. This width was rounded up to the nearest 25 mm, and it or the minimum beam width required for the fire rating of the

building was used, whichever was the greater. If the calculated width was greater than the depth, then the depth was increased in increments of 50mm and the width was recalculated until it was less than the depth.

Columns. All reinforced concrete columns were of square cross-section *h x h*. The initial sizing of a column in a reinforced concrete rigid frame was estimated by assuming that the column was short and unbraced, which meant that the ratio *le/h* must be less than 10 to 1. The effective height *le* was calculated as 1.2 times the clear height. The value of *h* obtained from this, was rounded up to the nearest 25 mm and it or the minimum column dimension required for the fire rating was used, whichever was the greater. All other reinforced concrete columns were part of the horizontal structural subsystem and were designed as short and braced. The size was initially calculated by taking the slenderness ratio (ratio of effective length to width) as less than or equal to 15, where the effective height was taken as 0.85 times the clear height. The *h* value, thus calculated, was rounded up to the nearest 25 mm and it or the minimum column dimension of square reinforced concrete columns, as determined by the fire rating, whichever was the greater, was used.

Initial Sizing in Steel Buildings

The initial sizes for floors, beams and columns in steel buildings were estimated as follows. When the vertical subsystem was being designed initially, the horizontal structural subsystem was unknown and was assumed to be a steel deck topped with reinforced concrete, and supported by steel beams. A steel deck unit was selected, together with the spacing of intermediate beams. These were required if the decking could not span the shorter bay dimension. In this case, intermediate beams spanned the longer dimension onto main beams, which spanned the shorter bay dimension. Initial beam sections were then selected using the method for steel beams described below. In contrast, when the system designed the horizontal subsystem, the floor system had already been selected and the initial size of the slab was found using the methods described above for reinforced concrete buildings.

Beams. The initial beam depth was calculated by taking the span to depth ratio as 12. The beam section with the smallest depth greater than that assumed, was then selected from the steel section database.

Columns. All initial column sections were selected from values obtained from published tables of Universal Columns by limiting the slenderness to 180. Slenderness was computed by dividing the effective length by the smaller radius of gyration. The chosen section had the smallest radius of gyration greater than that required for a slenderness of 180.

The function *Find_Assumed_Steel_Sizes_Braced_Frame,* (the main function), performed the following tasks. It initially selected the beam section, first in the narrow perspective and then in the wide. It then called the function *Choose_Steel_Beam_Section_With_Depth* using the initial beam depth as the selection parameter. This function read through a list of beam section data, which was created from the class hierarchy of steel sections, and selected

the appropriate beam section. It then used the *Check_Design* function, with the parameter *Element,* to ensure that a section had been found. If a section had been found, then the function continued and calculated an assumed radius of gyration for the columns. It then used this value as a parameter to choose a column section and again used *Check_Design* to determine whether or not it had managed to select a section from the data base list. If an appropriate section had been found, the function then called the *Change_Steel_Beam* function to update the assumed floor depth. If the function was unable to select either beam or column sections, then the design was eliminated from further consideration and deleted from the search tree. These actions were then repeated for the wide perspective of the design.

The next section explains how the detailing of braced frame design options was performed. Similar functions were applied to rigid frame and shear wall partial designs; however, their descriptions have not been included in this report.

APPENDIX DNOVA Functions

The following tables list the KAL functions used in NOVA.

General Functions

```
Calc_Buckling_Resistance_Of_Steel_In_Col [Column_Section]
Calc_Compr_Str_Of_Steel_In_Col [Section]
Calc_Percent_Optim []
Calc_Total_Bldg_Cost [Test_List, Fn]
Calc_Total_Bldg_Cost [Test_List, Fn]
Calc_Design_Strength_Of_Steel, [Section Type]
Check_Design_Parameters []
Check_If_Ready []
Check_If_User_Loc [Locations_Class]
Choose_Steel_Beam_Section_With_Depth_Method [Depth]
Choose_Steel_Beam_Section_With_SXX_Method [S_XX]
Choose_Steel_Column_Section_With_Area_Method [Area]
Choose_Steel_Column_Section_With_Radius_Of_Gyration_Method []
Choose_Steel_Diagonal_Section_With_Area_Method,
Copy_To_Horiz [Unit, Slot]
Create_New_Units []
Current_Sys []
Detail_Bldgs []
Evaluate_Alternatives []
Evaluate_Bldg_Method [Bldg]
Evaluation_Display_Method []
Find_Best_Alternatives []
Find_Alt_Name [Part Bldg]
Find_Next_Loc_Alt [Loc_Alt]
Get_Steel_Section_Value [Type, Section, Slot]
Horiz_Sys_P []
Median_Calc_Costs [Test_List, Cost_Slot]
Median_Target_Costs [Test_List, Fn]
Multiple_P [X, Y]
Rc_Bldg_P [Bldg]
Rccoldesign [Fcu, Fy, Donh, Nonbh, Monbhh]
Review_Evaluation_Features_Method []
Round01 [Realnumber]
Round25 [Realnumber]
Round_2_Places [Realnumber]
Round5 [Realnumber]
Roundpoint25 [Realnumber]
Roundup [Realnumber]
Select_Reinf_Centres, [Required_Area Max_Acceptable_Spacing
Min_Acceptable_Spacing Max_Acceptable_Diameter Min_Acceptable_Diameter]
Select_Reinforcement_Bars, [Required_Area Max_Acceptable_Number_Of_Bars
Min_Acceptable_Number_Of_Bars Max_Acceptable_Diameter Min_Acceptable_Diamete
Selected_Area Selected_Diameter Selected_Number_Of_Bars]
Setup_Arrays [Type]
Set_Up_Log_File []

Sort_Steel_Children [Parent, Link_Type]
Steel_Bldg_P [Bldg]
Test_Alternatives []
Try_Next_On_List [Bldg]
Valid_2D_N_Alternative, [x y]
```

```
Valid_2D_N_Location_Alternative, [x y]
Valid_2D_W_Alternative, [x y]
Valid_2D_W_Location_Alternative, [x y]
Valid_Intermed_Beams_Alternative, [x y]
Valid_Material_Alternative, [x y]
Valid_Support_Beams_Alternative, [x y]
Vert_Sys_P []
```

