CRANFIELD UNIVERSITY

COLLEGE OF AERONAUTICS

Department of Aerospace Science

Ph.D THESIS

Academic Year 1992-1994

WAHYU KUNTJORO

Expert System for Structural Optimization Exploiting Past Experience and A-priori Knowledge

Volume 1: Main Thesis

SUPERVISOR: PROF. A.J. MORRIS

NOVEMBER 1994

ABSTRACT

The availability of comprehensive Structural Optimization Systems in the market is allowing designers direct access to software tools previously the domain of the specialist. The use of Structural Optimization is particularly troublesome requiring knowledge of finite element analysis, numerical optimization algorithms, and the overall design environment.

The subject of the research is the application of Expert System methodologies to support non-specialists when using a Structural Optimization System. The specific target is to produce an Expert System as an adviser for a working structural optimization system. Three types of knowledge are required to use optimization systems effectively; that relating to setting up the structural optimization problem which is based on logical deduction; past experience; together with run-time and results interpretation knowledge. A knowledge base which is based on the above is set up and reasoning mechanisms incorporating case based and rule based reasoning, theory of certainty, and an object oriented approach are developed.

The Expert System described here concentrates on the optimization formulation aspects. It is able to set up an optimization run for the user and monitor the run-time performance. In this second mode the system is able to decide if an optimization run is likely to converge to a solution and advice the user accordingly.

The ideas and Expert System techniques presented in this thesis have been implemented in the development of a prototype system written in C++. The prototype has been extended through the development of a user interface which is based on XView.

ACKNOWLEDGMENT

For my part, I am firstly and mostly indebted to my mother Kusmiasih who sacrificed part of her life and happiness just for us her children. Her prayer and support is always an inspiration to me.

I wish to express my appreciation to my supervisor, Prof. A.J. Morris, for his exceptional guidance in conducting this research. Indeed, I have been fortunate to have him as supervisor.

I also would like to thank Dr. J. Saggu for fruitful discussions and advice in the subject of Expert System. My appreciation to Dr. L. Oswald for his help in providing the computational supports, and also to my colleague (Ph.D student) Mr. T. Kocak for the fruitful discussion in setting up a graphical-user-interface based on XView.

And of course, I wish to express my appreciation to my wife Linda for her understanding and support, in addition to the demands of a home and our little son Ilya.

CONTENTS

LIST OF FIGURES

CHAPTER 1:

INTRODUCTION

1.1. EXPERT SYSTEM AND STRUCTURAL OPTIMIZATION.

The design of a structure is a repetitive process involving a structural definition (synthesis) phase and an analysis phase to meet a set of requirements. A structure can be defined by choosing its type, configuration, and member sizes. The analysis is required to asses the structural responses (stress, strain, displacement, etc.) under environmental action (load, thermal, etc.). The structure performs satisfactorily if its responses are within acceptable values (allowable stresses, frequencies, etc.) which constitute the design constraints. In addition, the designer might choose a certain goal to be satisfied. The common practice for an aircraft structure is to achieve a minimum weight structure. Figure 1-1 illustrates the interaction in a design process.

Structural optimization methods have been developed to allow the design of minimum weight structures in an automative manner. By the use of such methods the design process outlined in the above paragraph can be computerized. The development of computers with powerful capabilities make the implementation of structural optimization methods practical proposition. Figure 1-2 shows the design flow which makes use of the optimization method. Most of the methods try to satisfy both the constraints and the goal.

Most structural optimization methods exploit principles developed in the allied discipline of mathematical programming. A certain degree of expertise is required to describe the structural problem in an appropriate mathematical form. This means the optimization methods and the physical behaviour of the structure to be considered must be well understood.

For a large structure, the analysis stage is performed using the finite element method. This requires the designer to model the structure being optimized by a finite element model. The designer must, therefore, understand the relation between a finite element analysis and an optimization method.

The increasing complexity and size of the structural problems are making the use of optimization methods a reasonable, if not obligatory, choice as an aid to the design of a structure. Unfortunately, most of the optimization packages are rather cumbersome to use and require a thorough knowledge of the methods in order to match the

solution algorithm to the structural design problem. In addition, the fact that the algorithms search for a solution in design space means that they can be subject to error and can give rise to misleading result if employed inexpertly. It is, therefore, imperative that a certain degree of expertise is required to use the optimization method correctly.

Normally the suppliers of structural optimization packages provide training, but there is no substitute for extensive experience which takes time to build up. An alternative approach, followed here, is to develop an Expert System containing the relevant knowledge so that the designers can use these packages effectively and efficiently. In this way the user would not need to be an expert in the use of structural optimization methods and the methods would be available to structural designers.

1.2. RESEARCH OBJECTIVE AND SYSTEM OVERVIEW.

The subject of this research is the application of Expert System methodologies for improving the performance of structural optimization systems. The specific target is, therefore, to produce an expert system as an adviser for a working structural optimization system.

No effort is devoted to developing a separate optimization program as this is considered as impractical. Instead, for its optimization tool, the Expert System should be able to use the currently available structural optimization systems in the market such as ASTROS, STARS, SAMCEF, and NASTRAN Sol-200. This expert system will function as a front end (and back end) to the optimization package. Assuming the user has sufficient knowledge of finite element modelling, the Expert System need concentrate only on the optimization formulation aspects.

Figure 1-3 gives an overall view of the interaction between the Expert System and the optimization package. The users interact with the expert system through a user interface. Based upon the information from the user (such as structural requirements, FEM model, and design variables) and the result of internal decision making within the system with respect to such aspects as the method and convergence criteria requirements (which can be overridden by the user), the system will produce driver files to run the optimization package. The driver files required depend on the optimization software used. After the optimization run, the Expert System can be consulted for the result. Based on the results a consultancy phase is entered and the driver files can be modified if necessary.

1.3. STRATEGY FOR SETTING UP AND MONITORING AN OPTIMIZATION.

To optimize a structure, in addition to preparing a finite element model of the structure, the designer needs to define the optimization algorithm and the design specification. An optimization algorithm, in general, consists of the method to be used, the number of iterations required, and the convergence criteria. Associated with this is the design specification which consists of design variables definitions, the constraints specifications (stress, displacement), and the gauges. The algorithm selection for a specified problem is a function of the analysis and constraints required. The finite element types used in finite element modelling have a profound influence on the method to be used. The size of the problem could also have an influence on the algorithm selection. The design specification usually is laid down by the designer.

The task of the Expert System is to guide the user in setting up the optimization strategy and the design specification. An Expert System is a knowledge based program that provides "expert quality" solutions to problems in a specific domain [Luger and Stubblefield 1989] which, in this research, is the structural optimization. Expert Systems are computer programs that use heuristic strategies to solve specific classes of problems. The common practice for an Expert System to reach a solution is by applying an inferencing strategy to a knowledge base. Thus the immediate task is to set up the knowledge base relating to the structural optimization and develop a reasoning or inferencing strategy for this knowledge base.

Inferencing is a process that attempts to use the available information to find conclusions from a set of statements or finds objects that match. The knowledge base stores information about the subject domain. It contains symbolic representations of experts's judgement rules and experience in a form that enables the inference engine to perform logical deductions upon the knowledge set [Yazdani 1986]. Forward or backward chaining (or their combination) is the inferencing strategy which is widely used. Forward chaining is sometimes called data driven because the inference engine uses the information which the user provides to move through a network of logical AND and OR statements until it reaches a terminal point, which is the goal. Backward chaining is the reverse of forward chaining. It starts with a hypothesis (an objective) and requests information to confirm or deny it. This method is also called goal driven.

The Expert System strategy developed in this research tries to set up the optimization strategy uses the two approaches. The first approach applies to an inferencing

strategy which basically is a forward chain to a knowledge base written by using an object oriented approach. The knowledge base stores the apriori knowledge required for setting up an optimization algorithm. The second approach tries to set up the optimization by recalling the system's past experience. A solution is found if the system finds in its knowledge base a similar design case to the current design problem. Using this second approach, during execution, the system searches its memory to find any similar case to the current design problem. If a similar design case is found, the solution for that case is proposed to the user. If no similar case is found the system will move to the first approach. This approach with past experience is used in case based reasoning.

These two solution strategies work in a complementary manner. The second approach termed the "memory approach" is written as a separate module from the first which is termed the "conventional approach". Both modules together represent a complete Expert System which is not only capable of finding a solution but also memorizing this solution and the details of the design case. With increasing design experience, the system becomes more capable in searching for a solution.

The mechanism adopted in the memory approach discussed above raises the question of defining similarity criteria. This is not straight-forward as appropriate similarity rules do not exist and are defined as part of this study. It is noted that some of the similarity parameters might not be deterministic in nature. Figure 1-4 shows a simplified sequence which represents the strategy discussed above.

In addition to setting up the optimization, the user must be able to monitor the performance of an optimization run in order to decide if (1) a solution is being approached and (2) at termination, the result is satisfactory. If the above situations are negative the system should offer appropriate explanations and indicate the required remedial action. This is implemented by developing a module which basically is an Expert System with forward chain inferencing and production rules. The knowledge required in this module, which is called the back-end module, is associated with the run-time situation and results interpretation. In addition, if the optimization run does not produce satisfactory results, based on the proposed remedial actions the back-end module should be able to modify the optimization set-up.

1.4. SYSTEM IMPLEMENTATION.

1.4.1. DRIVER FILES, FRONT-END, AND BACK-END.

The strategy mentioned in (1.3) is required to produce an appropriate optimization algorithm for a structural design problem. With regard to an optimization package, this algorithm should be written in a manner which is understood by that package. In addition to the algorithm, an optimization run requires the design specification and the finite element model. The design specification is related to the design variables, constraint set, and gauges. It is assumed that it is the user's responsibility to set up the FE model. A module is developed to write the optimization strategy and design specification into a file (or files) which can be read by the optimization package. These files are called driver files and the module which produces them is called the driver files module.

The front-end is that part of the total system from the system start up to the production of driver files. The conventional and memory approaches mentioned in (1.3) are part of this system's front-end which contains both procedural and Expert System programming methods. When the front-end activity is complete this is followed by an optimization run.

The system's back-end is invoked to monitor the progress of an optimization run. If an optimization run is predicted not to be heading for a solution (convergence) this module will cancel the run and advise the user of the possible explanation and actions to follow. At termination the result will be interpreted and if considered unsatisfactory the possible reason is explained with advice on actions to follow. Based on the actions proposed, the optimization set up can be modified, and another optimization run can be executed.

1.4.2. THE PROTOTYPE MODULES.

For system implementation, the STARS optimization package jointly developed by DRA and SCICON is used as the optimization tool. For performing the associated structural analysis, STARS calls the MSC NASTRAN finite element package. STARS requires three files (driver files) to run, which consists of a command data file (CDF), a design specification file (CONS), and a file which contains the FE model. The first two files are produced by the driver files module. Because STARS uses NASTRAN for its FE analysis, the FE model consists of a NASTRAN bulk data. The system is developed using C++ on a SUN SPARC II computer.

The command data file is an optimization algorithm which includes the optimization method, the number of iterations, and the convergence criteria. The design specification file (CONS) contains the design variable definition, the constraints specifications and the gauges. CDF and CONS are written in statements which are unique to STARS.

Figure 1-5 shows an outline of the modules available in the program. These modules are written using the object oriented approach.

- The controller is the module which controls the flow of information in the front end and basically is the main program.

- The design input module asks the users about the design case.

- The constraint module task is to handle information concerning constraint specifications and gauges.

- The variable module deals with all aspects regarding the design variables definitions.

- The element module deals with element data and operations. This module also contains various element types. Currently this module contains some elements which are found in the MSC NASTRAN finite element package.

- The driver files module has the task of producing driver files. Currently, it produces STARS files only.

- The memory module has the task of setting up the optimization by recalling the system's past experience.

- The conventional module has the task of setting up the optimization by applying inferencing technique to apriori knowledge.

- The system's backend which consists of the Run-time, Result Interpretation, and Modification modules, monitors the progress of an optimization run and at termination interprets the result. If there is any problems associated with optimization run, the system will search for an explanation and propose actions to follow. If required, based on these proposed actions, the modification module modifies the optimization set up.

Although not indicated in the diagram, there are several front-end modules, the design input module, the constraint module, the variable module, the element module, and the driver-files module, which are also used (shared) by the modification module.

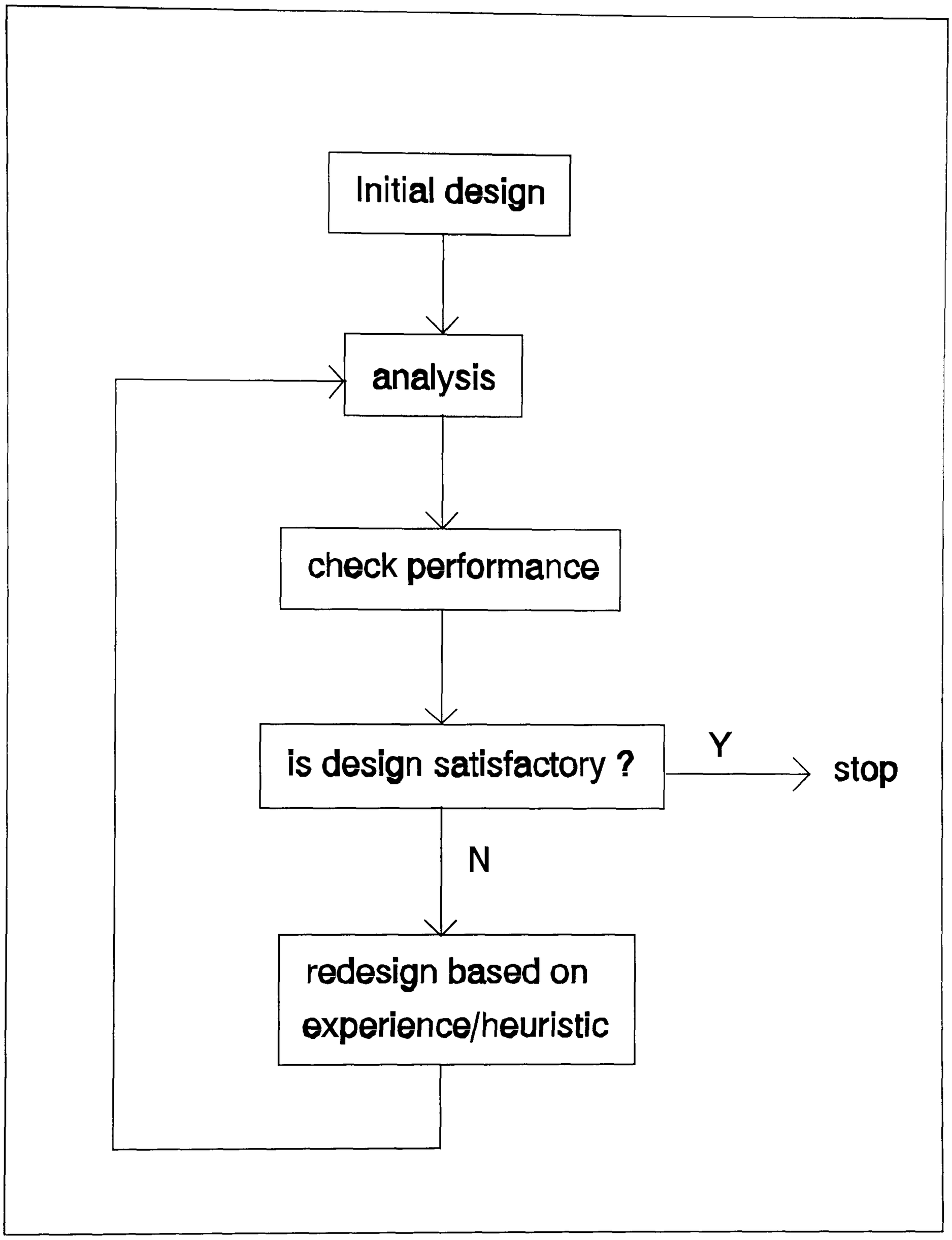


Figure 1-1 The conventional design process [Arora 1989].

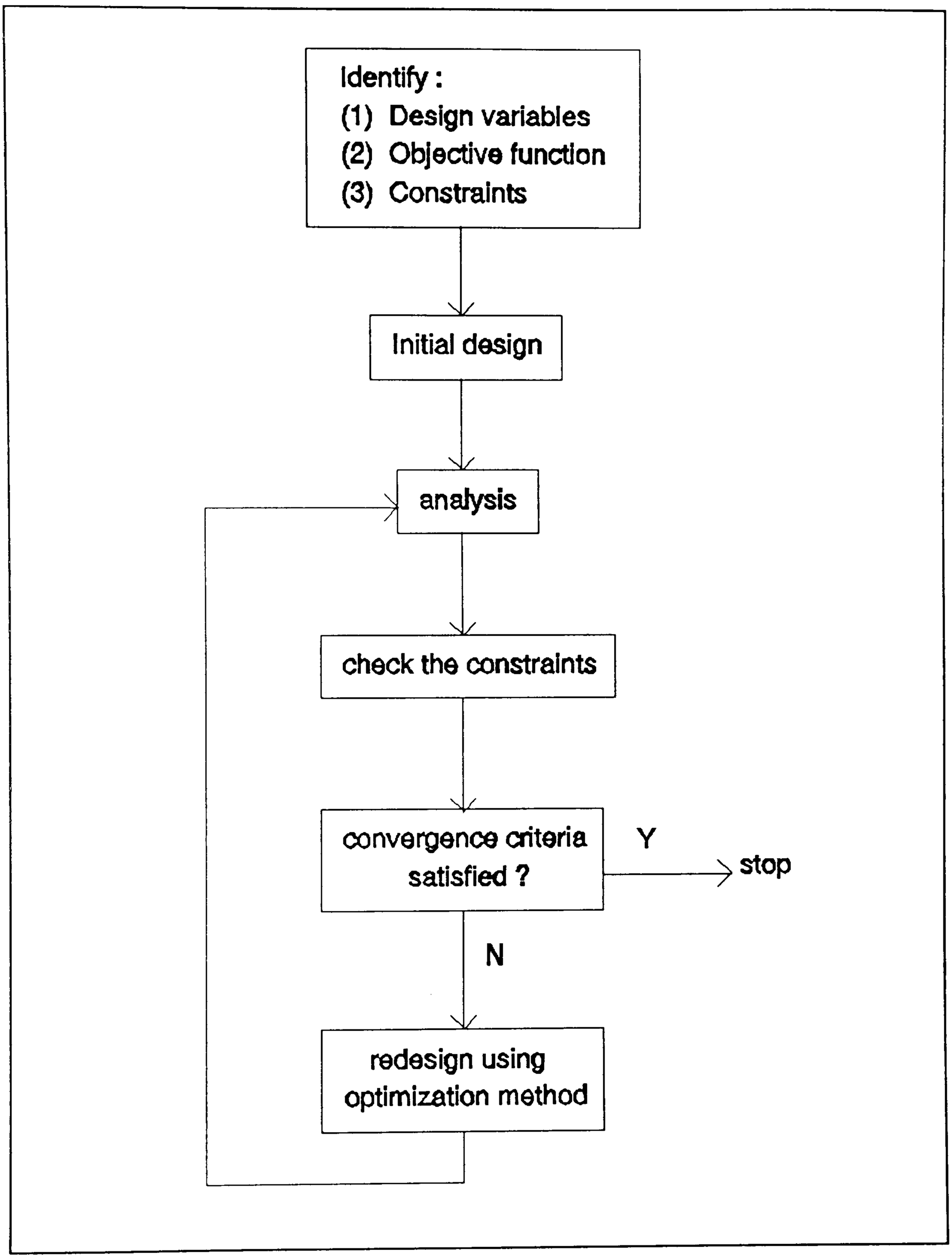


Figure 1-2 The optimum design process.

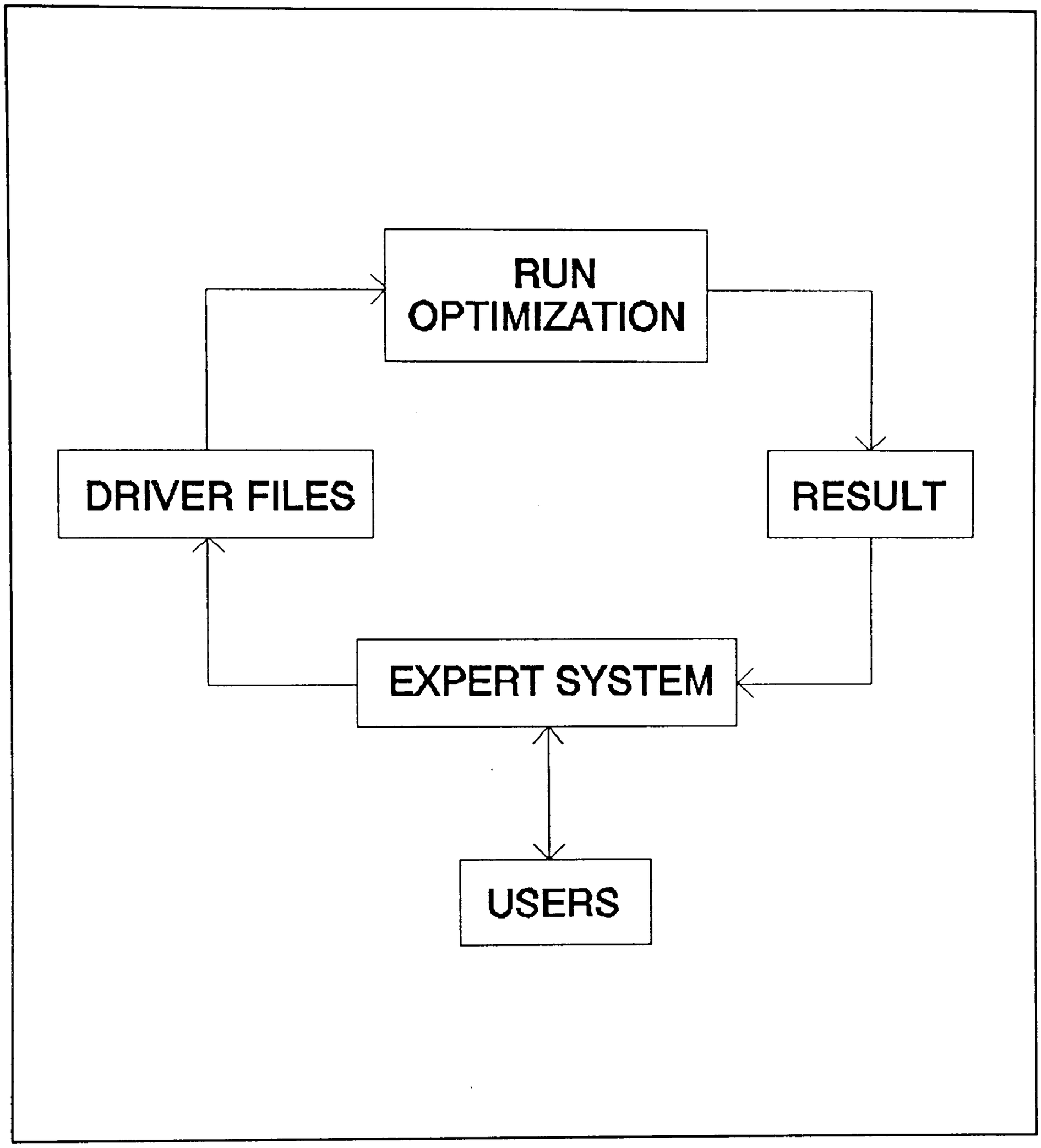


Figure 1-3 Expert System - optimization interaction.

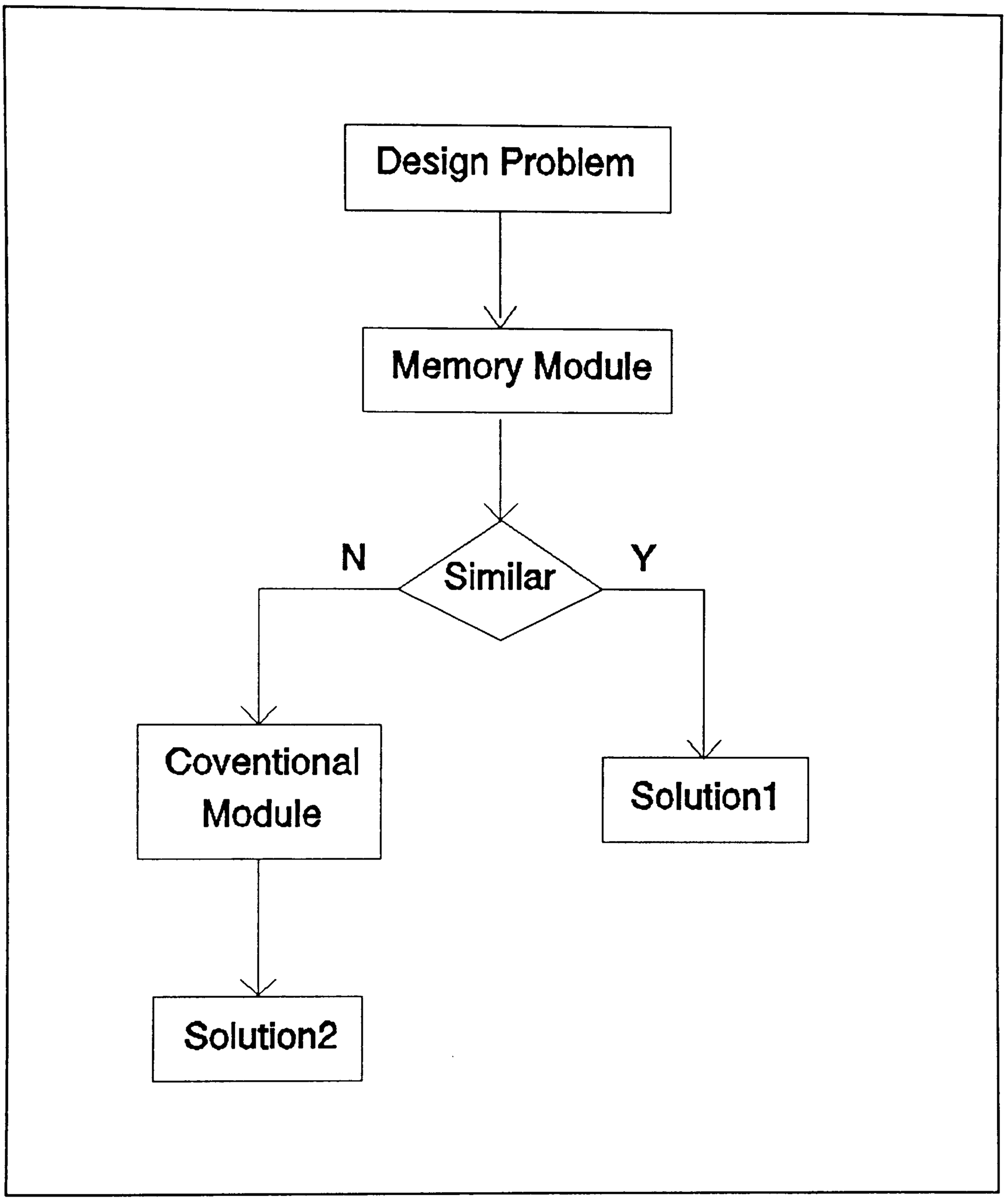


Figure 1-4 Strategy for finding a solution using the memory and conventional modules.

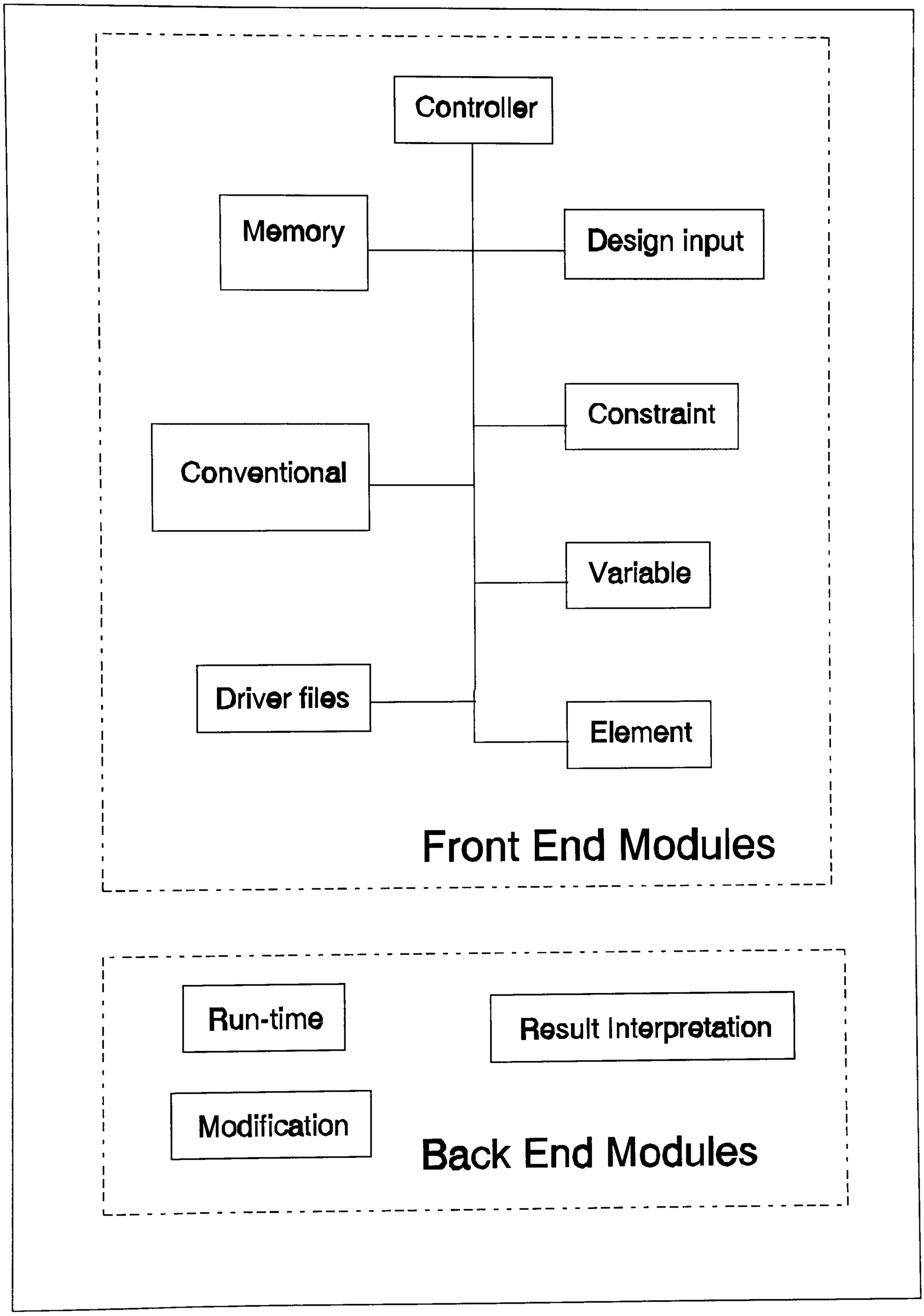


Figure 1-5 Program modules.

CHAPTER 2:

LITERATURE ON EXPERT SYSTEM FOR STRUCTURAL DESIGN AND OPTIMIZATION

Numerous studies have been reported in the literature relating to the development of Expert Systems related to design optimization. Some of these are reviewed briefly in this chapter. However, the first section is more general and reviews some successful examples of Expert System in different subjects. The following section then reviews the use of Expert Systems for structural analysis and design.

2.1. EXAMPLES OF SUCCESSFUL EXPERT SYSTEMS.

Several successful applications of the Expert System technology in various fields have been developed since the revival of Artificial Intelligence as a legitimate subject of research.

1. DENDRAL.

DENDRAL was designed to infer the structure of organic molecules from their chemical formulas from mass spectrographic information about the chemical bonds present in the molecules. Because organic molecules tend to be very large, the number of possible structures for these molecules tends to be huge. This problem of large search space is addressed by applying the heuristic knowledge of expert chemists to the structure elucidation problem. It represents one of the earliest Expert Systems and was developed at Stanford in the late 1960s.

2. MYCIN.

Developed at Stanford in the mid-1970s, MYCIN [Buchanan and Shortliffe 1985] was one of the first programs to address the problem of reasoning with uncertain or incomplete information. This Expert System uses expert medical knowledge to diagnose and prescribe treatment for spinal meningitis and bacterial infections of the blood. Written in INTERLISP, a dialect of LISP, with around 450 rules within its knowledge base, MYCIN established the methodology of contemporary Expert System implementation.

3. XCON.

XCON is an Expert System for configuring DEC computers. In 1981, XCON had about 500 rules and could configure the VAX 780. It was progressively refined until in 1984 with about 4000 rules, it configured most of the DEC product line.

4. PROSPECTOR.

This Expert System was developed at Stanford in the late 1970s. It is a diagnostic Expert System for determining the probable location and type of ore deposits based on geological information about a site. Its knowledge base contains about 1600 rules. Bayesian probability is used for treating uncertainties in information and rules.

5. INTERNIST.

This is an Expert System for performing diagnosis in the area of internal medicine. Developed further (with the name CADUCEUS), its knowledge base includes about 500 disease, 350 diseases manifestations, and about 100,000 symptomatic associations. CADUCEUS covers about 25 percent of the diseases of internal medicine.

The above Expert Systems can be regarded as classical Expert Systems and often mentioned in Expert System text books. These systems use rules (IF-THEN statements) for reasoning. Some systems, such as MYCIN and PROSPECTOR, address uncertainty in information and rules. As can be seen in later chapters, the Expert System presented in this thesis also use rules and has to deal with reasoning with uncertainty.

2.2. EXPERT SYSTEMS FOR STRUCTURAL ANALYSIS AND DESIGN.

The first successful application of Expert System technology in the structural analysis field appears to be SACON (Structural Analysis Consultant). Developed in LISP [Bennet and Engelmore 1979], SACON interacts with the user for the proper application of MARC finite element software. Rivlin et al. [1980] also attempted to develop a knowledge-based consultation system and to establish a finite element structural analysis knowledge base for the MARC finite element program in FORTRAN.

Chen and Hajela [1987] developed an expert system which serves as a consultant for finite element modelling (FEMOD). FEMOD gives the designer advice in modelling for a special finite element program EAL. It contains rules for selection of elements, node selection, mesh generation, selection of subdivision lines, and mesh refinement. FEMOD was developed using an Expert System shell AESOP with rule-based knowledge representation and backward chain inferencing.

A widely cited design Expert System is HI-RISE [Maher 1988]. HI-RISE is an Expert System for the preliminary structural design of high rise building of rectangular shape. The major concern of HI-RISE is to generate feasible configurations, to the level of detail needed to make a

selection from amongst alternatives, and to provide the initial estimate of geometric and mechanical properties for a detailed structural analysis. HI-RISE represents the design knowledge in the form of schemas and rules. The schemas contain the description of the design subsystems and components, and the rules represent design strategy and heuristic constraints.

An extension of HI-RISE is ALL-RISE [Sriram 1987]. It provides a framework for a knowledge-based synthesizer for all types of buildings. Multiple solutions are generated simultaneously, i.e there is no need for backtracking because ALL-RISE generates all the feasible alternatives.

Adeli and Al-Rijleh developed an Expert System for design of roof trusses, called RT-EXPERT. As described by Adeli and Balasubramanyam [1988] RT-EXPERT can advise the user on the appropriate type of the roof truss, selection of the layout of the truss, and the loading. The knowledge base and explanation facility is developed using INSIGHT 2+ Expert System shell. The mathematical computations, graphic algorithms, and the data file manipulation routines are developed in Turbo Pascal.

An Expert System which employ mathematical optimization was developed by Rogers and Barthelemy [1987]. It is in the form of an Expert System "pre-processor" for selecting the best combination of optimization techniques and one dimensional search method used in the optimization software Automated Design Synthesis (ADS) developed by Vanderplaats. This expert system does not address the problem of monitoring the optimization run and the result interpretation.

Harris et.al [1990] developed an Expert System package as an interface to the STARS optimization package. This system addresses problems of algorithm selection, quality of convergence, and constraint satisfaction. The Expert System was written using FLEX (Fortran Library for Expert System) developed by RAE. The knowledge used in FLEX comprises rules and facts. Rules are represented in the standard production rules. FLEX supports both forward and backward chaining inferencing. This Expert System does not have the capabilities to learn from experience.

Adelli and Balasubramanyam [1988] developed a coupled Expert System prototype for optimum (weight) design of bridge trusses subjected to moving loads. This prototype (called BTEXPERT) has the knowledge base which consists of the domain specific knowledge and control knowledge. The domain specific knowledge consists of parameters and rules where the rules are in the IF-THEN form. The control knowledge consists of control command for solving the problem. The inferencing has both forward and backward chaining.

This system does not address structures other than truss structure.

Berke et al. [1993] investigated the application of artificial neural networks to capture structural design expertise. An artificial neural network code, NETS, was used. A set of optimum design data were processed to obtain input and output pairs, which were used to develop a trained artificial neural network with the code NETS. Optimum designs for new design conditions were predicted by using the trained network. The nature of neural network in producing the solution makes it difficult for the users to understand the reason of any decision given by the network. This makes this system unsuitable as an optimization training media for non-experts.

This research is concerned with the development of an Expert System as an adviser for a working structural optimization system. The system is developed in such a manner that it covers the following:

1. It is able to handle various types of structure. This implies the use of the Finite Element method in the analysis stage.

2. It accommodates various optimization softwares available in the market.

3. It is able to perform the tasks associated with,
   - setting-up the optimization strategy,
   - monitoring the optimization run,
   - interpreting the optimization result,
   and, in case of unsatisfactory results, advising the users on actions to follow.

4. It has some capabilities to learn from past experience.

5. It is able to function as a training media for the users in learning the design for optimum structure. This implies the provision of a comprehensive explanation facility.

CHAPTER 3:

IDENTIFICATION OF TASKS INVOLVED IN A STRUCTURAL OPTIMIZATION

This chapter is devoted to identifying the tasks involved in performing an optimization for a structural design problem. Before discussing the tasks associated with a structural optimization system, several optimization methods and some basic concepts will be described first. Detailed discussions of the methods employed are not considered here as these are adequately considered elsewhere [Arora 1989; Morris 1982; Vanderplaats 1984]. The current chapter contains an overview to provide completeness.

3.1. OPTIMIZATION METHODS AND BASIC CONCEPTS.

The methods described in this section are those which are available in STARS optimization package and also commonly found in most commercial systems. These methods are quite well known and detail description of each method can be consulted in related references. Only general discussion are presented here.

3.1.1. FULLY STRESSED DESIGN.

Fully Stressed Design, or FSD, is applicable to the design of stress (strength) critical designs only and cannot deal with more general constraints such as displacement or frequency. This method is not really an optimization method in the context of numerical programming but instead is an automatic form of a design technique which has been used for many years before the advent of the computer [Vanderplaats 1984].

Fully Stressed Design procedures are based upon the assumption that in an optimal structure each member is subjected to its allowable stress under at least one of the loading cases. There is no explicit reference to an objective function (e.g. weight). This approach consists of the iterative application of analysis and redesign rule, where the redesign rule is based on stress ratio.

FSD algorithm is quite simple.
For member i of a structure :

$$X_i^{new} = \max_{j=1,NLC} \left(\frac{\sigma_{ij}}{\sigma} X_i^{old} \right) \qquad (3\text{-}1)$$

where X = size of member
NLC = number of load cases
σ_{ij} = stress at member
σ = allowable stress

Reasons for the significance of FSD concepts include [Kirsch 1981] :

1. Engineering experience indicates that a good design is often one in which each member is subjected to its allowable stress.

2. FSD can be proved to be optimal (weight) under certain circumstances.

3. FSD procedures are relatively efficient in comparison with many mathematical programming methods.

4. An FSD is often a good starting point for optimum design procedures based on mathematical programming.

3.1.2. OPTIMALITY CRITERIA.

The optimality criteria methods aims to obtain a design that satisfies a certain specified criterion and by so doing indirectly minimize the weight of the structure. The criterion is derived mathematically by differentiating the Lagrangian with respect to the design variables and imposing the Kuhn-Tucker optimality conditions. In deriving the optimality criterion and developing the algorithm, full use is made of the knowledge of the behaviour of the constraints imposed on the structure. Indeed, fully stressed design algorithm is a special case of optimality criteria based on stress constraint. The optimality criteria method is discussed in detail by Morris [1982].

3.1.3. MATHEMATICAL PROGRAMMING.

In mathematical programming, the structural design problem is formulated into mathematical expression. The non-linear constrained optimization problem can be formulated by,

finding $\{X\}$ so that

Objective Function : $F(\{X\})$ is minimum (3-2)

Subject to constraints,

$$g_j(\{X\}) \leqslant 0 \qquad j=1,m \tag{3-3}$$

$$h_k(\{X\}) = 0 \qquad k=1,l \tag{3-4}$$

where $\{X\} = (X_1, X_2, X_3, \ldots, X_n)$ are design variables, g_j and h_k is inequality and equality constraint respectively.

In finding the optimum, the design is changed iteratively. The iterative procedure is given by,

$$\{X^{q+1}\} = \{X^q\} + \alpha.S^q \qquad (3\text{-}5)$$

where S is the search direction in the design space and α is the length of search in finding the optimum of that search. The search procedure can be described as finding an optimum in a design space. For a two variables case, the procedure is illustrated in Figure 3-1. It shows the search sequences from the initial design $\{X^1\}$ up to the optimum point.

3.1.3.1 Pseudo-Newton Method.

The Pseudo-Newton method is a second-order projection method which makes use of the concept of finding an optimal searching along a Newton step which is projected onto the constraints. Based on a quadratic approximation to the objective function and linearized constraints the problem is,

$$\min F(\{X\}+\{h\}) = F(\{X\}) + \nabla F^T.h + 1/2\ h^T.H.h \qquad (3\text{-}6)$$
$$\text{subject to} \quad \{N^q\}^T\ h = 0$$

$\{N^q\}$ is the gradients of active constraints, H is the Hessian matrix of the objective function, and h is the vector of design variable changes.

It can be shown that the Lagrange multipliers, λ, are given

$$\lambda = (\ \{N^q\}^T\ H^{-1}\ \{N^q\}\ )^{-1}\ \ \{N^q\}^T\ H^{-1}\ \nabla F \qquad (3\text{-}7)$$

and consequently,

$$h = -\ (\ I - H^{-1}\ \{N^q\}\ (\ \{N^q\}^T\ H^{-1}\ \{N^q\}\ )^{-1}\ \ X$$
$$\{N^q\}^T\ )\ H^{-1}\ \nabla F \qquad (3\text{-}8)$$

Some of the advantages of using a Pseudo-Newton algorithm are that it both satisfies differential Kuhn-Tucker conditions and provides estimates of the Lagrange multipliers.

In its operation, any projection algorithm requires that the active constraints set is determined at each iteration. A basic active set strategy can be constructed using following steps at the end of each iteration:

1. Drop constraints with negative lagrangian multipliers.
2. Add violated constraints (after the analysis step).
3. Apply anti-zigg-zagg rules.

3.1.4. CONVERGENCE CRITERIA.

Convergence criteria is used to detect whether a design solution is an optimum one. There are several criteria commonly used. Some of them are,

1. The relative change in the design variables is less than a small predetermined value.

2. The relative change in the Objective Function is less than a specified tolerance.

3. The difference between the feasible weight and the actual weight is less than a small predetermined value.

4. The difference between the feasible objective function (weight) and a lower bound on the optimum is less than a specified tolerance.
   A lower bound for a design can be found by solving the dual problem.

3.2. OUTLINE OF SETTING-UP A TYPICAL STRUCTURAL OPTIMIZATION JOB.

There are several standard tasks which have to be performed by structural designers in the optimization of a design. These tasks are concerned with the way the problem is defined and modelled. There are eight main tasks which are outlined below and then considered in detail:

1. Problem Definition and Identification.
   (a). Identifying structural type (truss, frame, etc).
   (b). Problem type (static, non-linear, dynamic, etc).

2. Finite Element Modelling.
   This includes all aspects of FE modelling such as meshing and element type selection.

3. Definition of Objective Function.
   Minimum mass is common as a goal though other objective functions are now being actively considered as cost and manufacturability are key items in a modern concurrent engineering approach.

4. Variable Selection.
   This requires considering two aspects:

(a). Defining variables.
(b). Variable linking.

5. Relations between Design Variables and Analysis Model.

6. Constraint Definition.
Type of constraints (gauge, stress, displacement, etc).

7. Algorithm Selection.

8. Criteria of Convergence.
The selection of criteria.

3.2.1. PROBLEM DEFINITION.

Firstly, the type of structure to be optimized should be identified. The structure could be a truss, frame, plate, stiffened-shell, or other types, including combinations of structure types in a typical built-up construction. An important aspect is the analysis type. This is a function of the environment which defines the loadings, displacements, etc., and how the structure responds subject to these loads. These responses may require a static analysis, a normal modes analysis, a dynamic analysis, etc.