Element functions

```
Acheckset [N_Or_W, Arraycols, I, J]
Approximate_Supports_Detail [Bldg]
Approx_Rc_Supports_Detail [Bldg]
Approx_Steel_Supports_Detail [Bldg]
Aset, [N_Or_W i j]
Calc_Axial_Load_In_Reinforced, Concrete_Column_Due_To_Cols_And_Beams []
Calc_Axial_Load_In_Steel_Column_Due_To_Cols_And_Beams [Bldg]
Calc_No_Of_Supp_Beams_Incl_Interm [Bldg]
Calc_Number_Of_Support_Columns [Bldg]
Calc_Approx_Interm_Bms [Bldg, Lx]
Calc_Load_On_Frame_Due_To_Beams [Bldg]
Calc_Load_On_Walls_Due_Beams, [Bldg]
Calculate_Numbers_Of_Approximate_Support_Elements []
Calculate_Precast_Panels_Beam_Depth_Under_Floor []
Calculate_Precast_Panels_Supp_Beam_Depth [Bldg]
Calculate_Precast_Panels_Weight [Bldg]
Calculate_Reinforced_Concrete_Slab_Beam_Depth_Under_Floor []
Calculate_Reinforced_Concrete_Slab_Supp_Beam_Depth [Bldg]
Calculate_Reinforced_Concrete_Slab_Weight [Bldg]
Calculate_Ribbed_Slab_Beam_Depth_Under_Floor []
Calculate_Ribbed_Slab_Supp_Beam_Depth [Bldg]
Calculate_Ribbed_Slab_Weight [Bldg]
Calculate_Steel_Deck_Beam_Depth_Under_Floor [Bldg, Beam_Depth]
Calculate_Steel_Deck_Weight [Bldg]
Calculate_Volume_Of_Concrete_On_Steel_Deck [Slab_Depth, Deck_Unit]
Calculate_Waffle_Slab_Weight [Bldg]
Change_Rc_Beam_Size [Bldg, Depth_Slot]
Change_Steel_Beam [Bldg, Beam_Section]
Check_Design [Bldg, Type]
Column_Check [Bldg, Beam_Span]
Decrease_Rc_Beam_Size [Bldg, Decr_Depth, Depth_Slot, Decr_Width]
Decrease_Rf_Rc_Beam_Size [Bldg]
Delete_Unit_Because_Looped_Too_Many_Times [Bldg]
Find_Req_Area_Of_Reinf_In_Rc_Beam [Max_Sagging_Moment_In_Beam]
Find_Req_Area_Of_Reinf_In_Rc_Beam_Incl_Compression_Steel []
Find_Approx_Rc_Supports_Sizes [Bldg]
Find_Approx_Steel_Supports_Sizes [Bldg]
Find_Approx_Support_Sizes [Bldg]
Find_Default_Width_Of_Support_Beam [[Bldg Beam_Depth Beam_Depth_Under_Slab
 Effective_Depth Beam_Span Slab_Span]]
Find_Depth_Of_Beam_Under_Floor [Bldg, Beam_Depth]
Find_Initial_Rc_Support_Beam_Depth [Bldg]
Find_Initial_Steel_Support_Sizes [Bldg]
Find_Max_Rc_Beam_Depth [Bldg]
Find_Max_Steel_Beam_Depth [Bldg]
Find_Slab_Type_For_Approximate_Supports [Bldg]
Find_Weight_Of_Floor_Per_Mm_Squared [Bldg]
Get_Assumed_Sizes []
```

```
Increase_Rc_Beam_Size [Bldg, Incr_Depth, Depth_Slot, Incr_Width]
Increase_Rf_Rc_Beam_Size [Bldg]
Main_Support_Steel_Beam_Detail [Bldg, Section_Slot, Moment_Slot]
Rc_Beam_Check [Bldg, Beam_Span, Slab_Span, Beam_Depth]
Rc_Beam_Detail [Bldg, Max_Moment_In_Beam, Type]
Rc_Beam_Detail_And_Check [Bldg, Moment_Slot, Type]
Rc_Column_Detail [Bldg, Max_Axial_Load]
Rc_Main_Beam_Check [Bldg, Beam_Span, Intermediate_Beam_Span]
Rc_Support_Beam_Detail [Bldg, Beam_Depth_Slot, Beam_Width_Slot]
Rc_Supports_Detail [Bldg]
Set_Initial_Sizes [Bldg]
Steel_Beam_Check [Bldg, Beam_Span, Slab_Span, Beam_Section]
Steel_Beam_Section_Select, [BldgMax_Moment_In_Beam Beam_Sectionin]
Steel_Column_Detail [Bldg, Max_Axial_Load]
Steel_Main_Beam_Check [Bldg, Beam_Span, Slab_Span]
Steel_Supports_Detail [Bldg]
Supports_Detail [Bldg]
Support_Steel_Beam_Detail []
```

Braced Frame

```
Calculate_Dead_Load_Estimate_For_Braced_Frame []
Detail_Braced_Frame [Bldg]
Detail_Braced_Frame_Beam [Bldg]
Detail_Braced_Frame_Column [Bldg]
Detail_Braced_Frame_Diagonal [Bldg]
Find_Assumed_Steel_Sizes_In_Braced_Frame,
Increase_BF_Sizes [Bldg]
```

Rigid Frame

```
Calculate_Dead_Load_Estimate_For_Rigid_Frame []
Calculate_Forces_In_Rigid_Frame [Bldg]
Detail_Rc_Beam_In_Rigid_Frame [Bldg]
Detail_Rc_Beam_In_Rigid_Frame_For_Shear [Bldg]
Detail_Rc_Column_In_Rigid_Frame [Bldg]
Detail_Rigid_Frame [Bldg]
Detail_Steel_Beam_In_Rigid_Frame [Bldg]
Detail_Steel_Column_In_Rigid_Frame [Bldg]
Find_Assumed_Sizes_In_Rigid_Frame [Bldg, Type]
```

Shear Wall

```
Calc_Load_On_Wall_Due_To_Beams [Bldg]
Calculate_Compressive_Stress_In_Wall [Bldg]
Design_Shear_Wall [Bldg]
Detail_Shear_Wall [Bldg]
Detail_Steel_In_Wall [Bldg]
Find_Initial_Shear_Wall_Thickness [Bldg, Type]
Increase_Shear_Wall_Thickness [Bldg]
```

Floors

```
Calc_Shear_In_Flat_Slab [Bldg Ult_Load_On_Slab_Kn_Per_M_Sq]
Calculate_Ultimate_Load_On_Ribbed_Slab [Bldg]
Calculate_Ultimate_Load_On_Slab [Bldg Depth]
Calculate_Ultimate_Load_On_Waffle_Slab [Bldg]
Detail1_Rc_Slab [Bldg]
Detail1_Ribbed_Slab [Bldg]
Detail1_Waffle_Slab [Bldg]
Detail2_Ribbed_Slab [Bldg, Effective_Depth]
Detail2_Waffle_Slab [Bldg, Effective_Depth]
Detail_Precast_Panels [Bldg]
Detail_Rc_Slab [Bldg]
Detail_Ribbed_Slab [Bldg]
Detail_Steel_Deck [Bldg]
Detail_Waffle_Slab [Bldg]
Display_Waffle_Moulds []
Find_Effective_Depth_Of_Rc_Floor []
Find_Effective_Depth_Of_Two_Way_Spanning_Rc_Slab []
Find_Grid_Size_For_Rib [Bldg]
Find_Grid_Size_For_Waffle [Bldg,
Find_Intermediate_Beam_Spacing [Max_Spacing, Lx]
Find_List_Of_Steel_Deck_Units [Load, Insulation_Thickness, Lx]
Find_Max_Span [Unit, Load, Depth, Lx]
Find_Max_Span_Of_Precast_Panels []
Find_Required_Thickness_Of_Slab_On_Steel_Deck [Fire_Rating]
Find_Slab_Type [Bldg]
Identify_Rib_Mould_Class [Grid_Size]
Identify_Waffle_Mould_Class []
Increase_Flat_Slab_And_Column_Size [Bldg]
Increase_Flat_Slab_Size [Bldg]
Select_Precast_Floor_Unit [Span, Load]
Select_Rib_Mould [Depth, Grid_Size]
Select_Steel_Deck_Unit, [LoadiInsulation_ThicknessLX]
Select_Waffle_Mould [Depth, Grid_Size]
```