3.2.2. FINITE ELEMENT MODELLING.

For a complicated structure with a large number of elements, the Finite Element Method is the obvious choice for structural analysis. Many structural optimization packages such as STARS, ASTROS [Johnson et al. 1989], and NASTRAN SOL-200 [Miura 1988], offer finite element analysis module (which is NASTRAN in those three mentioned packages).

The finite element modelling problem involves tasks such as mesh generation, element selection, constraint definition (single or multi point constraints), and loading application. Certain type of elements are suitable only for certain optimization algorithms. With regard to the Expert System discussed here, the FE model is input data and assumed to be built around NASTRAN finite element analysis.

3.2.3. OBJECTIVE FUNCTION.

An objective function is a mathematical formulation of the entity to be optimized and is a function of the design variables. In general terms, an objective function could be anything defined as a goal to be achieved. In a structural problem, it is common to design a structure for minimum

mass (especially for aerospace structure) and structural mass is the objective function in this research. Nevertheless, any Structural Optimization Expert System must be open to the inclusion of alternative objective functions. The pressure to move away from mass is coming from the requirement that the more advance CAD systems consider a range of design criteria.

3.2.4. DESIGN VARIABLES.

The variables in structural optimisation can be member sizes and the configuration of the structure. The parameters defining the size of a structural members can be complicated. For example in a truss structure, the structural members might be a hollow tubes, where "size" can be defined as a combination of diameter and thickness. Hence, there are two variables for each member, the diameter and thickness. For different type of structural member (channel section, I-section, etc), there are different types of variables, which depend on the parameters defining the size.

Configuration optimization of a structure deals with finding an optimum (e.g. minimum mass) configuration of a structure. In many cases, the structural configuration is constrained by available space and structural functionality and has been fixed already when it goes to the optimization stage (hence only members size optimization is needed). The optimization tool in this research does not handle optimization of structural configuration.

A second aspect in this section involves the linkage of specific element design variables into linked design variables. This linkage process involves examining the structural behaviour patterns to define which elements are to be associated within a given design variable. The selection process can be augmented by the use of sensitivity analysis to improve the performance of the design variables through a relinkage process.

3.2.5. RELATION BETWEEN DESIGN VARIABLE AND ANALYSIS PROPERTY.

An optimization method deals with design variables whilst the finite element analysis deals with analysis properties. This means that a relationship between the design variable property and the analysis property must be determined.

For example, consider a truss structure with a hollow tube beams as members. The design variables are normally the bar diameter and thickness. If a finite element analysis is based on truss analysis (CONROD element in NASTRAN),

then the property analysis is cross-sectional area. A relationship between the design variables (thickness, diameter) and the property analysis (area) has to be defined. This depends on the optimization packages being used. For example, STARS (current version) only allows the cross-sectional area to be used as the design variable for its CONROD element. On the other hand, using the optimization module in NASTRAN (SOL 200), it is possible to define the relation function as

$$A = f(D,t) \qquad (3\text{-}9)$$

A = area
D = diameter
t = thickness

which for a tube is

$$A = 3.1416 \times D \times t \qquad (3\text{-}10)$$

3.2.6. CONSTRAINTS DEFINITION.

An optimization procedure is not only aiming at minimizing (or maximizing) the objective function, but it must also satisfy the set of constraints imposed. The constraints should be in accordance with the analysis type selected. The constraint types which can be handled depend not only on the associated finite element systems solution capabilities but on the element types themselves. Certain elements cannot be employed if analytic derivatives are required by the solution algorithm. A complex interrelation exists between the analysis system, the elements used to model the structure, the solution algorithm, etc.

3.2.7. ALGORITHM SELECTION.

The specific optimum seeking method selected directly affects the optimization algorithm to be applied. Some methods require constraint derivative while others do not. In STARS there are three methods available: Fully Stressed Design, Optimality Criteria, and Pseudo Newton. It is also possible to use them in combination. The number of iterations also needs to be determined. The selection of a specific algorithm or combination of algorithms represents one of the most complex aspects of the setting-up of an optimization run. The suitability of a solution strategy is difficult to asses if the design problem covers a combination of problem areas, statics, dynamics, aeroelasticity, etc, and the structure/loadings are large. In forming a solution strategy, past experience is, often, the main source

of information. If no experience is available the user has to proceed with extreme caution.

3.2.8. CRITERIA OF CONVERGENCE.

The selection of the convergence criteria actually depends on the problem type and the capabilities of the system. Most systems have the usual criteria which stop on the basis of small rates of change in specific parameters, design variables for example. Other systems can employ a dual bounding procedure but this may not be employed with non-derivative based algorithms.

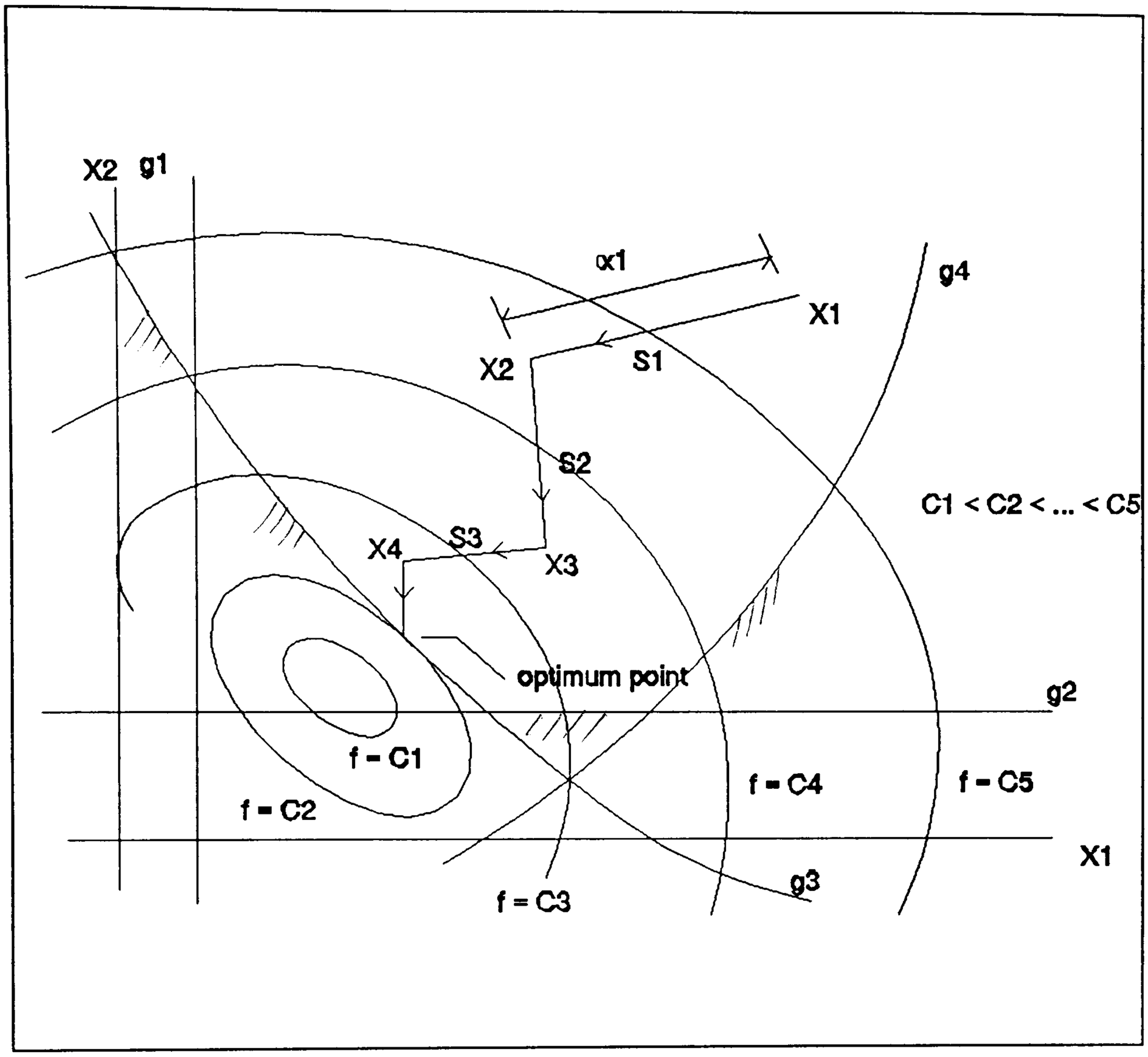


Figure 3-1 Searching for optimum in a two variable design space.

CHAPTER 4:

REASONING STRATEGIES

This chapter discusses aspects of Expert System (Knowledge Base Systems) methodologies placing emphasis on two aspects, the implementation of object oriented concepts in constructing a knowledge base system, and case based reasoning employing past experiences.

4.1. EXPERT SYSTEM: GENERAL DESCRIPTION.

4.1.1. EXPERT SYSTEMS.

An Expert System is a knowledge-based program that provides solutions to problems in specific domains [Luger and Stubblefield 1989]. Generally, its knowledge is extracted from a human expert (hence the term Expert System). The knowledge relates to both theoretical and practical aspects in the domain. These systems are able to use heuristic strategies to solve specific classes of problems. Because of the heuristic, knowledge intensive nature of expert level problem solving, Expert Systems are generally :

- Open to inspection, both in presenting intermediate steps and in answering questions about solution process.

- Easily modified, both in adding and deleting skills from the knowledge-base.

- Heuristic, in using (often imperfect) knowledge to obtain solutions.

Figure 4-1 shows a typical Expert System architecture.

A. The Knowledge-Base.

The three fundamental components of an expert system are the knowledge base, the inference engine, and the user interface. The knowledge base stores information about the subject domain. However, information in a knowledge base is not a passive collection of records and items which are found in a conventional database. Rather it contains symbolic representations of experts' rules of judgement and experience, in a form that enables the inference engine to perform logical deductions upon it [Yazdani 1986].

In general, knowledge can be represented in a number of ways. An example of four methods are shown below:

1. Rules in IF-THEN format : conditions are the rule element between IF and THEN, while the conclusions are elements which follow THEN.

2. Semantic Nets : these represent relations among objects in the domain by links between nodes. Semantic nets can be drawn up to assist the development of rules.

3. Frames : these are a generalized record structure with fields (or slots).

4. Horn clauses : this is a form of predicate logic on which PROLOG is based.

With the advent of Object Oriented Design programming, see (4.2), there are new developments which utilize this approach in constructing not only software but also knowledge base. Object oriented programming can also produce frames-like structure and simulate semantic nets. Object oriented programming is ideal to represent objects to be organized in taxonomies, and as optimization methods can be organized in this way then it is appropriate to employ this approach of programming. However, rules are a natural formalism for representing heuristic and problem solving knowledge [Luger, Stubblefield 1989]. For this reason rules are used in the Expert System for Structural Optimization for representing heuristic strategy and problem solving.

B. The Inference Engine.

B.1. Chaining.

Inferencing is a reasoning process that attempts to use the available information to find matching conclusions or objects. The most commonly used inferencing strategies are forward-chain and backward-chain.

Forward-chaining is sometimes called data-driven because the inference engine uses information that the user provides to move through a network of logical ANDs and ORs until it reaches a terminal point, which is the object. If the inference engine can not find an object by using the existing information, then it requests more. The only way to reach a solution is by satisfying all of its rules.

Suppose that the user of a structural optimization system decides to use a stress ratioing algorithm to solve a displacement constrained problem. The inferencing process would start the requirement that the design problem has a

displacement constraint. The chaining process starts with the inference engine firing the rule whose condition part deals with the displacement constraint and progresses from this point. Sooner or later it encounters a rule which states:

```
IF    ( constraint  = displacement  )  and
      ( algorithm = stress_ratioing )
THEN  inappropriate algorithm selected
```

Backward-chaining is the reverse of forward-chaining. It starts with a hypothesis (an object) and requests information to confirm or deny it. This method is also called goal-driven.

To illustrate backward-chaining we consider the stress-ratioing, displacement constrained proposition discussed above. In this case the inferencing mechanism works from a likely-looking conclusion to find the conditions which must apply for that conclusion to be true. The conclusion being sought is that the stress-ratioing algorithm is acceptable. When the propositions associated with this conclusion are examined it will be found no rule exists which implies that stress ratioing is inappropriate when displacement constraints are present.

The selection of chaining methods to be used depends on the nature of the problem. In our Expert System, forward-chaining is used for reasoning in the apriori knowledge for setting up the optimization and in the run-time and result-interpretation knowledge. This will be discussed in more details in chapter 6.

There are often situations where the Expert System must attempt to draw correct conclusions from uncertain evidence using rules which are not one hundred percent reliable. There are several ways of managing these uncertainties with the commonly used being the Bayesian approach and theory for certainty.

B.2. Incorporating Uncertainty.

The Bayesian approach to uncertainty is based on formal probability theory. Assuming a random distribution of events, probability theory allows the calculation of more complex probabilities from previously known results. Where the assumptions are met, Bayesian approaches offer the benefit of a well founded and statistically correct handling of uncertainty. However, many Expert System domains do not meet these requirements and must rely on more heuristic approaches.

The theory of uncertainty makes some simple assumptions for creating confidence measures and has some equally simple rules for combining these confidences as the program moves toward its conclusion. Based on a piece of evidence the certaintiness of a hypothesis, called as the certainty factor (CF), can be measured. The range of CF is between -1 and 1. As the certainty factor approaches 1 the evidence is stronger for a hypothesis; as CF approaches -1 the confidence against the hypothesis gets stronger; and a CF around 0 indicates that there is little evidence either for or against a hypothesis. In a rule base system, the reliability of a rule's conclusion (or hypothesis) can be measured if the CFs of rule premises and the CF associated with that rule is known. When two or more rules support the same result, the multiple CFs also can be combined.

Either the Bayesian approach or the theory of uncertainty can be critized for using numeric approaches to the handling of uncertain reasoning. Many would argue that it is unlikely that humans use these numeric valuation paradigms for reasoning with uncertainty. Some applications tend to require a more qualitative approach to the problem. If human experts are asked why their conclusions are uncertain, they tend to answer in qualitative relationships between features of the problem instance. However, if someone says that something is better than the other, one could ask how much better. If one compares ten systems and tries to put them in order with respect to their performances, probably numeric approach is better than qualitative relationship.

These probability aspects have an important role to play when questions relating to the similarity of design problems are addressed. As we shall observe, saying that two design problems are similar does not mean they are identical. Thus similarity has probability aspects associated with its use. This will be discussed in more details in chapter 5.

4.1.2. EXPERT SYSTEM DEVELOPMENT.

Expert System development requires a non-traditional development cycle based on early prototyping and incremental revision of the code. There is also complications due to the fact that (most) Expert Systems are jointly developed by the knowledge engineer (program builder) and the domain expert.

Expert Systems are built by progressive approximations, with the program's mistake leading to corrections and addition to the knowledge base. In a sense, the knowledge base is "grown" rather than constructed. The second major feature of Expert System programming is that the pro-

gram needs never be considered "finished". The modularity and easy modification available in the production system (rules) model make it natural to add to or modify rules [Luger and Stubblefield 1989].

Many of the successful Expert Systems have been the direct result of early research which has gone on since mid of the 1960s. Classic Expert System such as MYCIN required around fifty person-years to develop, while another successful system, DENDRAL, required almost forty person-years to develop.

However, there is an important aspect of Expert System programs, the average development time for Expert Systems has been drastically reduced across the decades of evolution. The emergence of Expert System shells was instrumental in reducing the design time.

An Expert System shell is an "expert" system which has everything (inference engine, explanation facility, knowledge base editor, and user interface) except the knowledge base. The system designer needs only to insert domain knowledge into the system by using a knowledge base editor. There is no doubt, that this system is a great help to the designer of an Expert System. The drawback of this system is its limitation to the numerically complex domain problems. With respect to the optimization problem it is difficult (if at all possible) to interfacing the commercially available Expert System shells with an optimization module/ software. In this research, the Expert System is developed as a generic system specially to deal with structural optimization problems.

4.2. OBJECT ORIENTED KNOWLEDGE BASE.

4.2.1. BASIC CONCEPTS OF OBJECT ORIENTED PROGRAMMING [Meyer 1988; Schildt 1990; Booch 1991].

Object oriented design is the construction of software systems in the form of structured collections of abstract data type implementations [Meyer 1988]. Every module in an object oriented architecture is built on a data abstraction. The principle of abstraction is used to construct solutions to problems without having to take into account the intricate details of the various component subproblems.

The modules of object oriented systems are called classes. An object is not a class, but an object is an instance of class. The class structure encompasses both properties and behaviour of an object which it defines. In other words, class is the template of object.

An object is a dynamic and complete entity. It has properties unique to itself and a set of operations (functions) on its properties. When required, an object is also able to interact with other objects. Indeed, an object oriented system is a system which operates on objects and objects interactions. Object also support the principle of data encapsulation which protects data within an object.

Object oriented systems also support the principle of inheritance where a class is a descendant of one or more other. Inheritance permits objects to be organized in taxonomies in which specialized objects inherit the properties and functions of more generalized objects. However, it might be possible for an object to redefine an inherited property or operation.

Polymorphism is another concept supported by object oriented program. Polymorphism essentially means that one name can be used for several related but slightly different purposes. The purpose of polymorphism is to allow one name to be used to specify a general class of actions. However, depending upon what type of data it is dealing with, a specific instance of the general case is executed.

Object oriented software is implemented using an object oriented language. The term object oriented language is used to separate this language type from conventional non-object-oriented languages such as Basic, Fortran, Pascal, and C. Examples of pure object oriented languages are Smalltalk, Eiffel and Simula; C++ is an extension of C which supports object oriented characteristics [Schildt 1990]. There are also available extensions of Lisp and Pascal which support object oriented principles.

In this research C++ is used to build the prototype. Unlike Lisp or PROLOG, C++ is not a traditional expert system language. The advantage of C++ is that it is not only capable of handling operations on symbols, but it is also an excellent numerical processor. Together with Fortran and Pascal, C++ is widely used in numerical analysis softwares. This numerically excellent capability would be a great advantage when it comes to the development of a coupled Expert System which deals with number crunching.

4.2.2. INFERENCING STRATEGY FOR THE OBJECT ORIENTED KNOWLEDGE BASE.

The properties of object oriented programming are very useful for the design of knowledge bases. In addition to the benefits of class inheritance for representing taxonomic knowledge, the objects interaction (message-passing) aspects of object oriented systems simplifies the representation of interacting components.

Having mentioned the good things of object oriented programming in the design of knowledge bases, it must be said that, production system rules are a natural formalism for representing heuristic and problem solving knowledge. The prototype developed as part of the current work combines both object oriented schemes and production rules for constructing the knowledge base. The object oriented scheme represents domain properties while rules reason about them. A simple way to show this relation is by writing a production rules where rules could send messages to objects and test the response in their premises. For example,

```
Rule :
   IF     ( method.analysis_test()    = true ) and
          ( method.constraint_test() = true )
   THEN   ( method.to_consider = true )                    (4-1)
```

The rule tests the object *method* to see if it can do the analysis and handle the constraints. It is the object *method* itself which does the analysis_test and the constraint_test operations. The object will see if it can do the analysis and accommodate the constraints requirements of the current design problem. If the rule premises are satisfied, then the object method will be considered for subsequent tests.

The problem with the above rule is that it is written in a procedural way. This type of programming leads to difficulties in writing the inference engine and the knowledge base as two separate modules. In an Expert System program, the user/programmer should be able to modify the rules (knowledge base) without interfering with the inference engine. A better way to code the above rule is,

```
IF ( analysis_test    = true ) and
   ( constraint_test = true )
THEN ( method_to_consider = true )                         (4-2)
```

The premises variables (analysis_test and constraint_test) have to be instantiated first. The instantiation is performed in an assignment/instantiation routine which sends enquiries to the appropriate object, in this case is *method*, about its capabilities in handling the analysis and constraints required. Functions analysis_test and constraint_test in the object *method* will provide the answers to these enquiries. If the conditions of the rule are satisfied then the conclusion method_to_consider is instantiated to true. For certain information, instead of asking questions to various objects, this assignment routine might ask the users to provide the answers.

From the discussion above it is clear that inferencing is applied to the heuristic knowledge which is represented

by IF-THEN rules while the object oriented domain supplies the information for variable (in the rule premises) assignment. This mechanism is shown in figure 4-2 and is nothing more than an Expert System with a production rule knowledge base which contains the heuristic knowledge. Either forward or backward chaining can be applied to these rules. The distinction is in the fact that additional information is provided by the knowledge relating to the optimization methods stored in an object oriented fashion.

The discussions above show a concept in which a knowledge base consists of domain properties (object oriented scheme in our case above) and IF-THEN rules which reason about them. This concept can also work as efficiently using records or frames to represent the domain properties (instead of the object oriented way).

4.3. CASE BASED REASONING.

Case based reasoning is an AI technique that adapts a solution obtained from past experiences (cases) for solving current problems. This reliance on previous experiences (or cases) is a hallmark of case-based reasoning. The idea of reasoning from relevant past cases is appealing because it corresponds to the process an expert uses to solve new problems.

A case basically contains a list of features which represent the nature of a case and a solution for that case. Any case specific information, such as possibility of case failure if a certain external perturbation is applied, can be added to a case. In case based systems, the case (or cases) itself is used as means of explanation. Hence a case should have adequate information about itself.

Case based reasoning differs from other types of reasoning and problem-solving techniques and is, in some instance, a direct reaction to the problems encountered in a rule-based (production-rules-based) reasoning system. The problems with rule-based systems which prompted a search for an alternative paradigm for problem solving concern knowledge acquisition and robustness:

(1). Knowledge Acquisition. It is known in rule based systems that collecting knowledge and encoding it into an IF-THEN form is difficult. It is often argued that experts do not actually use rules, it is their experience that makes them expert. An additional complication in setting-up rules is that it is necessary to trace interactions between rules to ensure that they could chain properly and that contradictions are eliminated.

(2). Robustness. Since all their knowledge is recorded in terms of rules, if a problem does not match any of the rules, the system could not solve it. In this context the rule based systems are brittle.
In fact it is possible to regard this third problem as a problem which is related to the knowledge quantity and also applies to other type of reasonings.

Case based reasoning offers advantages over rule-based system in the following ways:

- **Knowledge Acquisition**. Cases are more memorable than abstract rules. It is often easier for experts to remember and articulate specific examples of the problems and their solutions to those problems, than it is for them to describe their problem solving technique in terms of potentially large number of rules.

- **Learning from Experience**. The learning process is a natural mechanism in case based reasoning. Each time the system solves a problem, that problem and its solution are stored in memory as a new case. In this way, the case based systems progressively has more capacity in dealing with more situations.

- **Adaptivity**. The case based reasoning adapts the solution of similar past cases to solve new problems.

While adaptivity is regarded as one of basic steps in case based reasoning algorithms, in fact we regard this as one of the most difficult parts. If an exactly similar case can be found then the solution of that case can be directly applied for current problem. Problems occur if no exactly similar case is available. Generating an adaptation algorithm is not easy and must include a few rules from its rule-based "competitor". Pure case based reasoning will be simply lost if there is no case found to be similar. Thus, case based reasoning should be regarded as a complimentary technique to the rule based reasoning and both are implemented in this research.

The basic processing cycle of case based reasoning systems is "input a problem, retrieve similar past cases, adapt the selected case solution to the current problem, and store the new case along with its solution in "memory".
The above cycle is elaborated as follows:

(1). Input a problem. Upon input of a problem, the key features are extracted for use in retrieving cases with similar features. These features are called indexes.

(2). Retrieve similar past cases. The indexes are used to retrieve cases from the memory and the process of case retrieval is called case-indexing. A commonly used case-

indexing is the so called nearest-neighbour [Kolodner 1993]. The nearest-neighbour approaches let the user retrieve cases based on a weighted sum of features in the input case that match cases in memory.

(3). Select most relevant case(s). The similar cases retrieved are ranked and the one with highest rank available to be selected.

(4). Adapt the case solution for the current problem. If the current problem is the same or nearly the same as the selected case then the solution can be directly applied. However, often this is not true and the solution needs to be altered for the current problem.

(5). Update memory. The current problem and its successful solution is stored as a new case in memory.

With regard to our optimization problem, there are several questions which need to be answered:

1. What are the relevant features for any design case?
2. How can the case be described to the computer?
3. How to represent interdependency among features?
4. How to judge similarity?
5. How to address the ambiguities in similarity?
6. What if there is no case found to be similar?

These problems will be discussed in the next chapters.

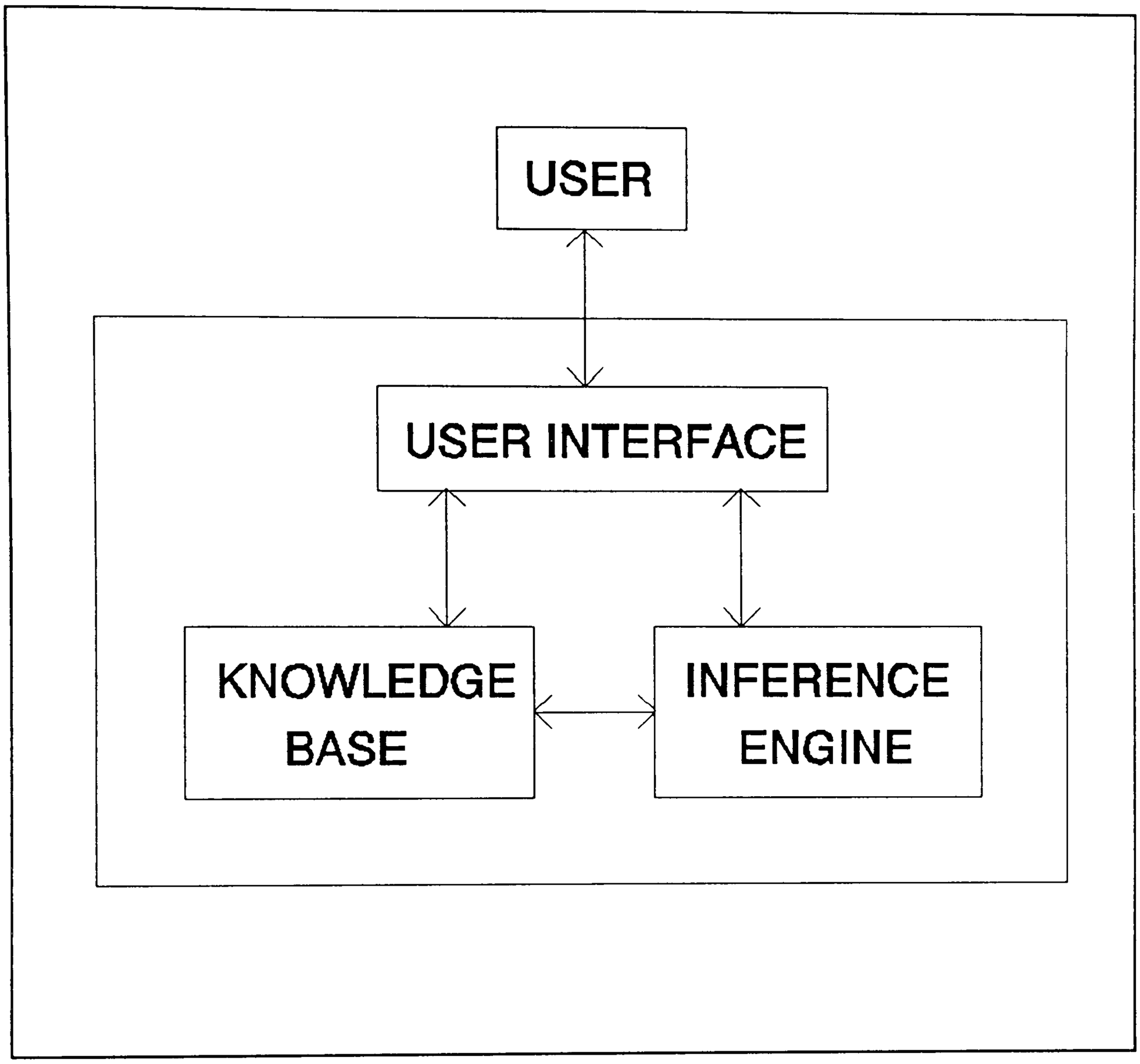


Figure 4-1 The architecture of a typical Expert System.

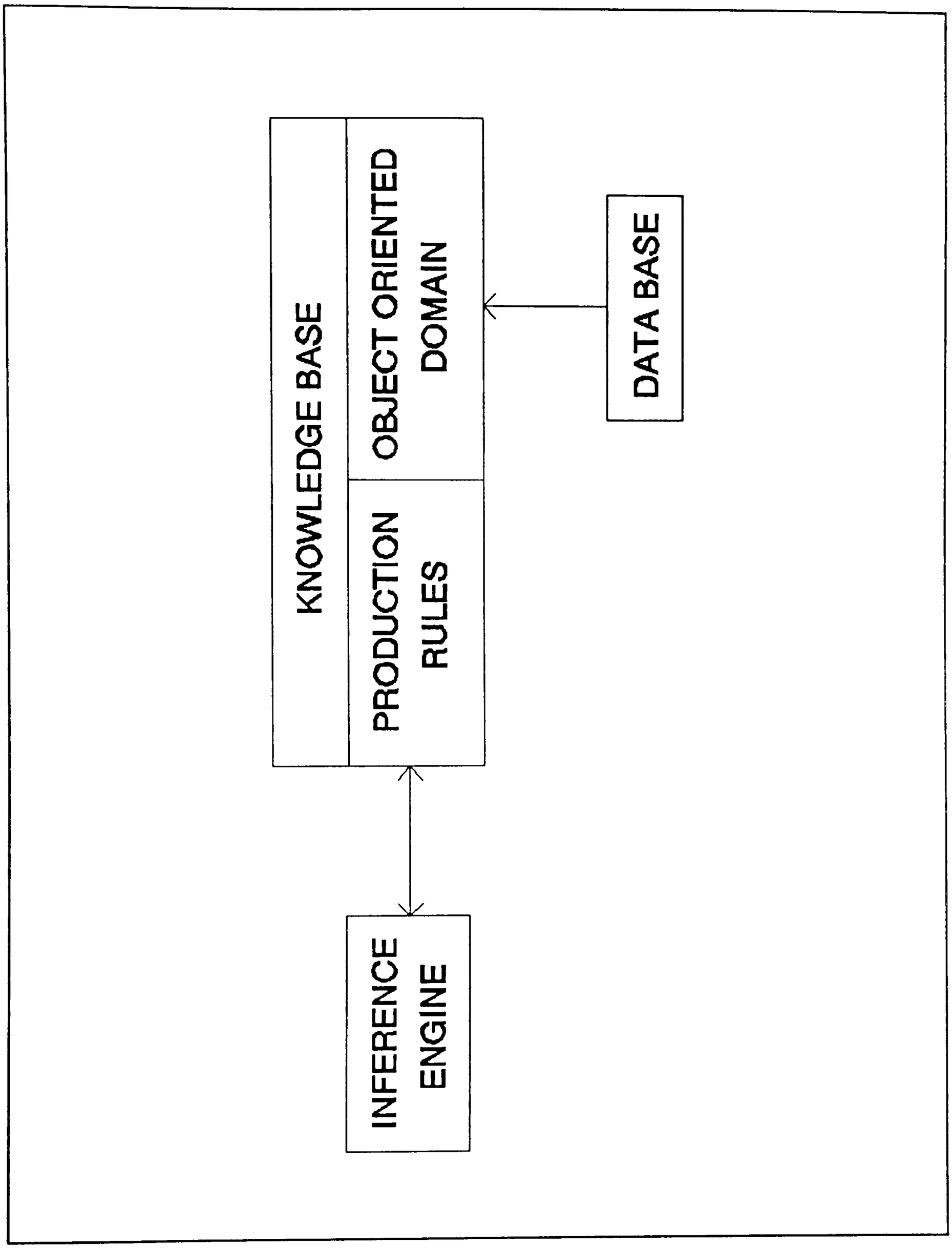


Figure 4-2 Inferencing strategy for the object oriented knowledge base.

CHAPTER 5

REASONING THROUGH PAST EXPERIENCES

5.1. STRUCTURAL OPTIMIZATION KNOWLEDGE.

For a structural optimization system to be effectively used the users must be able to set up the data to run the system efficiently, analyze the behaviour of the system during the running of the solution algorithms, and interprete the results. Setting up the system requires knowledge of the optimization requirements and can exploit past experience, where available. In addition to setting up the optimization the user must be able to monitor the performance of an algorithm at run-time in order to decide if a solution is being approached and, at termination if the result is satisfactory. Thus three types of knowledge can be identified as necessary to effectively use such systems:

1. Knowledge relating to setting up the structural optimization.
2. Knowledge which is based on past experiences.
3. Run-time and results interpretation knowledge.

Both the first and second knowledge types have the task of setting-up the optimization. However, the first type of knowledge is based solely on logical deduction and is a-priori in nature, while the second type of knowledge is based on previous experiences (past cases).

When the users input the design requirements/specifications, the system will consult the system past experience. If there is a similar case found in the memory then the solution for that case is proposed. If no similar case is found, then a solution strategy is formulated with whatever logical knowledge (a priori) is possessed by the system and is described in section 6.1. Once the optimization input data has been assembled the system initiates a solution run. Control is then handed over to the structural optimization program but the performance is monitored by the 'run-time' system. This assesses the effectiveness of the optimization run and intervenes as required.

This chapter looks into the detail of the knowledge based on past experiences. The a-priori and results interpretation knowledges are discussed in the next chapter.

5.2. KNOWLEDGE OF PAST EXPERIENCE.

As a supplement to apriori knowledge it is possible to appeal to past experience. The knowledge now is based ex-

plicitly on the outcome of previous optimization runs. The basic idea behind this approach is to try to capture the past design experiences (previous optimization runs) and use it to solve current problems. Just as human beings try to remember what has been done to solve a similar problem in the past, the Expert System will search for a similar case in its memory and uses the optimization set-up of that case for solving the current design problem. This kind of approach of solving new problems by adapting solutions that were used in previous cases is called case-based reasoning (CBR).

There are advantages of using the case-based reasoning. The most obvious one is the availability of a memory which contains previous cases and their solutions. The case-based reasoning systems are built on a memory of prior cases. Each time the system solves a problem, that problem and its solution are stored in memory as a case. In this way CBR systems can easily learn from experience; they do not have to waste effort resolving a similar problem which has been solved before. Storing a new case in memory can be regarded as a means of knowledge acquisition.

The case representation and similarity aspects depends on the nature of the problem at hands which in this case is structural optimization. Before discussing the structure of the past experience knowledge and its reasoning mechanism, the nature of setting up structural optimization design job will be described first.

5.2.1. THE NATURE OF SETTING UP A STRUCTURAL OPTIMIZATION DESIGN JOB.

Setting up an optimization job depends primarily on the design requirements. The type of analysis and constraint requirement directly affect the selection of optimization techniques. While a static case with stress constraint can be handled with fully stressing method, the addition of more complex constraints such as displacement or frequency will require more advanced optimization techniques. Some cases might require a combination of optimization techniques.

As the Finite Element method is used in the analysis stage of the optimization, Finite Element models are need to be prepared. The models are represented among other things by the number of nodes or grid points, number of elements, and type of elements. The FE model affects the selection of optimization algorithm as some elements require gradient information.

If the user decides to employ a large number of design variables this may require many optimization iterations

before the optimum point is found. For methods which needs constraint gradients, the selection of a large number of potentially active constraints will give rise to a requirement for an unacceptable time to locate an optimum. In both of these situations it might be necessary to limit the number of iterations or start with a non gradient based method. Whilst the standard convergence criteria, based on changes in direct parameters (e.g. design variables) from iteration to iteration have no influence on the selection process, the use of complex criteria could be influential. For example, the use of dual bonding criterion requires gradient information which is not required by a fully stressing method.

The above description points out that the type of analysis, constraint requirements, the Finite Element model, and the set up of design variables are important factors/parameters in setting up an optimization. This also means that, in the context of reasoning through past experience, those parameters should be considered in recalling the useful past cases. The influence of analysis and constraint are very obvious. Two designs with different analysis or (and) constraint requirements most probably need a different optimization approach. For example, a structure under static loads with stress constraint might be optimized using the fully stressed design method. The method can not be used for that same structure if the constraint set is changed to include displacements.

Whilst a change of analysis or constraint has clear implications for the algorithm selection the situation is more confused in the case of design variables numbers. Suppose for example, two structures have the same analysis and constraint set and the FE model is similar (which implies similar node and element parameters) but have a different numbers of design variables. If the first structure has (say) 20 variables and the second structure has 19 variables then it is logical to say that the same algorithm can solve both problem. But if one structure has 100 variables and a second 10 variables only, these two structure might need different solution approaches. The question of which number of design variables can be said to be similar and which ones are not remains, for the moment, an open question. We may also note that two problems may have the same number of design variables but number of elements within these variables and their pattern may be completely different.

The striking nature of structural optimization which might cause problem is that a slight change in design requirements might produce unexpected results. For example, consider a static case with a single displacement constraint. This structure normally can be optimized with the optimality criterion method without problem. However,

an addition of another active displacement constraint in design requirement might make the optimality criterion method ineffective to optimize the structure. Another example is in the dynamic case with a frequency constraint. An algorithm can be used to minimize the structural weight while keeping the natural frequency above, say, 10 Hz. If the frequency limit is increased, say, to 40 Hz, the same algorithm might not be able to meet the requirements. The reason probably is not caused by the effectiveness of the algorithm used but by, among other things, the nature of the structural characteristics which makes it impossible to have natural frequency as high as 40 Hz.

It has to be stressed here that the parameters mentioned above are not completely independent. For example, in a dynamic case, the number of degrees of freedom is important factor to consider in setting up the optimization. However, the number of degrees of freedom is not critical in a static case with stress constraints. The number of design variables is also more critical in non static cases than the static ones.

The complicated nature of structural optimization as discussed above makes the nearest neighbour techniques [Kolodner 1993] which are commonly used in case based reasoning systems unsuitable in dealing with structural optimization. A different reasoning strategy needs to be implemented.

5.2.2. KNOWLEDGE STRUCTURE AND SIMILARITY ASPECTS.

As mentioned earlier, the basic idea behind reasoning through past experience is to try to capture the past design experiences (previous optimization runs) and use it to solve current problems. In this case the system will search for a similar case in its memory and uses the optimization set-up of that case for solving the current design problem. To find a similar case to the current design problem, it is necessary to define similarity. This requires selecting criteria so that a specific design case can be categorized as similar to the current design problem. These criteria need to be expressed in terms of parameters which need to be defined to judge the similarity between the current design problem and the past design cases in the memory. The selection of these parameters is very important as the matching of the current design with a previous one is based on them exclusively. In CBR this procedure of extracting features to use in cases comparison is called index extraction where the indexes in this report are called the similarity parameters.

From the description on the nature of setting up an optimization strategy, the nearest neighbour strategy is

regarded as not suitable. Rules (IF-THEN rules) are more natural to represents the complexities and interdependency among the affecting parameters in setting up optimization. These parameters are adopted as the similarity parameters. Thus by using these similarity parameters, similarity rules can be defined. Using this approach a current design problem is defined as similar to a design case in the system memory if the similarity rules are satisfied.

The requirements for the similarity parameters is that their contribution in setting up the optimization should be significant and they should be able to define the uniqueness of a design case. Currently these relate to:

1. **Analysis**: statics, normal modes, dynamics, and combinations.

2. **Constraint**: stress, displacement, strain, buckling, natural frequency, dynamics, and combinations.

3. **Node number**: the number of nodes in the finite element model.

4. **Number of element**: the number of elements used to model the structure.

5. **Element types**: plate/membrane, beam, rod, and combinations.

6. **Number of design variables**: the number of design variables and the linkage pattern.

Design cases which the memory stores are treated as a collection of knowledge entities (past experiences) in the form of records. Each record basically contains a design case and its solution. These are expressed in terms of similarity parameters together with their values and the design solution for that particular case. A design solution basically is a definition of an optimization set-up which consists an optimization algorithm (an optimization method or methods combination), number of iterations, and convergence criteria. If the current design problem and a past design case are considered as similar, then this optimization algorithm is proposed for use.

Due to the nature of structural optimization problems, which has been discussed in 5.2.1, the use of past experience introduces certain ambiguities because the similarity concept does not, necessarily, gives rise to hard and fast rules. For example it is difficult to state with certainty that, two structures are not similar because one has 10 design variables and the other one 15 design variables and hence they need different optimization strategies, because it still might be possible that these two design problems

can be optimized effectively with exactly the same strategy. Similar ambiguities may arise in different situations, for example, in the case of two structure which have the same analysis and constraint requirements but a different number of nodes and elements.

Although there are ambiguities with regard to the similarity parameters, there are still some basic principles which can be followed in deriving the similarity rules. These principles are as follow,

1. **Analysis.**
   If the current design problem has the same analysis requirement as a design case in the memory record, then they are similar in analysis.

2. **Constraints.**
   If the current design problem has the same constraint requirement as a design case in the memory then they are similar in constraint.

   If the system is not able to find in its data base a design case with the same analysis and constraint set as the current design problem there is only a small possibility that any solution from the design cases in the memory can be used for the current problem.

3. **Node number.**
   If the current design problem has fewer nodes than the target one in the memory record then, from node number point of view, the solution in that record is acceptable for the current design.

4. **Element number.**
   If the current design problem has fewer elements than the target one in the memory record then, from element number point of view, the solution in that record is acceptable for the current design.

5. **Element type.**
   If the current design problem has same types of element as the one in the memory record then, from element point of view, the solution in that record is acceptable for the current design.

6. **Design Variable number.**
   If the current design problem has fewer design variables than the one in the memory record, currently being considered as similar, then the solution in that record is acceptable for the current design.

The rules related to the conditions (1) and (2) are deterministic in nature while the rules related to the conditions (3) to (6) are more fuzzy. If, in addition to

conditions (1) and (2) the rules related to conditions (3) to (6) are satisfied then the solution (an optimization set-up) in the record can be used. But, if those rules are not met it does not necessarily mean that the solution used in the earlier problem will fail for the current design problem.

As mentioned previously, the similarity parameters mentioned above are not completely independent and this to be taken into account in making judgments on similarity. A reasoning technique based on the production rules making use of the concept of certainty factors is implemented. Using the concept of case similarity, based on the similarity principles mentioned previously, the reasoner proposes an optimization strategy from a similar case for use.

The judgment whether cases are similar could not be based on statistical observations. We consider that this type of decision making relies more on the expertise (beliefs) of the individual making the decision. It is inappropriate to say, for example, that the prior probability of a case to be statics is 0 (no static cases ever found), or 1 (all structural cases found are statics). It is for this reason that in this research the concept of certainty factors is preferred to use than Bayesian conditional probability.

The similarity rules are written using IF-THEN rules with certainty factors attached to them. Each certainty factor (CF) express the level of user confidence (or belief) with regard to the trueness of the rule a CF is attached. These CFs can also be regarded as weight factors.

5.3. THE STRUCTURE OF DESIGN CASES.

As described above the basic idea is to try to capture design experience in order to be able to solve future problems. Thus design experience is codified into knowledge stored in the memory which can be regarded as a file containing a collection of knowledge (past experience) records. Each record contains a (past) design case and its solution, held in the form of the similarity parameters with their values and the successful optimization strategy for that particular case. The similarity rules are written in the form of production rules (IF-THEN rules) and are used to capture the design case record which is similar to the current design problem.

Each record in the knowledge base contains the following information (stored in fields),

- number of analysis,
- analysis types,
- number of constraints,
- constraint types,
- number of nodes,
- number of elements,
- element types,
- number of design variables,
- solution,
- comments.

Thus a typical record might be,

- number of analysis: 1;
- analysis types: statics;
- number of constraints: 2;
- constraints types: stress, displacement;
- number of nodes: 5;
- number of elements: 6;
- element types: rod;
- number of design variables: 6;
- solution: pseudo-newton algorithm, **n** number of iterations, a type of convergence criteria ;
- comments: a straight forward process ;

In addition to the above fields, each case might store additional information which is unique to that case, for example storing the associated structural drawing.

If the number of cases is small then the time spent for case searching and retrieval is unimportant. However it is inevitable that after many runs the number of cases stored becomes increasingly huge. Hence it is necessary to structure the cases to reduce the length of time spent for cases searching and retrieval. For our system it is proposed to classify cases as a tree structure. Cases are grouped according to the analysis type. The search will not go further in a group of cases if the analysis required by a current design problem does not match the analysis type of that group.

5.4. SIMILARITY RULES.

The contents of the rules are primarily based on the basic principles mentioned in (5.2) and the fact that similarity parameters are not completely independent has to be included in the rules.

A certainty factor is attached to each rule and reflects the designer's confidence in his rule. Value of CFs range from -1 to 1. As a CF approaches 1 the confidence in the rule is stronger; as CF approaches -1 the confidence

against the rule gets stronger; and a CF around 0 indicates that there is little evidence either for or against the rule.