Cost Functions

```
Calc_Approximate_Supports_Cost [Bldg]
Calc_Cost_Of_External_Walls []
Calc_Cost_Of_Stair_Core [Bldg, Height]
Calc_Cost_Of_Stairs [Bldg]
Calc_Superstructure_Floor_Area [Bldg]
Calc_Supports_Cost [Bldg]
Calculate_Approx_Rc_Floor_Cost [Bldg]
Calculate_Approx_Steel_Floor_Cost []
Calculate_Bldg_Floor_Cost [Bldg]
Calculate_Braced_Frame_Narrow_Cost [Bldg]
Calculate_Braced_Frame_Wide_Cost [Bldg]
Calculate_Cost [Bldg]
Calculate_Cost_Of_Finishes [Bldg]
Calculate_Precast_Panels_Cost [Bldg]
Calculate_Rc_Rigid_Frame_Narrow_Cost [Bldg]
```

```
Calculate_Rc_Rigid_Frame_Wide_Cost [Bldg]
Calculate_Rc_Vertical_System_Cost [Bldg]
Calculate_Reinforced_Concrete_Slab_Cost [Bldg]
Calculate_Ribbed_Slab_Cost [Bldg]
Calculate_Roof_Cost [Bldg]
Calculate_Shear_Wall_Narrow_Cost [Bldg]
Calculate_Shear_Wall_Wide_Cost [Bldg]
Calculate_Steel_Rigid_Frame_Narrow_Cost [Bldg]
Calculate_Steel_Rigid_Frame_Wide_Cost [Bldg]
Calculate_Steel_Vertical_System_Cost [Bldg]
Calculate_Steel_Deck_Cost [Bldg]
Calculate_Waffle_Slab_Cost [Bldg]
Cost_Rc_Beam [Depth, Width, Length, Steel_Mass_Per_Metre]
Cost_Rc_Column [Depth, Width, Length, Steel_Mass_Per_Metre]
Cost_Steel_Section [Section, Length, Type]
Log_Cost [Bldg, Amount, Descr]
Log_Cost_Beams [Bldg, Amount]
Log_Cost_Columns [Bldg, Amount]
Log_Cost_Diags [Bldg, Amount]
Log_Cost_Elem [Bldg, Amount, Descr]
Log_Item_Cost [Bldg, Item_Cost_List]
Log_Tot_Elemental_Cost [Bldg]
Write0_Cost_Log [W, Bldg, Slot]
Write1_Cost_Log [W, Cost_List]
Write_Cost_Log [W, System]
Write_Elemental_Costs [W, Bldg]
```

Evaluation Functions

```
Calculate_Buildability [Bldg]
Calculate_Clear_Space [Bldg]
Calculate_Column [Bldg]
Calculate_Flexibility [Bldg]
Calculate_Height [Bldg]
Calculate_Maintenance [Bldg]
Calculate_Prefab [Bldg]
Calculate_Sourcing [Bldg]
Calculate_Sway [Bldg]
Calculate_Time [Bldg]
Estimate_Column_Max [Bldg]
Estimate_Column_Min [Bldg]
Estimate_Height [Bldg]
Estimate_Maintenance_Max [Bldg]
Estimate_Maintenance_Min [Bldg]
Calculate_Percent_Optim, [Bldg Feat Res]
Weighting, [Feat Factor]
Review_Horiz_Evaluation_Features, []
Review_Vertical_Evaluation_Features, []
```

Design Process Functions

```
Check_Floor, [x]
Check_Intermed_Beams, [x]
```

```
Check_Material, [x]
Check_Support_Beams, [x]
Check_Vert_2D_N, [x]
Check_Vert_2D_N_Loc, [x]
Check_Vert_2D_W, [x]
Check_Vert_2D_W_Loc, [x]
Check_Vert_3D, [x]
Clean_Up, []
Cleanup_Of_Vertical_Subsystem, []
ClearHierarchy, []
Count []
Current_Sys, []
Design_First_Level_Down, [First_Level]
Design_Floor_Level, [Floor_Level]
Design_Horizontal_Subsystem, []
Design_Intermed_Beams_Level, [Intermed_Beams_Level]
Design_Material_Level, [Material_Level]
Design_Shear_Wall, [Bldg]
Design_Support_Beams_Level, [Support_Beams_Level]
Design_Vertical_2D_N_Level, [Vertical_2D_N_Level]
Design_Vertical_2D_N_Location_Level, [Vertical_2D_N_Location_Levelin]
Design_Vertical_2D_W_Level, [Vertical_2D_W_Level]
Design_Vertical_2D_W_Location_Level, [Vertical_2D_W_Location_Levelin]
Design_Vertical_Subsystem, []
Detail, [Bldg]
Detail_Floor_Level, []
Detail_V2Dloc_Level, []
Detail_Vertical_Subsystem, []
Generate_New_Units, []
Make_Beam_Steel_Section_List, []
Make_Column_Steel_Section_List, []
Make_Diag_Steel_Section_List, []
Make_List_Of_Floors_To_Detail, []
Make_List_Of_Steel_Deck_Units, []
Make_List_Of_V2Dlocs_To_Detail, []
MakeBeamLists, [BeamList]
MakeBeamLists2, [BeamList]
MakeBFVars, []
MakeELVars, []
MakeEVVars, []
MakeFLVars, []
MakeFNVars, []
MakeRFVars, []
MakeSDVars, []
MakeSWVars, []
```

Utility functions

```
Comparefn_Depth_Topping, [arg1 arg2]
Comparefn_Mould_Depth, [arg1 arg2]
Continue_Initial_Sizing, []
Drop_IdNum, [Alt]
Even, [x]
Fix, [number]
Get_Load_From_Span_Load_Table, [Alt]
Get_Span_From_Span_Load_Table, [Alt]
Input_User_Requirements, [Building_1]
LoadExcelProgram, []
```

```
Mod, [numDen]
Multiple_P, [numDen]
Percentage_Optimization, [Altblg_Value]
Quit, []
ReWrite_Short_Descr, [Bldg]
Set_Defaults, []
Set_N_Or_W, [Building]
Slot_Copy_Alts, [x]
Slot_Copy_Levels, [Alts_At_This_LevelLevel]
Update_Last_Level, []
```

Reporting Functions

```
Write_Excel_Design_Report, []
Write_Report_1, []
Write_Temp_Excel_Report1, []
Write_Temp_Excel_Report2, []
Write_Vert_Eval_Report, []
```

APPENDIX E NOVA Rules

This table lists the rules used in NOVA.