Each rule consists of an IF-THEN statement and a certainty factor CF. For example, the rule for similarity parameter relating to analysis, is in the form of:

IF *(analysis type of a design case)* = *(analysis type of the current design problem)*
THEN *analysis_similar is TRUE*
CF=0.15

The above rule simply says that, if the current design problem analysis requirement is the same as to the analysis type of a previous design case then the design is "similar" from analysis point of view. The designer's confidence on his rule towards the similarity of a case is shown by the CF value, in this case is 0.15. The opposite of the above analysis rule,

IF *(analysis type of a design case)* $\neq$ *(analysis type of the current design problem)*
THEN *analysis_similar is TRUE*
CF=-0.6

The rule says that, if the analysis required in current problem is different to the analysis of a previous design case then the design is unlikely to be similar (CF=-0.6) from analysis point of view.

As can be seen from previous analysis rules, rule premises have no CF values attached to. All premises are deterministic in nature. There is no chaining process during reasoning hence there is no CF propagation from a rule to another rule. The CF value contributes toward a final CF which is a combination of various CFs. The positive and negative values of CFs are judged and determined separately, unlike statistical probability where the sum of a probability and its negation is one. If fired, the second rule above will most probably kill off the chance of a case being similar, which is in accordance to the similarity principles discussed in (5.2).

Another example is the rule relating to the design variable which has the form

IF *(number of variable of a design case)* $\geq$ *(number of variable of the current design)*
THEN *variable_similar is TRUE*
CF=0.1

The rule states that, if a design case has the same or greater number of design variables than the current design

problem then that design case is acceptable for this parameter. The contribution of design variables towards the similarity of a case is rather small (0.1).

The parameters dependency can be shown by another rule which also deals with number of variables,

IF *(number of variable of a design case)* <
(number of variable of the current design)
AND *(analysis=static)*
THEN *variable similar is TRUE*
CF=0.05

From the rule it is clear that the logic of previous rule is not met, the current design problem has a larger number of variables than the design case. Due to the fact that the analysis is statics, when fired, this rule will not kill off the chance of a case to be regarded as "similar". This illustrates the relative importance of an analysis type to number of design variables with regard to the use of an optimization strategy.

From the discussion above it is clear that each similarity parameter is represented by more than one rule. During the reasoning process, for each similarity parameter, only one rule which represents the true current situation will be fired. The goal of the reasoning process is deciding whether a case is, based on similarity rules, acceptable (or unacceptable) and the confidence in the decision taken. The acceptability of a case is based on contribution of various similarity parameters and, hence, on the combination of CFs of the fired rules. This means that CF values play an important role in deciding whether a case could be regarded as similar. The CF value which is given to a rule should be based on the concept that that value at the end must support what the experts belief as true. In our problem, similarity, a case should be considered of having a chance to be similar if the criteria related to the analysis and constraint are satisfied; on the other hand a case is not similar if the same criteria above not satisfied. Figure 5-1 shows the structure of the past experiences knowledge base which contains both the similarity rules and the past cases.

Now it is necessary to calculate the contribution of various CFs toward a total CF value. This is identical to the problem of combining multiple CFs when two or more rules support the same result. Suppose CF(R1) is the certainty factor associated with the rule R1, and CF(R2) is the certainty factor associated with the rule R2; then the CF combination of both rules is calculated by [Luger and Stubblefield 1989] :

$$CF(R1) + CF(R2) - (CF(R1) \times CF(R2))$$ when CF(R1) and CF(R2) are positive

$$CF(R1) + CF(R2) + (CF(R1) \times CF(R2))$$ when CF(R1) and CF(R2) are negative

$$\frac{CF(R1) + CF(R2)}{1 - MIN(|CF(R1)|, |CF(R2)|)}$$ otherwise.

5.5. CASE SELECTION.

Faced with a problem, the task of a reasoner is to produce a solution by manipulating the knowledge to derive an effective solution strategy. From previous discussion it is clear that the knowledge structure consists of IF-THEN rules which addresses question of similarity and cases. The objective is to find an acceptable case in the memory and use its optimization set-up for solving the current problem.

The reasoning process is controlled by a case selector. The selector starts the case selection by retrieving a case from memory. The similarity rules are applied to evaluate the acceptability of the case. A confidence scoring based on the CF calculation in section 5.4.1 is performed to find the CF of the case. This mechanism is shown on figure 5-2. The cases with 'acceptable' CF values (higher than 0.27) are sorted using the bubble-sort technique and presented to the user for consideration.

The threshold value, 0.27, is selected because this number represents a 'reasonable' level of belief in similarity. In fact this number is slightly less than CF combination of analysis and constraint rules when both are satisfied (a past case has the same types of analysis and constraint with the current problem), which is 0.2775. This was chosen to allow minor adjustments due to the influence of other parameters.

However, due to the nature of structural optimization, there are situations in which the similarity-strategy proposed in this chapter could not cope well. Consider an optimization of a structure with natural frequency as the constraint and assume that only the first mode is considered. The constraint requires that at optimum the natural frequency of structure is higher or equal 20 Hertz. Suppose that in the system's memory, there is a design case which is exactly similar to the current design problem

(same structure, FE model, type of analysis, constraint, etc.), except that the case in the memory is required to have a natural frequency higher or equal 10 Hertz. Using our case based system with the strategy discussed above the optimization set up of the design case mentioned will be proposed with a high CF value. Still, there is no guarantee that the optimization will go to convergence in this case. There is possibility that due to the nature of structural configuration, it is impossible to achieve a natural frequency value of 20 Hz by resizing of structural components alone. Hence the optimization will not reach convergence. In the Expert System developed in this research, the task of finding the reasons of non-convergence in the situation illustrated above is given to the backend modules and the user will be given advice on modification required.

The final CF depends on the CF values in the similarity rules and these values are subjective in nature. The system's users are able to interrogate the system as to the system's decision on case selection and its level of confidence. It is important that users have control over the importance of a rule (represented by its CF) and can modify the CF values if appropriate.

As previously mentioned, the cases are structured according to a tree structure. The tree's main branches are determined according to the analysis types used. The search at any specific time is performed only in one tree branch. In other words a search will go deeper only in the branch representing the required analysis type. This is due to the analysis similarity parameter being regarded as a hard rule. Furthermore, the case selector will look for the cases with identical constraint type first. By using these measures the search can be channelled toward the most relevant cases only.

5.6. MEMORY UPDATE.

After a successful optimization run, the memory is updated by storing the new case. An important aspect is where to put this case in memory. As previously mentioned, the cases are structured as a tree with analysis types as the main branches. A new case will be stored in the slot with the class of similar analysis types.

The updating process also must include the "experience" of a case, especially the difficulties or problems a case encountered during an optimization run. Not every optimization process can run smoothly. Even if the optimization strategy itself is correct, there are still other sources of mistakes such as vibration restriction impos-

sible to achieve or constraints highly dependent on each other. For case based systems which rely on features comparison, it would be difficult to predict if an optimization run would end with any failures mentioned above. However, each type of structural case usually is associated with certain types of mistake possibilities. For example, an optimization of a static case with stress and displacement constraints might not go to convergence if the constraint values are mutually contradictory. Such mistake tendencies can be stored in the case in the comment field and serve as a reminder to the user not to repeat the same mistake.

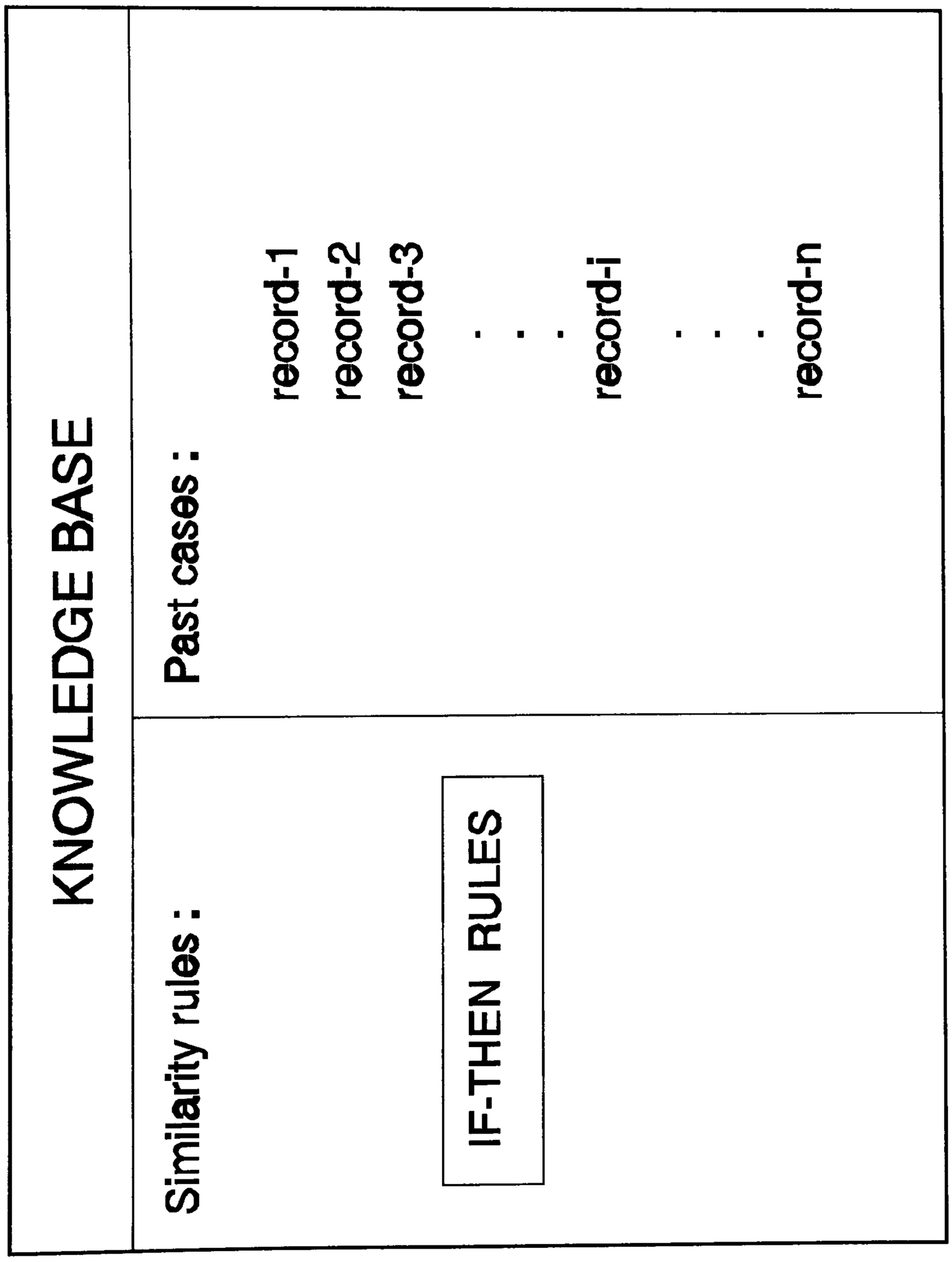


Figure 5-1 The structure of the past experience knowledge.

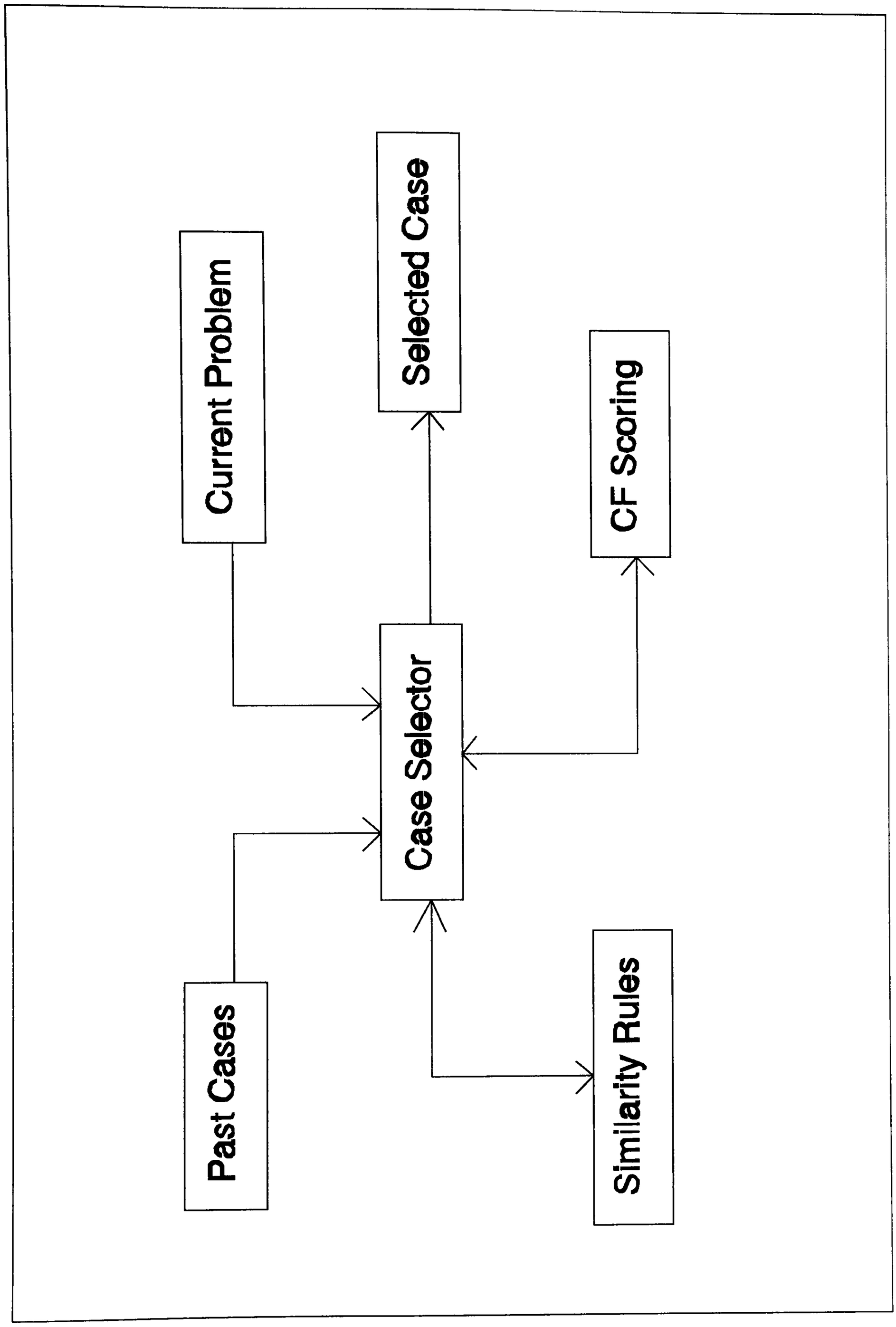


Figure 5-2 The process of case selection.

CHAPTER 6:

THE REASONING OF A-PRIORI AND RESULT INTERPRETATION KNOWLEDGE

As indicated in earlier chapters, for a structural optimization system to be effectively used the users must be able to set up the data to run the system efficiently, analyze the behaviour of the system during the running of the solution algorithms, and interpret the results. Setting-up the optimization run can exploit past experiences, and this has been discussed in detail in chapter five. In this chapter, another way of doing the set-up, through a-priori knowledge, is discussed. Also presented in this chapter is the discussion about monitoring the optimization run and interpreting the result.

6.1. KNOWLEDGE FOR SETTING UP THE STRUCTURAL OPTIMIZATION PROCESS.

6.1.1. THE PHILOSOPHY.

The knowledge being discussed here contains the information required to set up and efficiently run an optimization system. This basically consists of knowing which optimization technique (or their combination) to employ, the number of iterations, and the convergence criteria. The selection of the optimization strategy depends on the nature of the structural problem and the design requirements/specification.

The structural design specification generally consists of the following:

1. Type of structure (truss, stiffened panel, or others).

2. Type of analysis (static, dynamic, or others).

3. Constraints required (stress, displacement, or others).

4. Gauges limits.

5. Number of load cases.

6. Finite element model.

Once the specification is complete the selection of the solution algorithm is directly influenced by the decisions made at this early stage of the optimization process.

The parameters which affects the algorithm selection are:

1. **Analysis required** (static, dynamic, or others).
   Some methods are better suited to certain type of analysis, for example fully stressed design is appropriate for certain static analysis problems but will not work if a dynamic analysis is required.

2. **Constraints required** (stress, frequency, or others).
   Not every optimization method can handle all types of constraints. A Pseudo Newton method might work for a variety of constraints, but fully stressed design is appropriate for stress constraint only. In addition, for certain constraints such as flutter, their incorporation greatly augments the analysis requirements.

3. **Type of finite elements used** (membrane, beam, shell, or others).
   For some elements obtaining analytic gradient information is difficult, hence these are suitable for certain optimization algorithms only unless numerically derived gradients are used. The use of complex elements is not precluded but many optimization systems do not always allow the use of other than simple elements.

4. **Number of load cases.**
   The number of load cases affect the selection of optimization methods insofar as it influences the size of the problem. On this basis a large number of load cases might invalidate the use of gradient based methods which inflate in size in proportion to the cases analyzed.

5. **Number of design variables.**
   A problem which occurs with a large number of design variables requires many optimization iterations before the optimum point is found. For methods which need constraint gradients, the time required to locate the optimum might be unacceptable. In this situation it might be necessary to limit the number of iterations or start with a non gradient based method.

6. **Convergence criteria required.**
   Whilst the standard convergence criteria, based on changes in direct parameters (e.g. design variables) from iteration to iteration, have no influence on the selection process the use of complex criteria could be influential. For example, the use of dual bounding criterion requires the use of gradient information which is not immediately available when using a fully-stressing method (say).

The above provides an outline of the type of knowledge which allows the user to match the design requirement with

the capabilities of the optimization. The knowledge is not based explicitly on information obtained from previous runs but is derived by deducing the influence which the design model has on the selection of a solution strategy. Because it is based on logical deductions and not on past knowledge it is a priori in nature.

6.1.2. THE KNOWLEDGE STRUCTURE.

The set up knowledge base consists of production system rules and an object oriented scheme. The object oriented scheme is used to classify the optimization methods into objects with each method represented as an object. These objects contain information concerning the object capabilities and operations/functions to test whether a particular object can handle the design requirements. The production rules such as (4-2) are used to represent the heuristic and problem solving knowledge.

The object oriented domain has (currently) a simple scheme with a top class, procedure which has a child class, method. Procedure is a base class which contains a set of parameters and functions. These parameters and functions are inherited by method. Class method also has the constructor function for object instantiation. The objects instantiated by method are the optimization methods. The parameters in procedure are,

- **name**, name of optimization method,
- **analysis**, type of analysis which can be handled by a method,
- **constraint**, type of constraints which can be handled by a method,
- **element**, type of elements suitable for a method,
- **convergence**, type of convergence criteria which can be handled by a method,
- **Lagrange**, indicating the method's capability to produce Lagrange multiplier.

The purpose of class functions is to test whether an object (a method) can handle the design requirements. This is performed by comparing the design requirements with object capabilities. The functions required in procedure are,

- analysis_check(), to test whether an object can handle the analysis requirement,
- constraint_check(), to test whether an object can handle the constraints requirement,
- element_check(), to test whether an object can accept the types of elements used in a finite element model.

Currently there are five objects to be instantiated. These objects are optimization methods which can be run by

the structural optimization system, which, for the present report is taken to be STARS. In this case the methods are fully stressed design, optimality criteria, Pseudo Newton, and their combinations, fully stressing followed by optimality criteria or fully stressing followed by Pseudo Newton. The data/information of each object (optimization method) is read from a data base. For five objects there are five database files, one file for each object. The user can modify the data. More methods (for example from other optimization package) can be added. Each data base includes information such as name, analysis capability, constraint capability, convergence criteria, and Lagrangian information.

Figure 6-1 shows the knowledge base structure of the knowledge required to set up the optimization algorithm.

6.2. RUN-TIME AND RESULT INTERPRETATION KNOWLEDGE.

This is the knowledge which is concerned with aspects such as,
- deciding if an algorithm is heading for optimum,
- deciding that an optimum has been reached,
- the actions required if the above situations are negative.

Whether an optimization run is heading for an optimum usually can be predicted by studying the optimization trend. An extrapolation can be derived with respect to the convergence parameters on the change in structural weight or the gap between the actual weight and a lower bound. If an algorithm seems not to be heading for an optimum then the run-time knowledge should be able to explain the reason why it is not able to reach the optimum and give an alternative solution strategy if any exists. Examples of optimization routines having difficulties in locating a solution are numerous and occur, for example, in cases where vibration restrictions are impossible to achieve, or where the constraints are highly dependent.

In order to predict whether or not an optimization run is heading for an optimum we suspend the run after three or four iterations and studying its trend. An appropriate procedure is to extrapolate the gap between feasible weight and lower bound using the dual bounding process. For methods which do not produce a lower bound on the optimum weight, such as FSD, the gap between the actual and feasible weight is used. If it is predicted that an optimization run is not converging then this fact is presented to the user. However, the information presented has to cover a range of possibilities relating the failure, or otherwise, of the solution strategy. Some of these are related to the

form of the design problem itself and some are more algorithm based. It is worth noting impact that the user has when formulating the design problem is very significant in this area.

An optimum is reached when the convergence criteria are met and constraint set satisfied. This is known only after an optimization run stops, either because the convergence criteria are satisfied or the maximum iterations are reached (both do not, however, necessarily mean the optimum is found), and results produced which need interpretation. Successful results need to be explained to the user and offered to the 'past experience' data base but the interpreter needs to be able to explain why an optimization run does not produce an optimum result.

In the case where an optimization run does not result in an optimum, this is defined as a failure which can be associated with three situations,

- failure to go to convergence,
- false convergence, and
- system crash.

A 'failure to converge' means that an optimization run is predicted not to go to convergence or that it does not meet the convergence criteria after the maximum iteration is reached. False convergence is basically also a failure to converge. As an example consider the case of an algorithm stopping because there is no further change in mass which gives the appearance of a converged solution but the constraints are not satisfied. The causes of the such a failures cases are not easy to categorize. While there is theoretical base to trace the cause of convergence failure and false convergence there are situations where the failures are optimization software and machine dependent. The cause of a problem for one optimization software system might not be problem for an alternative optimization packages.

A system crash failure is definitely optimization software and machine dependent. For example a structure with a large number of members subject to stress constraints might give rise to a large number of stress constraints. This large number might exceed the constraint array setting for certain software and lead to a system crash. Another source of system crash, error in coding the driver files, has been very much reduced by developing modules which have the task of writing driver files.

The above situations, which lead to difficulties in categorizing and drawing-up the taxonomy of the knowledge base, make the use of production rules a natural choice for the knowledge base scheme. The information in the knowledge

base is based upon failures encountered during running STARS for various optimization design problem.

6.3. THE REASONER.

Faced with a problem, the task of a reasoner is to produce a solution by manipulating the knowledge to derive an effective solution strategy. It also has an important role in generating explanations for the user. This is done by means of an inferencing process. In the case of setting-up an optimization run by using the apriori knowledge, the reasoner produces a solution in the form of a definition of an optimization set-up. In the case of either a runtime situation or when apparent convergence is reached, faced with a failure, the task of the reasoner is to produce explanations why the failure occurs.

In this prototype the reasoner is based on forward chain inferencing (data driven reasoning) with the production rules (IF-THEN rules). The reasoning begins with the given facts of the problem which is the design requirement, and a set of rules. It searches for a solution by applying rules to facts to produce new facts which are, in turn, used to produce more new facts. This process continues until the goal is found. Data driven reasoning was selected, as opposed to a goal driven reasoning (backward chain), because:

1. **Many of the data are already available as the design requirement**. This is especially true for reasoning to produce a definition of an optimization set-up where all data required is available in the form of the design requirement and domain knowledge.

2. **It is difficult to form a goal**. In an optimization set-up it is difficult to predict values of the parameters used in an optimization set-up. With regard to the run-time and result interpretation problem it is also difficult to judge initially the reason for an error or failure.

The same reasoning/inferencing technique is used for a-priori, run-time, and results interpretation knowledge bases. This is possible because the inferencing is applied only to the production rules (IF-THEN rules) of all the knowledge approaches. However some of the data required by the reasoner is supplied from a different source:

1. The reasoning process for defining the optimization set-up with a priori knowledge requires information relating to the design requirements and objects (optimization methods) capabilities. While the design requirement (stored

in object *design*, see chapter seven) are specified by the user, the information concerning the optimization methods is provided by the object oriented domain. Each method (or methods combination), represented as an object, is tested one by one against the rules. The goal of this reasoning process is to generate an appropriate optimization set-up (differences lie in the method being used) which can be used to solve the design problem. Figure 6-2 shows the diagram of this process.

2. The goal of the reasoning process for the run-time and result interpretation knowledge base is the explanations associated with any run failure. The reasoning process requires data related to the optimization result and design specifications. The optimization results can be supplied by the users or automatically via interface to the optimization package. Figure 6-3 shows the diagram of the reasoning process of result interpretation. The diagram for the run-time process is similar.

The knowledge and reasoning mechanisms required in the Expert System for Structural Optimization have been discussed in chapter five and this chapter. In order to validate the concepts, it is necessary to assemble the required knowledge and programming requirement, and to construct the prototype. The prototype construction is discussed in the next chapter.

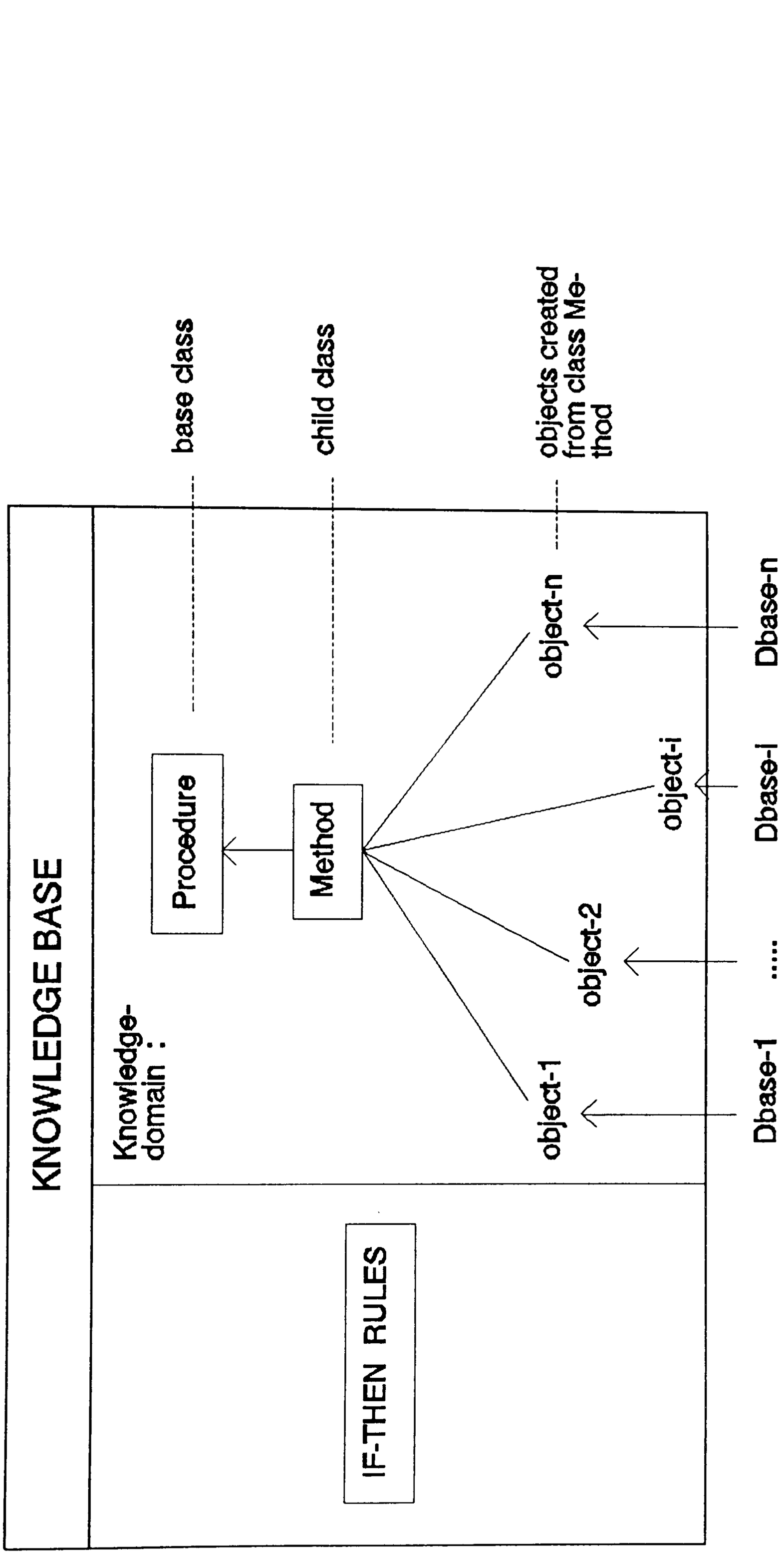


Figure 6-1 The structure of a-priori knowledge.

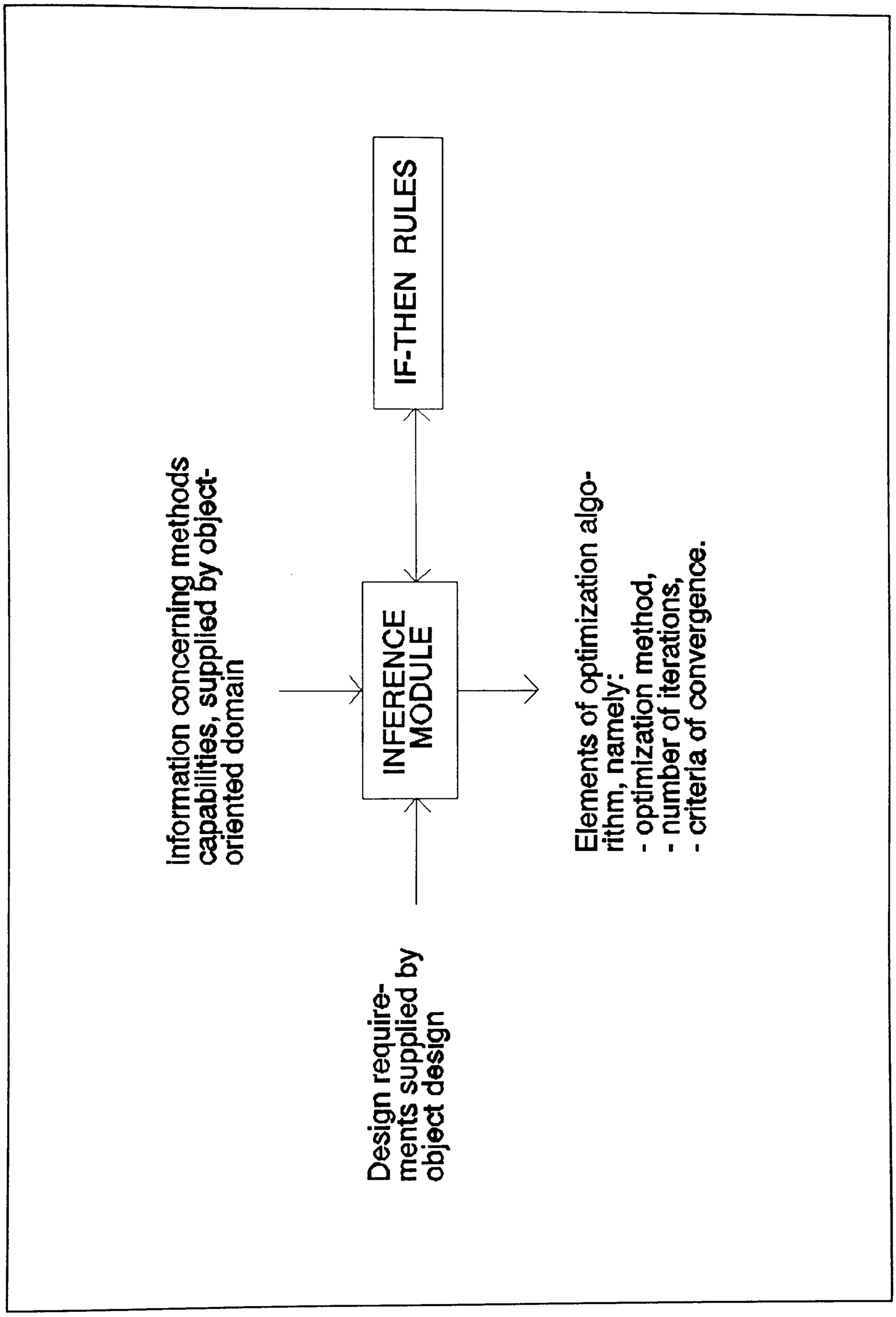


Figure 6-2 The process of the a-priori knowledge reasoning.

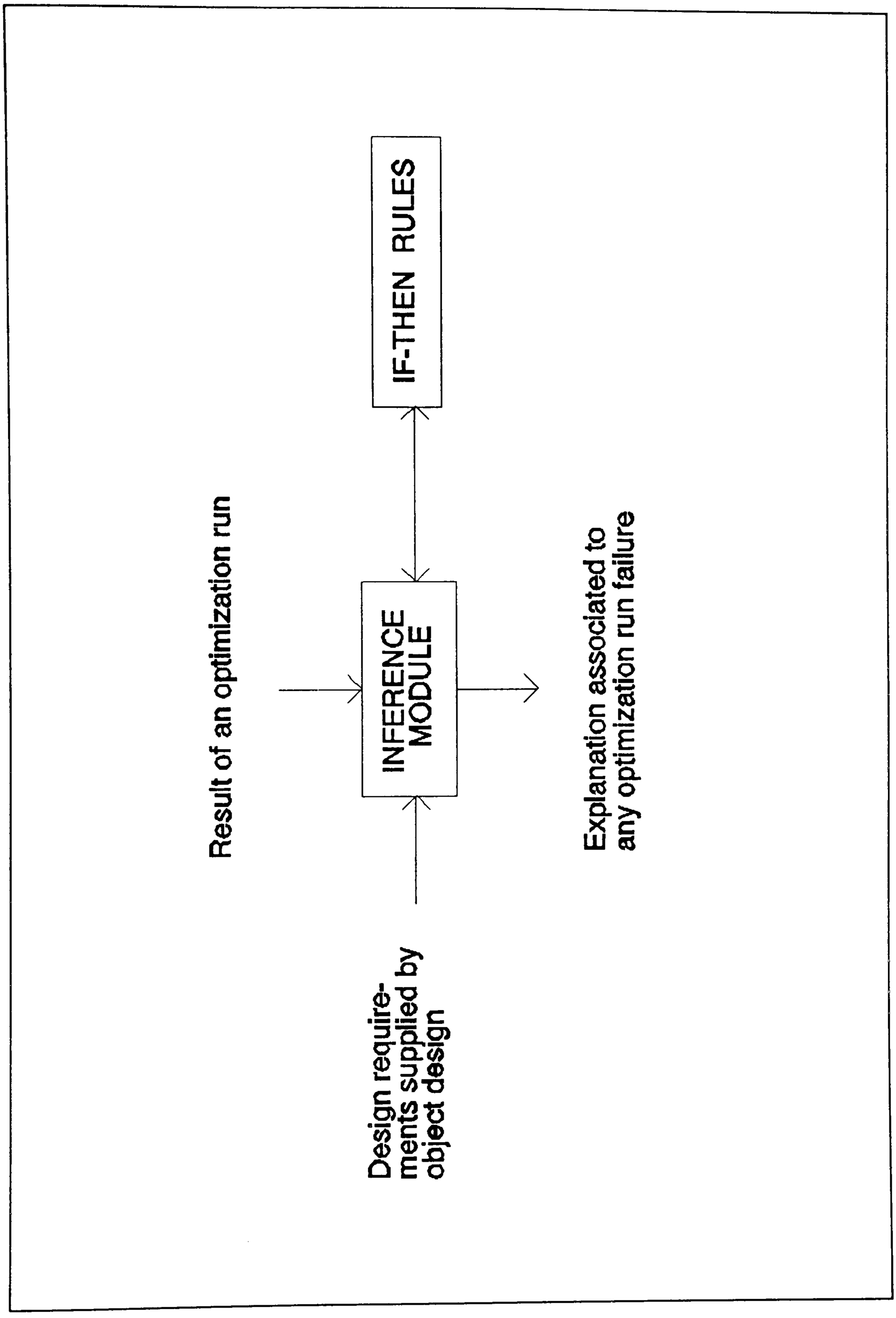


Figure 6-3 The process of the result interpretation reasoning.

CHAPTER 7:

PROTOTYPE DEVELOPMENT

The subject of the research presented in this report is the application of Expert System methodologies for improving the performance of structural optimization systems. The target is to produce an Expert System as an adviser for working structural optimization systems. The results from the earlier chapters have considered how the required knowledge may be constructed and reasoned with in order to produce interconnected components of an Expert System. In order to move towards a functioning system it would now be normal to construct an overall architecture. However, the concepts developed represent research conjectures and need validating before an architecture can be constructed. Thus, a prototype computer program is required to test idea and start the process of constructing a usable system.

This prototype is called ESSO (Expert System for Structural Optimization) and places the basic concepts into a program able to give guidance to designers who wish to employ optimization. The system will guide the user in set-ting up an optimization system, monitoring the optimization run, and interpreting the result or any failure associated with an optimization run. For trial implementation, the STARS optimization package is used as the optimization tool supported by ESSO. The prototype produces set-up files, which are called driver files, to run an optimization system. These files become the STARS CDF (command data) and CONS (design specification) files. The third file required by STARS, a file contains the finite element model, is to be prepared by the user.

At present, the prototype can handle static and dynamic problems only. But any type of constraint associated with both types of analysis can be handled. Currently, the prototype deals only with isotropic materials.

The modules available in the prototype are, the modules which constitute the system's front end,

- the design input module,
- the memory module,
- the conventional module,
- the element module,
- the variable module,
- the constraint module,
- the driver-files module,

and the modules which constitute the system back end,

- the run-time module,
- the result interpretation module,
- the modification module.

The prototype is written in C++ using object oriented design methodologies. Each module is written as a class with its own task. Some modules are defined with only one class while others have a parent and several child classes. Objects are created during executions and defined as global objects (similar in concept to global variables) which can be accessed by every module in the prototype.

A user interface is developed using XView [Heller 1990]. XView (X Window-System-based Visual/Integrated Environment for Workstations) is a user-interface toolkit to support interactive, graphics-based application running under the X Window System. XView applications are written using the C language. It features an object-oriented style interface and provides a set of prebuilt, user-interface objects such as canvases, scrollbars, menus and control panels. The appearance and functionality of these objects follow the OPEN LOOK Graphical User Interface (GUI) specification.

7.1. THE DESIGN INPUT MODULE.

The task of the design input module is to obtain information relating to the design requirement such as the analysis requirement, constraint requirement, number of design variables, and general information of the FE model. This module consists of one class, Design, which has the parameters for defining the design requirement and the operations/functions relating to obtaining the design requirement. The class Design defines the object *design*. All design specifications are stored in this object and these can be used by other modules through object passing. This object only deals with a general design requirement. More detail specifications are dealt with by other modules. For example, with regard to the constraint requirement, object *design* stores only type of constraints (such as stress, displacement) while details of the constraint values are handled by the module constraint. Figure 7-1 shows the input-output diagram of this module.

The data for this module, in the form of design specification, is supplied by the user and stored in *design*. Data in this object is required by the memory, conventional, variable, element, constraint, driver-files, and back-end modules. Data transfer is performed by sending object *design* to those modules.

7.2. THE MEMORY MODULE.

Following the execution of the design input module, the memory module is invoked. This contains the reasoner and the knowledge base for the past experience knowledge. The detail description of the mechanism inside this module has been discussed in chapter five. If there is a **similar** case found then the execution proceeds to the variable and element modules, otherwise the conventional module is executed. Figure 7-2 shows the input-output diagram of the memory module.

The information required by the memory module is supplied by object *design* in the form of the design requirements. The output of this module, definition of the optimization set-up, is used by the driver-files module.

7.3. THE CONVENTIONAL MODULE.

The conventional module contains the reasoner and the knowledge base for the a priori knowledge. The mechanism inside the module has been described in detail in chapter six. Figure 7-3 shows the input-output diagram of this module.

Similar to the memory module, the information required by the conventional module is supplied by object *design*. The output of this module, definition of the optimization set-up, is sent to the driver-files module.

7.4. THE ELEMENT MODULE.

The task of the element module is to deal with those operations related to the finite element model such as elements numbering and displaying elements characteristics. The details of the FE model are supplied by the user. This module has a base class, Element, which itself has three child classes, Plate_membrane, Beam, and Axial_rod. The class Beam has four child classes, those are I-beam, channel-beam, T-beam, and rectangular_beam. The classes under the class Element contain information about element characteristics and relations between design variable properties and analysis properties. Any element type used in the FE model is associated to an appropriate element class and entitles to the characteristics belong to that class. For example, a CBEAM (a NASTRAN element type) element type will be associated to the Beam class, and if the beam used is having a rectangular cross section then it will be associated further to the rectangular_beam class. The output of this module is an FE data and element

characteristics stored in various element objects. The input-output diagram of this module is shown in figure 7-4.

Other information required by this module is the design requirement which are concerned with the general FE model and supplied by object *design*. The output of this module is required by the variable and driver-files modules.

7.5. THE VARIABLE MODULE.

The variable module has the task of dealing with operations relating to variable formulation. This module consists of one class, Variable. Object *variable* is created from this class. The *variable* asks the user to supply variable specifications such as finite element types and element numbers in each design variable and stores them in this object. It also advises the users if incompatible element types (having different analysis properties) are grouped in the same design variable, for example if the rod elements is put under the same design variable as the plate elements. This module needs information relating to the element characteristics. The input-output diagram of the element module is shown in figure 7-5.

In order for the variable module to run, it needs information relating to the design requirement which is supplied by object *design*, and the elements characteristics supplied by various element objects. The user also has to supply variable specifications. The object variable is required by the driver-files module.

7.6. THE CONSTRAINT MODULE.

The operations relating to details of the design constraints are handled by the constraint module. This module has a base class, Constraint. This class has several child classes which deal with more specific type of constraints namely, Static_constraint, Normal_modes_constraint, Buckling_constraint, and Dynamic_constraint. Depending upon the analysis and constraint requirements, the appropriate objects are created. For example, if a static analysis with stress constraint is required, then an object which is called *static_constraint* is instantiated. Figure 7-6 shows the input-output diagram of this module.

This module needs detail information which is related to constraint specification supplied by the user. Some of the information is supplied by the object *design*. The

products of this module, various constraint objects, are required by the driver-files module.

7.7. THE DRIVER-FILES MODULE.

The task of the driver-files module is to produce files which are used to run the optimization software. In this prototype, the driver-files module produces files for running the STARS optimization package, namely the command data file (CDF) and the design specification file (CONS). We may note that STARS is being used to represent a general structural optimization system and the prototype is not written in a STARS specific manner. The ideas and concepts being trialed here are generally applicable to any system which employs driver files. The CDF consists of a set of instructions for running and controlling the optimization algorithms and includes the optimization method to use, number of iterations, and convergence criteria. The CONS defines the variable set-up and constraint set. The driver files module consists of two modules. The first module, the cdf-module, produces the CDF file while the second module, the cons-module, produces the CONS file. The information required by the driver files module to produce these two files is obtained from the modules executed previously. For different optimization packages, the driver files module has to be modified accordingly. Figure 7-7 shows the input-output diagram of this module.

7.8. THE MODULES OF THE SYSTEM'S BACK-END.

The back-end consists of the run-time module, the result interpretation module, and the modification module. With the driver files and FE model (prepared by the user) available, the optimization system can be run. This will be suspended after three iterations to check whether the optimization is converging. This check is performed with the aid of an extrapolation routine in the run-time module. If convergence is predicted the optimization run is resumed otherwise the module searches for explanations. If a point is reached where the convergence criteria selected by the user are satisfied (or the maximum number of iterations is reached) the result interpretation module checks the optimization result to ensure that this point is indeed optimal. For satisfactory results, the current design problem and its optimization set up is appended to the case memory in the memory module. If this module regards the optimization result as unsatisfactory it offers an explanation of the reasons for failure and corrective actions to follow. If required, the modification module

will modify the optimization set up for another optimization run.

The run-time and result interpretation modules are Expert System modules and have been discussed in detail in chapter six. The modification module has the task of modifying the optimization set-up if required. The modification is based on the actions proposed by the run-time module or the result interpretation module. The modification module also needs design data and optimization set up produced by the front-end. For design/constraint data processing, the modification module makes use the objects structures of the design input module, the variable module, the element module, and the constraint module. The modification instructions are sent to the driver files module. Figure 7-8 shows the input-output diagram for this module in a very simplified manner.

7.9. THE DATA FLOW DIAGRAM.

The overall architecture of the prototype is shown in figure 7-9. Figure 7-10 shows the data flow diagram.