Rules

```
Detail_Rule, []
Evaluate_Rule, []
Find_Defs_1_Rule, []
Find_Defs_2_Rule, []
Generate_Rule, []
Rule_About_1_Hour_Fire_Rating, []
Rule_About_2_Hour_Fire_Rating, []
Rule_About_4_Hour_Fire_Rating, []
Rule_About_Floors_For_Rc_Bldgs, []
Rule_About_Floors_For_Steel_Bldgs, []
Rule_About_Precast_Panel_Select, []
Rule_About_Precast_Panel_Span, []
Rule_About_Rib_Grid_Size, []
Rule_About_Rib_Mould_Select, [Altbldg|Test_Class]
Rule_About_Steel_Deck_Select, []
Rule_About_Waffle_Grid_Size, []
Rule_About_Waffle_Mould_Select, []
Rule_About_Waffle_Slabs, []
Rule_Abt_Max_Span_Precast_Panels, []
Rule_For_Aparts, []
Rule_For_Buildability, []
Rule_For_Car_Park, []
Rule_For_ClrSpce_In_Car_Parks, []
Rule_For_ClrSpce_In_Hotels_Aparts, []
Rule_For_ClrSpce_In_Offices, []
Rule_For_ClrSpce_In_Other_Bldgs, []
Rule_For_Column, []
Rule_For_Flexibility_Not_Req, []
Rule_For_Flexibility_Req, []
Rule_For_Height_In_Urban_Areas, []
Rule_For_Height_Outof_Urban_Areas, []
Rule_For_Hotel, []
Rule_For_Large_Job, []
Rule_For_Maintenance, []
Rule_For_Non_Prestg_Buildings_Cost, []
Rule_For_Offices, []
Rule_For_Prefab_Important, []
Rule_For_Prefab_Not_Important, []
Rule_For_Prestg_Buildings_Cost, []
Rule_For_Reinf_Concrete_Bldg, []
Rule_For_Small_Job, []
Rule_For_Sourcing_Important, []
Rule_For_Sourcing_Not_Important, []
Rule_For_Steel_Building, []
Rule_For_Sway, []
Rule_For_Time_Important, []
Rule_For_Time_Not_So_Important, []
Rule1_About_1_Way_Spanning_Slabs, []
Rule1_About_Car_Parks, []
Rule1_About_Combination_Systems, []
Rule1_About_Concrete_Strength, []
```

```
Rule1_About_Core, []
Rule1_About_Cover_To_Reinfmnt, []
Rule1_About_Den_Reinfmnt_In_Slabs, []
Rule1_About_Flat_Slabs, []
Rule1_About_I_Way_Slabs, []
Rule1_About_Offices, []
Rule1_About_Orthogonal_2D_Sys, []
Rule1_About_Parts_Finishes_Weight, []
Rule1_About_Rc_Slab_Span, []
Rule1_About_Reinf_In_Shear_Wall, []
Rule1_About_Ribbed_Slab_Span, []
Rule1_About_Rigid_Frame_N, []
Rule1_About_Rigid_Frame_Wide, []
Rule1_About_Shear_Wall_N, []
Rule1_About_Shear_Wall_Parms, []
Rule1_About_Shear_Wall_Thick, []
Rule1_About_Shear_Wall_Wide, []
Rule1_About_Steel_Deck_Span, []
Rule1_About_Steel_Material, []
Rule1_About_Steel_Sections, [Altbldg|Test_Class]
Rule1_About_Strength_Shear_Steel, []
Rule1_Abt_Grade_Structural_Steel, []
Rule1_Abt_Reinf_Concrete_Material, []
Rule1_For_Braced_Frame, [Altbldg|Test_Class]
Rule1_For_Reinf_In_Rc_RFB, []
Rule1_For_Shear_In_Flat_Slabs, [Altbldg|Test_Class]
Rule1_For_Shear_Stress_In_Rc_RFB, []
Rule1_For_Steel_Yield_Stress, []
Rule1A_For_Braced_Frame, [Altbldg|Test_Class]
Rule1A_For_Shear
Rule2_About_1_Way_Spanning_Slabs, []
Rule2_About_Car_Parks, []
Rule2_About_Combination_Systems, []
Rule2_About_Core, []
Rule2_About_Flat_Slabs, []
Rule2_About_Offices, []
Rule2_About_Parts_Finishes_Weight, []
Rule2_About_Rc_Slab_Span, []
Rule2_About_Reinf_In_Shear_Wall, []
Rule2_About_Ribbed_Slab_Span, []
Rule2_About_Shear_Wall_Thick, []
Rule2_About_Steel_Deck_Span, []
Rule2_Abt_Reinf_Concrete_Material, []
Rule2_For_Braced_Frame, [Altbldg|Test_Class]
Rule2_For_Reinf_In_Rc_RFB, []
Rule2_For_Shear_Stress_In_Rc_RFB, []
Rule2_For_Steel_Yield_Stress, []
Rule2A_For_Braced_Frame, [Altbldg|Test_Class]
Rule3_About_Rc_Slab_Span, []
Rule3_About_Steel_Deck_Span, []
Rule3_Abt_Reinf_Concrete_Material, []
Rule4_About_Rc_Slab_Span, []
Rule4_About_Steel_Deck_Span, []
Ruleank_R, []
Test_Rule, []
```

Rulesets

Global:Rs_For_Chk_Det_RF_Alts
    Rule1_For_Shear_Stress_In_Rc_RFB
    Rule1_For_Too_Little_Reinf_In_RcRFB
    Rule1_For_Too_Much_Reinf_In_RcRFB
    Rule2_For_Shear_Stress_In_Rc_RFB
    Rule2_For_Too_Little_Reinf_In_RcRFB
    Rule1_For_Reinf_In_Rc_RFB
    Rule2_For_Reinf_In_Rc_RFB

Global:Rs_For_Chk_Det_BF_Alts
    Rule1_For_Braced_Frame
    Rule2_For_Braced_Frame
    Rule1A_For_Braced_Frame
    Rule2A_For_Braced_Frame

Global:Rs_For_Chk_Det_Pre_Panels_Alts
    Rule_About_Precast_Panel_Select;

Global:Rs_For_Chk_Det_Rc_Slab_Alts
    Rule1_For_Shear_In_Flat_Slabs
    Rule1A_For_Shear_In_Flat_Slabs