7.10. PROTOTYPE IMPLEMENTATION OF THE KNOWLEDGE.

There are four modules in ESSO which contain the knowledge bases and reasoners namely, the memory module, the conventional module, the run-time module, and the result interpretation module. The tasks of the memory and conventional modules are for setting up the optimization. The run-time module monitors the optimization run. The optimization results will be interpreted by the result interpretation module. Details of how the knowledge may be constructed and reasoned with have been described in chapters five and six. The following sections describe their implementation in the prototype.

The knowledge base of the run-time and result interpretation module, were constructed using procedural rules, IF-THEN rules. The knowledge base of the conventional modules combine the domain/subject knowledge with IF-THEN rules. This domain knowledge contains the domain properties while the rules reason about them. The program implementation of forward chain inferencing is adapted from the original code in Levine et al. [1990]. The approach used in the memory module is based on case based reasoning. Similarity rules were set up to find similar cases among the past cases recorded in the memory module.

The domain knowledge of the conventional module was structured according to object oriented methodologies. Each optimization method is treated as an object and stored in a file. This represents the optimization method capabilities in handling design problems.

In the memory module, the domain knowledge contains past experiences in dealing with design problems. Each design experience is stored as a record. Each record, which represents a design case, contains the design problem and the set up of the optimization strategy for solving the design problem. Whether cases can be regarded as similar depend on the value of certainty factors of the cases after similarity rules are applied.

7.11. THE CONVENTIONAL KNOWLEDGE BASE.

The knowledge base of the conventional module combines both object oriented schemes and production rules. The object oriented scheme represents domain properties while rules reason about them. A simple way to show this relation is by writing production rules which send messages to objects and test the response in their premises. It must be noted that in an Expert System program, the user/programmer should be able to modify the rules (knowledge base) without interfering with the inference engine. This is shown in the following example,

```
IF ( analysis_test    = true ) and
   ( constraint_test = true )
THEN ( method_to_consider = true )          (7-1)
```

The premises variables (analysis_test and constraint_test) have to be instantiated first. The instantiation is performed in an assignment/instantiation routine which sends enquiries to a candidate object, in this case is assumed to be *fsd* (the Fully Stressed Design method), about its capabilities in handling analysis and constraints required. Functions analysis_test and constraint_test in the object *fsd* will provide the answers to these enquiries. If the conditions of the rule is satisfied then the conclusion method_to_consider is instantiated to true. For certain information, instead of asking questions to various objects, this assignment routine might ask the users to provide the answers.

The above illustration leads to the discussion about the domain/subject knowledge (written in an object oriented scheme) and production rules in the conventional module.

7.11.1 THE DOMAIN/SUBJECT KNOWLEDGE.

The domain knowledge contains the information about objects capabilities. The objects in this case are the optimization methods. Object oriented methodologies are used to represents this domain knowledge. For this purpose, two classes are defined, those are class **Procedure** and class **METHOD**. **Procedure** is a base class with **METHOD** as its child class.

The task of the class **Procedure** is to deal with matters relating to optimization algorithms. It contains parameters which define the capabilities and properties of optimization algorithms. This class also contains procedures for interrogating the available algorithms with regard to their capabilities in handling a current design problem. Object *procedure* is created from this class. The class Procedure is defined as follows (written in C++ terminology,

```
class Procedure {
public :
  /* the parameters below refer to algorithm capabilities
     in dealing with */
  // max number of load case
  int load_case,
  // lagrange multiplier.
      lagrange;
  // number of analysis types, number of constraint types.
  int analysis_num, constraint_num;
  // number of convergence type, number of element type
     available.
  int convergence_num, elem_avail_num;
  // combine analysis, combine constraint.
  int combine_analysis, combine_constraint;
  // stress constraint;
  int stress_constraint;
  // an algorithm might need stress constraint (e.g. FSD).
  int need_stress;
  // analysis types
  char analysis [10][40],
  // constraint types
       constraint[10][40],
  // convergence criteria types
       convergence[10][40],
  // element (FE) types
       elem_avail[10][40];

  /* method currently available */
  // number of method
  int num_of_method;
```

```
// method name
char method_name[5][20];

// procedure to invoke available methods.
void case_method(void);

/* procedures below are to test whether an algorithm can
   handle current design requirement with respect to fol-
   lowing aspects   */
// analysis
  int analysis_checking(void);
  // constraint
  int constraint_checking(void);
  // load case
  int loadcase_checking(void);
  // element (FE) types
  int element_checking(void);

} procedure;   // object procedure is defined.



class METHOD : public Procedure {
public :
  // method name
  char name[10];
  // procedure for reading the file
  void Dbase_read(char f[10]);
} method[5]; // five objects are created.
```

The class **METHOD** is the child of the class **Procedure**. During program run several object are created where each object represents an algorithm. The name of the available algorithms is stored in a file called *method.dat*. The data relating to an algorithm is read from a file. For example, data for the fsd algorithm is stored in a file called *fsd.dat*. There are five algorithms available in ESSO. The data for each file is written in a standard sequence as follows,

Parameters:	Explanation:
analysis_num;	Number of analysis.
analysis[i];	Types of analysis (array).
combine_analysis;	Capability in handling analysis combination.
constraint_num;	Number of constraints.
constraint[i];	Type of constraints (array).
combine_constraint;	Capability in handling constraint combination.
stress_constraint;	Capability in handling stress constraint.

need_stress;	If an algorithm needs stress constraint.
convergence_num;	Number of convergence criteria.
convergence[i];	Type of convergent criteria (array).
load_case;	Number of load case.
lagrange;	Capability to calculate lagrange multiplier.
elem_avail_num;	Number of element types.
elem_avail[i];	Element names (array).

For example, the capabilities of the optimality criterion algorithm is coded in *opc.dat* as follows,

```
3                    ---> number of analysis capability
static
normal_modes
buckling
0                    ---> not suited to analysis combination
4                    ---> number of constraints capability
stress
displacement
buckling
natural_frequency
0                   ---> not suited to constraint combination
0                    ---> can not handle stress constraint
0                    ---> does not need stress constraint
3
ENDP
ENDF
ENDT
20                   ---> number of load cases capability
1                    ---> able to calculate lagrange multi-
                          plier
7                    ---> number of available element types
CQUAD4
CTRIA3
CSHEAR
CBEAM
CBAR
CROD
CONROD
```

7.11.2. THE IF-THEN RULES.

The IF-THEN rules are used for reasoning. The rules are stored in a file called *convrule.dat*. The coding of these rules in C++ is implemented using *switch - case* operation as follows,

```
#ifdef RULE

    // if-then statements

     // if part:
     case 1: {
      if ( ..... )   s=1;
      break;
     }
     case 2: {
      if ( ..... )   s=1;
      break;
     }

     case 3: {
      if ( ..... )   s=1;
      break;
     }
        .
        .
        .

     case i: {
      if ( ..... )   s=1;
      break;
     }
        .
        .
        .


       case n: {
        if ( ..... )   s=1;
        break;
       }

     }


     // see if the then part should be invoked, i.e s=1
     if (s!=1) {
        ....;
        goto ...;
     }


     // invoke the then part

     switch(sn)   {
       // then part
       case 1: {
          ( ..... )
          instantiate();
```

```
    break;
  }
  case 2: {
    ( ..... )
    instantiate();
    break;
  }
        .
        .
        .
    case i: {
      ( ..... )
      instantiate();
      break;
    }
        .
        .
        .

    case n: {
      ( ..... )
      instantiate();
      break;
    }

}

#endif


#ifdef VARLIST

  // list of variables
  strcpy(varlt[1],"...");
  strcpy(varlt[2],"...");
        .
        .
        .
  strcpy(varlt[k],"...");

  // list of variables in rule premises
  strcpy(clvarlt[1],"...");                    //rule-1
  strcpy(clvarlt[2],"...");
  strcpy(clvarlt[11],"...");                   //rule-2
        .
        .
        .
  strcpy(clvarlt[21],"...");                   //rule-n
  strcpy(clvarlt[22],"...");
  strcpy(clvarlt[23],"...");

#endif
```

```
#ifdef CLASS_PAR

   ......

#endif


#ifdef CHECK_INST
void apriori_reasoning::check_instantiation(METHOD object)
{
  i=1;
  while ( (strcmp(v,varlt[i])!=0) && i<=30 )  i++;
  if (instlt[i]!=1) {
    instlt[i]=1;

    switch(i)  {
      case 1: {
        ............
        break;
      }
      case 2: {
        ............
        break;
      }
            .
            .
            .
      case m: {
        ............
        break;
      }

    }
  }
}

#endif
```

This file consists of four sections in which each section is coded between *#ifdef* and *#endif* statements. Those sections are,

- rules (between *#ifdef RULE* and *#endif*)
- list of variables (between *#ifdef VARLIST* and *#endif*)
- variable definition (between *#ifdef CLASS_PAR* and *#endif*)
- instantiation (between *#ifdef CHECK_INST* and *#endif*).

In the rules section, the rules (referred to by rule number) to be considered depend on the value of *sn*. The number after *case* is the rule number, from 1 to n. In the

'if part', if rule premises (within parentheses) are satisfied, then the value of *s* is instantiated to 1. In that case the 'then part' of that rule (within parentheses) will be fired. The function *instantiate()* is part of the inferencing mechanism.

7.11.3. EXPLANATION FACILITY.

Associated with each rule (IF-THEN statement) is an explanation-statement of that rule. This explanation-statement describe the contents of the rule using easy to understand sentences. When a rule is fired, the associated statement is recalled. The coding of the explanation-statement is as follow:

```
//case 1:
strcpy(texrule[1].text[0]," .... ");
strcpy(texrule[1].text[1]," .... ");
          .
          .
          .
strcpy(texrule[1].text[q-2]," .... ");
strcpy(texrule[1].text[q-1]," .... ");
texrule[1].textnum=q;

          .
          .
          .

//case i:
strcpy(texrule[i].text[0]," .... ");
strcpy(texrule[i].text[1]," .... ");
          .
          .
          .
strcpy(texrule[i].text[r-2]," .... ");
strcpy(texrule[i].text[r-1]," .... ");
texrule[i].textnum=r;

          .
          .
          .

//case n:
strcpy(texrule[n].text[0]," .... ");
strcpy(texrule[n].text[1]," .... ");
          .
          .
          .
strcpy(texrule[n].text[s-2]," .... ");
strcpy(texrule[n].text[s-1]," .... ");
texrule[n].textnum=s;
```

The number of explanation statements is the same as the number of rules which is *n*. The parameters q, r, and s are the number of "sentences"in each of statement. The explanation-statements for the conventional module are stored in the file *convexpl.dat*. The user should learn this file and the associated knowledge base file *convrule.dat*.

7.12. THE MEMORY KNOWLEDGE BASE.

The knowledge base of the memory module also consists of the domain knowledge and the IF-THEN rules. The domain knowledge is the system's memory of its past experience which can be regarded as a file containing a collection of knowledge (past experience) records. Each record contains a (past) design case and its solution, held in the form of the similarity parameters (for similarity parameters and similarity rules please see chapter 5) with their associated values and the successful optimization strategy for that particular case. The IF-THEN rules are the similarity rules and are used to reason in capturing the design case record which is similar to the current design problem.

7.12.1 THE DOMAIN KNOWLEDGE.

Each record in the knowledge base contains the following information (stored in fields),

- number of analysis,
- analysis types,
- number of constraints,
- constraint types,
- number of nodes,
- number of elements,
- element types,
- number of design variables,
- solution which consists of algorithm,
  number of iterations, and convergence criteria,
- comments, comments on the optimization process,
- filename, the filename where the FE data are stored.

Thus a typical record might be,

- number of analysis: 1;
- analysis types: statics;
- number of constraints: 2;
- constraints types: stress, displacement;
- number of nodes: 5;
- number of elements: 6;
- element types: beam;

- number of design variables: 6;
- solution: pseudo-newton algorithm, **n** number of iterations, a type of convergence criteria ;
- comment: a straight forward optimization run ;
- filename: framnsin.dat;

Such record can be written in C++ using *Structure*. The DESIGN_CASE structure below is used to represent the system's memory.

```
struct DESIGN_CASE {
    int index;
    int num_analysis;
    int num_constraint;
    int element_type_num;
    int element_num;
    int node_num;
    int design_var;
    int algor_iteration;
    int fsd_iteration;
    int used_rule[20];
    int num_fired_rule;
    float used_cf[20];
    float cf_value;
    char analysis[7][40];
    char constraint[7][40];
    char element_type[7][40];
    char algor_name[10];
    char crit_conv[10];
    char comment[10][80];
    char filename[40];
    struct TEXTRULE explain_rule[25];
};
```

Records of the system's past experience can be seen in *memory.dat*. Some members of the above data structure are used for reasoning process only and do not appear in the data file. The *TEXTRULE* is a structure which is used to store the explanation-statements and these statements are stored in *memrule.dat* together with the similarity rules.

7.12.2. THE SIMILARITY RULES.

The similarity rules are written in the following format:

```
// rule 1
if ( ..............      &&
     ..............      &&
        .
        .
        .
```

```
      ..............          )  {
   ..................;
   rule[1]=TRUE;
   cf[1]=x.xx;
}
// text rule 1
strcpy(textrule[1].text[0]," .... ");
strcpy(textrule[1].text[1]," .... ");
         .
         .
         .
strcpy(textrule[1].text[q-2]," .... ");
strcpy(textrule[1].text[q-1],"cf=x.xx");
textrule[1].textnum=q;

         .
         .
         .

// rule i
if ( ..............        &&
      ..............        &&
         .
         .
         .
      ..............           )  {
   ..................;
   rule[i]=TRUE;
   cf[i]=x.xx;
}
// text rule i
strcpy(textrule[i].text[0]," .... ");
strcpy(textrule[i].text[1]," .... ");

         .
         .
         .

strcpy(textrule[i].text[r-2]," .... ");
strcpy(textrule[i].text[r-1],"cf=x.xx");
textrule[i].textnum=r;

         .
         .
         .

// rule n
if ( ..............        &&
      ..............        &&
         .
         .
         .
```

```
      ..............          )   {
   ..................;
   rule[n]=TRUE;
   cf[n]=x.xx;
}
// text rule n
strcpy(textrule[n].text[0]," .... ");
strcpy(textrule[n].text[1]," .... ");
         .
         .
         .
   strcpy(textrule[n].text[s-2]," .... ");
   strcpy(textrule[n].text[s-1],"cf=x.xx");
   textrule[n].textnum=s;
```

A certainty factor (cf) with a value of *x.xx* is attached to each similarity rule. Although the rules above show AND (&&) relationships, the rule premises could also include OR relationships. As shown above, the explanation-statements are written below associated rules. The similarity rules are stored in the file *memrule.dat*.

7.13. THE KNOWLEDGE BASE OF RUN-TIME AND RESULT INTERPRETATION MODULES.

The run-time and result interpretation modules use procedural rules as the knowledge base. The structure of the rules (IF-THEN rules) are similar to the rules in the conventional module (omitting the objects). The knowledge base of the run-time interpretation module is stored in *runtrule.dat* while the explanation-statements are in *runtexpl.dat*. The knowledge base of the result interpretation module is stored in *reslrule.dat*, while the explanation statements are in *reslexpl.dat*.

7.14. OPTIMIZATION PROCEDURE.

Figure 7-11 shows a flow chart of the procedure in an optimization job as implemented in the ESSO prototype. It shows the sequence of modules execution but it does not show the interactions which can occur at any stage during the optimization process.

When the ESSO execution begins the first step the users have to do is to supply the design requirements. The design input module asks the user to supply
- type of analysis,
- set of constraint,

- FE data which includes the number of nodes, number of elements, element types, and number of load cases,
- number of design variables.

Based on the above information, the memory module searches for a similar case in its data base and if one is found the solution (the algorithm, number of iterations, and convergence criteria used) is proposed to the user. If required this solution can be modified. Having accepted the solution the execution moves to the design variables specification, otherwise the conventional module is invoked.

The conventional module, similar to the memory module, searches for an optimization solution using the information provided during the design input stage. This is performed by a reasoning process to an a-priori knowledge (see chapter six). The user can query the solution proposed by this module and accept or change it. Having accepted the solution the execution moves to the element specification.

To specify the design variables the user needs to define the element types for every design variable. If incompatible element types are grouped in one design variable the user will be warned and asked to make modification. The reason of incompatibility will be explained. If everything goes well then the user is asked to specifiy the element numbers for each design variable.

Depending upon the type of analysis used the system asks the user to supply the associated set of constraints. For example, if a static analysis is required then the user will be asked to supply constraints such as displacement, stress, or strain, and not natural frequency. For an analysis type the user has to supply one or more constraint types associated with that analysis. Most constraints require the users to supply upper and lower value of the constraint. The users are also asked to supply gauge limits for each design variable.

At this stage the system runs the driver-files module. If required by the user the driver files can be displayed. With the driver files and FE model available (the FE model prepared by the user) the optimization system can start running.

The performance of the optimization run is monitored by the run-time module which is part of the back-end module. As mentioned in (7.8) this is performed by suspending the run after three iterations to check whether the optimization is converging. If convergence is predicted the run is resumed. Optimization results will be interpreted by the result interpretation module. If results are not satisfactory, the system will search for

explanations and proposes actions to follow. Based on the proposed actions, if required the modification module will modify the optimization set up for another optimization run.

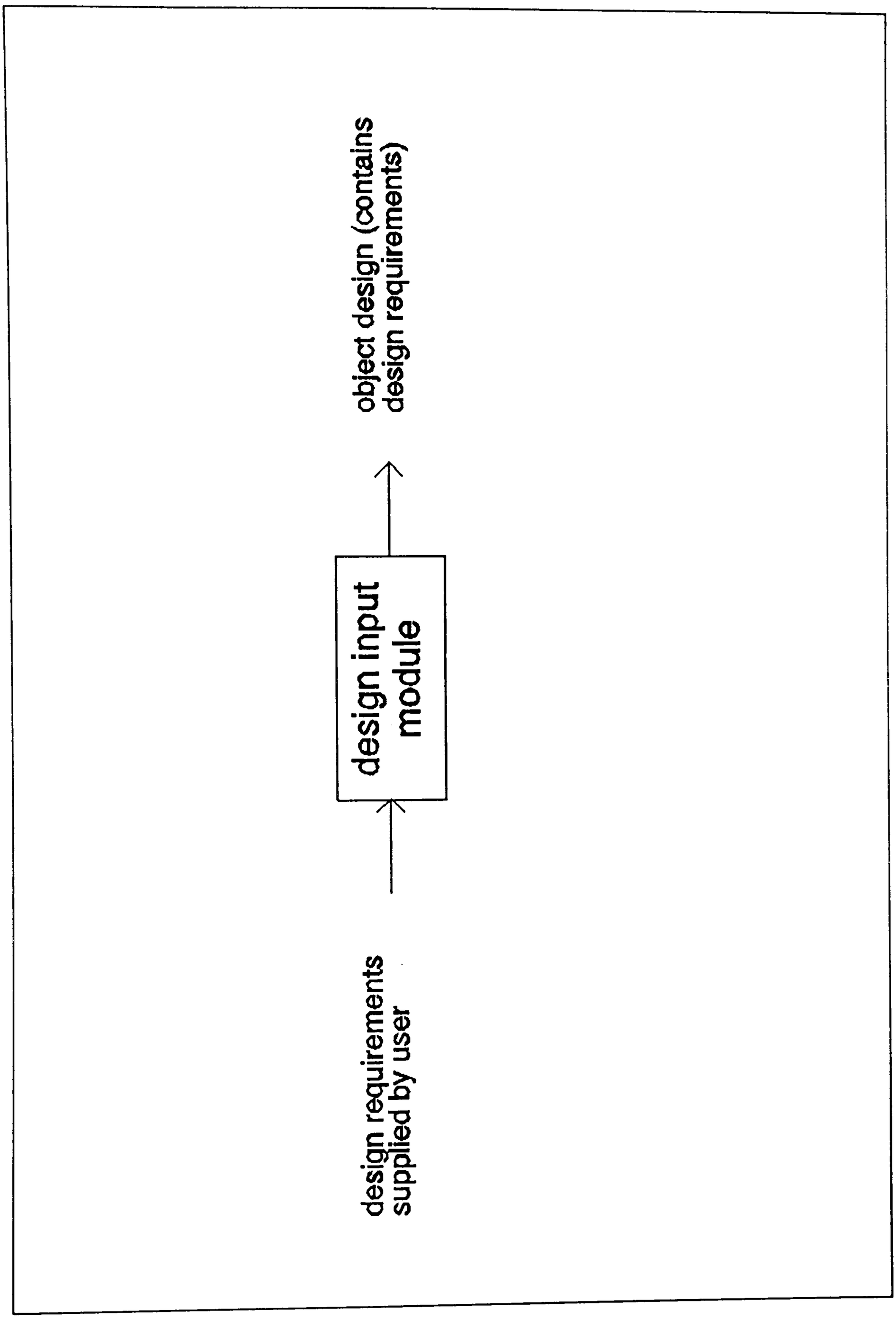


Figure 7-1 Input-output diagram of the design-input module.

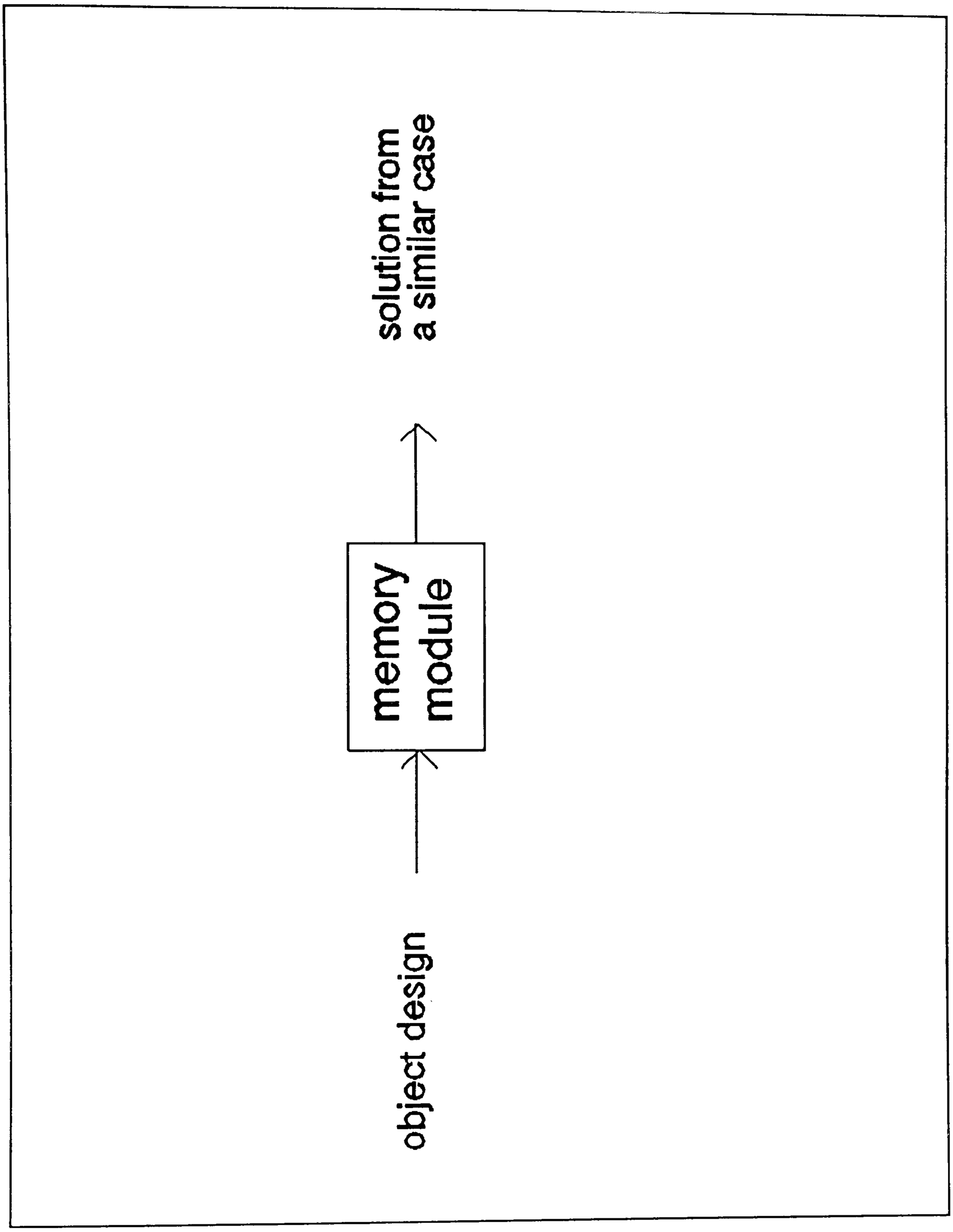


Figure 7-2 Input-output diagram of the memory module.

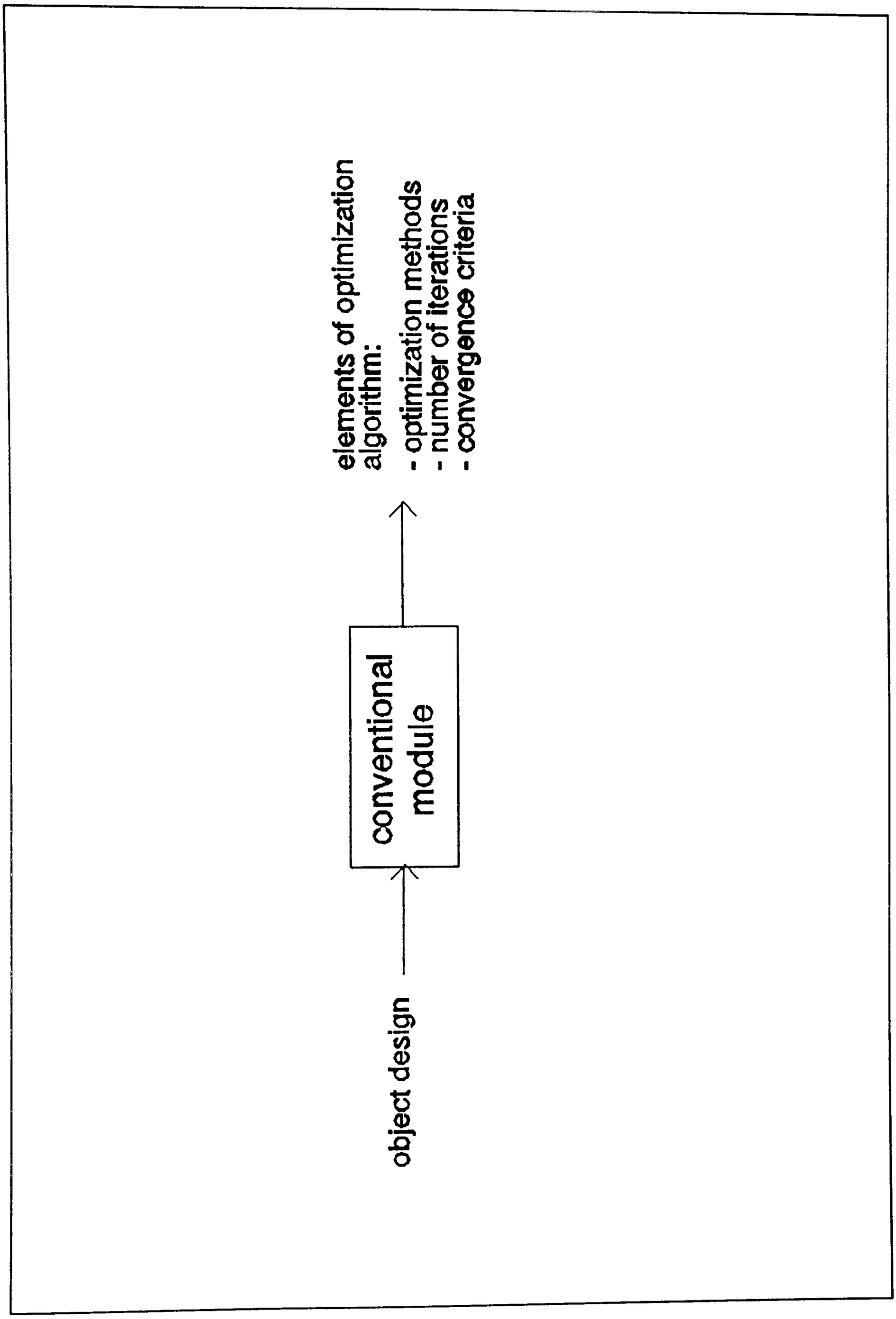


Figure 7-3 Input-output diagram of the conventional module.

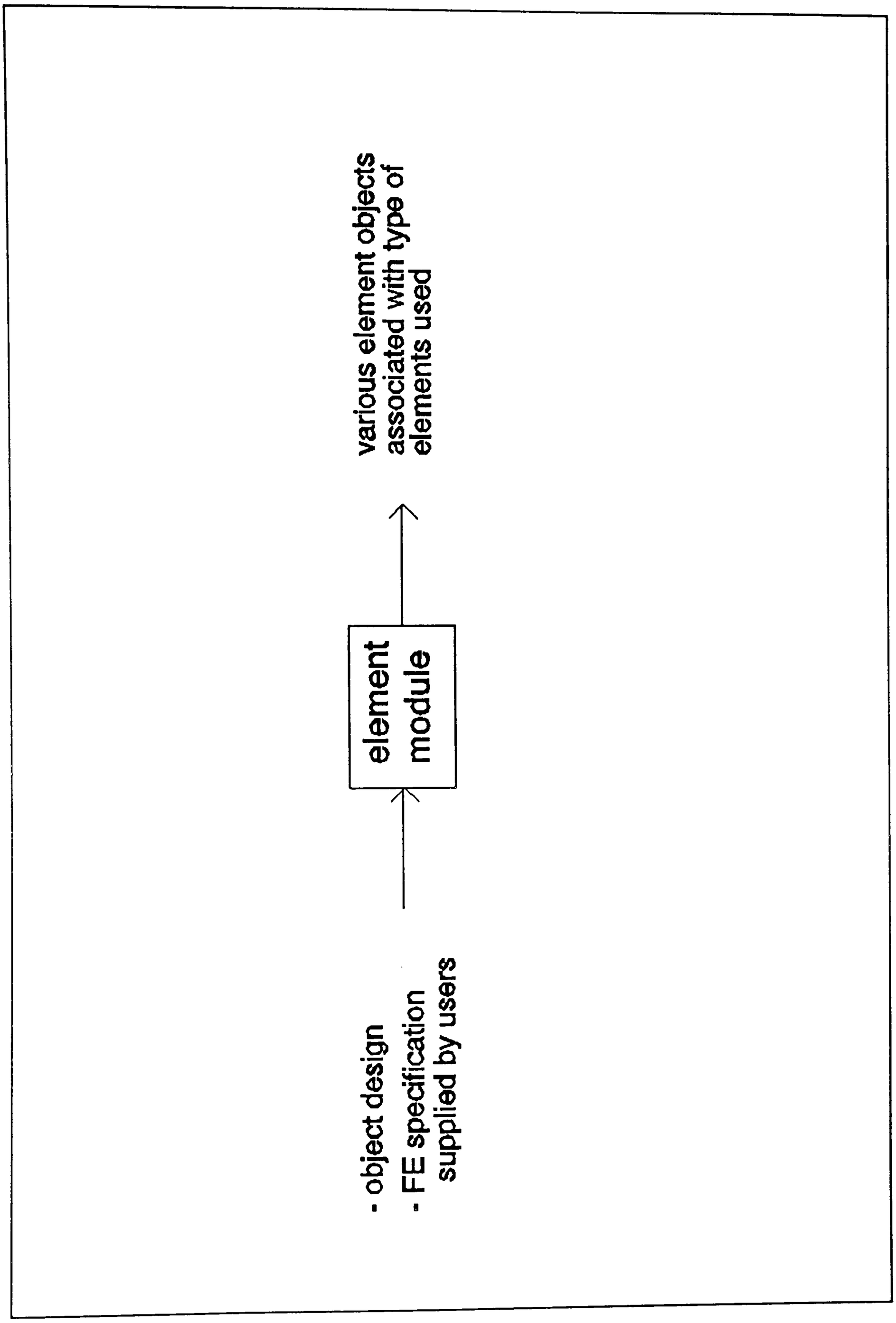


Figure 7-4 Input-output diagram of the element module.

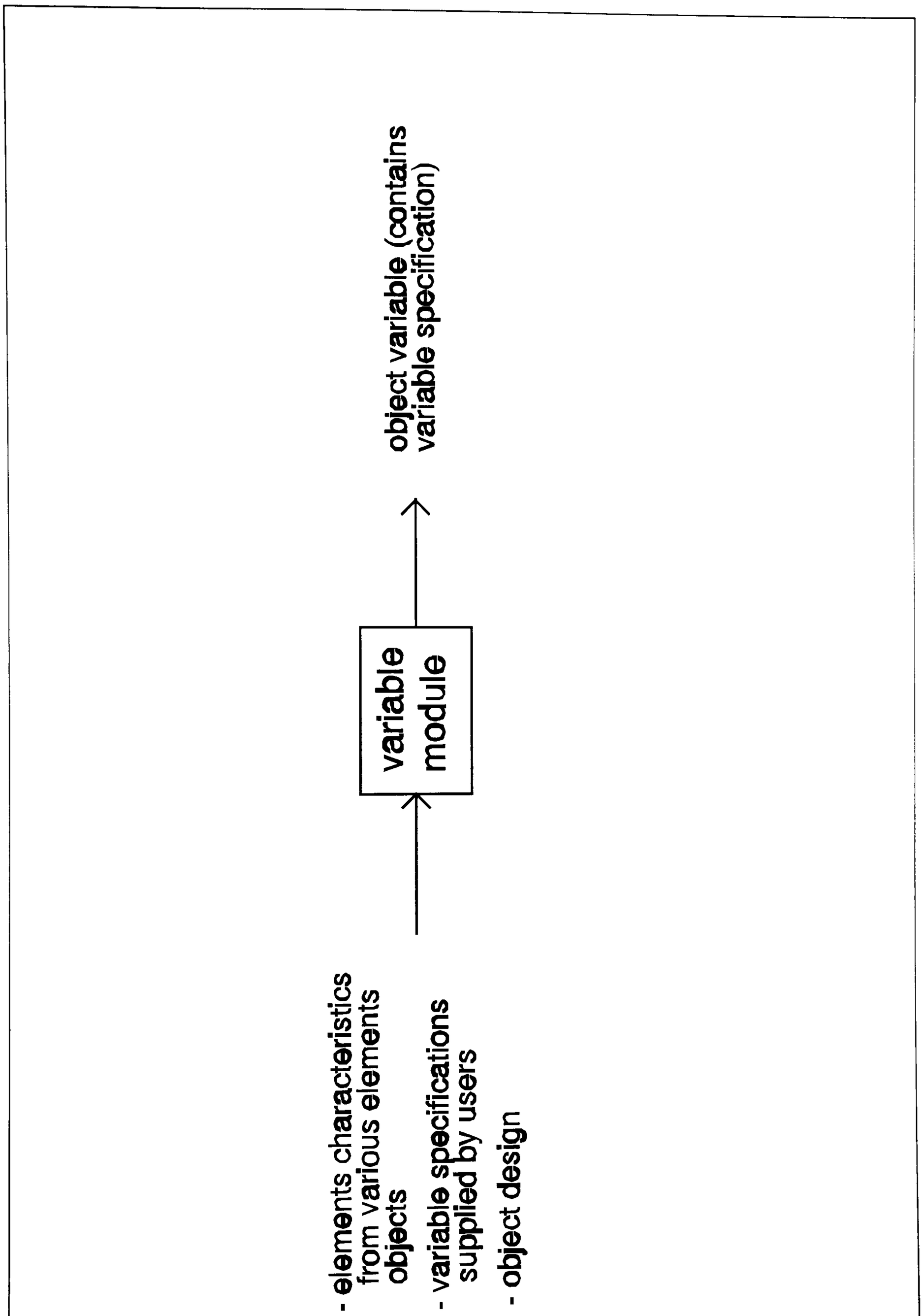


Figure 7-5 Input-output diagram of the variable module.

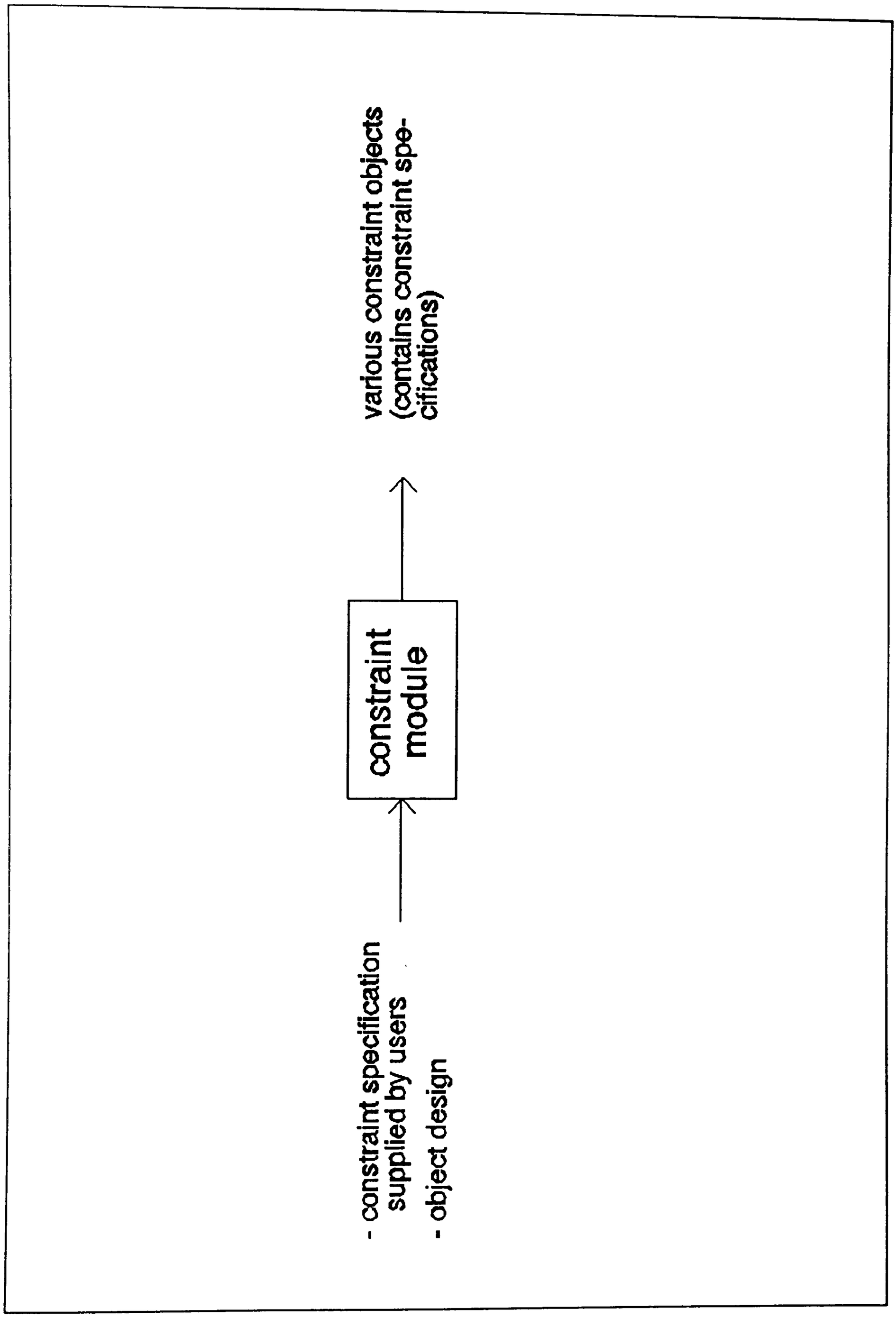


Figure 7-6 Input-output diagram of the constraint module.

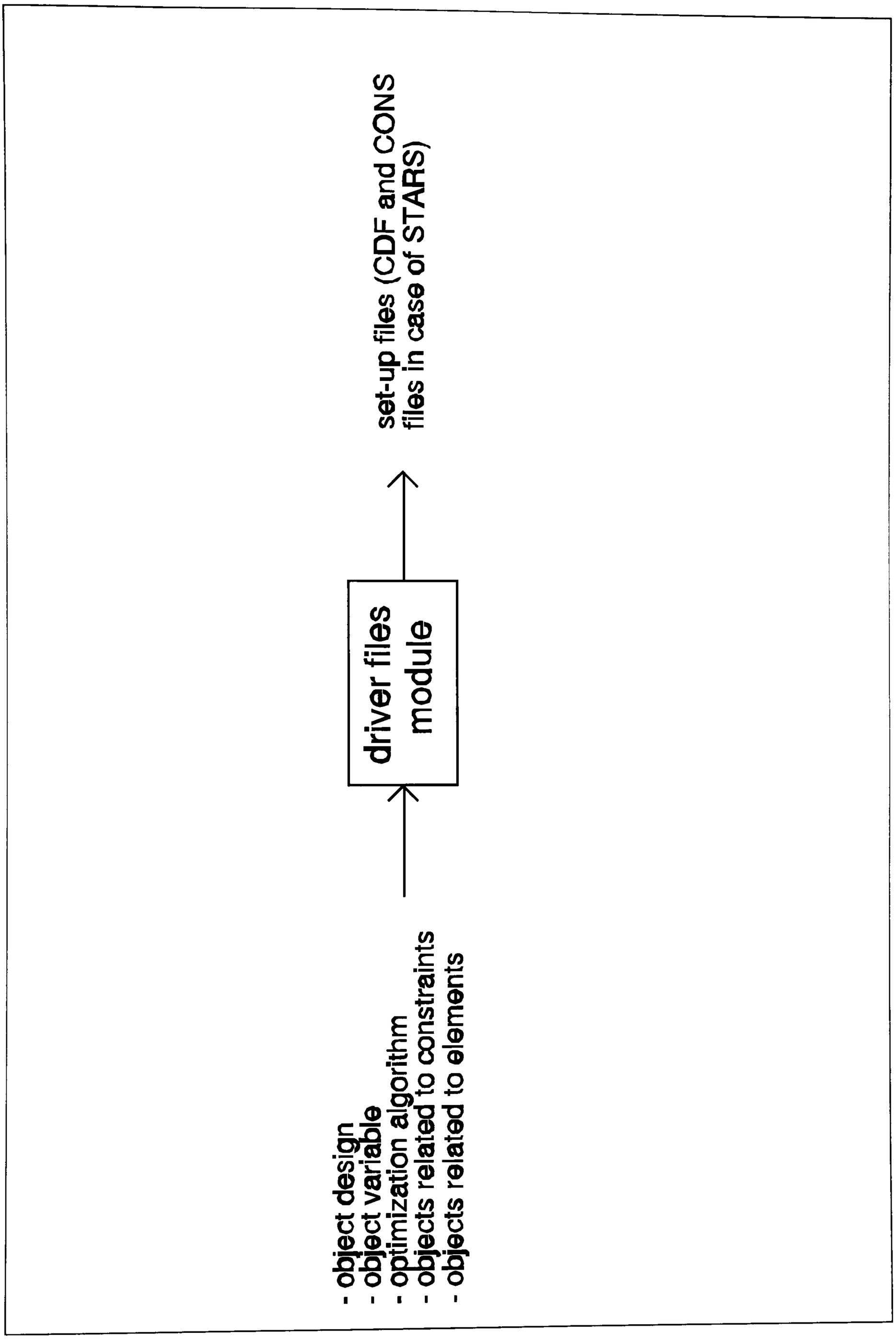


Figure 7-7 Input-output diagram of the driver-files module.