Global:Rs_For_Chk_Det_Ribbed_Slab_Alts
    Rule_About_Rib_Mould_Select;

Global:Rs_For_Chk_Det_Steel_Deck_Alts
    Rule_About_Steel_Deck_Select;

Global:Rs_For_Chk_Det_Waffle_Slab_Alts
    Rule_About_Waffle_Mould_Select
    Rule_About_Waffle_Grid_Size
    Rule_About_Rib_Grid_Size

Global:Rs_For_Chk_Det_Elements_Alts
    Rule1_About_Steel_Sections;

Global:Rs_For_Chk_Det_SW_Alts
    Rule1_About_Shear_Wall_Thick
    Rule2_About_Shear_Wall_Thick
    Rule1_About_Reinf_In_Shear_Wall
    Rule2_About_Reinf_In_Shear_Wall

APPENDIX F Class Diagrams

Building System Object Classes

During development of the NOVA design tool an object model was created, by abstraction from the requirement specification. The work started with the identification of the structural design objects and the subsequent grouping of those objects, where possible, into a hierarchy of classes. Objects placed in the same hierarchy are those which exhibited similar behaviour. The following diagrams show key groups of building system object identified.

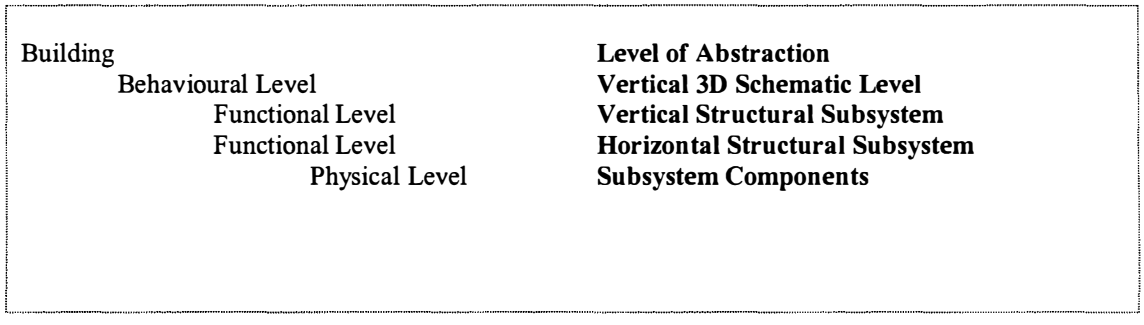


Figure F.1 Object classes, in the building hierarchy (the Product Model).

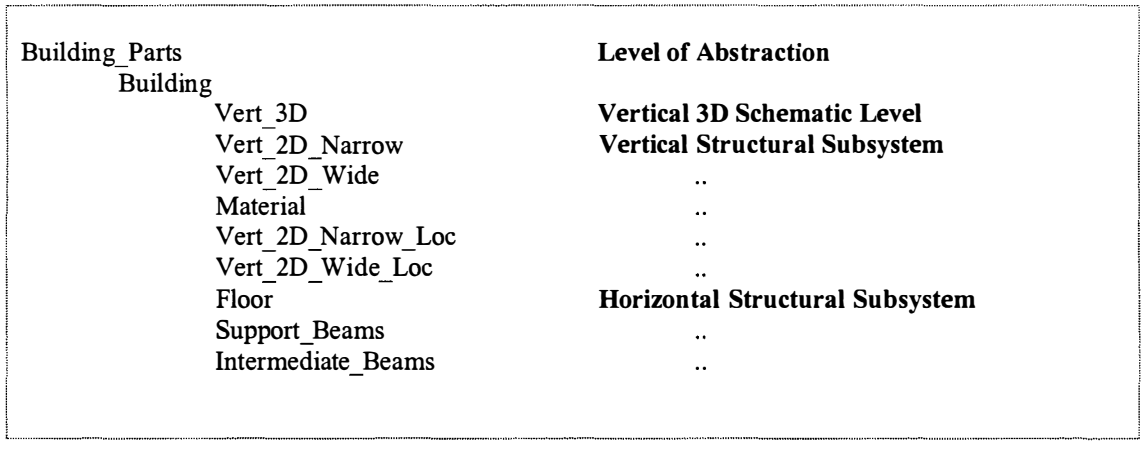


Figure F.2 Object classes, which constitute the levels in the building hierarchy.

The design object classes, which make up the building hierarchy, are shown below in Figure F.3, in the form of one completed design with appropriate alternatives attached at each level.

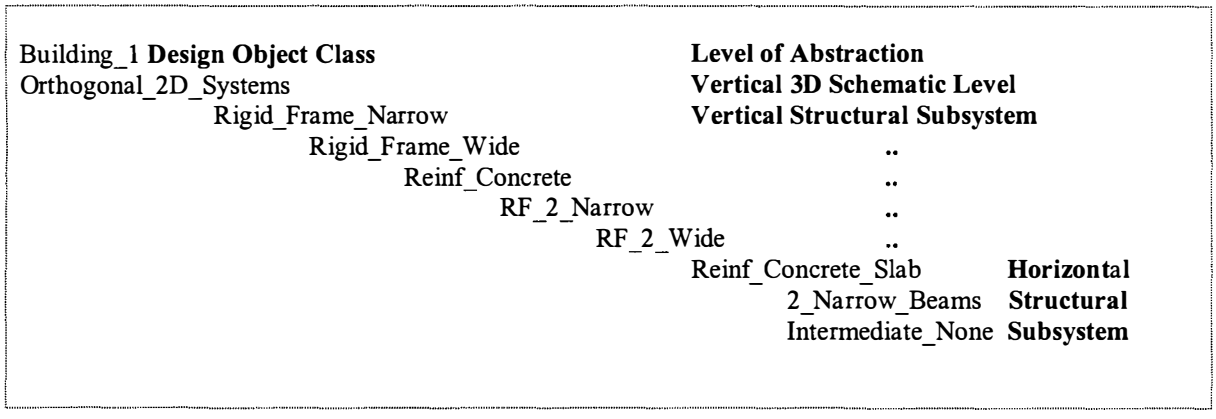


Figure F.3 Object classes in a completed design, which is displayed hierarchically.

At each level in the hierarchy, the system designers were required to provide an appropriate set of design options, which made up the alternatives designs, which the system could generate for that level. These design options were represented by the *Alternatives* Classes. These classes were organised in an object hierarchy, which is shown in Figure F.4.