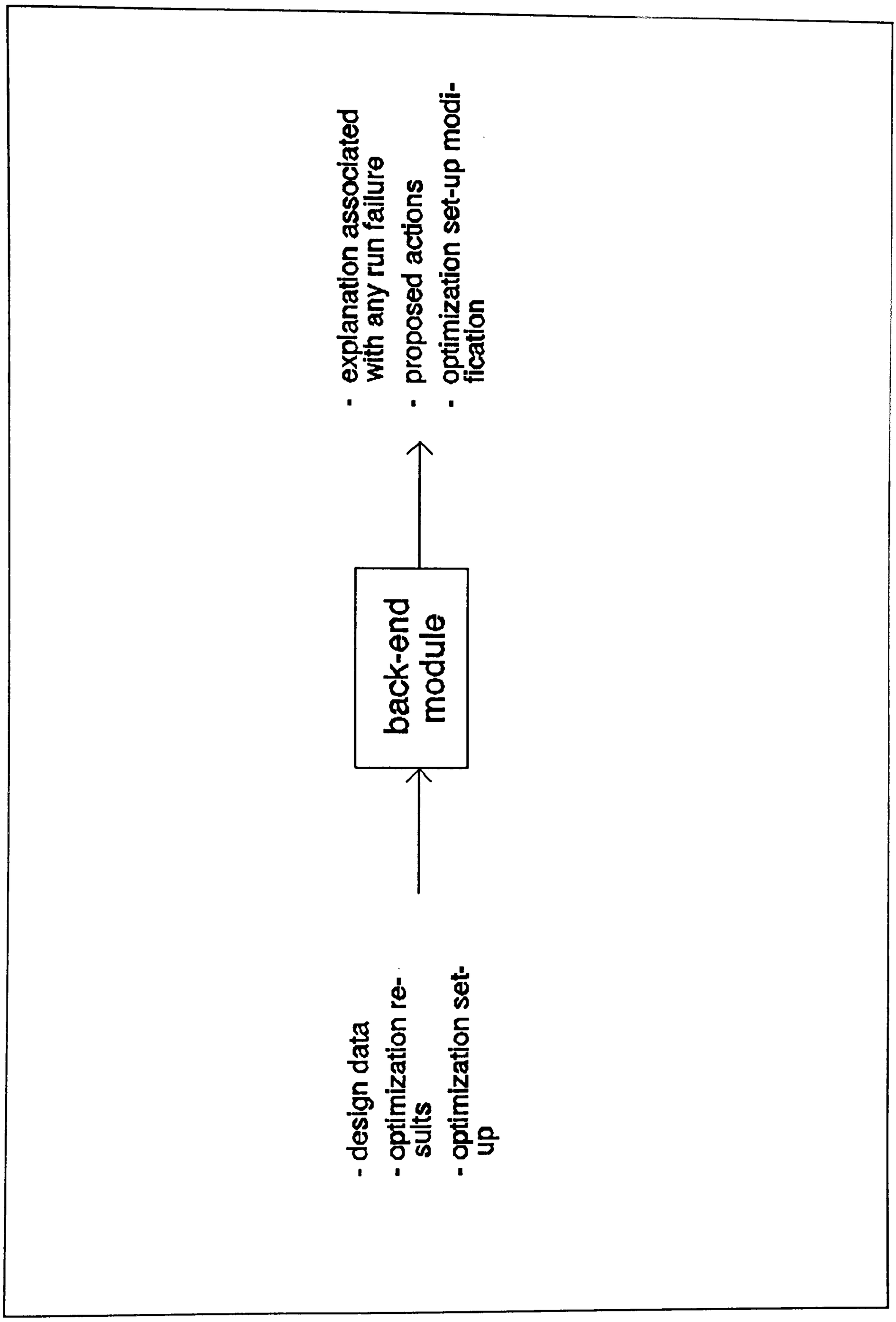


Figure 7-8 Input-output diagram of the back-end module.

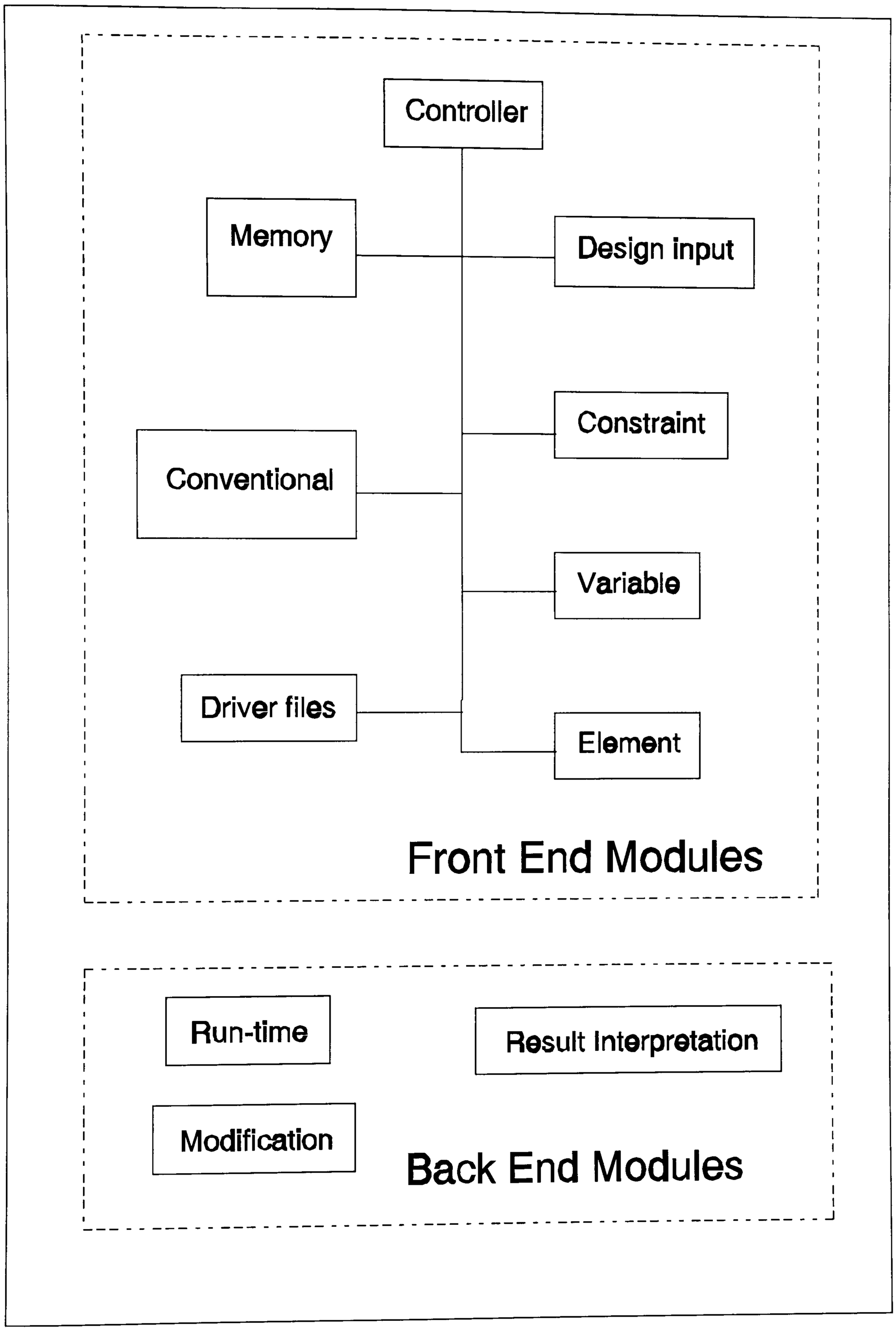


Figure 7-9 Architecture of the program.

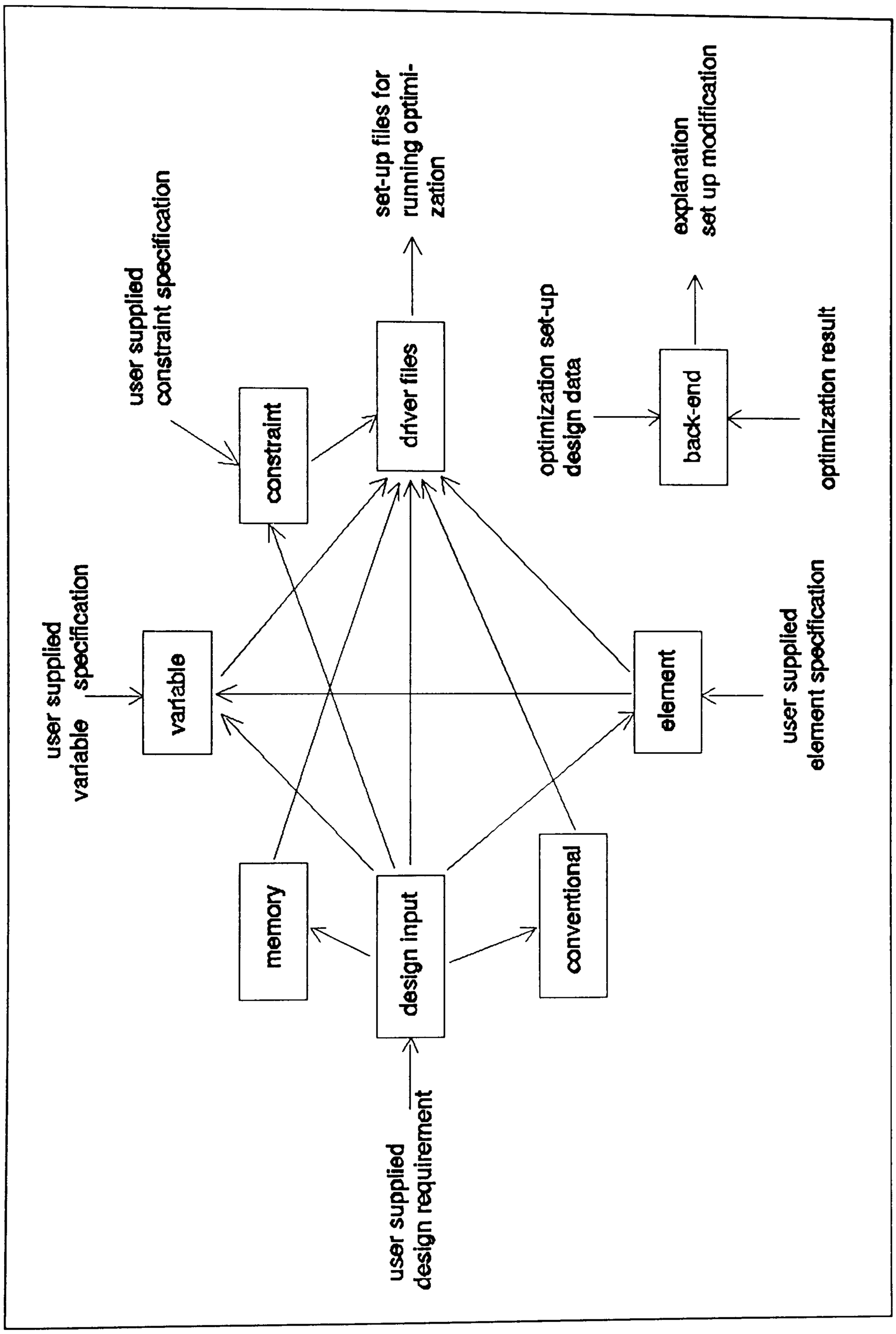


Figure 7-10 The data flow diagram of the modules.

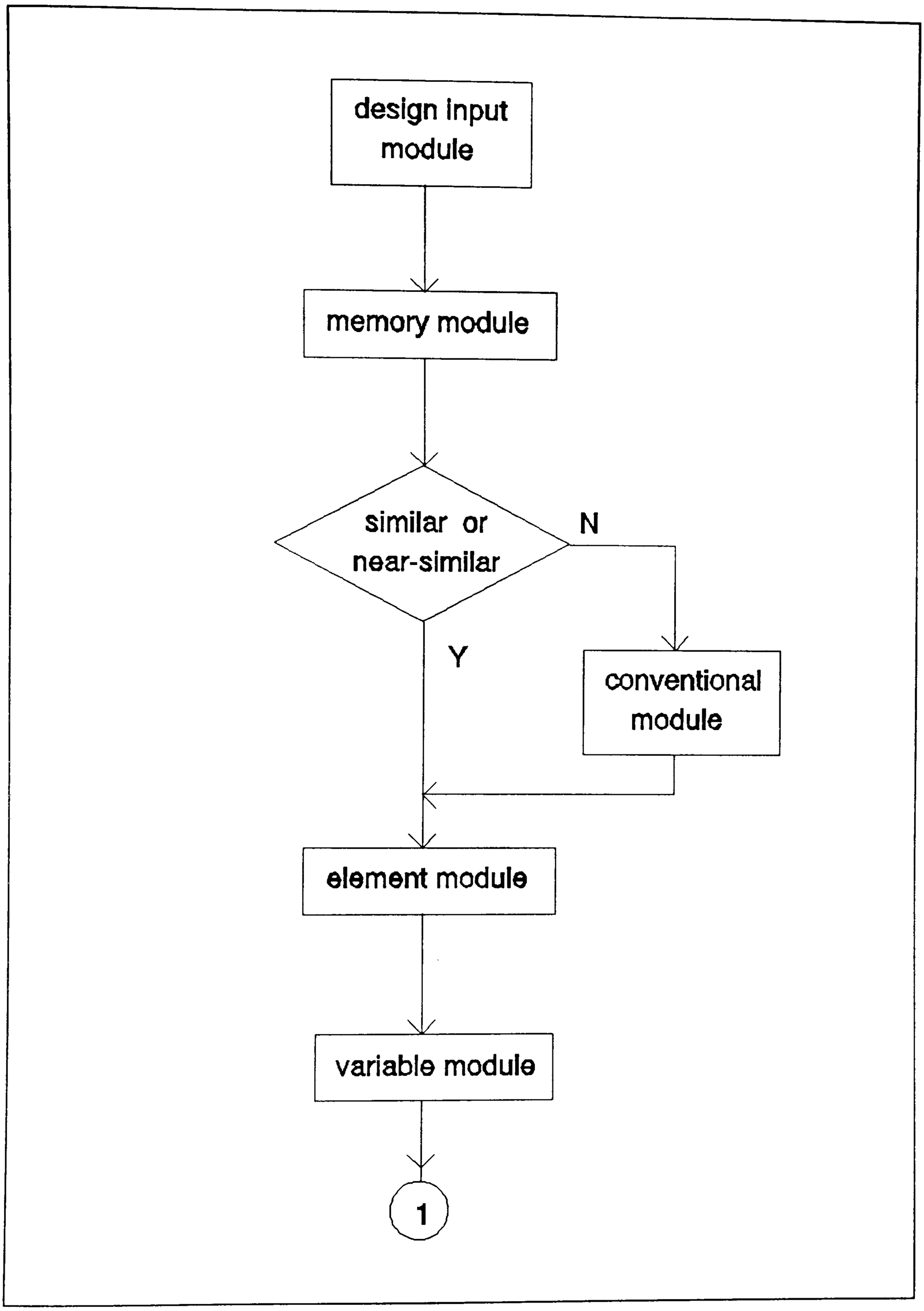


Figure 7-11 (continued)

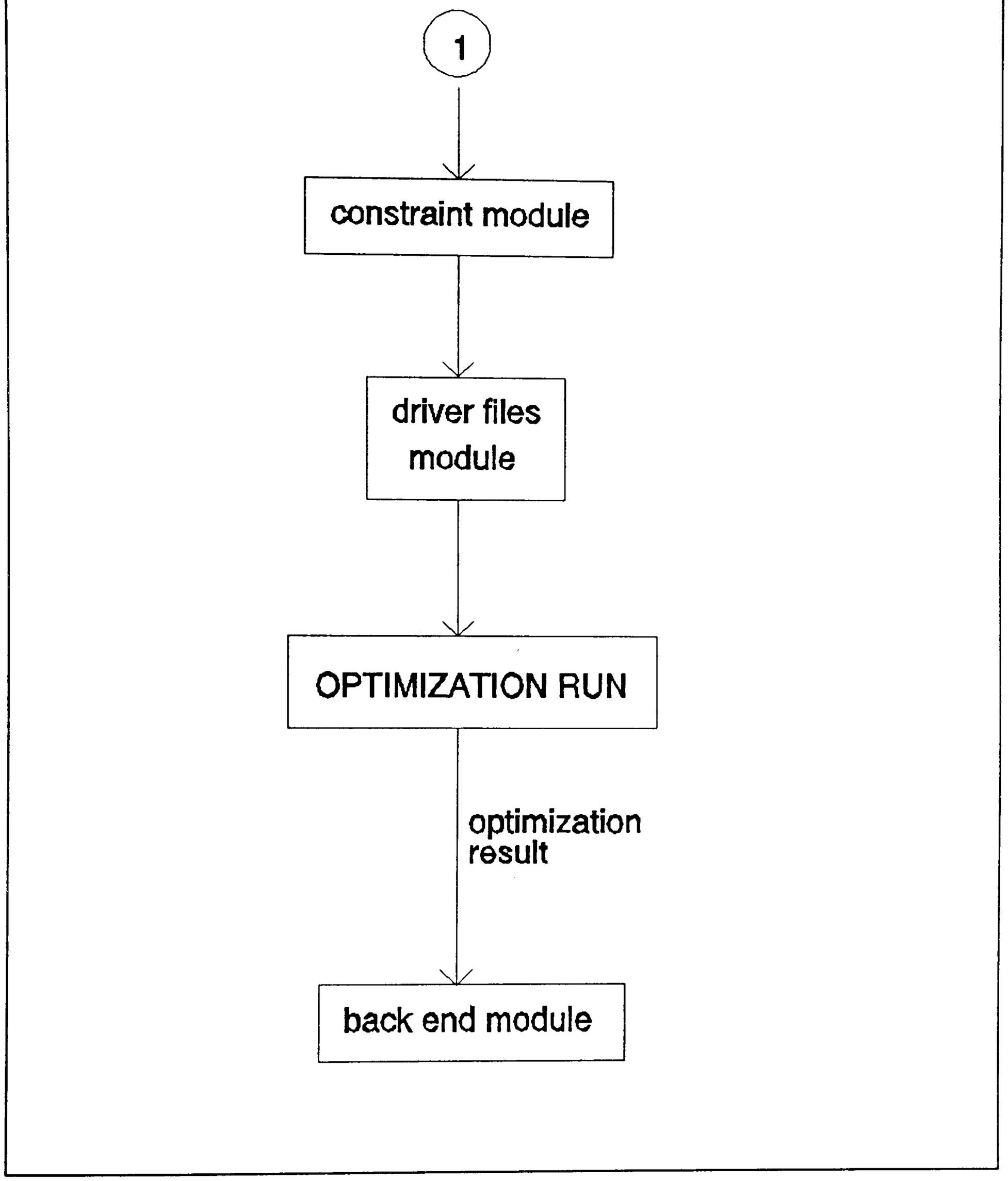


Figure 7-11 The sequence of modules execution.

CHAPTER 8:

VALIDATION

The ideas and Expert System techniques presented in the previous chapters need validation and for that purpose a prototype Expert System For Structural Optimization (ESSO) has been constructed (see chapter seven). Intensive tests using various design cases have been performed using the prototype and two of these, one static case and one dynamic case, are presented in this chapter.

8.1. STATIC CASE EXAMPLE.

8.1.1. PROBLEM DESCRIPTION.

A wing structure (cantilever-wing) is designed to carry a static load which is assumed to be applied at the wing tip. The structure consists of upper and lower panels, front and rear spars, and wing ribs. The length of the wing is 3600 mm (half-span). The root-chord length is 2400 mm with a maximum thickness of 600 mm. The tip-chord length is 800 mm with a maximum thickness of 200 mm. The finite element model of the wing is shown in figure 8-1 and consists of 90 nodal points and 112 elements. CQUAD4 membrane elements are used for the wing panels and ribs. For front and rear spars CSHEAR elements are used. The FE model in NASTRAN format is shown in table 8-1.

The tensile strength of the material is 250.0 MPa while the compressive strength is -120.0 MPa. The other material data used is:

Modulus Young = 79000.0 MPa
poisson's ratio = 0.33
mass density = 2.8E-6 Kg/mm^3

The structure is to be designed to be of minimum weight subject to limits on stresses given above and on the lateral displacement of 25 mm at nodes 81 to 85 (wing-tip). The gauge limits are as follows, thickness of the structural elements should be not less than 0.6 mm. It is specified that maximum thickness of upper and lower panel is 10 mm, the maximum thickness of spars is 3.5 mm, and the maximum thickness of ribs is 2.5 mm.

8.1.2. ESSO SESSION.

ESSO starts the session by asking the user to supply information relating to,

- type of analysis, in this case statics,
- type of constraints, stress and displacement.

The user must now supply a general description of the finite element model of the structure, which consists of,

- the number of nodal/grid points, which is 90,
- the element types, which in this case are CQUAD4 and CSHEAR,
- the number of elements, which is 112,
- and the number of load cases, in this case is 1.

Ideally the system should draw this information directly from the FE (NASTRAN) file, but at its current stage ESSO cannot do this. The user now informs the system of the number of design variables required, 6 in this example. It is up to the user to define the set up of the design variables which link specific finite elements to a design variable. For example if the panel of a wing bay (between two ribs) is desired to have a uniform thickness, then all finite elements in that bay are grouped in one design variable.

Based on the above information, ESSO searches its memory using the reasoning process described earlier and finds one "similar" case, that is,

case index number 3 (from static cases in memory),
number of analysis = 1, statics,
number of constraint type is 2, stress and displacement,
number of nodal points = 24,
number of elements = 26,
number of element types = 1, CQUAD4,
number of design variables = 5,
name of solution algorithm = Pseudo Newton,
maximum number of solution iteration = 4,
convergence criteria = dual gap.

The finite element model for the above case is shown in figure 8-2. The certainty factor of this case (indicating the degree to which this solution algorithm can solve the current problem successfully) is 0.381. The same analysis and constraint types give positive contribution to similarity, though the number of nodal points, elements, and design variables of this past case are smaller than the current design problem. This lack of total agreement does not become a kill off factor because the analysis type is static and ESSO believes there is a reasonable chance of success in using this case. The resulting algorithm proposed as a result of this process is the Pseudo Newton with maximum iteration numbers of 4. It is assumed that the user agrees with this proposal.

The next step is design variables set up. The user informs ESSO of the element types in each design variable (there are six design variables in the present case), and ESSO will check whether the elements grouping is appropriate. The element types of the design variables are,

variable-1 = CQUAD4,
variable-2 = CQUAD4,
variable-3 = CQUAD4,
variable-4 = CQUAD4,
variable-5 = CSHEAR,
variable-6 = CQUAD4.

The above grouping is accepted by ESSO and the user then has to supply the element numbers for each of these design variables,

variable-1 = 33,34,49,50,81,82,97,98 (the first upper and lower panel bays near the wing root)
variable-2 = 35,36,51,52,83,84,99,100 (the second upper and lower panel bays)
variable-3 = 37-48, 53-64 (the rest of the upper panel)
variable-4 = 85-96, 101-112 (the rest of the lower panel)
variable-5 = 65-80 (the front and rear spars)
variable-6 = 113-144 (the ribs).

The gauge and stress constraints are now supplied:

Gauges:

1. lower-limit = 0.6
   upper-limit = 10.0
   element = 33-64, 81-112 (upper and lower panels)

2. lower-limit = 0.6
   upper-limit = 3.5
   element = 65-80 (front and rear spars)

2. lower-limit = 0.6
   upper-limit = 2.5
   element = 113-144 (ribs)

Stress: lower-limit = -120.0
upper-limit = 250.0
for elements 33-144.

ESSO also asks the user to supply the displacement constraint:

lower-limit = -25.0
upper-limit = 25.0, to Y-direction
at nodes = 81-85.

At this stage ESSO has performed the necessary reasoning to construct a solution strategy and the user has supplied the description of the design problem. It is now a relatively straightforward for ESSO to create appropriate input data sets for the structural optimization, thus, ESSO now informs the user that the front end session is completed and the STARS driver files "essocdf.dat" (see table 8-2) and "essocons.dat" (see table 8-3) are produced. ESSO also requests the user to run STARS. When executed, the STARS run is suspended after 3 iterations. This suspension is invoked to allow ESSO to examine the progress of the optimizer and asses the likely success of the current run to converge to an optimum.

ESSO employs a number of parameters to predict convergence or otherwise as described earlier. In the present case the iteration history up to the third iteration (can be read from the STARS result file "essorslt.dat") is shown below.

iteration	feasible weight	dual weight
0	127.9	48.3
1	109.5	77.0
2	106.5	99.6
3	106.2	104.8

Based on the above information, the run-time application predicts that the optimization will reach convergence at maximum iteration (maximum iteration=4). In fact the above data shows that the optimization run has produced a very good design at iteration 3.

The STARS run is resumed with one more iteration to complete. After the optimization stops at fourth iteration, the result application is invoked to check whether this design can be regarded as optimum. This ESSO application uses dual weight of the last iteration (iteration 4) and the most non-feasible constraint (if any) together with any violated limit value. In the present case we find that at the optimum,

- feasible weight = 106.2, dual weight = 106.1,

- the "violated" constraint is displacement at node 83 equal to 25.0 mm with the limit value of 25.0 mm.

Based on this data, ESSO concludes that the design at iteration 4 can indeed be regarded as optimum because the constraints are satisfied while the dual gap is satisfied.

Because the optimization produces an optimum design, the ESSO memory is updated with this design problem.

8.2. DYNAMIC CASE EXAMPLE.

8.2.1. PROBLEM DESCRIPTION.

The same structure as in the first case (8.1) is used again for this second design case. Instead of the static design requirement, now the structure is desired to have a minimum weight while maintaining its lowest natural frequency above 2.50 cycles/second. This is a dynamic case with frequency constraint. The FE model and the set up of the design variables of the first case are retained. The same material is also used with similar gauge limits on elements. The FE model is similar to the one in figure 8-1 omitting the static forces. Table 8-4 shows the NASTRAN data of the model.

8.2.2. ESSO SESSION.

Similar to the first case, ESSO starts the session by asking the user to supply information relating to,

- type of analysis, which in this case is dynamics,
- type of constraint, which is natural frequency.

This is followed by the general description of the finite element model of the structure, and this consists of,

- number of nodal/grid points, which is 90,
- element types, which in this case are CQUAD4 and CSHEAR,
- number of elements, which is 112,
- and number of load cases, which in this case is 1.

The user then must inform the system on the number of design variables required, which we have selected to be 6.

Based on the above information, ESSO suggests the following as a suitable "similar" case,

```
case index number 1 (from dynamic cases in memory),
number of analysis = 1, dynamics,
number of constraint type is 1, natural frequency,
number of nodal points = 90,
number of elements = 148,
number of element types = 3, CQUAD4, CSHEAR, CONROD,
number of design variables = 8,
name of solution algorithm = Pseudo Newton,
maximum number of solution iteration = 7,
convergence criteria = dual gap
```

The finite element model of the above case can be seen in figure 8-3. The certainty factor of this case is 0.473. In addition to the front and rear spars, the structure in the "similar" proposed case has a middle spar. We now assume that the user is satisfied that there is a good chance of success in using this case to supply the optimization solution strategy for the present example. The algorithm proposed is the pseudo newton with maximum iteration numbers of 7. It is assumed that the user prefers to increase the maximum iteration numbers to 10.

The next step is design variables set up. The element types associated with the design variables are,

```
variable-1 = CQUAD4,
variable-2 = CQUAD4,
variable-3 = CQUAD4,
variable-4 = CQUAD4,
variable-5 = CSHEAR,
variable-6 = CQUAD4,
```

and the element within each design variable,

```
variable-1 = 33,34,49,50,81,82,97,98 (the first upper and
             lower panel bays near the wing root)
variable-2 = 35,36,51,52,83,84,99,100 (the second upper
             and lower panel bays)
variable-3 = 37-48, 53-64 (the rest of the upper panel)
variable-4 = 85-96, 101-112 (the rest of the lower panel)
variable-5 = 65-80 (the front and rear spars)
variable-6 = 113-144 (the ribs).
```

Similar to the first case, there is no fix-element.

The gauges limit are specified as:

```
1. lower-limit = 0.6
   upper-limit = 10.0
   element     = 33-64, 81-112 (upper and lower panels)

2. lower-limit = 0.6
   upper-limit = 3.5
   element     = 65-80 (front and rear spars)

2. lower-limit = 0.6
   upper-limit = 2.5
   element     = 113-144 (ribs)
```

Next, ESSO asks the user to supply the lower limit value of the frequency and the associated mode number. In this case,

```
mode number = 1
natural frequency = 2.50.
```

At this stage ESSO inform the user that the front end session is completed and the STARS driver files "essocdf.dat" (see table 8-5) and "essocons.dat" (see table 8-6) are produced. ESSO also requests the user to run STARS. As with the previous "statics" case the run is suspended after 3 iterations.

The iteration history for these initial iterations is:

iteration	feasible weight	dual weight
0	127.9	35.9
1	80.4	35.9
2	107.9	35.9
3	168.7	35.9

Based on the above information, the run-time application predicts that the optimization will not converge to the optimum at maximum iteration (maximum iteration=10).

At this point, the optimizer has been able to raise the natural frequency of the structure from the starting value of 1.54 Hz to 2.0 Hz. However, ESSO believes that the optimum will not be reached because there is insufficient scope to make changes due to the design variable set up. ESSO, thus, proposes that number of design variables be increased. In response to this the number of design variables is increased to 17.

The element types associated with the design variables are,

variable-1 = CQUAD4, the first bay of upper and lower panels,
variable-2 = CQUAD4, the second bay of upper and lower panels,
variable-3 = CQUAD4, the third bay of upper and lower panels,
variable-4 = CQUAD4, the fourth bay of upper and lower panels,
variable-5 = CQUAD4, the fifth bay of upper and lower panels,
variable-6 = CQUAD4, the sixth bay of upper and lower panels,
variable-7 = CQUAD4, the seventh bay of upper and lower panels
variable-8 = CQUAD4, the eighth bay of upper and lower panels,
variable-9 = CQUAD4, the ribs,
variable-10 = CSHEAR, the first bay of front and rear spars,

variable-11 = CSHEAR, the second bay of front and rear spars,
variable-12 = CSHEAR, the third bay of front and rear spars,
variable-13 = CSHEAR, the fourth bay of front and rear spars,
variable-14 = CSHEAR, the fifth bay of front and rear spars,
variable-15 = CSHEAR, the sixth bay of front and rear spars,
variable-16 = CSHEAR, the seventh bay of front and rear spars,
variable-17 = CSHEAR, the eighth bay of front and rear spars.

The associated elements within each design variable can be seen in table 8-7.

Having completed the modification, ESSO produces "essocdf.dat" and "essocons.dat". The "essocons.dat" is shown in table 8-7. ESSO then asks the user to run STARS.

After 3 iterations STARS run is suspended. The ESSO's run-time application asks the user to supply data on feasible and dual weights from iteration 0 to iteration 3 which is shown below,

iteration	feasible weight	dual weight
0	127.9	35.9
1	72.1	34.7
2	71.6	48.3
3	76.8	56.4

Based on the above information, the run-time application predicts that the optimization will reach convergence at maximum iteration (maximum iteration=10) and asks the user to resume the optimization run.

The optimization run stops at maximum iteration and the ESSO's result application is called to check the results. It asks the user to supply the feasible and dual weights of the last iteration (iteration 10) and also the most non-feasible constraint (if any) together with the violated limit value. From the "essorslt.dat",

- feasible weight = 78.3, dual weight = 66.0,

- the "violated" constraint is the natural frequency of 2.39 Hz which is still below the limit value of 2.5 Hz.

From the data above ESSO concludes that the design is not satisfactory, but the constraint-gap is relatively small, under five percent. At this stage, ESSO asks the user to supply the history of frequency during optimization run which is as follow,

f0	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10
1.54	1.73	2.05	2.29	2.42	2.36	2.42	2.38	2.41	2.40	2.39

The data above shows the frequency at every iteration from initial value, f0, to the last iteration, f10. Based only on the above data, in fact the optimization is converging to a design point where the natural frequency of the structure is close to 2.4 Hz (still infeasible). From this fact ESSO concludes that the problem is caused by the algorithm itself (pseudo newton). To solve this problem (an infeasible design) ESSO suggests the user to increase slightly the constraint value. This is accepted by increasing the natural frequency to 2.65 Hz.

With this slight modification, the STARS is re-run. The optimization data after ten iteration is

- feasible weight = 95.9, dual weight = 72.5,
- the natural frequency is 2.53 Hz which is above the original limit value of 2.50 Hz.

The data above shows that the design is not yet optimum but it is a useful feasible design (with respect to the original design requirement) and probably not that far away from the optimum point. The frequency history as shown below,

f0	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10
1.54	1.75	2.09	2.37	2.56	2.48	2.56	2.48	2.54	2.48	2.53

shows that the optimization is converging to a design point with the frequency close to 2.5 Hz. This supports the fact points out by ESSO that the pseudo newton method of STARS could not cope well with this dynamic problem.

Finally ESSO updates its memory by appending this case and asking the user to put comments on the running of the optimization. The comments might look like this,

- natural frequency requirement of at least 2.5 Hz.
- initial frequency of structure (iter-0) = 1.54 Hz.

- needs effective material distribution through appropriate design variable set-up
- optimization tends to point to a slightly infeasible design
- the trick is to slightly increase the constraint to 2.65 Hz.

Table 8-1: The FE data of the Static Case Example.

```
==========================================================
TITLE = WING STATIC-CASE STRESS-DISPLACEMENT CONSTRAINT
SUBCASE=1
LOAD=1
BEGIN BULK
GRID            1                0.    300.      0.          123456
GRID            2                0.    270.    600.          123456
GRID            3                0. 221.052   -600.          123456
GRID            4                0. 142.104  -1200.          123456
GRID            5                0.    140.   1200.          123456
GRID            6                0.-142.105  -1200.          123456
GRID            7                0.   -200.   1200.          123456
GRID            8                0.-221.052   -600.          123456
GRID            9                0.   -300.    600.          123456
GRID           10                0.   -300.      0.          123456
GRID           11              450.    275.      0.             456
GRID           12              450.   247.5    550.             456
GRID           13              450. 202.631   -550.             456
GRID           14              450. 130.813   1100.             456
GRID           15              450. 130.262  -1100.             456
GRID           16              450.-130.263  -1100.             456
GRID           17              450.-186.875   1100.             456
GRID           18              450.-202.632   -550.             456
GRID           19              450.   -275.    550.             456
GRID           20              450.   -275.      0.             456
GRID           21         900.0001    250.      0.             456
GRID           22         900.0001    225.    500.             456
GRID           23         900.0001  184.21   -500.             456
GRID           24         900.0001 121.625   1000.             456
GRID           25         900.0001 118.421  -1000.             456
GRID           26         900.0001-118.421  -1000.             456
GRID           27         900.0001 -173.75   1000.             456
GRID           28         900.0001-184.211   -500.             456
GRID           29         900.0001   -250.    500.             456
GRID           30         900.0001   -250.      0.             456
GRID           31            1350.    225.      0.             456
GRID           32            1350.   202.5    450.             456
GRID           33            1350. 165.789   -450.             456
GRID           34            1350. 112.438900.0001             456
GRID           35            1350. 106.579-900.000             456
GRID           36            1350.-106.579-900.000             456
GRID           37            1350.-160.625900.0001             456
GRID           38            1350. -165.79   -450.             456
GRID           39            1350.   -225.    450.             456
GRID           40            1350.   -225.      0.             456
----------------------------------------------------------
```

(continued)

Table 8-1

GRID	41	1800.	200.	0.	456
GRID	42	1800.	180.	400.	456
GRID	43	1800.	147.368	-400.	456
GRID	44	1800.	103.25	800.0001	456
GRID	45	1800.	94.73701	-800.000	456
GRID	46	1800.	-94.7375	-800.000	456
GRID	47	1800.	-147.369	-400.	456
GRID	48	1800.	-147.5	800.0001	456
GRID	49	1800.	-200.	400.	456
GRID	50	1800.	-200.	0.	456
GRID	51	2250.	175.	0.	456
GRID	52	2250.	157.5	350.	456
GRID	53	2250.	128.948	-350.	456
GRID	54	2250.	94.06251	700.0001	456
GRID	55	2250.	82.89521	-700.000	456
GRID	56	2250.	-82.8956	-700.000	456
GRID	57	2250.	-128.948	-350.	456
GRID	58	2250.	-134.375	700.0001	456
GRID	59	2250.	-175.	350.	456
GRID	60	2250.	-175.	0.	456
GRID	61	2700.	150.	0.	456
GRID	62	2700.	135.	300.	456
GRID	63	2700.	110.527	-300.	456
GRID	64	2700.	84.87501	600.	456
GRID	65	2700.	71.0535	-600.	456
GRID	66	2700.	-71.0537	-600.	456
GRID	67	2700.	-110.527	-300.	456
GRID	68	2700.	-121.25	600.	456
GRID	69	2700.	-150.	300.	456
GRID	70	2700.	-150.	0.	456
GRID	71	3150.	125.	0.	456
GRID	72	3150.	112.5	250.	456
GRID	73	3150.	92.10591	-250.	456
GRID	74	3150.	75.68751	500.	456
GRID	75	3150.	59.21181	-500.	456
GRID	76	3150.	-59.2119	-500.	456
GRID	77	3150.	-92.1059	-250.	456
GRID	78	3150.	-108.125	500.	456
GRID	79	3150.	-125.	250.	456
GRID	80	3150.	-125.	0.	456
GRID	81	3600.	100.	0.	456
GRID	82	3600.	90.00001	200.	456
GRID	83	3600.	73.68501	-200.	456
GRID	84	3600.	66.50001	400.	456
GRID	85	3600.	47.37001	-400.	456

(continued)

Table 8-1

GRID	86		3600.	-47.3700	-400.	456
GRID	87		3600.	-73.6850	-200.	456
GRID	88		3600.	-95.0000	400.	456
GRID	89		3600.	-100.	200.	456
GRID	90		3600.	-100.	0.	456
MAT1	1	79000.		0.33	2.80-6	
PSHELL	33	1	1.			
PSHEAR	65	1	1.			
FORCE	1	81		5000.0	0.0	1.0 0.0
FORCE	1	82		10000.0	0.0	1.0 0.0
FORCE	1	83		12000.0	0.0	1.0 0.0
FORCE	1	84		10000.0	0.0	1.0 0.0
FORCE	1	85		5000.0	0.0	1.0 0.0
CQUAD4	33	33	5	14	12	2 0.
CQUAD4	34	33	2	12	11	1 0.
CQUAD4	35	33	14	24	22	12 0.
CQUAD4	36	33	12	22	21	11 0.
CQUAD4	37	33	24	34	32	22 0.
CQUAD4	38	33	22	32	31	21 0.
CQUAD4	39	33	34	44	42	32 0.
CQUAD4	40	33	32	42	41	31 0.
CQUAD4	41	33	44	54	52	42 0.
CQUAD4	42	33	42	52	51	41 0.
CQUAD4	43	33	54	64	62	52 0.
CQUAD4	44	33	52	62	61	51 0.
CQUAD4	45	33	64	74	72	62 0.
CQUAD4	46	33	62	72	71	61 0.
CQUAD4	47	33	74	84	82	72 0.
CQUAD4	48	33	72	82	81	71 0.
CQUAD4	49	33	1	11	13	3 0.
CQUAD4	50	33	3	13	15	4 0.
CQUAD4	51	33	11	21	23	13 0.
CQUAD4	52	33	13	23	25	15 0.
CQUAD4	53	33	21	31	33	23 0.
CQUAD4	54	33	23	33	35	25 0.
CQUAD4	55	33	31	41	43	33 0.
CQUAD4	56	33	33	43	45	35 0.
CQUAD4	57	33	41	51	53	43 0.
CQUAD4	58	33	43	53	55	45 0.
CQUAD4	59	33	51	61	63	53 0.
CQUAD4	60	33	53	63	65	55 0.
CQUAD4	61	33	61	71	73	63 0.
CQUAD4	62	33	63	73	75	65 0.
CQUAD4	63	33	71	81	83	73 0.
CQUAD4	64	33	73	83	85	75 0.

(continued)

Table 8-1

CQUAD4	81	33	6	16	18	8	0.
CQUAD4	82	33	8	18	20	10	0.
CQUAD4	83	33	16	26	28	18	0.
CQUAD4	84	33	18	28	30	20	0.
CQUAD4	85	33	26	36	38	28	0.
CQUAD4	86	33	28	38	40	30	0.
CQUAD4	87	33	36	46	47	38	0.
CQUAD4	88	33	38	47	50	40	0.
CQUAD4	89	33	46	56	57	47	0.
CQUAD4	90	33	47	57	60	50	0.
CQUAD4	91	33	56	66	67	57	0.
CQUAD4	92	33	57	67	70	60	0.
CQUAD4	93	33	66	76	77	67	0.
CQUAD4	94	33	67	77	80	70	0.
CQUAD4	95	33	76	86	87	77	0.
CQUAD4	96	33	77	87	90	80	0.
CQUAD4	97	33	10	20	19	9	0.
CQUAD4	98	33	9	19	17	7	0.
CQUAD4	99	33	20	30	29	19	0.
CQUAD4	100	33	19	29	27	17	0.
CQUAD4	101	33	29	39	37	27	0.
CQUAD4	102	33	30	40	39	29	0.
CQUAD4	103	33	40	50	49	39	0.
CQUAD4	104	33	39	49	48	37	0.
CQUAD4	105	33	50	60	59	49	0.
CQUAD4	106	33	49	59	58	48	0.
CQUAD4	107	33	60	70	69	59	0.
CQUAD4	108	33	59	69	68	58	0.
CQUAD4	109	33	70	80	79	69	0.
CQUAD4	110	33	69	79	78	68	0.
CQUAD4	111	33	80	90	89	79	0.
CQUAD4	112	33	79	89	88	78	0.
CQUAD4	113	33	17	14	12	19	0.
CQUAD4	114	33	19	12	11	20	0.
CQUAD4	115	33	20	11	13	18	0.
CQUAD4	116	33	18	13	15	16	0.
CQUAD4	117	33	27	24	22	29	0.
CQUAD4	118	33	29	22	21	30	0.
CQUAD4	119	33	30	21	23	28	0.
CQUAD4	120	33	28	23	25	26	0.
CQUAD4	121	33	37	34	32	39	0.
CQUAD4	122	33	39	32	31	40	0.
CQUAD4	123	33	40	31	33	38	0.
CQUAD4	124	33	38	33	35	36	0.

(continued)

Table 8-1

```
--------------------------------------------------------------------------------
CQUAD4     125      33      48      44      42      49      0.
CQUAD4     126      33      49      42      41      50      0.
CQUAD4     127      33      50      41      43      47      0.
CQUAD4     128      33      47      43      45      46      0.
CQUAD4     129      33      58      54      52      59      0.
CQUAD4     130      33      59      52      51      60      0.
CQUAD4     131      33      60      51      53      57      0.
CQUAD4     132      33      57      53      55      56      0.
CQUAD4     133      33      68      64      62      69      0.
CQUAD4     134      33      69      62      61      70      0.
CQUAD4     135      33      70      61      63      67      0.
CQUAD4     136      33      67      63      65      66      0.
CQUAD4     137      33      78      74      72      79      0.
CQUAD4     138      33      79      72      71      80      0.
CQUAD4     139      33      80      71      73      77      0.
CQUAD4     140      33      77      73      75      76      0.
CQUAD4     141      33      88      84      82      89      0.
CQUAD4     142      33      89      82      81      90      0.
CQUAD4     143      33      90      81      83      87      0.
CQUAD4     144      33      87      83      85      86      0.
CSHEAR      65      65       5      14      17       7
CSHEAR      66      65      14      24      27      17
CSHEAR      67      65      24      34      37      27
CSHEAR      68      65      34      44      48      37
CSHEAR      69      65      44      54      58      48
CSHEAR      70      65      54      64      68      58
CSHEAR      71      65      64      74      78      68
CSHEAR      72      65      74      84      88      78
CSHEAR      73      65       4      15      16       6
CSHEAR      74      65      15      25      26      16
CSHEAR      75      65      25      35      36      26
CSHEAR      76      65      35      45      46      36
CSHEAR      77      65      45      55      56      46
CSHEAR      78      65      55      65      66      56
CSHEAR      79      65      65      75      76      66
CSHEAR      80      65      75      85      86      76
ENDDATA
================================================================================
```

(end of table)

Table 8-2: The command data file of the Static Case Example.

```
==========================================================================
START
COMMAND DATA PNEWTON
C
      SET NSIV=1;
      SET MXIT=4;
      SET APRT=YES;
      SET ACTG=YES;
      SET IFLM=YES;
      DO NSIN;
      DO OPTIN;
C
      DO ANALYSIS;
      DO INSPECT;
      DO OPTX;
C     PSEUDO NEWTON ITERATIONS
L010: DO ITER;
      DO QNEWTON;
      DO OPTX;
      IF(ENDC.EQ.YES) GOTO L031;
      IF(NOIT.EQ.3) DO HISTORY;
      IF(NOIT.EQ.3) SUSPEND;
      IF(ENDI.EQ.YES) GOTO L040;
      GOTO L010;
L031: DISPLAY CONVERGENCE TO THE LOWER BOUND WEIGHT;
      GOTO L050;
L032: DISPLAY CONVERGENCE TO DESIGN VARIABLES;
      GOTO L050;
L033: DISPLAY CONVERGENCE TO FEASIBLE VS ACTUAL WEIGHT;
      GOTO L050;
L034: DISPLAY CONVERGENCE TO FEASIBLE WEIGHT;
      GOTO L050;
L040: DISPLAY MAX. NUMBER OF ITERATIONS REACHED;
C
L050: SET APRT=YES;
      DO ANALYSIS;
      DO INSPECT;
      DO HIST;
      STOP;
      END;
C     ANALYSIS PROCEDURE;
      PROC OPTX;
      DO ANALYSIS;
      DO ACTSET;
      DO DERV;
      DO CONV;
      DO INSPECT;
      END OPTX;
ENDATA
==========================================================================
```

Table 8-3: The design/constraint file of the Static Case Example.

```
================================================================================
C234567890123456789012345678901234567890123456789012345678901234567890
DESIGN              1          33
DESIGN              1          34
DESIGN              1          49
DESIGN              1          50
DESIGN              1          81
DESIGN              1          82
DESIGN              1          97
DESIGN              1          98
DESIGN              2          35
DESIGN              2          36
DESIGN              2          51
DESIGN              2          52
DESIGN              2          83
DESIGN              2          84
DESIGN              2          99
DESIGN              2          100
DESIGN              3          37
DESIGN              3          38
                    .
                    to
                    .
DESIGN              3          47
DESIGN              3          48
                    .
                    to
                    .
DESIGN              3          63
DESIGN              3          64
                    .
                    to
                    .
DESIGN              4          95
DESIGN              4          96
DESIGN              4          101
DESIGN              4          102
                    .
                    to
                    .
DESIGN              4          111
DESIGN              4          112
DESIGN              5          65
--------------------------------------------------------------------------------
```

(continued)

Table 8-3

DESIGN	5	66	
	.		
	to		
	.		
DESIGN	5	79	
DESIGN	5	80	
DESIGN	6	113	
DESIGN	6	114	
	.		
	to		
	.		
DESIGN	6	143	
DESIGN	6	144	
DESIGN	6	117	
GAUGE	0.600000	10.0000	33
GAUGE	0.600000	10.0000	34
	.		
	to		
	.		
GAUGE	0.600000	10.0000	63
GAUGE	0.600000	10.0000	64
GAUGE	0.600000	10.0000	81
GAUGE	0.600000	10.0000	82
	.		
	to		
	.		
GAUGE	0.600000	10.0000	111
GAUGE	0.600000	10.0000	112
GAUGE	0.600000	3.50000	65
GAUGE	0.600000	3.50000	66
	.		
	to		
	.		
GAUGE	0.600000	3.50000	79
GAUGE	0.600000	3.50000	80
GAUGE	0.600000	2.50000	113
GAUGE	0.600000	2.50000	114
	.		
	to		
	.		
GAUGE	0.600000	2.50000	143
GAUGE	0.600000	2.50000	144
STRESS	-120.000	250.000	33
STRESS	-120.000	250.000	34
STRESS	-120.000	250.000	35
STRESS	-120.000	250.000	36

(continued)

Table 8-3

STRESS	-120.000	250.000	37
STRESS	-120.000	250.000	38
	.		
	to		
	.		
STRESS	-120.000	250.000	143
STRESS	-120.000	250.000	144
DISP-Y	-25.0000	25.0000	81
DISP-Y	-25.0000	25.0000	82
DISP-Y	-25.0000	25.0000	83
DISP-Y	-25.0000	25.0000	84
DISP-Y	-25.0000	25.0000	85
ENDATA			

(end of table)

Table 8-4: The FE data of the Dynamic Case Example.

```
==========================================================
TITLE = WING DYNAMIC-CASE FREQUENCY CONSTRAINT
BEGIN BULK
GRID            1              0.    300.      0.          123456
GRID            2              0.    270.    600.          123456
GRID            3              0. 221.052   -600.          123456
GRID            4              0. 142.104  -1200.          123456
GRID            5              0.    140.   1200.          123456
GRID            6              0.-142.105  -1200.          123456
GRID            7              0.   -200.   1200.          123456
GRID            8              0.-221.052   -600.          123456
GRID            9              0.   -300.    600.          123456
GRID           10              0.   -300.      0.          123456
GRID           11            450.    275.      0.             456
GRID           12            450.   247.5    550.             456
GRID           13            450. 202.631   -550.             456
GRID           14            450. 130.813   1100.             456
GRID           15            450. 130.262  -1100.             456
GRID           16            450.-130.263  -1100.             456
GRID           17            450.-186.875   1100.             456
GRID           18            450.-202.632   -550.             456
GRID           19            450.   -275.    550.             456
GRID           20            450.   -275.      0.             456
GRID           21         900.0001    250.      0.             456
GRID           22         900.0001    225.    500.             456
GRID           23         900.0001  184.21   -500.             456
GRID           24         900.0001 121.625   1000.             456
GRID           25         900.0001 118.421  -1000.             456
GRID           26         900.0001-118.421  -1000.             456
GRID           27         900.0001 -173.75   1000.             456
GRID           28         900.0001-184.211   -500.             456
GRID           29         900.0001   -250.    500.             456
GRID           30         900.0001   -250.      0.             456
GRID           31             1350.    225.      0.             456
GRID           32             1350.   202.5    450.             456
GRID           33             1350. 165.789   -450.             456
GRID           34             1350. 112.438900.0001             456
GRID           35             1350. 106.579-900.000             456
GRID           36             1350.-106.579-900.000             456
GRID           37             1350.-160.625900.0001             456
GRID           38             1350. -165.79   -450.             456
GRID           39             1350.   -225.    450.             456
GRID           40             1350.   -225.      0.             456
GRID           41             1800.    200.      0.             456
GRID           42             1800.    180.    400.             456
----------------------------------------------------------
```

(continued)

Table 8-4

```
--------------------------------------------------------------------------------
GRID           43              1800. 147.368   -400.                        456
GRID           44              1800.  103.25800.0001                        456
GRID           45              1800.94.73701-800.000                        456
GRID           46              1800.-94.7375-800.000                        456
GRID           47              1800.-147.369   -400.                        456
GRID           48              1800.  -147.5800.0001                        456
GRID           49              1800.   -200.    400.                        456
GRID           50              1800.   -200.      0.                        456
GRID           51              2250.    175.      0.                        456
GRID           52              2250.   157.5    350.                        456
GRID           53              2250. 128.948   -350.                        456
GRID           54              2250.94.06251700.0001                        456
GRID           55              2250.82.89521-700.000                        456
GRID           56              2250.-82.8956-700.000                        456
GRID           57              2250.-128.948   -350.                        456
GRID           58              2250.-134.375700.0001                        456
GRID           59              2250.   -175.    350.                        456
GRID           60              2250.   -175.      0.                        456
GRID           61              2700.    150.      0.                        456
GRID           62              2700.    135.    300.                        456
GRID           63              2700. 110.527   -300.                        456
GRID           64              2700.84.87501    600.                        456
GRID           65              2700. 71.0535   -600.                        456
GRID           66              2700.-71.0537   -600.                        456
GRID           67              2700.-110.527   -300.                        456
GRID           68              2700. -121.25    600.                        456
GRID           69              2700.   -150.    300.                        456
GRID           70              2700.   -150.      0.                        456
GRID           71              3150.    125.      0.                        456
GRID           72              3150.   112.5    250.                        456
GRID           73              3150.92.10591   -250.                        456
GRID           74              3150.75.68751    500.                        456
GRID           75              3150.59.21181   -500.                        456
GRID           76              3150.-59.2119   -500.                        456
GRID           77              3150.-92.1059   -250.                        456
GRID           78              3150.-108.125    500.                        456
GRID           79              3150.   -125.    250.                        456
GRID           80              3150.   -125.      0.                        456
GRID           81              3600.    100.      0.                        456
GRID           82              3600.90.00001    200.                        456
GRID           83              3600.73.68501   -200.                        456
GRID           84              3600.66.50001    400.                        456
GRID           85              3600.47.37001   -400.                        456
--------------------------------------------------------------------------------
```

(continued)

Table 8-4

```
GRID            86              3600.-47.3700   -400.                   456
GRID            87              3600.-73.6850   -200.                   456
GRID            88              3600.-95.0000    400.                   456
GRID            89              3600.   -100.    200.                   456
GRID            90              3600.   -100.      0.                   456
MAT1             1  79000.                0.33 2.80-6
PSHELL          33         1       1.
PSHEAR          65         1       1.
CQUAD4          33        33       5       14       12        2       0.
CQUAD4          34        33       2       12       11        1       0.
CQUAD4          35        33      14       24       22       12       0.
CQUAD4          36        33      12       22       21       11       0.
CQUAD4          37        33      24       34       32       22       0.
CQUAD4          38        33      22       32       31       21       0.
CQUAD4          39        33      34       44       42       32       0.
CQUAD4          40        33      32       42       41       31       0.
CQUAD4          41        33      44       54       52       42       0.
CQUAD4          42        33      42       52       51       41       0.
CQUAD4          43        33      54       64       62       52       0.
CQUAD4          44        33      52       62       61       51       0.
CQUAD4          45        33      64       74       72       62       0.
CQUAD4          46        33      62       72       71       61       0.
CQUAD4          47        33      74       84       82       72       0.
CQUAD4          48        33      72       82       81       71       0.
CQUAD4          49        33       1       11       13        3       0.
CQUAD4          50        33       3       13       15        4       0.
CQUAD4          51        33      11       21       23       13       0.
CQUAD4          52        33      13       23       25       15       0.
CQUAD4          53        33      21       31       33       23       0.
CQUAD4          54        33      23       33       35       25       0.
CQUAD4          55        33      31       41       43       33       0.
CQUAD4          56        33      33       43       45       35       0.
CQUAD4          57        33      41       51       53       43       0.
CQUAD4          58        33      43       53       55       45       0.
CQUAD4          59        33      51       61       63       53       0.
CQUAD4          60        33      53       63       65       55       0.
CQUAD4          61        33      61       71       73       63       0.
CQUAD4          62        33      63       73       75       65       0.
CQUAD4          63        33      71       81       83       73       0.
CQUAD4          64        33      73       83       85       75       0.
CQUAD4          81        33       6       16       18        8       0.
CQUAD4          82        33       8       18       20       10       0.
CQUAD4          83        33      16       26       28       18       0.
CQUAD4          84        33      18       28       30       20       0.
CQUAD4          85        33      26       36       38       28       0.
CQUAD4          86        33      28       38       40       30       0.
CQUAD4          87        33      36       46       47       38       0.
```

(continued)

Table 8-4

CQUAD4	88	33	38	47	50	40	0.
CQUAD4	89	33	46	56	57	47	0.
CQUAD4	90	33	47	57	60	50	0.
CQUAD4	91	33	56	66	67	57	0.
CQUAD4	92	33	57	67	70	60	0.
CQUAD4	93	33	66	76	77	67	0.
CQUAD4	94	33	67	77	80	70	0.
CQUAD4	95	33	76	86	87	77	0.
CQUAD4	96	33	77	87	90	80	0.
CQUAD4	97	33	10	20	19	9	0.
CQUAD4	98	33	9	19	17	7	0.
CQUAD4	99	33	20	30	29	19	0.
CQUAD4	100	33	19	29	27	17	0.
CQUAD4	101	33	29	39	37	27	0.
CQUAD4	102	33	30	40	39	29	0.
CQUAD4	103	33	40	50	49	39	0.
CQUAD4	104	33	39	49	48	37	0.
CQUAD4	105	33	50	60	59	49	0.
CQUAD4	106	33	49	59	58	48	0.
CQUAD4	107	33	60	70	69	59	0.
CQUAD4	108	33	59	69	68	58	0.
CQUAD4	109	33	70	80	79	69	0.
CQUAD4	110	33	69	79	78	68	0.
CQUAD4	111	33	80	90	89	79	0.
CQUAD4	112	33	79	89	88	78	0.
CQUAD4	113	33	17	14	12	19	0.
CQUAD4	114	33	19	12	11	20	0.
CQUAD4	115	33	20	11	13	18	0.
CQUAD4	116	33	18	13	15	16	0.
CQUAD4	117	33	27	24	22	29	0.
CQUAD4	118	33	29	22	21	30	0.
CQUAD4	119	33	30	21	23	28	0.
CQUAD4	120	33	28	23	25	26	0.
CQUAD4	121	33	37	34	32	39	0.
CQUAD4	122	33	39	32	31	40	0.
CQUAD4	123	33	40	31	33	38	0.
CQUAD4	124	33	38	33	35	36	0.
CQUAD4	125	33	48	44	42	49	0.
CQUAD4	126	33	49	42	41	50	0.
CQUAD4	127	33	50	41	43	47	0.
CQUAD4	128	33	47	43	45	46	0.
CQUAD4	129	33	58	54	52	59	0.
CQUAD4	130	33	59	52	51	60	0.

(continued)

Table 8-4

```
CQUAD4     131      33      60      51      53      57      0.
CQUAD4     132      33      57      53      55      56      0.
CQUAD4     133      33      68      64      62      69      0.
CQUAD4     134      33      69      62      61      70      0.
CQUAD4     135      33      70      61      63      67      0.
CQUAD4     136      33      67      63      65      66      0.
CQUAD4     137      33      78      74      72      79      0.
CQUAD4     138      33      79      72      71      80      0.
CQUAD4     139      33      80      71      73      77      0.
CQUAD4     140      33      77      73      75      76      0.
CQUAD4     141      33      88      84      82      89      0.
CQUAD4     142      33      89      82      81      90      0.
CQUAD4     143      33      90      81      83      87      0.
CQUAD4     144      33      87      83      85      86      0.
CSHEAR      65      65       5      14      17       7
CSHEAR      66      65      14      24      27      17
CSHEAR      67      65      24      34      37      27
CSHEAR      68      65      34      44      48      37
CSHEAR      69      65      44      54      58      48
CSHEAR      70      65      54      64      68      58
CSHEAR      71      65      64      74      78      68
CSHEAR      72      65      74      84      88      78
CSHEAR      73      65       4      15      16       6
CSHEAR      74      65      15      25      26      16
CSHEAR      75      65      25      35      36      26
CSHEAR      76      65      35      45      46      36
CSHEAR      77      65      45      55      56      46
CSHEAR      78      65      55      65      66      56
CSHEAR      79      65      65      75      76      66
CSHEAR      80      65      75      85      86      76
ENDDATA
```

(end of table)

Table 8-5: The command data file of the Dynamic Case Example.

```
==========================================================
START
COMMAND DATA PNEWTON
C
      SET NSIV=1;
      SET MASS=STR;
      SET MODE=1;
      SET MXIT=10;
      SET APRT=YES;
      SET ACTG=YES;
      SET IFLM=YES;
      DO NSIN;
      DO OPTIN;
C
      DO ANALYSIS;
      DO INSPECT;
      DO OPTX;
C     PSEUDO NEWTON ITERATIONS
L010: DO ITER;
      DO QNEWTON;
      DO OPTX;
      IF(ENDC.EQ.YES) GOTO L031;
      IF(NOIT.EQ.3) DO HISTORY;
      IF(NOIT.EQ.3) SUSPEND;
      IF(ENDI.EQ.YES) GOTO L040;
      GOTO L010;
L031: DISPLAY CONVERGENCE TO THE LOWER BOUND WEIGHT;
      GOTO L050;
L032: DISPLAY CONVERGENCE TO DESIGN VARIABLES;
      GOTO L050;
L033: DISPLAY CONVERGENCE TO FEASIBLE VS ACTUAL WEIGHT;
      GOTO L050;
L034: DISPLAY CONVERGENCE TO FEASIBLE WEIGHT;
      GOTO L050;
L040: DISPLAY MAX. NUMBER OF ITERATIONS REACHED;
C
L050: SET APRT=YES;
      DO ANALYSIS;
      DO INSPECT;
      DO HIST;
      STOP;
      END;
C     ANALYSIS PROCEDURE;
      PROC OPTX;
      DO ANALYSIS;
      DO ACTSET;
      DO DERV;
      DO CONV;
      DO INSPECT;
      END OPTX;
ENDATA
==========================================================
```

Table 8-6: The design/constraint file of the Dynamic Case Example.

```
==========================================================================
C234567890123456789012345678901234567890123456789012345678901234567890
DESIGN              1         33
DESIGN              1         34
DESIGN              1         49
DESIGN              1         50
DESIGN              1         81
DESIGN              1         82
DESIGN              1         97
DESIGN              1         98
DESIGN              2         35
DESIGN              2         36
DESIGN              2         51
DESIGN              2         52
DESIGN              2         83
DESIGN              2         84
DESIGN              2         99
DESIGN              2         100
DESIGN              3         37
DESIGN              3         38
                    .
                    to
                    .
DESIGN              3         47
DESIGN              3         48
                    .
                    to
                    .
DESIGN              3         63
DESIGN              3         64
                    .
                    to
                    .
DESIGN              4         95
DESIGN              4         96
DESIGN              4         101
DESIGN              4         102
                    .
                    to
                    .
DESIGN              4         111
DESIGN              4         112
DESIGN              5         65
--------------------------------------------------------------------------
```

(continued)

Table 8-6

```
DESIGN              5          66
                    .
                   to
                    .
DESIGN              5          79
DESIGN              5          80
DESIGN              6         113
DESIGN              6         114
                    .
                   to
                    .
DESIGN              6         143
DESIGN              6         144
DESIGN              6         117
GAUGE        0.600000    10.0000          33
GAUGE        0.600000    10.0000          34
                    .
                   to
                    .
GAUGE        0.600000    10.0000          63
GAUGE        0.600000    10.0000          64
GAUGE        0.600000    10.0000          81
GAUGE        0.600000    10.0000          82
                    .
                   to
                    .
GAUGE        0.600000    10.0000         111
GAUGE        0.600000    10.0000         112
GAUGE        0.600000     3.50000         65
GAUGE        0.600000     3.50000         66
                    .
                   to
                    .
GAUGE        0.600000     3.50000         79
GAUGE        0.600000     3.50000         80
GAUGE        0.600000     2.50000        113
GAUGE        0.600000     2.50000        114
                    .
                   to
                    .
GAUGE        0.600000     2.50000        143
GAUGE        0.600000     2.50000        144
NFREQ               1        2.50
ACTNFREQ            1
ENDATA
```

(end of table)

Table 8-7: The modified design/constraint file of the Dynamic Case Example.

```
================================================================================
C234567890123456789012345678901234567890123456789012345678901234567890
DESIGN              1         33
DESIGN              1         34
DESIGN              1         49
DESIGN              1         50
DESIGN              1         81
DESIGN              1         82
DESIGN              1         97
DESIGN              1         98
DESIGN              2         35
DESIGN              2         36
DESIGN              2         51
DESIGN              2         52
DESIGN              2         83
DESIGN              2         84
DESIGN              2         99
DESIGN              2         100
DESIGN              3         37
DESIGN              3         38
DESIGN              3         53
DESIGN              3         54
DESIGN              3         85
DESIGN              3         86
DESIGN              3         101
DESIGN              3         102
DESIGN              4         39
DESIGN              4         40
DESIGN              4         55
DESIGN              4         56
DESIGN              4         87
DESIGN              4         88
DESIGN              4         103
DESIGN              4         104
DESIGN              5         41
DESIGN              5         42
DESIGN              5         57
DESIGN              5         58
DESIGN              5         89
DESIGN              5         90
DESIGN              5         105
DESIGN              5         106
DESIGN              6         43
DESIGN              6         44
DESIGN              6         59
DESIGN              6         60
--------------------------------------------------------------------------------
```

(continued)

Table 8-7

DESIGN	6	91
DESIGN	6	92
DESIGN	6	107
DESIGN	6	108
DESIGN	7	45
DESIGN	7	46
DESIGN	7	61
DESIGN	7	62
DESIGN	7	93
DESIGN	7	94
DESIGN	7	109
DESIGN	7	110
DESIGN	8	47
DESIGN	8	48
DESIGN	8	63
DESIGN	8	64
DESIGN	8	95
DESIGN	8	96
DESIGN	8	111
DESIGN	8	112
DESIGN	9	113
DESIGN	9	114
	.	
	to	
	.	
DESIGN	9	143
DESIGN	9	144
DESIGN	10	65
DESIGN	10	73
DESIGN	11	66
DESIGN	11	74
DESIGN	12	67
DESIGN	12	75
DESIGN	13	68
DESIGN	13	76
DESIGN	14	69
DESIGN	14	77
DESIGN	15	70
DESIGN	15	78
DESIGN	16	71
DESIGN	16	79
DESIGN	17	72
DESIGN	17	80

(continue)

Table 8-7

```
-------------------------------------------------------------------
GAUGE          0.600000   10.0000          33
GAUGE          0.600000   10.0000          34
                    .
                   to
                    .
GAUGE          0.600000   10.0000          63
GAUGE          0.600000   10.0000          64
GAUGE          0.600000   10.0000          81
GAUGE          0.600000   10.0000          82
                    .
                   to
                    .
GAUGE          0.600000   10.0000         111
GAUGE          0.600000   10.0000         112
GAUGE          0.600000   3.50000          65
GAUGE          0.600000   3.50000          66
                    .
                   to
                    .
GAUGE          0.600000   3.50000          79
GAUGE          0.600000   3.50000          80
GAUGE          0.600000   2.50000         113
GAUGE          0.600000   2.50000         114
                    .
                   to
                    .
GAUGE          0.600000   2.50000         143
GAUGE          0.600000   2.50000         144
NFREQ                 1      2.50
ACTNFREQ              1
ENDATA
===================================================================
```

(end of table)

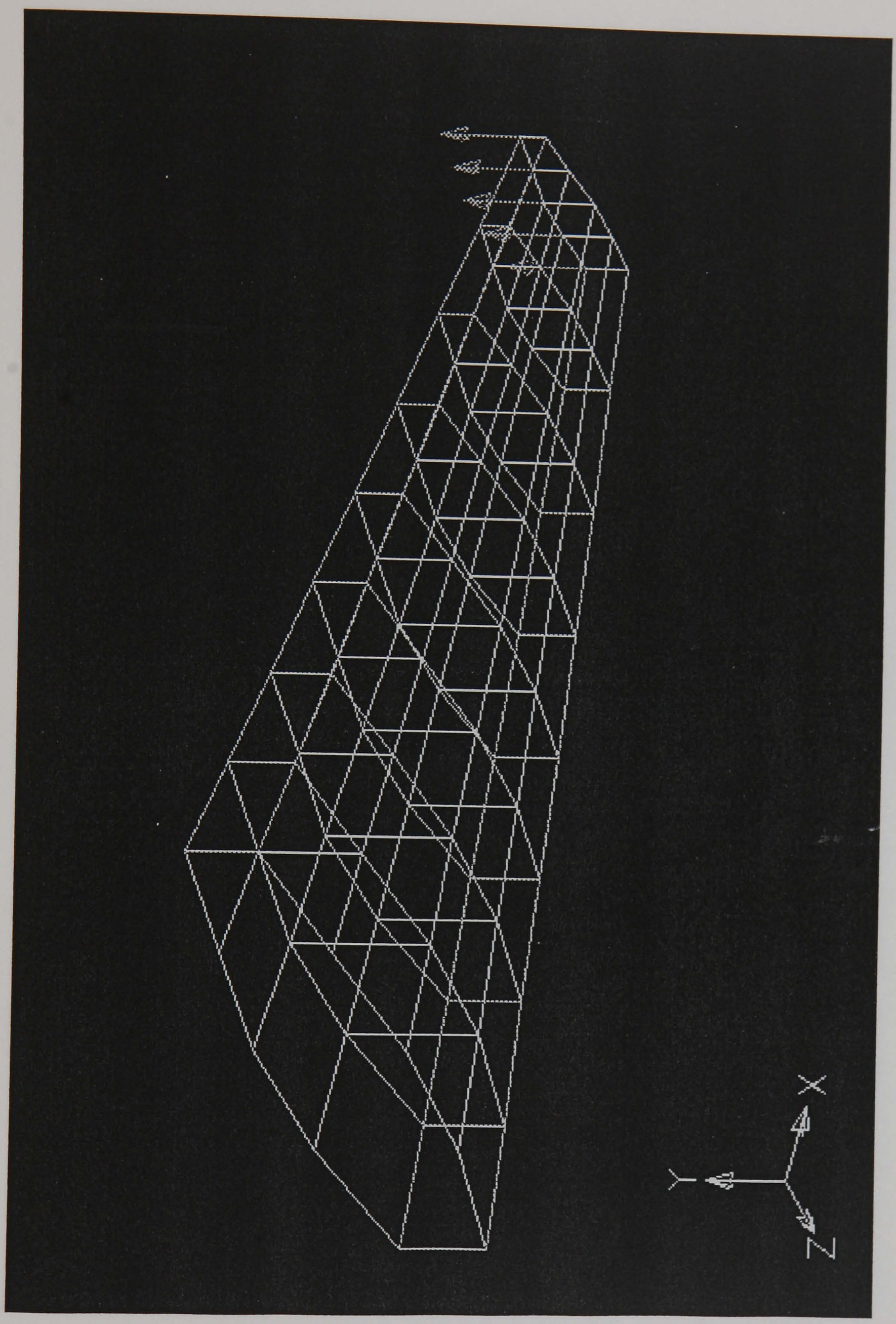


Figure 8-1 The FE model of the Static Case Example.

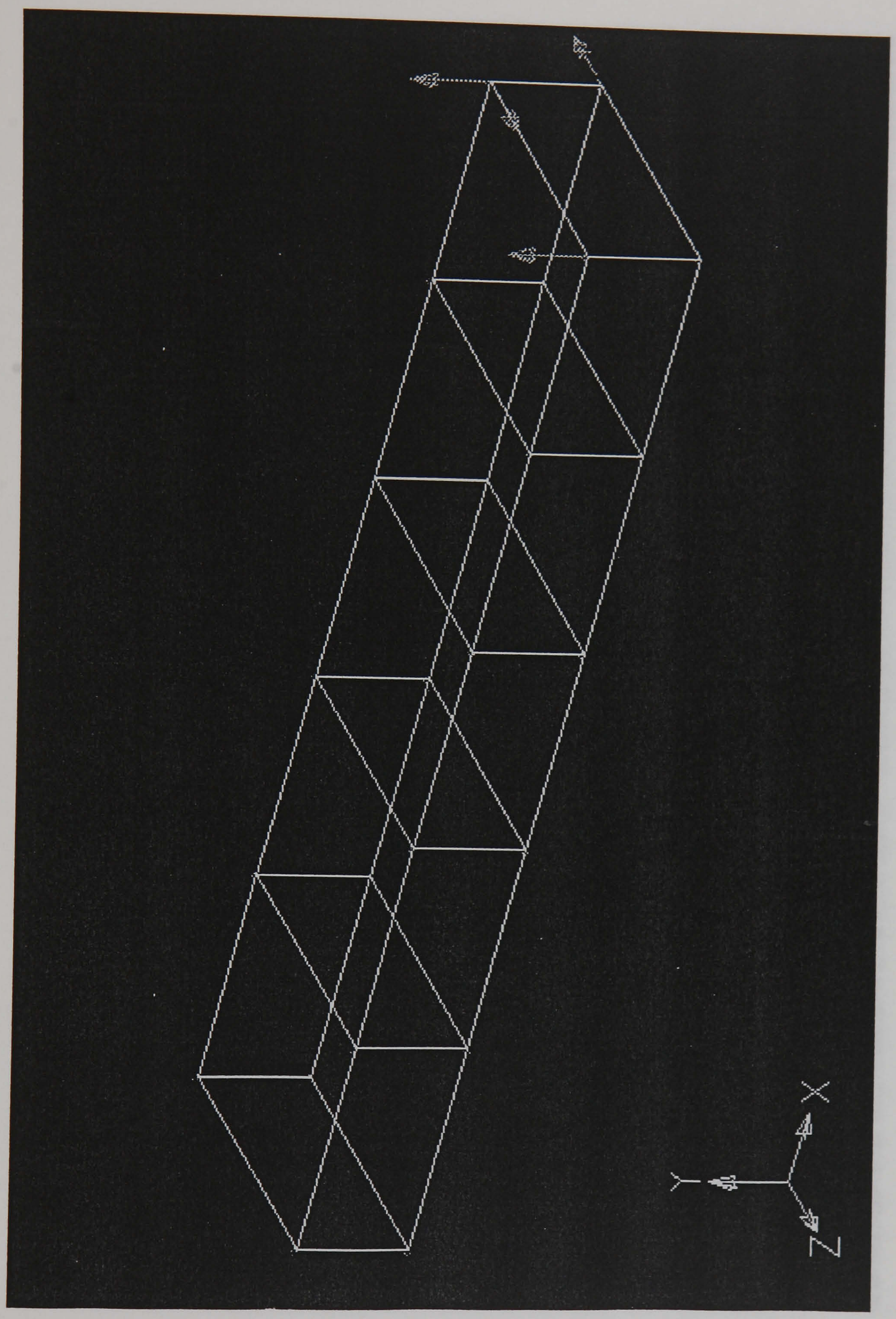


Figure 8-2 The FE model of the "similar" static past case.

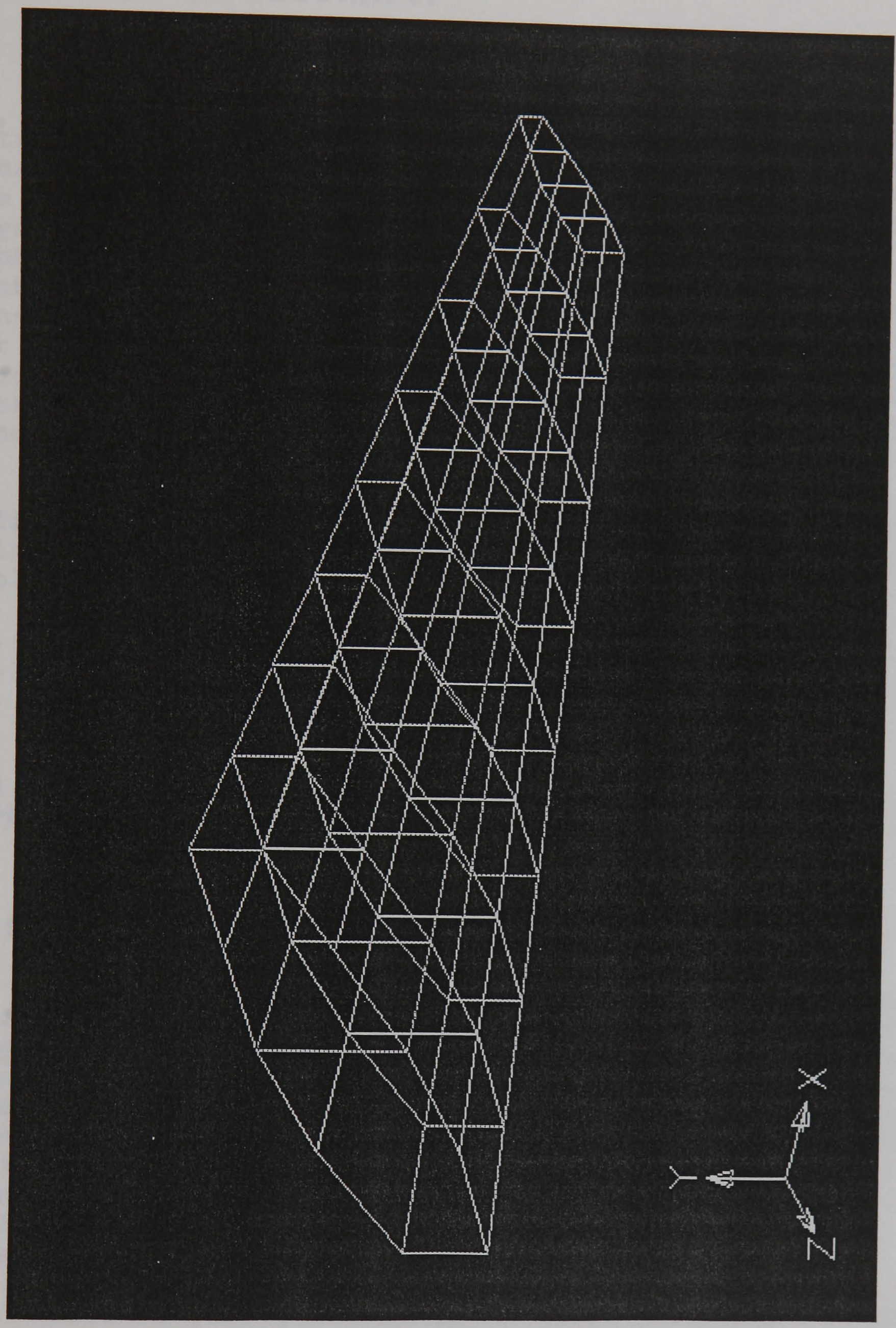


Figure 8-3 The FE model of the "similar" dynamic past case.

CHAPTER 9

FURTHER DEVELOPMENT

The research and development of an Expert System for structural optimization has been discussed in the previous chapters. Currently, the system prototype (ESSO) functions as an adviser for the user of structural optimization systems. It assists the user in constructing the optimization strategy, monitoring the optimization run, interpreting the optimization results, and in case of unsatisfactory results modifying the optimization set up. At this present stage ESSO can deal with static and dynamic problems and includes within the constraint set limits on stress, displacement, natural frequency, dynamic amplitude, and element gage.

However, in order for the prototype to work more effectively in dealing with the design optimization of aircraft structures it is necessary to investigate the following aspects:

1. Deepening the present knowledge.
2. Extending the knowledge to include aeroelastic design.
3. Extending the knowledge to include the design for manufacturing.

In the following sections, the above aspects and their implication for the ESSO are discussed in more detail.

9.1. DEEPENING THE PRESENT KNOWLEDGE.

Other than direct inputs from experts, relevant inputs research papers are an important sources of knowledge for the expert system for structural optimization. For example, research done by Kellerer [1992] gives a better understanding of the effect of design variable linking on the optimization of wing structures under static loading. This shows the effect of design variable definition for the wing region (spars, ribs, upper and lower panels), and the influence of the number of design variables in characterising the optimization problem.

Brooker [1992] attempted to reduce vibration levels of helicopter structures through structural redesign utilising optimization techniques. Both STARS and NASTRAN solution 200 were used in his research. This provides considerable insight into the use of optimization for dynamic modelling of structures.

Currently the expert system described here only deals with the isotropic materials. To be able to deal effectively with modern structures, the system should also posses the capability to work with composite structures. The system should be able to advise the user in the following aspects:

1. Selection of suitable material types such as, carbon, kevlar, or glass fibres, for the parts to be designed.
2. Combination of ply orientations in a laminate.

It is unnecessary to include all possible ply orientations (from 0 to 90 degree), from practical point of view it is sufficient to consider [0], [90], and [45/-45] only.

Much of the decision making in a design process is based on the functionality of the components, the type of forces applied to the structure (axial, shear, etc.), and the environment in which the components operate. To deepen the structural optimization knowledge of the system, it is necessary to be more specific about the functionality of the structure. With regard to the aircraft structure, the expert system should classify the structures into wing, fuselage, horizontal stabilizer, fin, engine nacelle, undercarriage, etc. The decision on the linkage of design variables, material selection, advice on the type of finite elements used, depend on the functionality of the component. For example, in setting up the design variables, wing spars is not usually linked to the upper or lower panel; the shear element (CSHEAR in NASTRAN) is normally used to model the spar structure; to effectively carry the shear load, a [45/-45] laminate or a laminate with a high portion of plies in this orientation is normally used if composite material is preferred. In the upper wing panel, it is normally preferred to link the panel in one bay (between two ribs) to a single design variable; Al-7075 is appropriate if isotropic material is selected, or if composite material is preferred then a carbon laminate with majority of plies orientated to [0] is appropriate. A plate element is normally used for the finite element modelling and the user advised not to forget the compression buckling constraint. The kind of knowledge mentioned above is very much depended on the functionality of the part/component to be designed.

Figure 9-1 shows a simplified diagram of an aircraft structural component with arrows indicating the flow of information. It is interesting to note that knowledge stored in the spar is not only appropriate for the wing but also for the fin and horizontal stabilizer. This suggests that there are a number of similarities between the design considerations for, and structural modelling of spars for wing, fin, or horizontal stabilizer.

9.2. DESIGN OPTIMIZATION FOR AEROELASTICITY.

Aeroelasticity is concerned with those physical phenomena which involve significant mutual interaction among inertial, elastic and aerodynamic forces [Dowell et al. 1989]. This section is concerned only with structural optimization which includes flutter as a design requirement. Structural optimization with aeroelastic design requirements has been performed in the past and is reported in several reports [Souahi 1986; Kiusalaas 1972; Lansing et al. 1977; Wilkinson et al. 1976; Haftka 1973].

Flutter calculation is concerned with finding the critical flutter speed and involves [Rodden et al. 1979]:

1. Modelling the structure with finite elements.
2. Calculating the aerodynamic forces based on geometric and the aerodynamic considerations.
3. Performing a modal analysis which requires the solution of the free-vibration eigenvalue problem.
4. Performing a flutter analysis by solving the appropriate flutter equation.

The task involve in flutter optimization is enormous. This is due to the fact that a complete description of the aeroelastic behaviour of the structure must be provided at each step of the optimization routine requiring the solution of a free-vibration eigenvalue problem, the interconnection of structural and aerodynamic grids, and the generation of unsteady air forces. The theory used in developing air forces depends on flight speed (subsonic, supersonic, or transonic). The flutter solution is defined in terms of complex eigenvalues and the corresponding complex eigenvectors. The details of the approach used to perform flutter optimization is outside the scope of this thesis and should be consulted in the relevant references some of them are already mentioned above.

To optimize a structure with respect to strength and flutter requirements, it is common to obtain a design which is strength optimum followed by flutter optimization. The strength optimum design can be achieved by the fully stressing method. It is known that the fully stressing method does not always produce the least weight design. However it is unnecessary to use more advanced method to meet the strength requirement as at the end it is the flutter requirement which is more likely to be active.

The flutter optimization phase can be performed by two approach. The first approach consists of two stages; the first stage involves a structural redesign to achieve the required flutter speed; the second stage minimized the weight by keeping the flutter speed constant. In the second

approach, the weight is minimized with at the same time trying to meet the flutter speed requirement.

The knowledge base required to assist the designer in optimizing for strength and flutter requirements should include the following guidelines:

1. An indication that the system should optimize the structure for strength requirements first (possibly using the fully stressed design method).
2. For this strength optimization a large number of design variables can be used.
3. Due to the enormous calculation required, a large number of design variables is prohibitive in flutter optimization. Hence the flutter optimization stage is divided into two stages.
4. The first involves optimizing the structure with a reasonably small number of design variables.
5. The second stage increase the number of design variables in the region where the elements are more effective (high constraint sensitivity).

9.3. DESIGN FOR MANUFACTURING KNOWLEDGE.

9.3.1. IMPLICATION OF MANUFACTURING TO DESIGN PROCESS.

The manufacturing cost of an aircraft structure (airframe) is a significant factor in the success of the design. Thus it is necessary, during the design process, to consider the manufacturing implications of a design decision. The product should be designed so that it can be economically manufactured. This means that the designers should have information with respect to the manufacturing aspects.

A designer has to ensure that his design is not only functionally good but also optimum from a production point of view. A good design is one in which the components are designed for the selected material and manufacturing processes to meet a specific functionality. Thus, it is useless to have a structural design which is optimum (say, least weight) and meet all strength and stiffness constraints but too complicated and expensive to be manufactured. When a structural designer decides that a part should be made of composite instead of aluminium, or a part should be machined from solid instead of forging, the designer makes one of the most important decision in the whole design procedure by committing production to a particular manufacturing process. Such a decision should be made only after examining all possible manufacturing processes and the costs associated with it.

It is recognized that many designers do not posses the required manufacturing knowledge and rely on the input from production engineers when a manufacturing process decision has to be made. In an automated design process this gap should be bridged, and AI is potentially useful for this purpose. The manufacturing and production implications of the design can be relayed to the designer in the early stages of the design process through the use of AI techniques. Research on this subject has been performed by Saggu [1991] using the design of aircraft flap as a specific example. The rest of this section is devoted to the implications of design for manufacturing for the structural optimization knowledge base.

9.3.2. DESIGN FOR MANUFACTURING IN EXPERT SYSTEM FOR STRUCTURAL OPTIMIZATION.

In performing a structural design and moving that design into manufacturing stage, optimization is part of the design stage. Any optimization method is mathematical in nature. This suggests that providing the problem can be formulated in mathematical terms, it is probable that the optimization process will run happily and find the optimum (say, minimum weight) without taking into account the manufacturability of the design.

It is, however, possible for part of the optimization task set up process to take into account manufacturing aspects in a limited way via minimum gauges on thickness. It is known, for example, that a milling process should not be applied to machine sheet plate below certain thickness due to problems of vibrations in the sheet. ESSO does not directly hold this kind of information in its knowledge base, hence it is the designers responsibility to specify the correct minimum thickness (associated with the intended machining process) for the gauge constraint.

Design for manufacturing is directly associated with material selection, the machining, and assembly process and their associated costs. These aspects should always be taken into account during the design process. To include these aspects in the ESSO knowledge base requires an extension to the present base. To perform this extension, the scheme mentioned in (9.1) and illustrated in figure 9-1 is still applicable because design for manufacturing is directly associated with the type of component being considered.

Let us consider an upper wing panel. From a functionality point of view, there are two material types which are suitable, those are Al-7075 aluminium (high strength aluminium) and carbon fibre composite. The final selection between the two depends not only on which

material is superior, or which material produced at least weight, but also on other factors such as:

- the material cost,
- the manufacturing capability possessed in producing the design made of aluminium or composite,
- the cost of producing the design made of aluminium or composite,
- the capability and the cost associated with parts assembly.

Clearly there is no point in designing the panel in composite, eventhough it produces a least weight structure, if the company does not have the production facility for manufacturing composite parts such as autoclave, tape laying, or composite material cutter. In this situation, the question arises as to whether it is cost effective to make new investments in composite production facilities.

If aluminium is selected it is still necessary to decide which type of panel is to be used, conventional (riveted skin-stiffener) type or integral (integral skin-stiffener machined from block of aluminium) type. Selecting the integral type of panel requires heavy milling machine facilities, especially for large airplanes. When conventional panel type is selected there will be questions such as what kind of cross-sectional shape adopted for stiffeners. From a buckling point of view, the use of a stiffer stiffener (high EI) is better, but this might lead to a more complicated shape (such as top-hat or Y stiffeners) which is more difficult and expensive to produce. There is also question as to whether plate bending process is adequate or extrusion process is needed for manufacturing stiffeners.

The discussion above suggests that design for manufacturing depends on the structural component being considered. As mentioned previously the scheme mentioned in (9.1) is appropriate to accommodate design for manufacturing. Each component can be treated as an object. The information relating to design for manufacturing associated with each component can be stored in the form of component objects. For example, with respect to manufacturing, an object could contain information relating to:

- possible material, either composite or aluminium,
- possible machining process and gauges associated with each type of process,
- assembly process (bonding, riveting),
- costs associated with a manufacturing type, etc.

In addition to the information associated with the manufacturing process, each object also contains

information related to the functionality of the component. In this respect, an object could contain:

- function of the component, e.g. ribs stabilize the panel and transfer air pressure on panels, webs carry shear loads, etc,
- type of loading associated with the component, e.g. stiffeners mainly carry axial load, webs carry shear loads,
- possible material, e.g. AL-7075 aluminium for upper panel, AL-2024 aluminium for lower panel,
- if composite material, suggested laminate orientations, e.g. [45/-45] for shear webs,
- suggested finite element type, e.g. NASTRAN C-SHEAR for shear webs, CQUAD4 for panels,
- behavioral constraints required, e.g. buckling constraint for stiffener, von Misses strength for lower panel, etc.

With this scheme, the optimization process would be performed only when the task associated with design for functionality and manufacturing is completed.

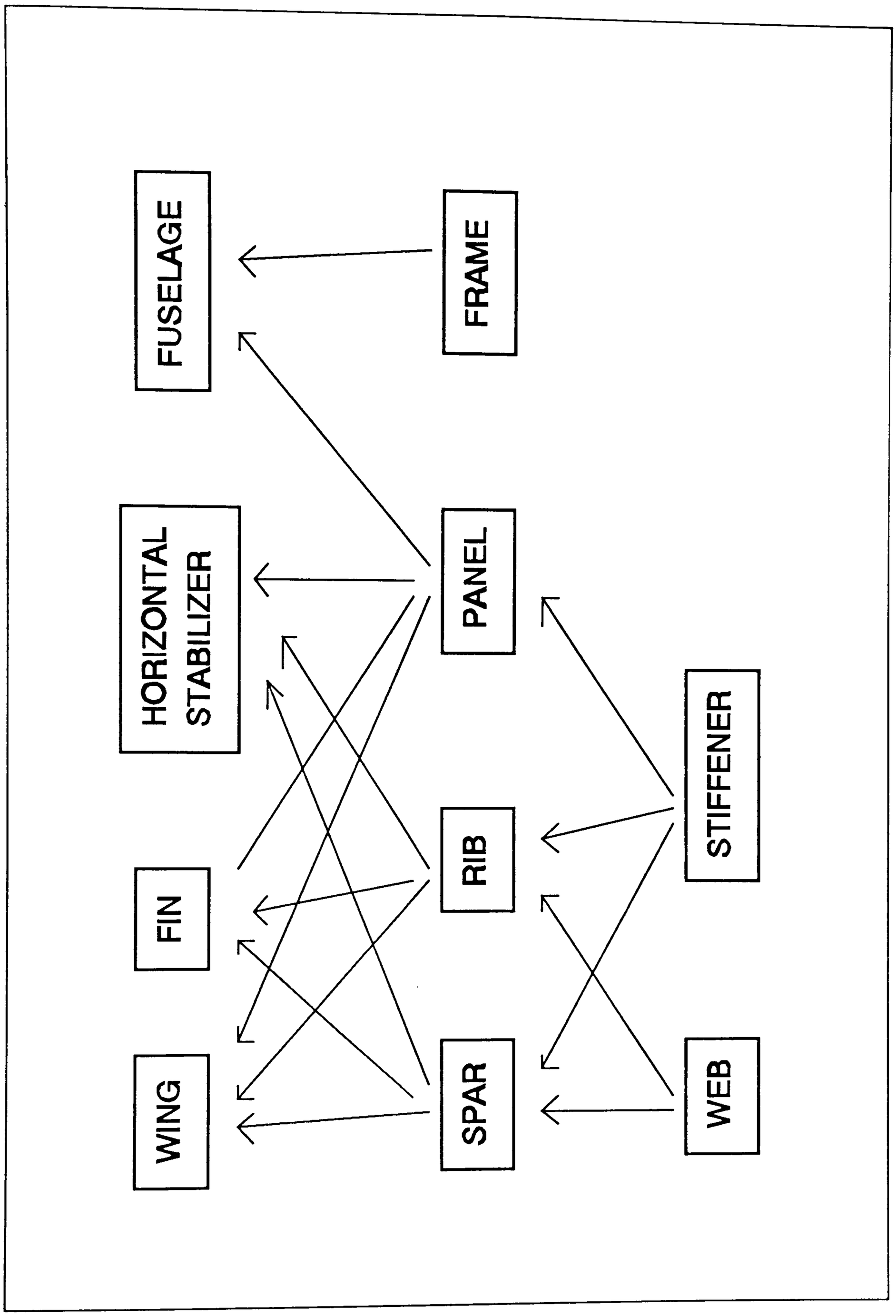


Figure 9-1 Simplified diagram of aircraft structural component.