Alternatives

Vert_3D_Alternatives	**Level in the building hierarchy**
Core	**Available Options**
Orthogonal_2D_Systems	..
Vert_2D_Narrow_Alternatives	**Level in the building hierarchy**
Braced_Frame_Narrow	**Available Options**
Rigid_Frame_Narrow	..
Shear_Wall_Narrow	..
Vert_2D_Wide_Alternatives	**Level in the building hierarchy**
Rigid_Frame_Wide	**Available Options**
Braced_Frame_Wide	..
Shear_Wall_Wide	..
Material_Alternatives	**Level in the building hierarchy**
Reinf_Concrete	**Available Options**
Steel	..
Floor_Alternatives	**Level in the building hierarchy**
Reinf_Concrete_Slab	**Available Options**
Ribbed_Slab	..
Steel_Deck	..
Waffle_Slab	..
Precast_Panels	..
Support_Beams_Alternatives	**Level in the building hierarchy**
a0_Beams	**Available Options**
a2_Narrow_Beams	..
a2_Wide_Beams	..
a4_Beams	..
Intermed_Beams_Alternatives	**Level in the building hierarchy**
Intermed_Narrow	**Available Options**
Intermed_None	..
Intermed_Wide	..

Figure F.4 **Object classes, which make up the building design alternatives.**

A separate object hierarchy; the *Location Alternatives,* was used to represent the various layout plans, which were available for use with the structural systems, both in the narrow and wide perspectives, at the *Vertical 2D* level. These classes could have been included in the *Alternatives* hierarchy, however they were split out into a separate class to simplify identification and reduce complication. These location alternatives are shown in Figure F.5.

Location _Alternatives

- Vert_2D_Narrow_Loc_Alternatives
  - Rigid_Frame_Narrow_Loc_Alternatives
    - RF_2_Narrow
    - RF_3_Narrow
    - RF_4_Narrow
    - RF_All_Narrow
    - RF_User_Loc_Narrow
  - Braced_Frame_Narrow_Loc_Alternatives
    - BF_2_Narrow
    - BF_3Plus_Narrow
    - BF_Corners_Narrow
    - BF_Inside_Narrow
    - BF_User_Loc_Narrow
  - Shear_Wall_Narrow_Loc_Alternatives
    - SW_2_Narrow
    - SW_3_Narrow
    - SW_4_Narrow
    - SW_All_Narrow
    - SW_Shaft_Narrow
    - SW_User_Loc_Narrow
- Vert_2D_Wide_Loc_Alternatives
  - Braced_Frame_Wide_Loc_Alternatives
    - BF_2_Wide
    - BF_3Plus_Wide
    - BF_Corners_Wide
    - BF_Inside_Wide
    - BF_User_Loc_Wide
  - Rigid_Frame_Wide_Loc_Alternatives
    - RF_All_Wide
    - RF_3_Wide
    - RF_2_Wide
    - RF_4_Wide
    - RF_User_Loc_Wide
  - Shear_Wall_Wide_Loc_Alternatives
    - SW_2_Wide
    - SW_All_Wide
    - SW_Shaft_Wide
    - SW_3_Wide
    - SW_4_Wide
    - SW_User_Loc_Wide

Each item at this level represents a different configuration option.

Figure F.5 Object classes, which make up the location alternatives hierarchy.

Object classes were also required to model the different types of composite physical units. The precast concrete units group is shown below in Figure F.7. Each lower level class represents a separate B11, precast unit.

Precast_Units	**Physical components.**
B11	
B11_46	..
B11_56	..
B11_66	..
B11_76	..
B11_86	..
B15	
B15_56	..
B15_66	..
B15_76	..
B15_86	..
B15_96	..
.....	

Figure F.6 Object classes, which represent precast concrete units.

Other object classes were required to represent the non-physical components of the structural design domain. These less visible system entities included the elements of the building plan, which was used to guide the building design process, and the default design parameters. The building plan was represented by the attributes of the *Schedule* unit.

System_Schedules	Building Plan, which includes the sequence of design activities.
Schedule	

Figure F.7 The Schedule object class, contains the plan for the design process.

Defaults	Design Parameters

Figure F.8 The Default Design Parameters object class.

Other non-physical entities such as the evaluation features, were also represented by object classes, these are shown in Figure F.9. A separate class was created for each evaluation feature.

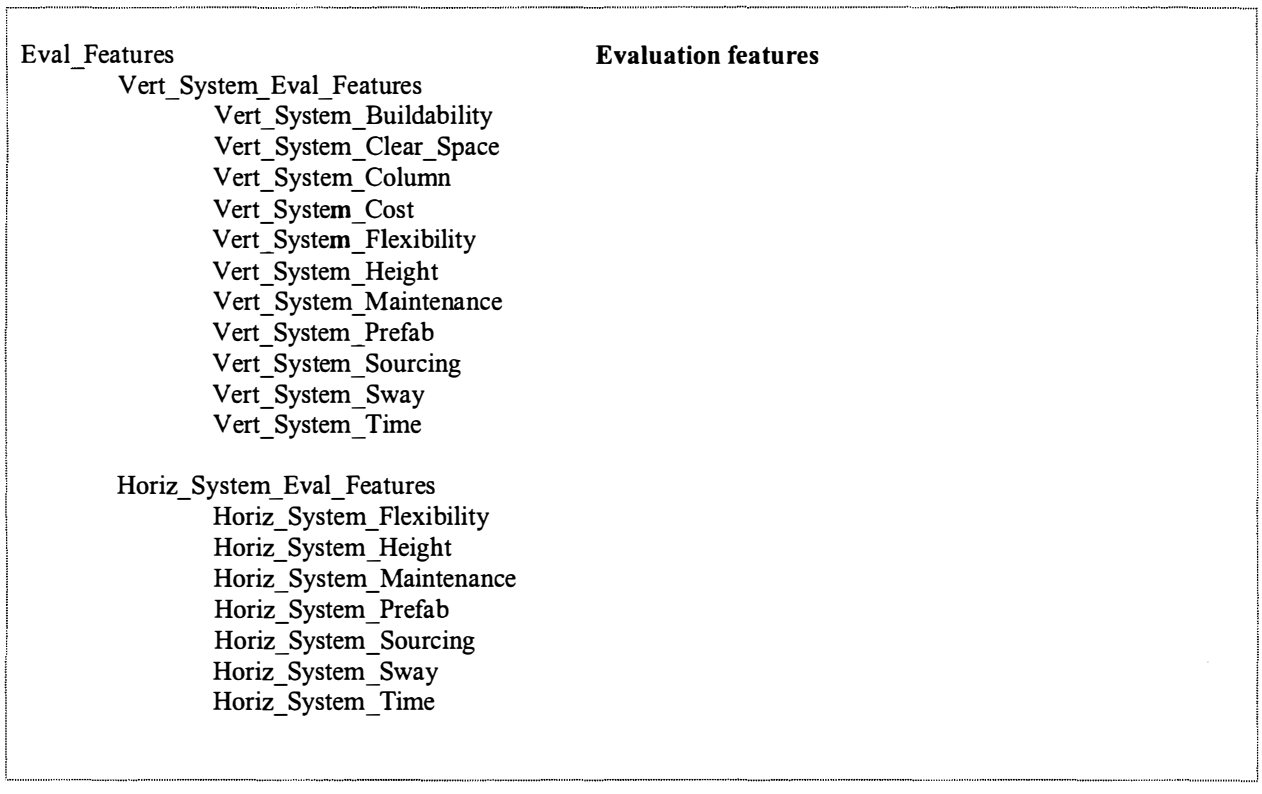


Figure F.9 The Evaluation Features object class.