SUMMARY

A research and development program focused on the creation of an Expert System for Structural Optimization has been carried out. The ideas and techniques presented in the preceding chapters have indicated the potential of using Expert System as an adviser for the user of working structural optimization systems.

Three types of knowledge are identified as necessary components for a structural optimization knowledge base in order for it to work effectively:

- knowledge relating to setting up the structural optimization based on logical deduction, and which is a-priori in nature,
- knowledge based on past experience,
- run-time and results interpretation knowledge.

Associated with each type of knowledge, a knowledge base structure and a reasoning strategy have been developed. The ideas and techniques advanced in this thesis have been implemented in the development of a prototype system called ESSO (Expert System for Structural Optimization).

This Expert System uses the Finite Element Method, in the form of MSC NASTRAN, for the analysis stage to deal with various types of structure. The DRA/PSI STARS structural optimization package is used for the optimization tool. Although the selection of a specific optimization (and Finite Element) package restricts the portability of the Expert System, it has validated the concept developed during the research programme.

The prototype was written using C++ on a SUN SPARC-II computer. This has been extended with the development of a user-interface which is based on XView. A User's Manual had been prepared which contains a guidance on using ESSO and details of the prototype programming. The prototype system had been trialed successfully using static and dynamic optimization problem examples.

For an Expert System for structural optimization to work effectively, it should have access to the optimization data (structural responses, constraint values, sensitivity data, objective function values, etc.) during system run. This issue becomes important for an Expert System which is intended to work with proprietary optimization system packages because the optimization codes (computer programs) are usually not available (for outsider). The direct application to a specific system requires an agreement which opens the way for the Expert System developer to have

access to the optimization package. Unfortunately, in the development of ESSO, we do not have means to access the optimization data during a STARS session. This means that for interpreting the optimization trend or results ESSO requires the users to read STARS output and enter the data manually for ESSO which is not an ideal way for an Expert System to work.

Research on Expert Systems for Structural Optimization is an on-going research and by no means complete. Further development is necessary and is anticipated. The author suggests that the "final" form of this Expert System should be an Expert System for aircraft structural design with structural optimization as part of the system. The Expert System should be extended to include the problems of aeroelasticity. The proposed system should also take into account the implication of manufacturing and production in structural design.

In chapter nine, the direction of further prototype development as mentioned above is discussed in detail. It is suggested that for the system to be able to deal effectively with aircraft structural design problems, it is necessary to be more specific with the functionality of the structures. This requires the system to classify the aircraft structure into its components such as wing, fuselage, fin, and treats these components as objects. Design for manufacturing is directly associated with the type of component being considered and its functionality, hence the classification of structure is considered to be the proper way to represent the knowledge.

REFERENCES

Adeli, H. and Balasubramanyam, K V. 1988.
Expert Systems for Structural Design.
Prentice Hall, New Jersey, USA.

Arora, J S. 1989.
Introduction to Optimum Design.
McGraw-Hill Book Company, 1st edition, USA.

Bartholomew, P. 1991.
STARS Theoretical Manual.
SD-Scicon UK Ltd., UK.

Bennet, J S. and Engelmore, R S. 1979.
SACON: A Knowledge Based Consultant for Structural Analysis.
Proceedings of the 6th International Joint Conference on Artificial Intelligence. Tokyo.

Berke, L., Patnaik, S N. and Murthy, P L N. 1993.
Application of Artificial Neural Networks to the Design Optimization of Aerospace Structural Component.
NASA Technical Memorandum 4389.

Booch, G. 1991.
Object Oriented Design with Applications.
The Benjamin/Cummings Publishing Company, Cal., USA.

Brooker, S M. 1992.
Vibration Control of Helicopters through Structural Optimization.
MSc Thesis, COA - Cranfield Institute of Technology, UK.

Buchanan, B G. and Shortliffe, E H. 1985.
Rule-Based Expert Systems.
Addison-Wesley Publishing Company, USA.

Castillo, E. and Alvarez, E. 1991.
Expert Systems: Uncertainty and Learning.
Computational Mechanics Publications & Elsevier Applied Scence. UK. 1991.

Chen, J L. and Hajela, P. 1987.
FEMOD : A Consultative Expert System for Finite Element Modeling.
29th Structures, Structural Dynamics and Materials Conference.

Coxhead, P. 1987.
Starting LISP for AI.
Blackwell Scientific Publications, UK.

CRYSTAL, The Expert System Builder.
Intelligent Environment.

Dowell, E H., Curtiss, H C,Jr., Scanlan, R H. and Sisto, F. 1989.
A Modern Course in Aeroelasticity.
Kluwer Academic Publishers, The Netherlands.

Dutta, S. and Bonissone, P. 1990.
Integrating Case Based and Rule Based Reasoning: The Possibilistic Connection.
INSEAD Working Papers 90/45/TM. France.

Felt, L R., Grisham, A F. and Dotson, B F. 1987.
Knowledge-Based (Expert) Systems for Structural Analysis and Design.
28th Structures, Structural Dynamics and Materials Conference.

Forsyth, R (Ed.). 1984.
Expert Systems - Principles and case studies.
Chapman and Hall Computing, UK.

Functional Specification for Remind Development .
Version 1.1
Cognitive Systems, Inc.

Gero, J S (Ed.). 1989.
Artificial Intelligence in Design.
Computational Mechanics Publications Springer-Verlag UK.

Gero, J S (Ed.). 1991.
Artificial Intelligence in Design '91.
Butterworth-Heidemann Ltd., UK.

Haftka, H. 1973.
Automated Procedure for Design of Wing Structures to Satisfy Strength and Flutter Requirements.
NASA TN D-7264.

Harris, J ap C; et.al. 1990.
An Application of an Expert System Techniques to Structural Optimization in a Fortran Environment.
RAE/TM/Mat/Str/1147.

Heller, D. 1990.
XView Programming Manual. An OPEN LOOK Toolkit for X11.
O'Reilly & Associates, Inc. USA.

Johnson, E H., Neil, D J., Herendeen, D L. and Canfield, CA. 1989.
Automated Structural Optimization System (ASTROS) - User Training Workshop. AFWAL-TR-88-3101.

Kellerer, H. 1992.
Influence of Structural Modeling and Design Variable Linking on the Results of Structural Optimization of a Wing-Box Structure under Stress Constraints. Diplomarbeit, Technische Universitat Munchen & COA - Cranfield Institute of Technology, UK.

Kirsch, U. 1981.
Optimum Structural Design.
McGraw-Hill Book Company, 1st edition, USA.

Kiusalaas, J. 1972.
Minimum Weight Design of Structures via Optimality Criteria. NASA TN D-7115.

Kolb, M A. 1992.
Constraint-Based Component-Modeling for Knowledge-Based Design. AIAA 92-1192.

Kolodner, J. 1993.
Case-Based Reasoning.
Morgan Kaufmann Publishers, Inc. USA.

Kuntjoro, W. 1991.
Preliminary Study on the Development of an Expert System for Structural Optimization.
MSc Thesis, COA - Cranfield Institute of Tecnology, UK.

Lansing, W., Lerner, E. and Taylor, R F. 1977.
Application of Structural Optimization for Strength and Aeroelastic Design Requirements. AGARD-R-664.

Levine, R., Drang, D. and Edelson, B. 1990.
AI and Expert Systems, A Comprehensive Guide.
McGraw Hill, 2nd edition, USA.

Luger, G F. and Stubblefield, W A. 1989.
Artificial Intelligence and The Design of Expert Systems.
The Benjamin/Cummings Publishing Company, Inc., Cal., USA.

Maher, M L. 1988.
HI-RISE : An Expert System for Preliminary Structural Design. In Rychener (1988).

Meyer, B. 1988.
Object Oriented Software Construction.
Prentice-Hall, 1st edition, UK.

Milne, R.
Case Based Reasoning - An Introduction. (Lecture Notes)
Intelligence Application Limited.

Miura, H. 1988.
MSC/NASTRAN Version 66, Handbook for Structural Optimization.
The Macneal-Schwendler Corporation.

Morris, A J. 1982.
Foundation of Structural Optimization : A Unified Approach.
John Wiley & Sons Ltd., 1st edition, UK.

MSC/NASTRAN Version 66A, User's Manual. 1989.
The Macneal-Schwendler Corporation.

MSC/XL Version 3, User's Manual. 1991.
The Macneal-Schwendler Corporation.

NAFEMS. 1986.
A Finite Element Primer.
DTI - NEL, Glasgow, UK.

Rivlin, J M., Hsu, M.B. and Marcal, P V. 1980.
Knowledge Based Consultation for Finite Element Structural Analysis.
AFWAL-TR-80-3069. USAF Flight Dynamics Laboratory, Wright-Patterson AFB, Ohio.

Rodden, W P., Harder, R L. and Bellinger, E D. 1979.
Aeroelastic Addition to NASTRAN. NASA CR 3094.

Rogers, J L. and Barthelemy, J M. 1987.
An Expert System for Choosing the Best Combination of Options in a General-Purpose Program for Automated Design Synthesis.
Engineering with Computers, Vol. 1.

Rychener, M D (Ed.). 1988.
Expert Systems for Engineering Design.
Academic Press, Inc., London.

Sacks, R. and Buyukozturk, O. 1988.
Expert Interactive Design of Reinforced-Concrete Columns under Biaxial Bending.
in, Expert Systems in Engineering
(Edited by Pham,DT).
IFS Publications/Springer-Verlag, UK.

Saggu, J S. 1991.
IKADE : An Intelligence Knowledge Assisted Design Environment Incorporating Manufacturing and Production Information.
PhD Thesis, COA - Cranfield Institute of Technology, UK.

Schildt, H. 1986.
Artificial Intelligence - Using C.
Osborne - McGraw-Hill, 1st edition, USA.

Schildt, H. 1987.
Advanced Prolog.
Osborne - McGraw-Hill, Berkeley, Cal., USA.

Schildt, H. 1990.
Using Turbo C++.
Osborne - McGraw-Hill, 1st edition, USA.

Shapiro, S C (Ed.). 1987.
Enciclopedia of Artificial Intelligence Vol. 1,
John Willey & Sons, USA.

Shapiro, S C (Ed.). 1992.
Reasoning, Case Based; in Encyclopedia of Artificial Intelligence Second Edition.
John Wiley & Sons, Inc.

STARS User Manual, Version 17. 1991.
SD-Scicon UK Ltd., UK.

STARS Application Manual, Version 17. 1991.
SD-SCICON UK Ltd., UK.

Souahi, A. 1986.
Structural Optimization of Aircraft Lifting Surfaces to Satisfy Flutter Requirements.
PhD Thesis, COA - Cranfield Institute of Technology, UK.

Sriram, D. 1987.
ALL-RISE : A Case Study in Constraint-Based Design.
in, The International Journal for Artificial Intelligence in Engineering, vol. 2, no. 4.

Swartout, W R. 1987.
Explanation. In Shapiro (1987).

Thomson, W T. 1986.
Theory of Vibration with Applications, 2nd edition.
George Allen & Unwin Ltd., London.

Vaish, A K. and Kamil, H. 1988.
An Expert System for Structural Analysis and Design.
29th Structures, Structural Dynamics and Materials Conference.

Vanderplaats, G N. 1984.
Numerical Optimization Techniques for Engineering Design : With Applications.
McGraw-Hill Book Company, 1st edition, USA.

Wallsgrove, R. 1988.
Explanation in Expert Systems.
MSc Thesis, College of Manufacturing, Cranfield Institute of Technology, UK.

Wilkinson, K., Lerner, E. and Taylor, R F. 1976.
Practical Design of Minimum Weight Aircraft Structures for Strength and Flutter Requirements.
Journal of Aircraft, Vol.13 No.8.

Yazdani, M (Ed.). 1986.
Artificial Intelligence, Principles and Applications.
Chapman and Hall, 1st edition, UK.

Zadeh, L A. 1983.
Commonsense Knowledge Representation Based on Fuzzy Logic.
Computer, October 1983.

CRANFIELD UNIVERSITY

COLLEGE OF AERONAUTICS
Department of Aerospace Science

Ph.D THESIS

Academic Year 1992-1994

WAHYU KUNTJORO

Expert System for Structural Optimization Exploiting Past Experience and A-priori Knowledge

Volume 2: User Manual of the Expert System for Structural Optimization (ESSO)

SUPERVISOR: PROF. A.J. MORRIS

NOVEMBER 1994

CONTENTS

LIST OF FIGURES

CHAPTER 1:

BASICS OF OPEN LOOK GRAPHICAL USER INTERFACE

The user interface for the Expert System for Structural Optimization (ESSO) is implemented using the XView (XWindow-System-Based Visual/Integrated Environment for Workstations) toolkit running under the X Window System. XView provides a set of prebuilt, user-interface objects such as canvases, scrollbars, menus and control panels. XView is written in the C language. The appearance and functionality of these objects follow the OPEN LOOK Graphical User Interface (GUI) specification. Basic operations of the OPEN LOOK GUI is presented in this chapter. Only subjects which are directly related to the Expert System for Structural Optimization (ESSO) are presented.

1.1. OVERVIEW OF X WINDOW SYSTEM.

X controls a bit-mapped display in which every pixel (dot on the screen) is individually controllable. This allows drawing of pictures in addition to text. Like other windowing systems, X divides the screen into multiple input and output areas called windows. Each window can act as an independent virtual terminal. Using a terminal emulator, not only can windows run ordinary text-based applications but also applications designed to take advantage of the graphic power of the bitmapped display.

Similar to other window systems, the user provides input through a pointer, usually a mouse but could just as well be a trackball or tablet, and a keyboard. The pointer allows the user to point at certain graphics on the screen and use the buttons on the mouse to control a program without using the keyboard. It is also used to direct the input focus of the keyboard from window to window with only one application at a time able to receive keyboard input.

Each application in X need not consist of only a single window. Any part of an application can have its own separate window. Such child windows are only visible within the confines of their parent window. An application can also creates multiple frames, each of which has its own window (or windows).

XView provides a set of windows that include:

- Canvases on which programs can draw.
- Text subwindows with built-in editing capabilities.

- Panels containing items such as buttons, choice items, and sliders.
- TTY subwindows that emulate character-based terminals.

These windows are arranged as subwindows within frames, which are themselves windows. Frames can be transitory or permanent. Transient interactions with the user can also take place in menus which can pop-up anywhere on the screen.

As mentioned before, the XView implements the OPEN LOOK GUI. The OPEN LOOK can be implemented on any operating system, windowing system or graphic display. It can be mapped easily into the X Window System. Figure 1-1 shows a sample of OPEN LOOK workspace.

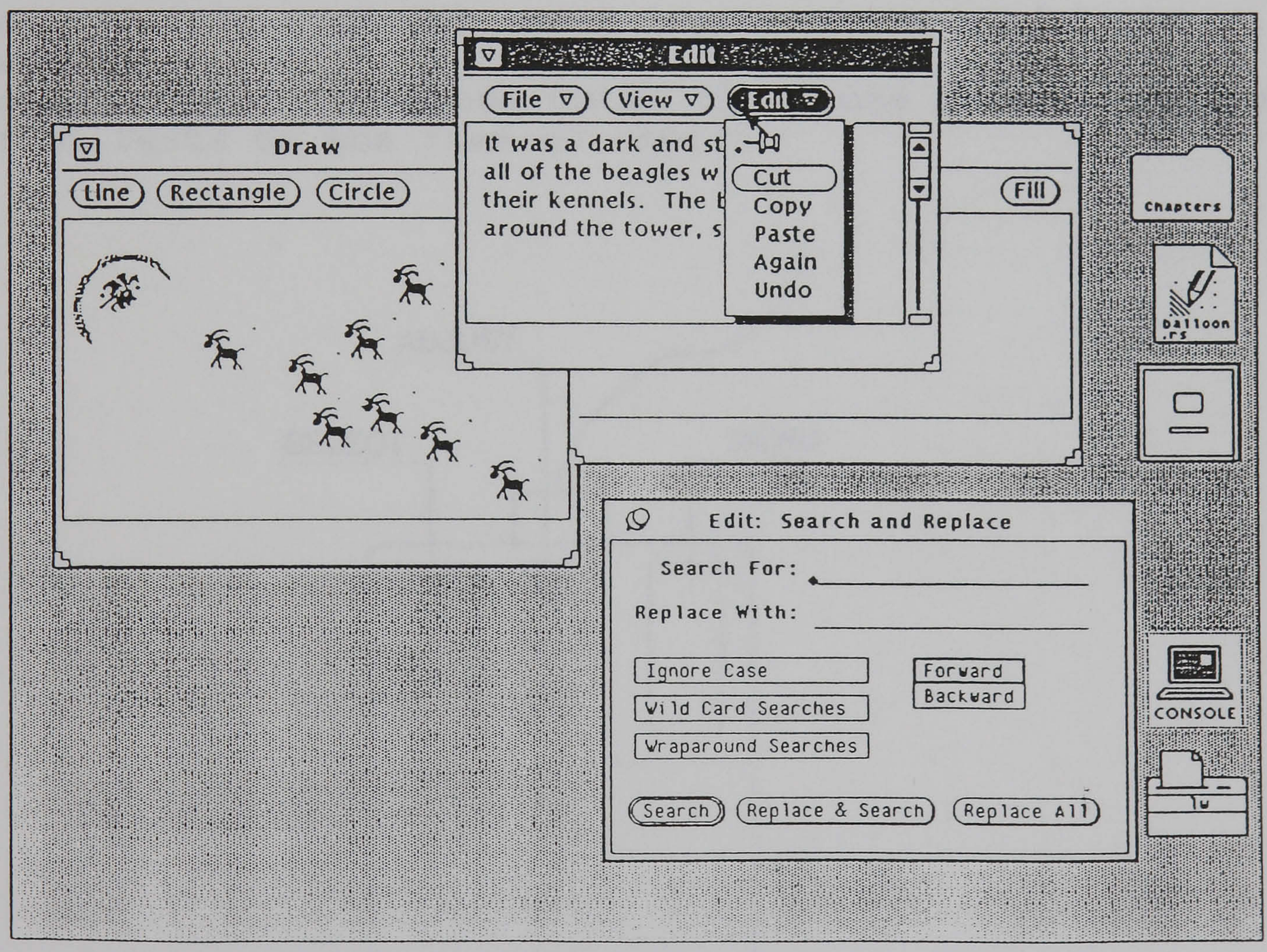


Figure 1-1 A Sample of OPEN LOOK workspace
(from Heller [1990])

1.2. MOUSE.

The operation on a window can be activated through a mouse. The mouse is primarily used to move the cursor (symbolized usually as an arrow or a solid box) around the windows. The mouse has three buttons (see Figure 1-2) which in XView their actions are defined as:

- The **Left Mouse Button** (also called **select** button) which is normally used to select or activate an operation associated with a panel button,
- the **Middle Mouse Button** (also called **adjust** button) which can be set to activate certain operations depending on the application, and
- the **Right Mouse Button** (also called **menu** button) which is normally used to display menu from which a selection is made.

The basic terminologies which are related to mouse operation are:

- **Click**, to quickly press and release the mouse button.
- **Double-click**, to click the mouse button twice in rapid succession.
- **Drag**, to hold down the mouse button while moving the mouse.
- **Point**, to move the mouse until the mouse pointer on the screen rests on the item of choice.

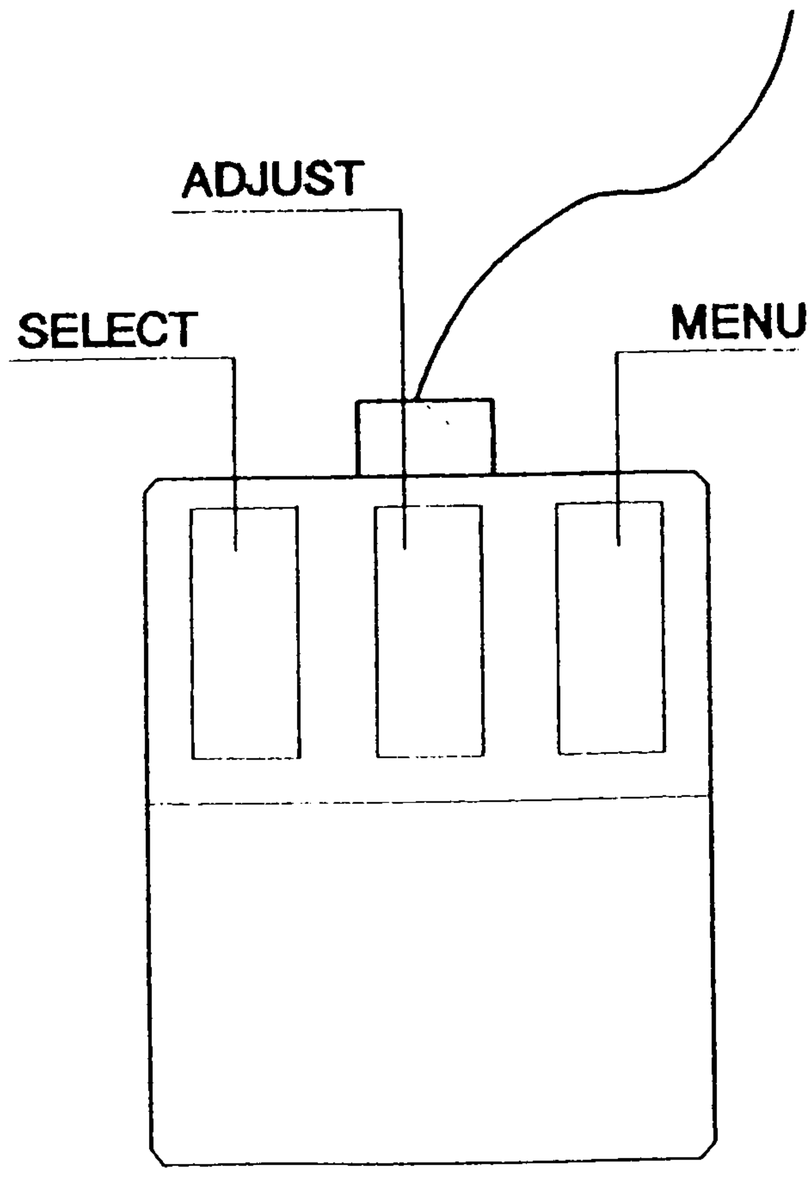


Figure 1-2 Three-button mouse.

1.3. FRAME.

A frame is a container of other windows. It manages the geometry and placement of subwindows that do not overlap and are fixed within the boundary of the frame. In OPEN LOOK, the subwindows in a frame do not overlap one another. However frames might overlap. Subwindow types include canvases, text subwindows, panels, and scrollbars. These subwindow cannot exist without a parent frame to manage them. Figure 1-3 shows an example of a screen that displays several frames, each one containing at least one subwindow.

Figure 1-4 shows a typical frame and its elements.
- **Title bar** is part at the top of the frame and usually contains the frame label. The title, Edit No file, suggests that this is a frame which contain a Text subwindow.

- **Resize corner** is used to change the size of a frame. This is performed by pointing the cursor to the resize corner push the select button and drag the frame corner to its required new size.

- **The close button** is used to close (iconize) the frame (and its subwindows).

- **The control panel** (if available) contains one or more panel items. In our example it has 'Button' button and 'Menu Button' menu. A menu button is recognized by the small triangle near it. To activate a panel button, place the cursor on top of the button and select it by clicking the select (left) button of the mouse.
To activate the menu button, point to the menu, push the mouse's menu (right) button, drag the mouse to the required menu-item (a file, a value, or others) and release the mouse's button to select.

- **The vertical scrollbar** is used to scroll the window up and down. If required, the horizontal scrollbar can also be created.

1.4. PANELS.

A panel (or control area) is an unbordered region of a window where controls such as buttons and settings are displayed. The panel shown in Figure 1-4 represents the typical positioning of the control area, that is under the title bar and above the subwindow. Figure 1-5 shows examples of panel items from the OPEN LOOK GUI.

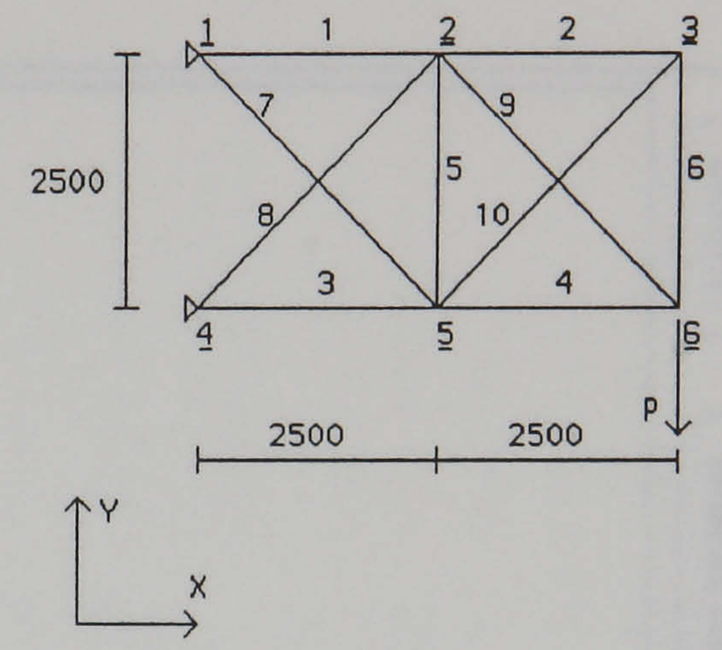
EXPERT SYSTEM FOR STRUCTURAL OPTIMIZATION

Quit | MSC/XL | STARS | Run ES | EditFile | Example | Proposed_case | Explain_reason | ReadMe_first

coas3{aep103}41:

Example1

CANTILEVER-TRUSS



Task: Optimize the structure shown.

A truss structure with 6 nodes and 10 mem-, bers with a static load P at node 6.
The structure was modelled using NASTRAN, truss members were modelled with CROD.
The tensile strength is 400 MPa, compressive strength is 250 MPa, E = 69,000 MPa, and Poisson's ratio is 0.3.
The downward displacement at node 6 should be less than 50 mm.
P = 10,000 N.
At optimum, the size of members 1,2,3,4 must be the same, and members 5,6,7,8,9,10 also must have the same size (could be different from members 1,2,3,4).

EXPERT SYSTEM FOR STRUCTURAL OPTIMIZATION

GIVE DESCRIPTION OF STRUCTURE
1. Truss
2. Frame
3. Full-monocoque structure
4. Semi-monocoque structure
5. Solid
6. Other
Type in your structural description (1-6) : 1

Figure 1-3 A screen with several frames.

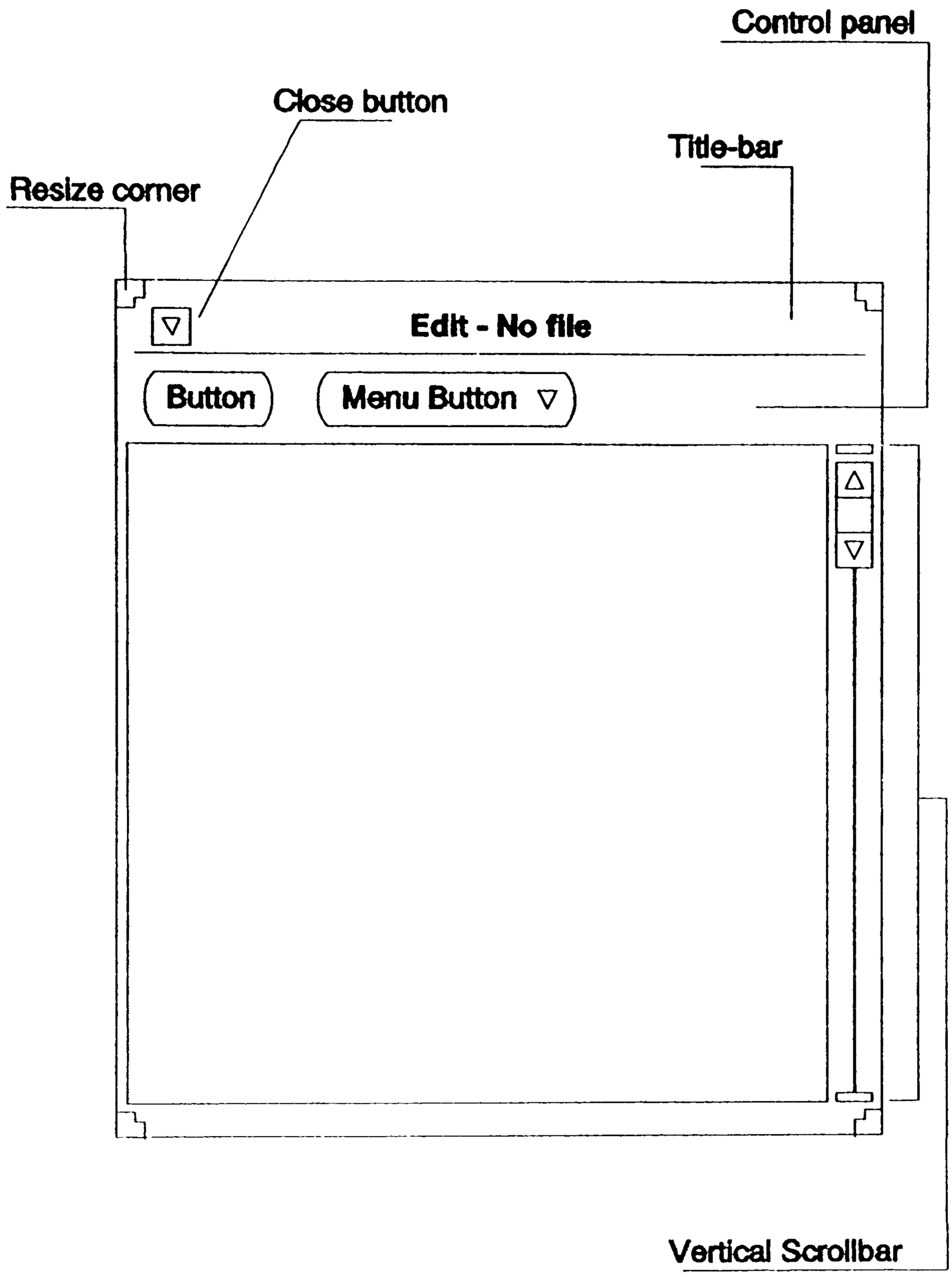


Figure 1-4 A typical frame and its elements.

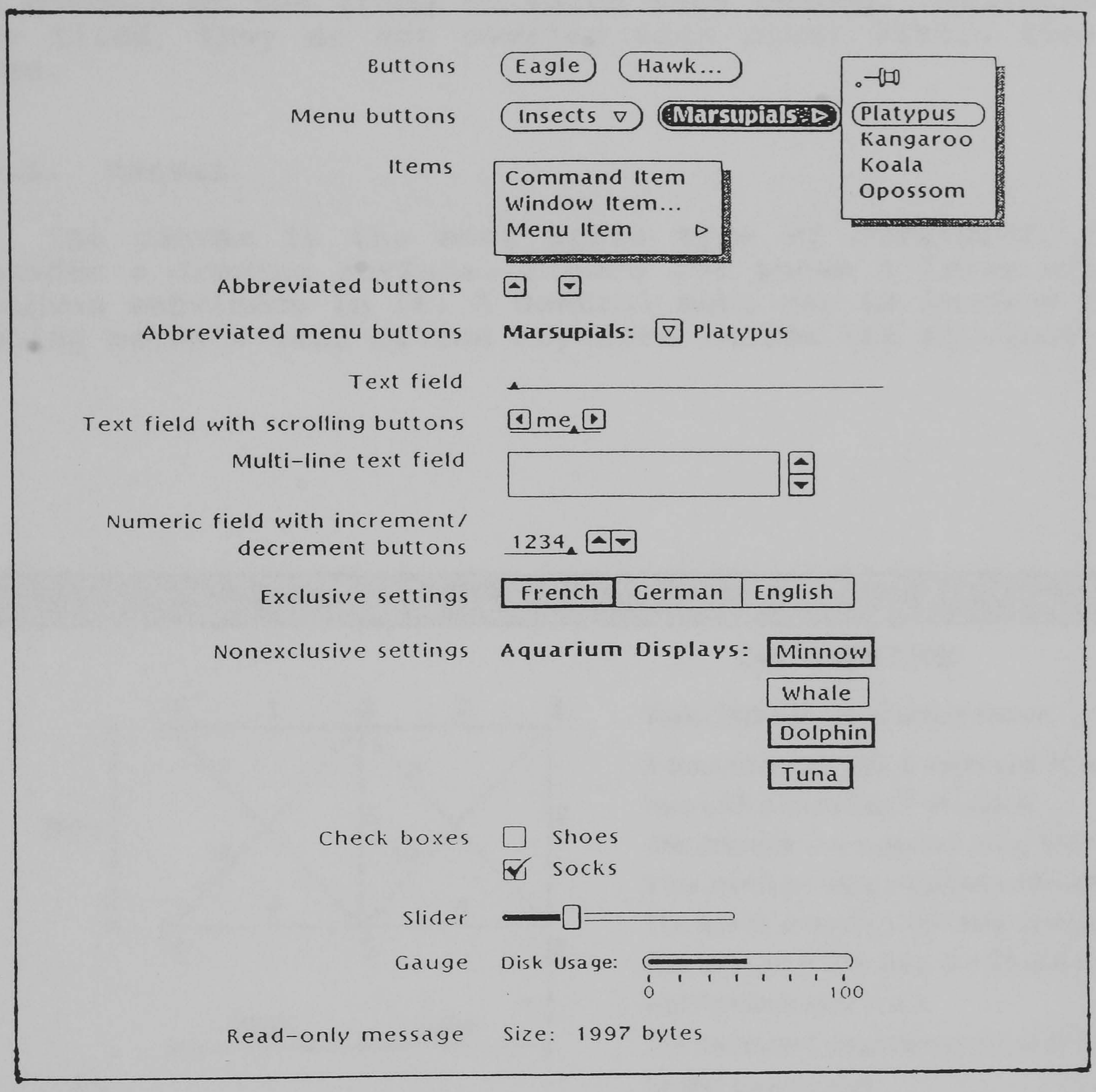


Figure 1-5 Panel items in OPEN LOOK
(from Heller [1990])

1.5. SUBWINDOWS.

Subwindows differ from frames in several basic ways. They never exist independently; they are always owned and maintained by a frame or another window, and they may not themseleves own frames. While frames can be moved freely around the screen, subwindows are constrained to fit within the borders of the frame to which they belong. Subwindows also tiled, they do not overlap each other within their frame.

1.5.1. Canvas.

The canvas is the most basic type of subwindow. It provides a drawing surface. Figure 1-6 shows a frame with a canvas subwindow in it. A control menu can be invoked by pushing mouse's menu button anywhere inside the subwindow.

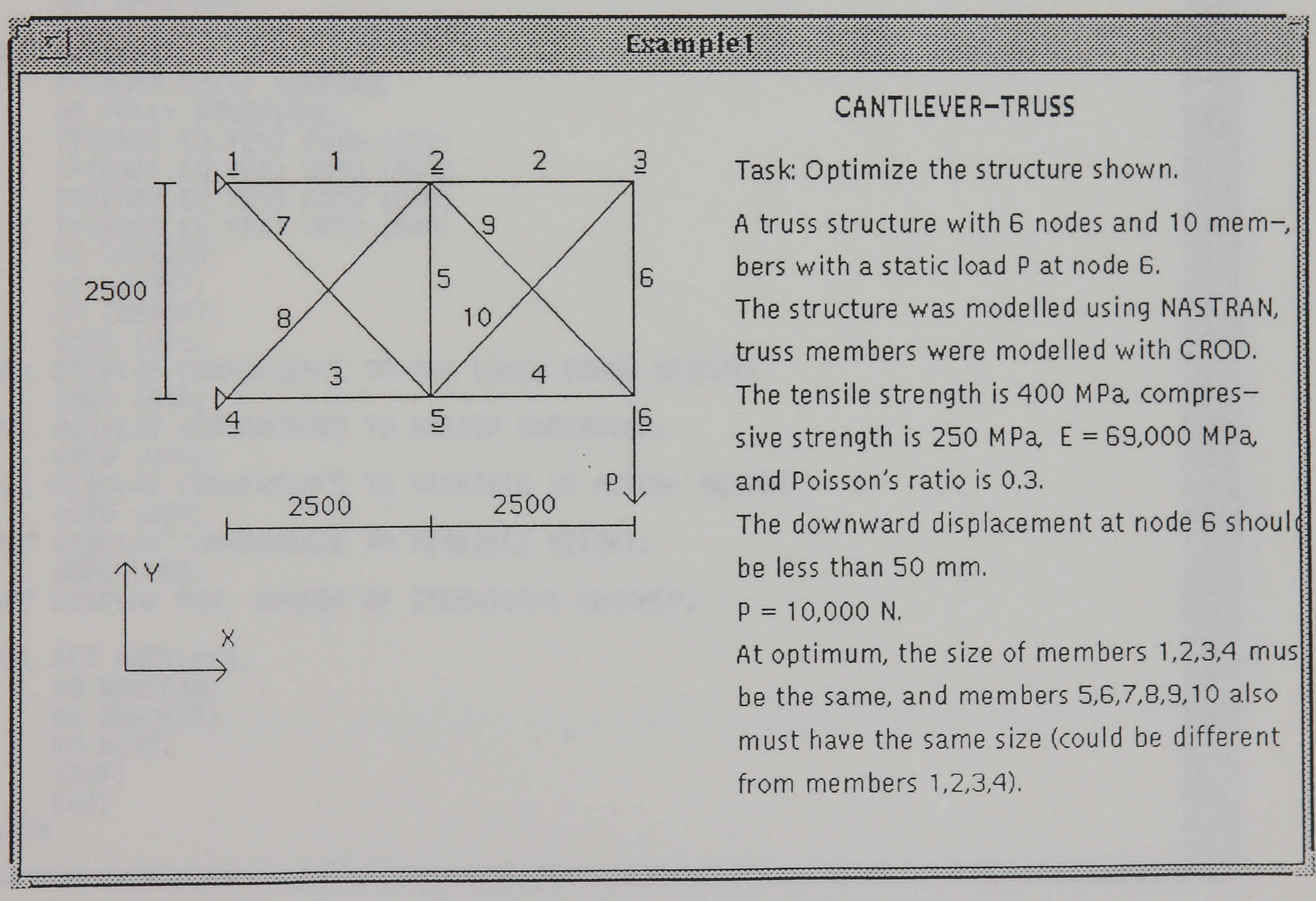


Figure 1-6 Canvas subwindow.

1.5.2. Text Subwindow.

Another basic window type is a text subwindow. It provides basic text editing capabilities. Figure 1-7 shows a frame with a text subwindow in it. Similar to a canvas subwindow, a control menu can be invoked by pushing mouse's menu button anywhere inside the subwindow.

EditWindow

```
START
COMMAND DATA FSD
C
      SET MXIT=8;
      SET APRT=YES;
      SET ACTG=YES;
      DO NSIN;
      DO OPTIN;
C
      DO NASTRAN;
      DO ACTSET;
      DO INSPECT;
      SET APRT=NO;
C     FULLY STRESSING ITERATIONS
L010: DO ITER;
      IF(NOIT.EQ.4) SUSPEND;
      DO FULLY STRESSING;
      IF(ENDT.EQ.YES) GOTO L032;
      IF(ENDF.EQ.YES) GOTO L033;
      IF(ENDP.EQ.YES) GOTO L034;
      IF(ENDI.EQ.YES) GOTO L040;
      DO NASTRAN;
      DO ACTSET;
      DO INSPECT;
      GOTO L010;
L031: DISPLAY CONVERGENCE TO THE LOWER BOUND WEIGHT;
      GOTO L050;
L032: DISPLAY CONVERGENCE TO DESIGN VARIABLES;
      GOTO L050;
L033: DISPLAY CONVERGENCE TO FEASIBLE VS ACTUAL WEIGHT;
      GOTO L050;
L034: DISPLAY CONVERGENCE TO FEASIBLE WEIGHT;
      GOTO L050;
L040: DISPLAY MAX. NUMBER OF ITERATIONS REACHED;
C
L050: SET APRT=YES;
      DO NASTRAN;
      DO INSPECT;
      DO HIST;
      STOP;
      END;
ENDATA
```

Figure 1-7 Text subwindow.

1.5.3. TTY Subwindow.

The TTY (or terminal emulator) subwindow emulates a standard terminal. The ESSO uses the **term subwindow.** Basically it is a TTY subwindow that can be edited like a text subwindow. It may also contain a scrollbar. Figure 1-8 shows a term subwindow with a control panel which contains a text field and a button.

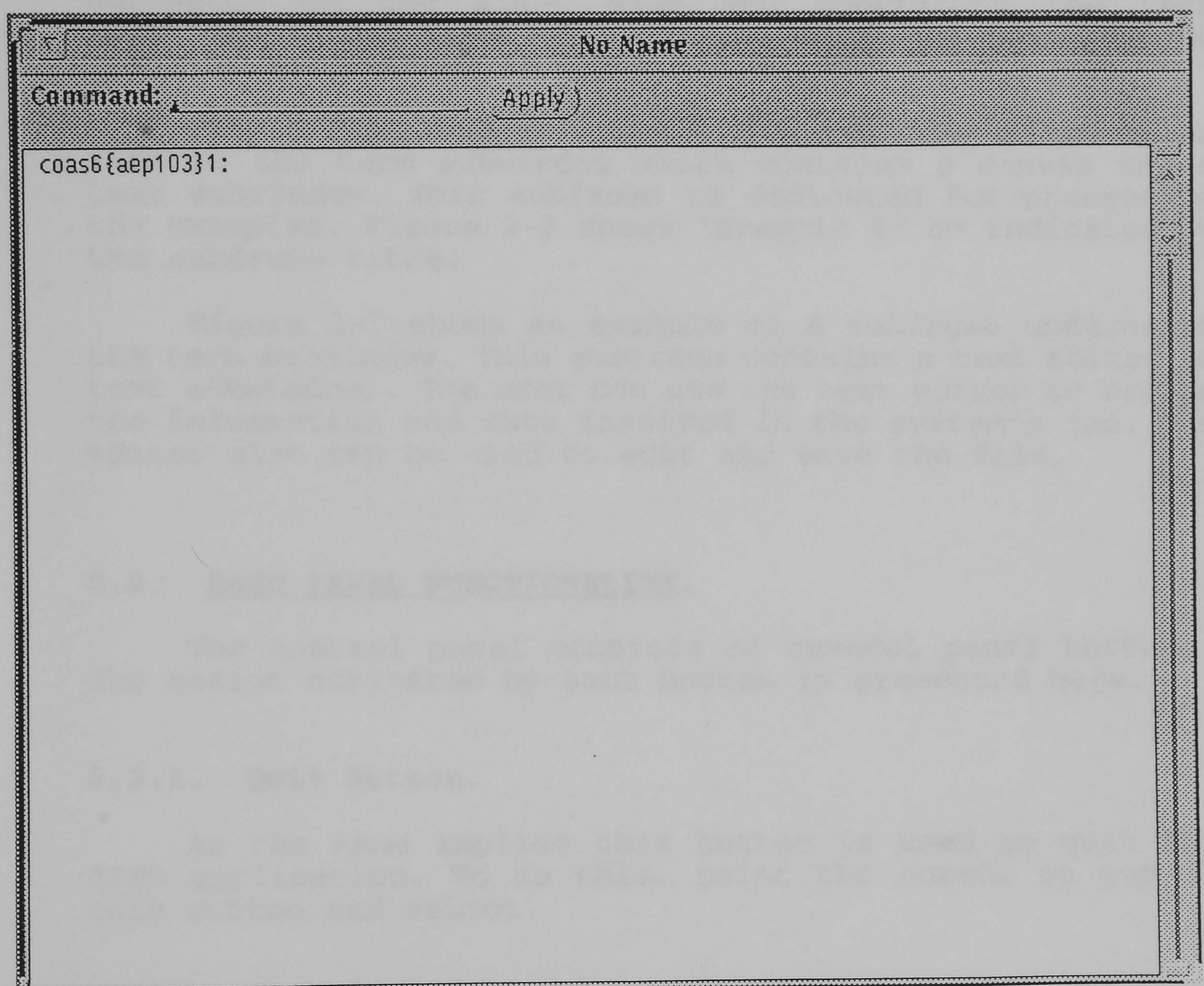


Figure 1-8 Term subwindow.