A series of objects were also required to model the user interface specified in the requirements. The objects in this part of the model included the session windows, input buttons and output displays required to facilitate interaction with the system user.

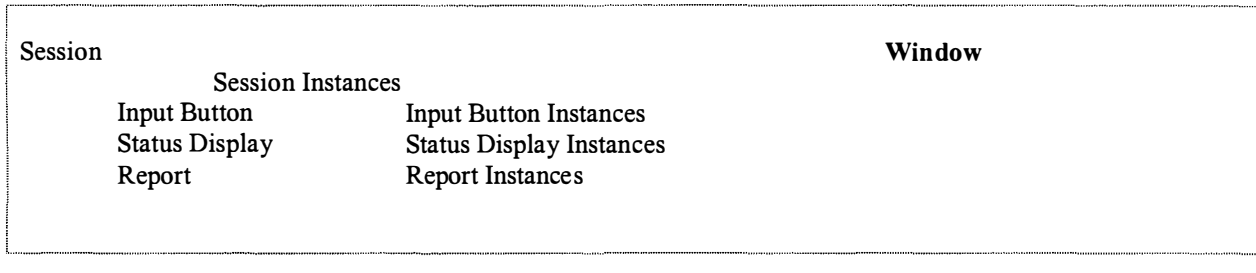


Figure F.10 The Session object class and some of its associate classes.

APPENDIX G Class Attributes

CLASS: Defaults	Superclass: Root
Attributes:	Characteristics:
Ass_Cover_To_Bottom_Steel	VALUE_TYPE, NUMBER Ass_Cover_To_Bottom_Steel = 50.0
Ass_Cover_To_Steel_In_Slabs	VALUE_TYPE, NUMBER Ass_Cover_To_Steel_In_Slabs = 35.0
Ass_Cover_To_Top_Steel	VALUE_TYPE, NUMBER Ass_Cover_To_Top_Steel = 50.0
Ass_Steel_Density_In_Slabs	"Density in Tonne-Per-Cu-M. Used when approximating the cost of the slabs at the vertical system stage." VALUE_TYPE, NUMBER Ass_Steel_Density_In_Slabs = 0.0785
Assd_Stl_Den_In_Slabs_On_Stl_Dk	Assd_Stl_Den_In_Slabs_On_Stl_Dk, "Density inTonne-Per-Cu.M. used when calculating the cost of steel decks" VALUE_TYPE, NUMBER Assd_Stl_Den_In_Slabs_On_Stl_Dk = 0.03925
Concrete_Design_Strength	VALUE_TYPE, NUMBER Concrete_Design_Strength = 35.0
Cover_To_Main_Steel_In_Columns	VALUE_TYPE, NUMBER Cover_To_Main_Steel_In_Columns = 40.0
Estimated_Costs_Except_Superstr	VALUE_TYPE, NUMBER Estimated_Costs_Except_Superstr = 2036350.605
Grade_Of_Structural_Steel	VALUE_TYPE, NUMBER MINIMUM_VALUE, 43 MAXIMUM_VALUE, 55 Grade_Of_Structural_Steel = 50.0
List_Of_Bar_Diameters	List_Of_Bar_Diameters, "Diameters of Reinforcing Bars to be considered when Detailing." MULTIPLE VALUE_TYPE, NUMBER List_Of_Bar_Diameters, 6, 8, 10, 12, 16, 20, 25, 32, 40, 50
Max_Diam_Of_Bars_In_Beams	Max_Diam_Of_Bars_In_Beams, "Max Diameter of reinforcing bars in Rc Beams." VALUE_TYPE, NUMBER Max_Diam_Of_Bars_In_Beams = 32
Max_Diam_Of_Bars_In_Ribs	Max_Diam_Of_Bars_In_Ribs, "Max Diameter of reinforcing bars in Rc Beams." VALUE_TYPE, NUMBER Max_Diam_Of_Bars_In_Ribs = 25
Ass_Spec_Weight_Of_Pre_Pnls	VALUE_TYPE, NUMBER Ass_Spec_Weight_Of_Pre_Pnls = 0.00001
Ass_Spec_Weight_Of_Wfle_Slbs	VALUE_TYPE, NUMBER Ass_Spec_Weight_Of_Wfle_Slbs = 0.00001
Max_Number_Of_Reinforcing_Bars	Max_Number_Of_Reinforcing_Bars, "the acceptable maximum number of reinforcing bars to be considered as acting together to give a total effective area." VALUE_TYPE, NUMBER Max_Number_Of_Reinforcing_Bars = 24
Max_Shear_Wall_Thickness	VALUE_TYPE, NUMBER Max_Shear_Wall_Thickness = 400.0
Max_Spacing_Of_Reinforcing_Bars	VALUE_TYPE, NUMBER Max_Spacing_Of_Reinforcing_Bars = 1000.0
Maximum_Section_Depth	Maximum_Section_Depth, "Used when detailing to find minimum depth section possible" VALUE_TYPE, NUMBER Maximum_Section_Depth = 20000
Min_Dimen_Of_Square_Rc_Cols	Min_Dimen_Of_Square_Rc_Cols, "based on min 300X300 in Rc

	Design Manual and the fire rating" VALUE_TYPE, NUMBER Min_Dimen_Of_Square_Rc_Cols = 300.0
Min_Number_Reinforcing_Bars	VALUE_TYPE, NUMBER Min_Number_Reinforcing_Bars = 1
Min_Rc_Beam_Width	Min_Rc_Beam_Width, "based on RC Design Manual and fire rating" VALUE_TYPE, NUMBER Min_Rc_Beam_Width = 200
Min_Shear_Wall_Thickness	VALUE_TYPE, NUMBER Min_Shear_Wall_Thickness = 180
Min_Spacing_Of_Reinforcing_Bars	VALUE_TYPE, NUMBER Min_Spacing_Of_Reinforcing_Bars = 65.0
Min_Topping_For_Ribs	VALUE_TYPE, NUMBER Min_Topping_For_Ribs = 75.0
Min_Topping_For_Waffles	VALUE_TYPE, NUMBER Min_Topping_For_Waffles = 75
Partitions_Finishes_Est	Partitions_Finishes_Est, "estimate of weight of partitions and finishes N per mm*2" VALUE_TYPE, NUMBER Partitions_Finishes_Est = 0.0028
Percent_Of_Ext_Surface_In_Wndws	VALUE_TYPE, NUMBER Percent_Of_Ext_Surface_In_Wndws = 5
Spn_Eff_Dpth_Rtio_Fr_Ribbd_Slbs	Spn_Eff_Dpth_Rtio_Fr_Ribbd_Slbs, "from IStructE Manual Section 4.2.6.2" VALUE_TYPE, NUMBER Spn_Eff_Dpth_Rtio_Fr_Ribbd_Slbs = 20.8
Spn_Eff_Dpth_Rtio_Fr_Wafle_Slbs	Spn_Eff_Dpth_Rtio_Fr_Wafle_Slbs, "from IStructE Manual Section 4.2.6.2" VALUE_TYPE, NUMBER Spn_Eff_Dpth_Rtio_Fr_Wafle_Slbs = 18.72
Steel_Yield_Stress	Steel_Yield_Stress, "yield stress of steel in N per mm*2" VALUE_TYPE, NUMBER Steel_Yield_Stress = 460.0
Spec_Wght_Of_Reinf_Concrete	VALUE_TYPE, NUMBER Spec_Wght_Of_Reinf_Concrete = 0.0000236
Steel_Yield_Stress_For_Columns	VALUE_TYPE, NUMBER Steel_Yield_Stress_For_Columns = 460.0
Table_For_Insulation_Thickness	Table_For_Insulation_Thickness, "table used to find required slab thickness for composite steel deck floors - Table 4.3 from Newman (1983)" MULTIPLE Table_For_Insulation_Thickness, 23.60_0.5_065.0, 23.60_1.0_090.0, 23.60_1.5_105.0, 23.60_2.0_115.0, 23.60_3.0_135.0, 23.60_4.0_150.0, 18.64_0.5_055.0, 18.64_1.0_065.0, 18.64_1.5_075.0, 18.64_2.0_085.0, 18.64_3.0_115.0, 18.64_4.0_130.0
Wt_Of_Concr_For_Steel_Deck	Wt_Of_Concr_For_Steel_Deck, "in kN per cubic m VALUE_TYPE, NUMBER MINIMUM_VALUE, 18.64 MAXIMUM_VALUE, 23.6 Wt_Of_Concr_For_Steel_Deck = 23.6
Yield_Strength_Of_Shear_Steel	VALUE_TYPE, NUMBER Yield_Strength_Of_Shear_Steel = 460.0
Youngs_Modulus_Of_Concrete	Youngs_Modulus_Of_Concrete,

	"Young's Modulus for concrete" VALUE_TYPE, NUMBER Youngs_Modulus_Of_Concrete = 26567.52403
Youngs_Modulus_Of_Rein_Steel	VALUE_TYPE, NUMBER Youngs_Modulus_Of_Rein_Steel = 210000
Youngs_Modulus_Of_Struct_Steel	"Young's Modulus for structural steel VALUE_TYPE, NUMBER Youngs_Modulus_Of_Struct_Steel = 205000
Max_Precast_Panel_Span_Gn_Load	VALUE_TYPE, NUMBER Max_Precast_Panel_Span_Gn_Load = 11000
Max_Diam_Of_Bars_In_Columns	VALUE_TYPE, NUMBER Max_Diam_Of_Bars_In_Columns = 32
Max_No_Of_Bars_In_Beams	Max_No_Of_Bars_In_Beams, "max number of reinforcing bars in rc beams" VALUE_TYPE, NUMBER Max_No_Of_Bars_In_Beams = 8
Max_No_Of_Bars_In_Columns	Max_No_Of_Bars_In_Columns, "max number of reinforcing bars in rc beams" Max_No_Of_Bars_In_Columns = 16
Max_No_Of_Bars_In_Ribs	Max_No_Of_Bars_In_Ribs, "max number of reinforcing bars in rc beams" VALUE_TYPE, NUMBER Max_No_Of_Bars_In_Ribs = 2
Min_Diam_Of_Bars_In_Beams	Min_Diam_Of_Bars_In_Beams, "min diameter of reinforcing bars in rc beams" VALUE_TYPE, NUMBER Min_Diam_Of_Bars_In_Beams = 10
Min_Diam_Of_Bars_In_Columns	Min_Diam_Of_Bars_In_Columns, "min diameter of reinforcing bars in rc beams" VALUE_TYPE, NUMBER Min_Diam_Of_Bars_In_Columns = 10
Min_Diam_Of_Bars_In_Ribs	Min_Diam_Of_Bars_In_Ribs, "min diameter of reinforcing bars in rc beams" VALUE_TYPE, NUMBER Min_Diam_Of_Bars_In_Ribs = 10
Min_No_Of_Bars_In_Beams	Min_No_Of_Bars_In_Beams, "min number of reinforcing bars in rc beams" VALUE_TYPE, NUMBER Min_No_Of_Bars_In_Beams = 2
Min_No_Of_Bars_In_Columns	Min_No_Of_Bars_In_Columns, "min number of reinforcing bars in rc beams" VALUE_TYPE, NUMBER Min_No_Of_Bars_In_Columns = 4
Min_No_Of_Bars_In_Ribs	Min_No_Of_Bars_In_Ribs, "min number of reinforcing bars in rc beams" VALUE_TYPE, NUMBER Min_No_Of_Bars_In_Ribs = 2
Min_Rc_Slab_Depth	VALUE_TYPE, NUMBER Min_Rc_Slab_Depth = 125

CLASS: Eval_Features	Superclass: Root
Attributes:	Characteristics:
Description	
Importance	ALLOWABLE_VALUES, Extremely, Very, Quite, Not_So, Irrelevant
Importance_Factor	VALUE_TYPE, NUMBER
Target_Max	VALUE_TYPE, NUMBER Target_Max = 0
Target_Min	Target_Min, VALUE_TYPE, NUMBER Target_Min = 0
Target_Set	ALLOWABLE_VALUES, Yes, No
Type_Of_Target	ALLOWABLE_VALUES, Max, Min, Achieve, Any
Methods:	
Feature_Calculation	
CLASS: Vert_System_Column	Superclass: Vert_System_Eval_Features
Attributes:	Characteristics:
Description	Description = "Max_Col_Size_Cm**2"
Importance	ALLOWABLE_VALUES, Extremely, Very, Quite, NotSo, Irrelevant Importance = NotSo
Importance_Factor	VALUE_TYPE, NUMBER Importance_Factor = 1.0
Target_Max	Target_Min, VALUE_TYPE, NUMBER Target_Max = 5000.0
Target_Min	Target_Min, VALUE_TYPE, NUMBER Target_Min = 400.0
Target_Set	Target_Set, ALLOWABLE_VALS, Yes, No Target_Set = Yes
Type_Of_Target	Type_Of_Target, ALLOWABLE_VALUES, Max, Min, Achieve, Any Type_Of_Target = Min
Methods:	
Feature_Calculation	Calculate_Column(Bldg);

Table G1 Class attributes for major classes