CHAPTER 2:

ESSO: ITS ENVIRONMENT AND RUN PROCEDURE

2.1. ESSO WORKSPACE.

ESSO workspace is shown in Figure 2-1. It consists of a base frame with *Expert System for Structural Optimization* title written on the title bar. Underneath the title bar is a control panel with several buttons. The subwindow on the upper left under the control panel is a term subwindow which emulates the terminal. The rest of workspace on the right side and underneath the term subwindow is provided for subframes which are activated by the relevant buttons in the control panel.

Figure 2-2 shows an example of a subframe on the right side of the term subwindow which contains a canvas and a text subwindow. This subframe is dedicated for presenting the examples. Figure 2-2 shows "Example 1" as indicated by the subframe title.

Figure 2-3 shows an example of a subframe underneath the term subwindow. This subframe contains a text editor (a text subwindow). The user can use the text editor to browse the information and data involved in the system's run. The editor also can be used to edit and save the file.

2.2. ESSO PANEL FUNCTIONALITY.

The control panel consists of several panel buttons. The action activated by each button is presented here.

2.2.1. Quit Button.

As the name implies this button is used to quit the ESSO application. To do this, point the cursor on top of this button and select.

2.2.2. MSC/XL Button.

MSC/XL is a CAD package from the Macneal-Schwendler Corporation which is built specially for NASTRAN pre/post-processor. To call MSC/XL, select the **MSC/XL** button. In ESSO, MSC/XL is used to visualize the FE model of similar past cases. For drawing the structural geometry and developing the FE model, MSC/XL should be invoked from the system's command/shell window.

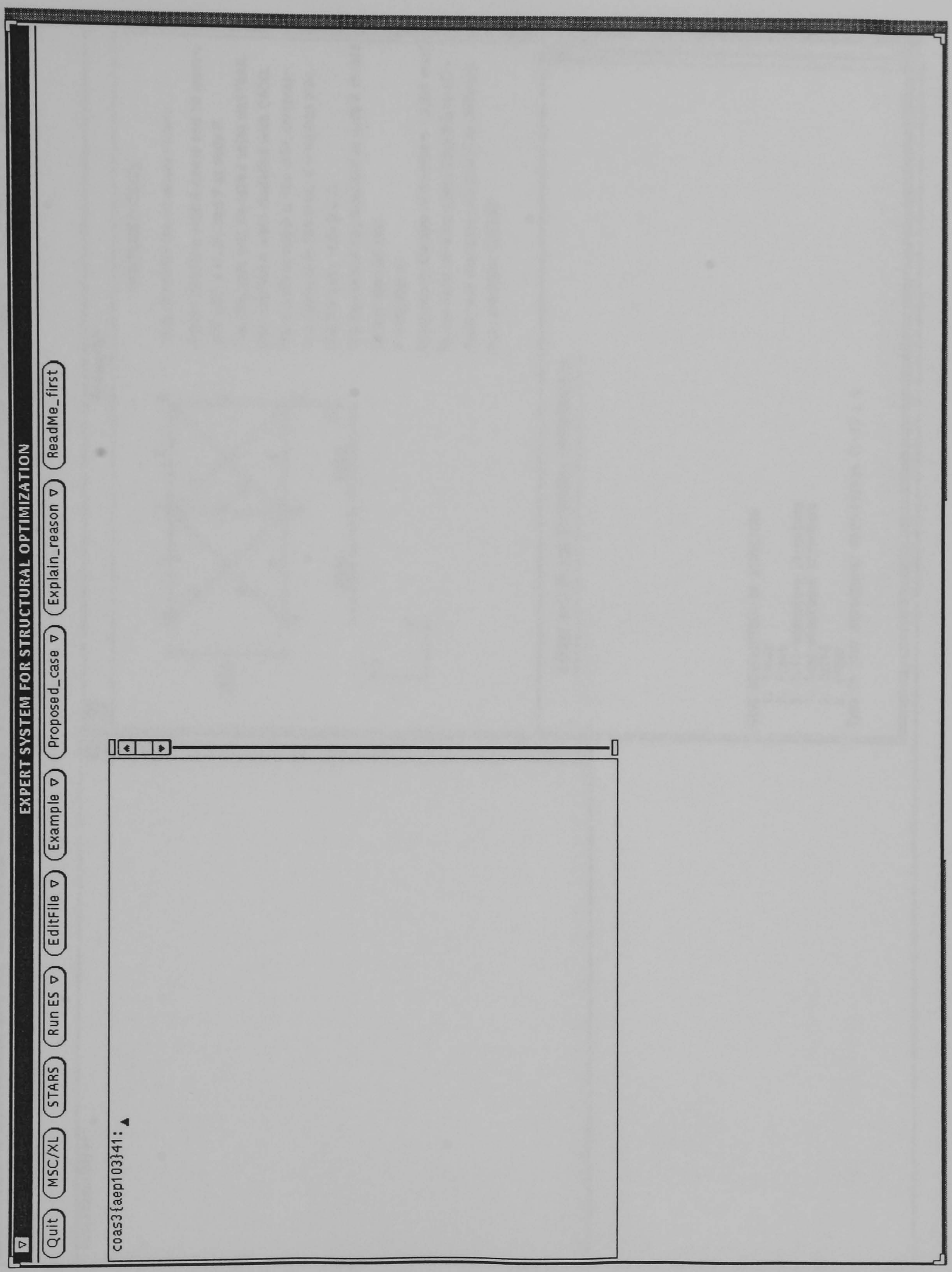


Figure 2-1 ESSO Workspace.

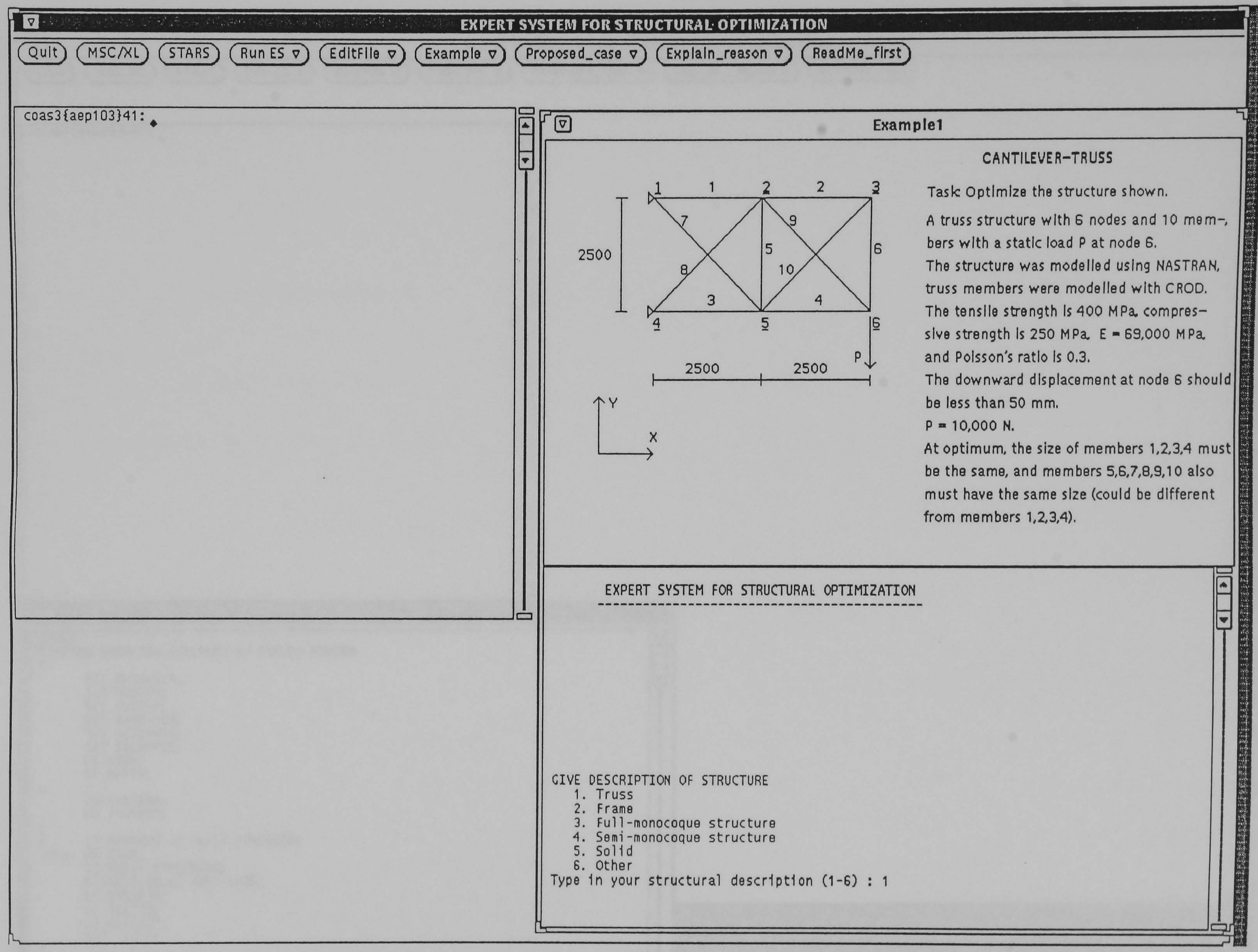


Figure 2-2 ESSO with Example-Window active.

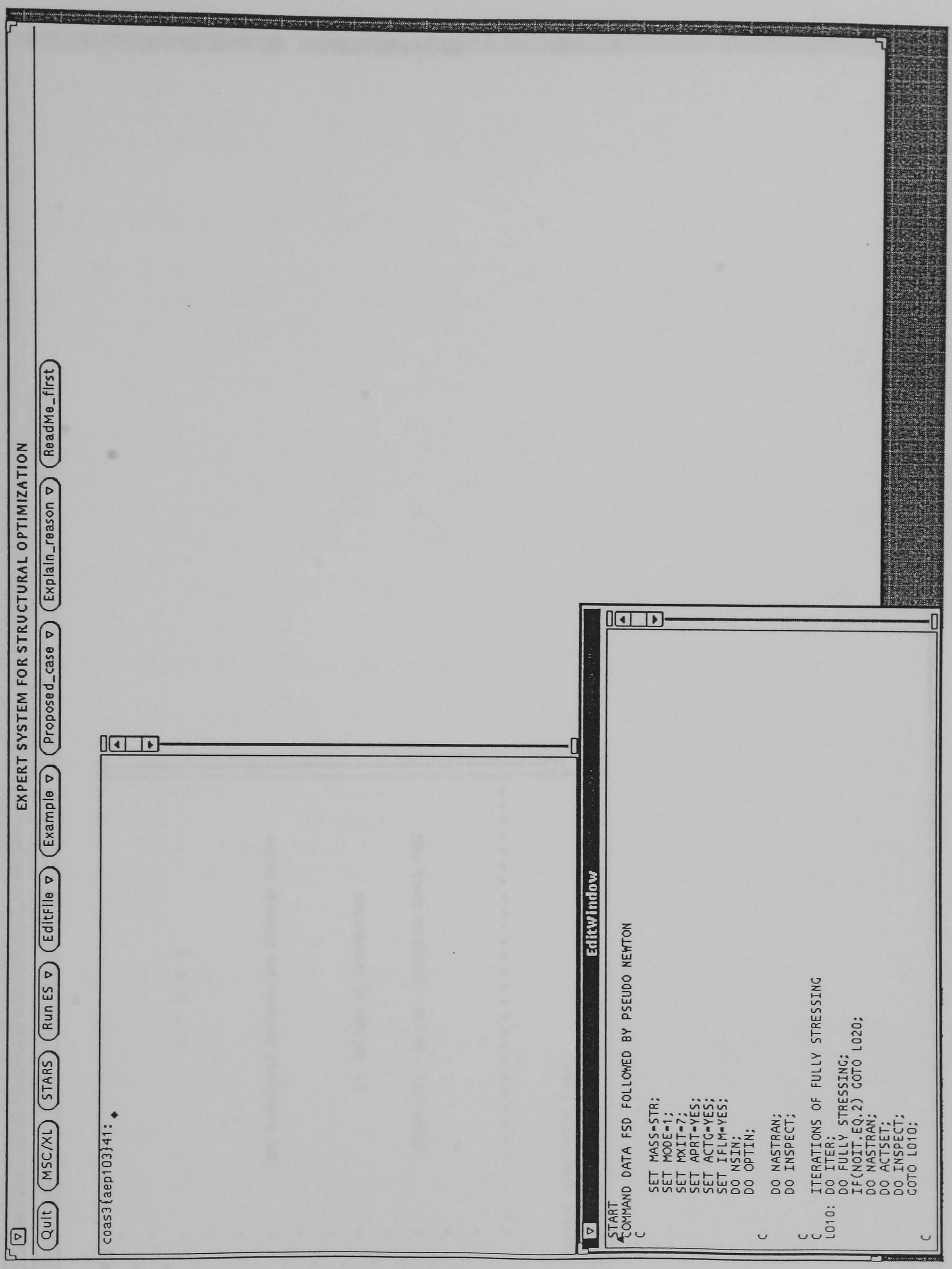


Figure 2-3 ESSO with text editor active.

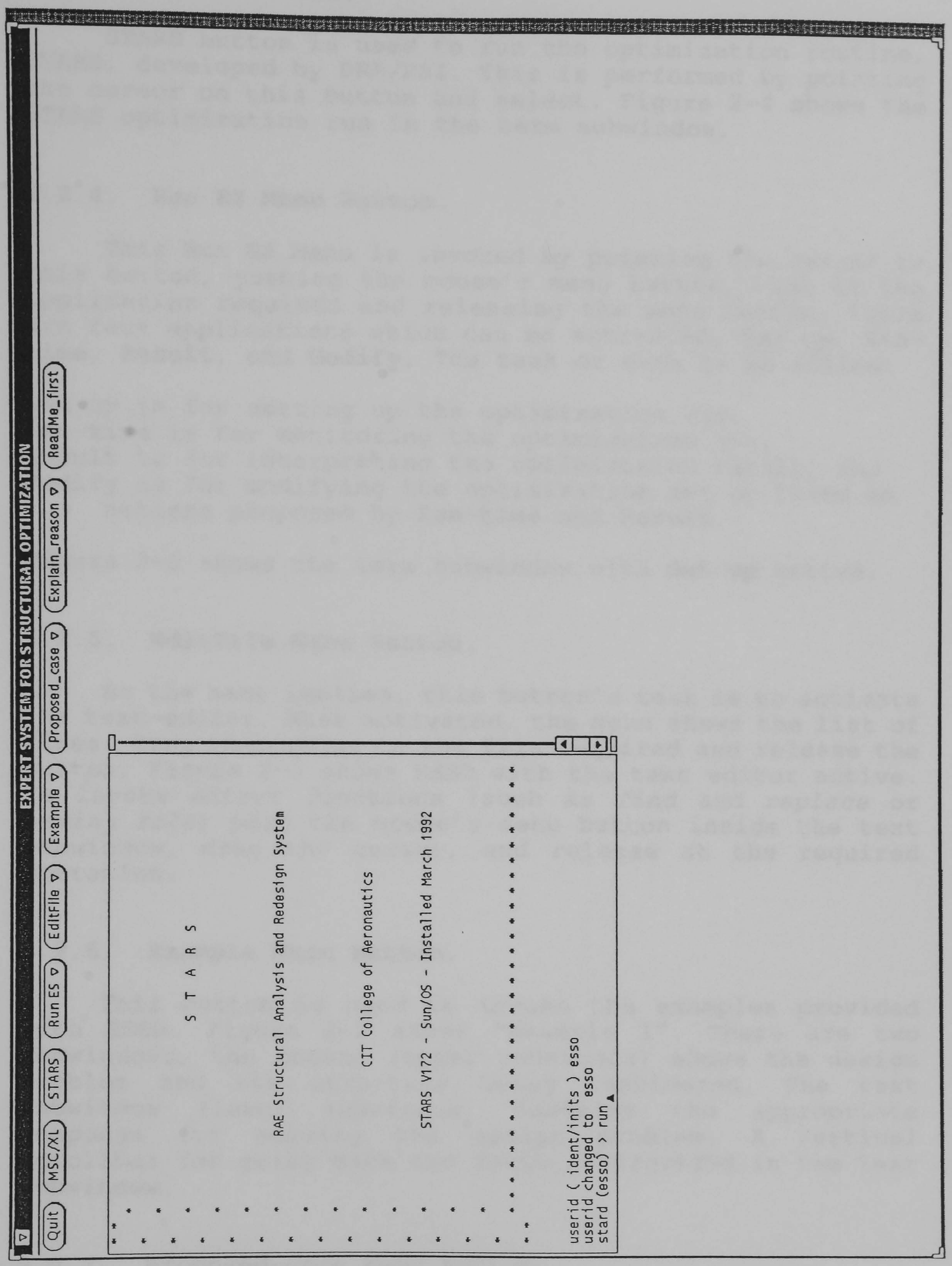


Figure 2-4 Term subwindow with STARS active.

2.2.3. **STARS Button.**

STARS button is used to run the optimization routine, STARS, developed by DRA/PSI. This is performed by pointing the cursor on this button and select. Figure 2-4 shows the STARS optimization run in the term subwindow.

2.2.4. **Run ES Menu Button.**

This Run ES Menu is invoked by pointing the cursor to this button, pushing the mouse's menu button, drag to the application required and releasing the menu button. There are four applications which can be activated, **Set-up**, **Run-time**, **Result**, and **Modify**. The task of each is as follow:

Set-up is for setting up the optimization run,
Run-time is for monitoring the optimization run,
Result is for interpreting the optimization result, and
Modify is for modifying the optimization set up based on actions proposed by **Run-time** and **Result**.

Figure 2-5 shows the term subwindow with **Set-up** active.

2.2.5. **EditFile Menu Button.**

As the name implies, this button's task is to activate the text-editor. When activated, the menu shows the list of files. Drag the cursor to the file required and release the button. Figure 2-3 shows ESSO with the text editor active. To invoke editor functions (such as *find and replace* or *saving file*) push the mouse's menu button inside the text subwindow, drag the cursor, and release at the required operation.

2.2.6. **Example Menu Button.**

This button is used to invoke the examples provided with ESSO. Figure 2-2 shows "Example 1". There are two subwindows, the canvas (upper subwindow) shows the design problem and the structure being considered. The text subwindow (lower subwindow) contains the appropriate response for solving the design problem. A vertical scrollbar for going back and forth is provided in the text subwindow.

2.2.7. **Proposed-case Menu Button.**

In ESSO, one way of setting up an optimization is by recalling similar past cases. If similar past cases (to the current design problem) exist, these cases can be accessed

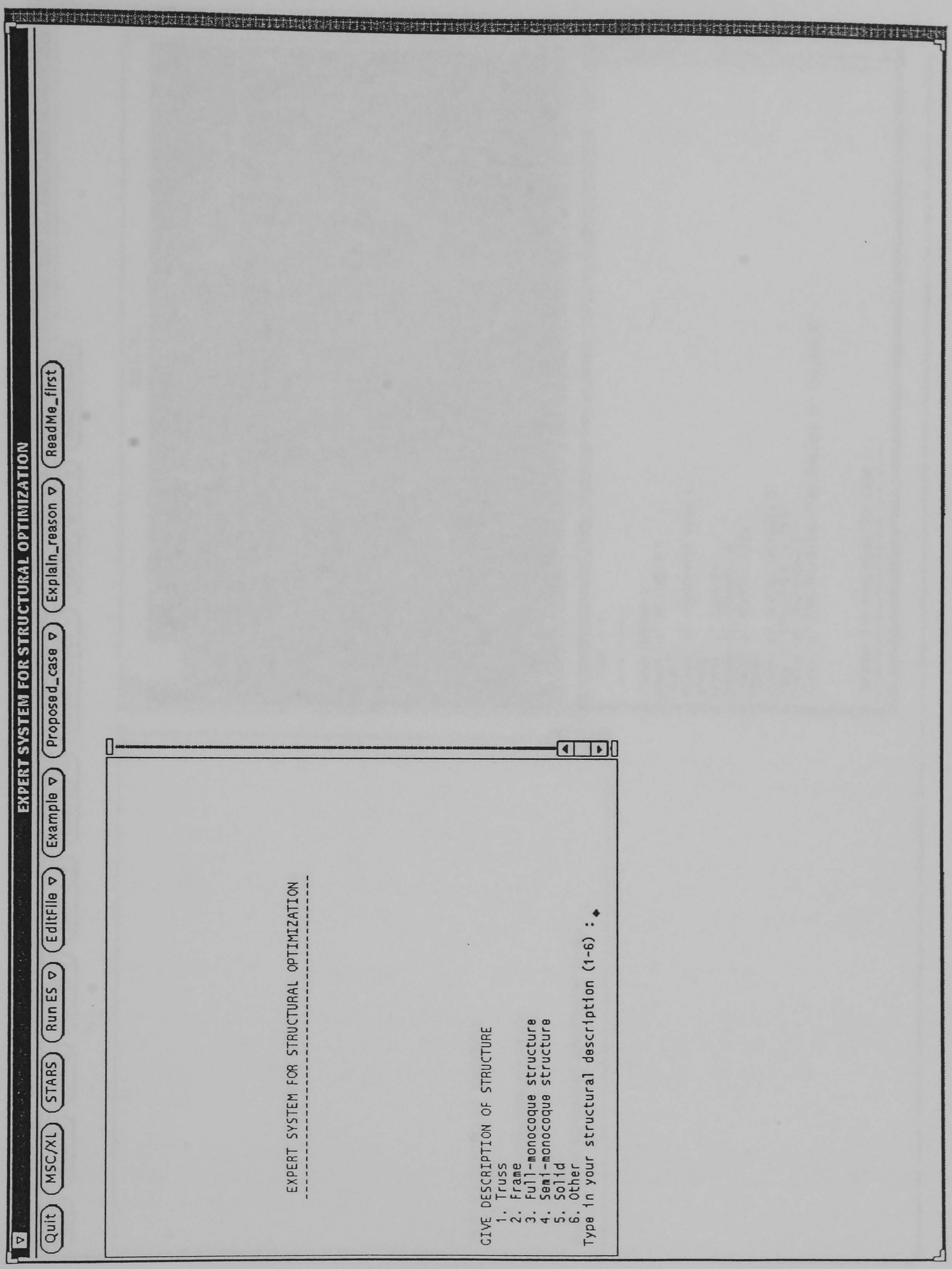


Figure 2-5 Term subwindow with Set-up active.

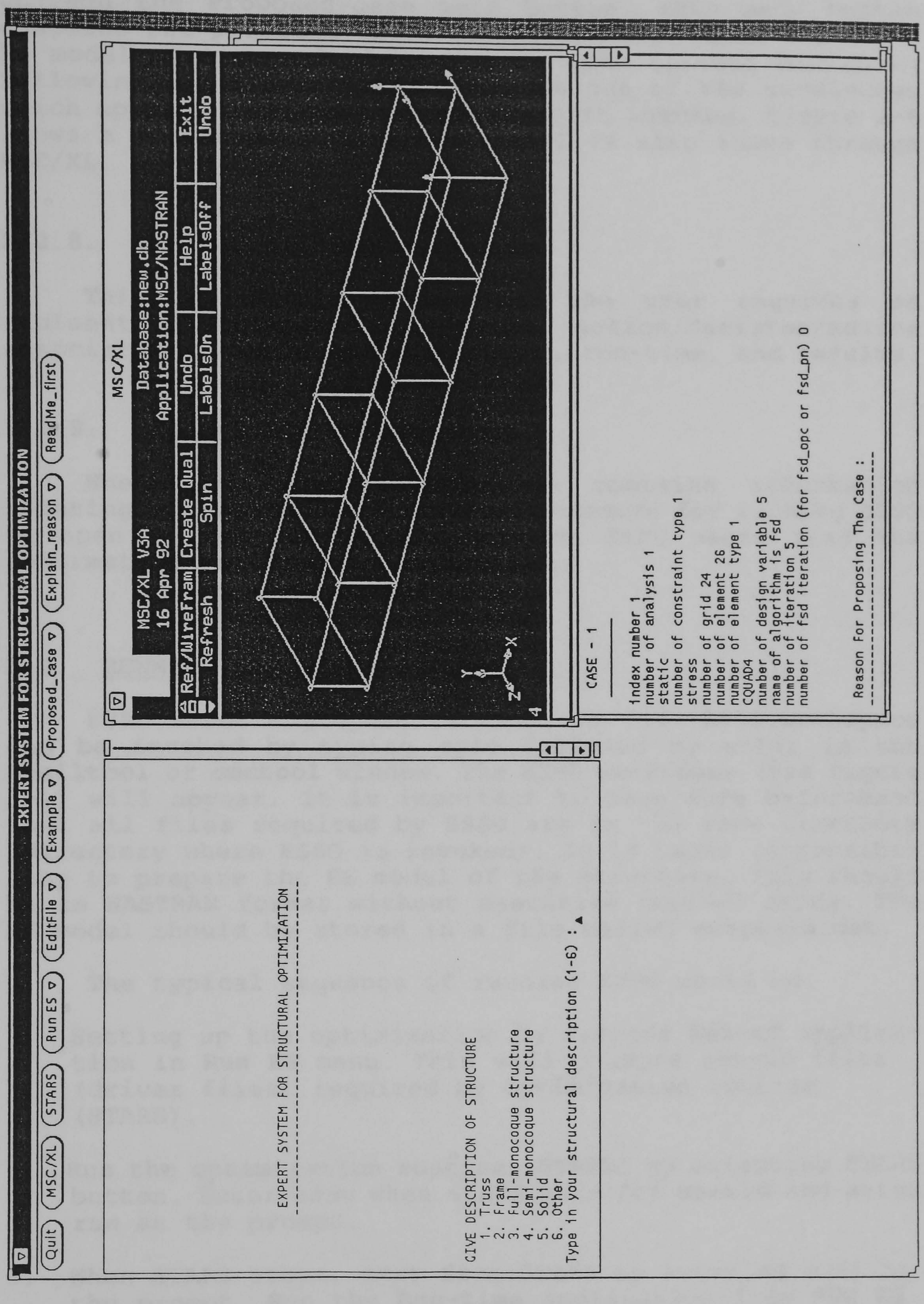


Figure 2-6 A similar case proposed by ESSO.

through the **Proposed-case** menu button. This menu button contains the proposed cases in descending order. The FE model of the past cases can be seen through MSC/XL by following the instructions given in one of the subwindows which appears when any proposed case is invoked. Figure 2-6 shows a proposed case. The FE model is also shown through MSC/XL.

2.2.8. **Explain_reason Menu Button.**

This button is invoked if the user requires an explanation or reason for any action/decision/advice associated with optimization set up, run-time, and results.

2.2.9. **ReadMe_first Button.**

When invoked, a file which contains information relating to the requirements and procedure for running ESSO is open. It is very important that ESSO users read the information in this file carefully.

2.3. **RUNNING ESSO**.

ESSO needs a graphic terminal to run. ESSO workspace can be invoked by typing *esso* followed by *enter* in the **shelltool** or **cmdtool** window. The ESSO workspace (see figure 2-1) will appear. It is important to make sure beforehand that all files required by ESSO are in the same directory (directory where ESSO is invoked). It is users responsibility to prepare the FE model of the structure. This should be in NASTRAN format without executive control cards. The FE model should be stored in a file called **essonsin.dat**.

The typical sequence of running ESSO would be:

1. Setting up the optimization by running **Set-up** application in **Run ES** menu. This will produce set-up files (driver files) required by optimization routine (STARS).

2. Run the optimization routine (STARS) by selecting STARS button. Enter *esso* when STARS asks for **userid** and enter *run* at the prompt.

3. When STARS stops, exit from STARS by entering *exit* at the prompt. Run the **Run-time** application from **RUN ES** menu. This predicts the possibility of an optimization run converging. If convergence is in sight go to step 4. If the optimization is not predicted to be converging the system will search for explanations and

propose actions to follow. To implement these actions, go to step 6.

4. Resume the STARS run. This is done by selecting the **STARS** button, enter *esso* as userid, and enter *resume* at the prompt.

5. When STARS stops, run the **Result** application from **RUN ES** menu. This application interprets the optimization result. If the result is satisfactory the ESSO session stops, otherwise the system will search for explanations and proposes actions to follow. To implement these actions, go to step 6.

6. Run the **Modify** application from **RUN ES** menu. This application will modify the optimization set up based on the actions proposed in step 3 or 4.

7. Go to step 2.

For first time users, the best way to become familiar with ESSO is by running the examples given in **Example** menu. Call the example and at the same time run the **Run ES**. Run the case in **Example** by supplying the information required by applications in **Run Es** according to the response given in the example.

To exploit MSC/XL from inside ESSO for viewing the structural model of proposed past cases (or to do other tasks) the users have to do the following:

- Select the **MSC/XL** button after ESSO workspace appears on the screen. This action has to be done before the users run other ESSO applications.

- Iconize the MSC/XL workspace into an icon.

- At this stage the users can run ESSO applications as normal.

- When required the users can invoke MSC/XL by opening the MSC/XL icon. When the MSC/XL shows a blank window the users have to restore the MSC/XL window. This is done by pressing the menu button on the MSC/XL title bar, drag to the "restore" option and release the button.
However, it is better to run MSC/XL from outside ESSO. To do this, open a **command-tool** or **shell-tool** window and make **esso** sub-directory as the active directory. Enter *xl3a* at the prompt.

CHAPTER 3:

EDITING ESSO KNOWLEDGE BASE

For an Expert System to be able to work effectively, it is important that its knowledge can be easily modified. New knowledge needs to be added to the existing knowledge base. It is also necessary to modify the contents of the knowledge base when the current knowledge is no longer relevant. Another important aspect of any knowledge base system is explanation. When required, the system should be able to give its reasoning for any decision taken. This chapter explains the way the ESSO knowledge bases are coded in C++ language and how to edit or modify them. For any modification in the knowledge base, the associated explanation also needs modification, and this is also discussed in this chapter.

3.1. ESSO KNOWLEDGE BASE.

There are four modules in ESSO which contain the knowledge bases and reasoners namely, the memory module, the conventional module, the run-time module, and the result interpretation module. The task of the memory and conventional modules are for setting up the optimization. The run-time module monitors the optimization run. The optimization results will be interpreted by the result interpretation module.

The knowledge base of the run-time and result interpretation modules were constructed using procedural rules, IF-THEN rules. The knowledge base of the conventional module combines the domain/subject knowledge with IF-THEN rules. This domain knowledge contains the domain properties while the rules reason about them. The approach used in the memory module is based on case based reasoning. Similarity rules were set up to find similar cases among the past cases recorded in the memory module.

The domain knowledge of the conventional module was structured according to object oriented methodologies. Each optimization method is treated as an object and stored in a file. This represents the optimization method capabilities in handling design problems.

In the memory module, the domain knowledge contains past experiences in dealing with design problems. Each design experience is stored as a record. Each record, which represents a design case, contains the design problem and the set up of the optimization strategy for solving the

design problem. Whether cases can be regarded as similar depend on the value of certainty factors of the cases after similarity rules are applied.

3.2. THE CONVENTIONAL KNOWLEDGE BASE.

The knowledge base of the conventional module combines both object oriented schemes and production rules. The object oriented scheme represents domain properties while rules reason about them. A simple way to show this relation is by writing production rules which send messages to objects and test the response in their premises. For example,

```
Rule :
   IF     ( fsd.analysis_test()    = true ) and
          ( fsd.constraint_test() = true )
   THEN   ( fsd.to_consider = true )                    (3-1)
```

The rule tests the object *fsd* (fully stressed design) if it can do both the analysis and constraints. It is the object *fsd* itself which does the analysis_test and constraint_test operations. The object will see if it can do analysis and constraints requirements of current design problem. If the rule premises are satisfied, then the fsd method will be considered for subsequent tests.

The problem with the above rule is that it is written in a procedural way. This type of programming leads to difficulties in writing the inference engine and the knowledge base as two separate modules. In an Expert System program, the user/programmer should be able to modify the rules (knowledge base) without interfering with the inference engine. A better way to code the above rule is,

```
IF ( analysis_test    = true ) and
   ( constraint_test = true )
THEN ( method_to_consider = true )                  (3-2)
```

The premises variables (analysis_test and constraint_test) have to be instantiated first. The instantiation is performed in an assignment/instantiation routine which sends enquiries to the appropriate object, in this case *fsd*, about its capabilities in handling analysis and constraints required. Functions analysis_test and constraint_test in the object *fsd* will provide the answers to these enquiries. If the conditions of the rule is satisfied then the conclusion method_to_consider is instantiated to true. For certain information, instead of asking questions to various objects, this assignment routine might ask the users to provide the answers.

The above illustration leads to the discussion about the domain/subject knowledge (written in an object oriented scheme) and production rules in the conventional module and their editing.

3.2.1. The Domain/Subject Knowledge.

The domain knowledge contains the information about objects capabilities. The objects in this case are the optimization methods. Object oriented methodologies are used to represents this domain knowledge. For this purpose, two classes are defined, those are class **Procedure** and class **METHOD**. **Procedure** is a base class with **METHOD** as its child class.

The task of the class **Procedure** is to deal with matters relating to optimization algorithms. It contains parameters which define the capabilities and properties of optimization algorithms. This class also contains procedures for interrogating the available algorithms with regard to their capabilities in handling a current design problem. Object *procedure* is created from this class. The class Procedure is defined as follows (written in C++ terminology,

```
class Procedure {
public :
  /* the parameters below refer to algorithm capabilities
     in dealing with */
  // max number of load case
  int load_case,
  // lagrange multiplier.
      lagrange;
  // number of analysis types, number of constraint types.
  int analysis_num, constraint_num;
  // number of convergence type, number of element type
     available.
  int convergence_num, elem_avail_num;
  // combine analysis, combine constraint.
  int combine_analysis, combine_constraint;
  // stress constraint;
  int stress constraint;
  // an algorithm might need stress constraint (e.g. FSD).
  int need_stress;
  // analysis types
  char analysis [10][40],
  // constraint types
       constraint[10][40],
  // convergence criteria types
       convergence[10][40],
  // element (FE) types
       elem_avail[10][40];
```

```
/* method currently available */
// number of method
int num_of_method;
// method name
char method_name[5][20];

// procedure to invoke available methods.
void case_method(void);

/* procedures below are to test whether an algorithm can
   handle current design requirement with respect to fol-
   lowing aspects   */
   // analysis
   int analysis_checking(void);
   // constraint
   int constraint_checking(void);
   // load case
   int loadcase_checking(void);
   // element (FE) types
   int element_checking(void);

} procedure;   // object procedure is defined.


class METHOD : public Procedure {
public :
   // method name
   char name[10];
   // procedure for reading the file
   void Dbase_read(char f[10]);
} method[5]; // five objects are created.
```

The class **METHOD** is the child of the class **Procedure**. During program run several object are created where each object represents an algorithm. The name of the available algorithms are stored in a file called *method.dat*. The data relating to an algorithm is read from a file. For example, data for the fsd algorithm is stored in a file called *fsd.dat*. There are five algorithms available in ESSO.

3.2.2. The IF-THEN rules.

The IF-THEN rules are used for reasoning. The coding of these rules in C++ is implemented using *switch - case* operation as follows,

```
switch(sn)  {

  // if part:
```

```
case 1: {
 if ( ..... )   s=1;
 break;
}
case 2: {
 if ( ..... )   s=1;
 break;
}
case 3: {
 if ( ..... )   s=1;
 break;
}
    .
    .
    .

case i: {
 if ( ..... )   s=1;
 break;
}
    .
    .
    .

case n: {
 if ( ..... )   s=1;
 break;
  }

}


// see if the then part should be invoked, i.e s=1
if (s!=1) {
    ....;
    goto ...;
}


// invoke the then part

switch(sn)  {

  // then part
case 1: {
   ( ..... )
   instantiate();
   break;
}

case 2: {
   ( ..... )
   instantiate();
```

```
    break;
}
        .
        .
        .

case i: {
    ( ..... )
    instantiate();
    break;
}
        .
        .
        .

    case n: {
        ( ..... )
        instantiate();
        break;
    }

}
```

Which rule (referred to by rule number) to be considered depends on the value of *sn*. The number after *case* is the rule number, from 1 to n. In the 'if part', if rule premises (within parentheses) are satisfied, then the value of *s* is instantiated to 1. In that case the 'then part' of that rule (within parentheses) will be fired. The function *instantiate()* is part of the inferencing mechanism and users should not worry about this.

3.2.3. Editing the Knowledge Base.

(1). The Domain Knowledge.

The part of the domain knowledge which might need modification (editing) is the files which contain the data referring to the algorithm capabilities. The data for each file is written in a standard sequence as follows,

Parameters:	Explanation:
analysis_num;	Number of analysis.
analysis[i];	Types of analysis (array).
combine_analysis;	Capability in handling analysis combination.
constraint_num;	Number of constraints.
constraint[i];	Type of constraints (array).
combine_constraint;	Capability in handling constraint combination.

stress_constraint;	Capability in handling stress constraint.
need_stress;	If an algorithm needs stress constraint.
convergence_num;	Number of convergence criteria.
convergence[i];	Type of convergent criteria (array).
load_case;	Number of load case.
lagrange;	Capability to calculate lagrange multiplier.
elem_avail_num;	Number of element types.
elem_avail[i];	Element names (array).

For example, the capabilities of the optimality criterion algorithm is coded in *opc.dat* as follows,

```
3                    ---> number of analysis capability
static
normal_modes
buckling
0                    ---> not suited to analysis combination
4                    ---> number of constraints capability
stress
displacement
buckling
natural_frequency
0                    ---> not suited to constraint combination
0                    ---> can not handle stress constraint
0                    ---> does not need stress constraint
3
ENDP
ENDF
ENDT
20                   ---> number of load cases capability
1                    ---> able to calculate lagrange multi-
                          plier
7                    ---> number of available element types
CQUAD4
CTRIA3
CSHEAR
CBEAM
CBAR
CROD
CONROD
```

(2). <u>The IF-THEN rules</u>.

The rules are stored in a file called *convrule.dat* which is coded as follows,

```
#ifdef RULE

    // if-then statements
```

```
// if part:
case 1: {
 if ( ..... )   s=1;
 break;
}
case 2: {
 if ( ..... )   s=1;
 break;
}
case 3: {
 if ( ..... )   s=1;
 break;
}
    .
    .
    .

case i: {
 if ( ..... )   s=1;
 break;
}
    .
    .
    .

case n: {
 if ( ..... )   s=1;
 break;
}

}


// see if the then part should be invoked, i.e s=1
if (s!=1) {
    ....;
    goto ...;
}


// invoke the then part

switch(sn)   {
  // then part
case 1: {
   ( ..... )
   instantiate();
   break;
}
case 2: {
   ( ..... )
   instantiate();
   break;
}
```

```
        .
        .
        .

case i: {
   ( ..... )
   instantiate();
   break;
}
        .
        .
        .

      case n: {
         ( ..... )
         instantiate();
         break;
      }

   }

#endif



#ifdef VARLIST
// list of variables
strcpy(varlt[1],"...");
strcpy(varlt[2],"...");
        .
        .
        .
strcpy(varlt[k],"...");


// list of variables in rule premises
strcpy(clvarlt[1],"...");                    //rule-1
strcpy(clvarlt[2],"...");
strcpy(clvarlt[11],"...");                   //rule-2
            .
            .
            .
   strcpy(clvarlt[21],"...");                //rule-n
   strcpy(clvarlt[22],"...");
   strcpy(clvarlt[23],"...");

#endif
```

```
#ifdef CLASS_PAR
   ......
#endif


#ifdef CHECK_INST
void apriori_reasoning::check_instantiation(METHOD object)
{
  i=1;
  while ( (strcmp(v,varlt[i])!=0) && i<=30 )   i++;
  if (instlt[i]!=1) {
    instlt[i]=1;

    switch(i)  {
      case 1: {
        ............
        break;
      }
      case 2: {
        ............
        break;
      }
           .
           .
           .
      case m: {
        ............
        break;
      }

    }
  }
}

#endif
```

This file consists of four sections in which each section is coded between *#ifdef* and *#endif* statements. Those sections are,

- rules (between *#ifdef RULE* and *#endif*)
- list of variables (between *#ifdef VARLIST* and *#endif*)
- variable definition (between *#ifdef CLASS_PAR* and *#endif*)
- instantiation (between *#ifdef CHECK_INST* and *#endif*).

An editing process normally is started with modifying the rule premise and/or conclusion, adding new rules, or deleting any existing rules. The rule premise might be a

single premise or multiple premises with logical **AND** or **OR**. All new variables in rule premises should be written in the list of variables and list of variables in rule premise. Those new variables should be defined in the variable definition section. How a variable gets its value should be specified in the instantiation section. In carrying out the editing process, familiarity with C++ will help users. The users are advised to study the *convrule.dat* to understand the relation between those four sections.

3.2.4. Explanation Facility.

Associated with each rule (IF-THEN statement) is an explanation-statement of that rule. This explanation-statement describe the contents of the rule using easy to understand sentences. When a rule is fired, the associated statement is recalled. The coding of the explanation-statement is as follow:

```
//case 1:
strcpy(texrule[1].text[0]," .... ");
strcpy(texrule[1].text[1]," .... ");
          .
          .
          .
strcpy(texrule[1].text[q-2]," .... ");
strcpy(texrule[1].text[q-1]," .... ");
texrule[1].textnum=q;

          .
          .
          .

//case i:
strcpy(texrule[i].text[0]," .... ");
strcpy(texrule[i].text[1]," .... ");
          .
          .
          .
strcpy(texrule[i].text[r-2]," .... ");
strcpy(texrule[i].text[r-1]," .... ");
texrule[i].textnum=r;

          .
          .
          .

//case n:
strcpy(texrule[n].text[0]," .... ");
strcpy(texrule[n].text[1]," .... ");
          .
          .
          .
```

```
strcpy(texrule[n].text[s-2]," .... ");
strcpy(texrule[n].text[s-1]," .... ");
texrule[n].textnum=s;
```

The number of explanation statements is the same as the number of rules which is *n*. The parameters q, r, and s are the number of "sentences" in each of statement. The explanation-statements for the conventional module are stored in the file *convexpl.dat*. The user should learn this file and the associated knowledge base file *convrule.dat*.

3.3. THE MEMORY KNOWLEDGE BASE.

The knowledge base of the memory module also consists of the domain knowledge and the IF-THEN rules. The domain knowledge is the system's memory of its past experience which can be regarded as a file containing a collection of knowledge (past experience) records. Each record contains a (past) design case and its solution, held in the form of the similarity parameters (for similarity parameters and similarity rules please see chapter 5 in the main thesis) with their associated values and the successful optimization strategy for that particular case. The IF-THEN rules are the similarity rules and are used to reason in capturing the design case record which is similar to the current design problem.

3.3.1. The Domain Knowledge.

Each record in the knowledge base contains the following information (stored in fields),

- number of analysis,
- analysis types,
- number of constraints,
- constraint types,
- number of nodes,
- number of elements,
- element types,
- number of design variables,
- solution which consists of algorithm, and number of iterations,
- comments, comments on the optimization process,
- filename, the filename where the FE data are stored.

Thus a typical record might be,

- number of analysis: 1;
- analysis types: statics;

- number of constraints: 2;
- constraints types: stress, displacement;
- number of nodes: 5;
- number of elements: 6;
- element types: rod;
- number of design variables: 6;
- solution: pseudo-newton algorithm, **n** number of iterations ;
- comments: a straight forward optimization run ;
- filename: framnsin.dat;

Such record can be written in C++ using *Structure*. The DESIGN_CASE structure below is used to represent the system's memory.

```
struct DESIGN_CASE {
    int index;
    int num_analysis;
    int num_constraint;
    int element_type_num;
    int element_num;
    int node_num;
    int design_var;
    int algor_iteration;
    int fsd_iteration;
    int used_rule[20];
    int num_fired_rule;
    float used_cf[20];
    float cf_value;
    char analysis[7][40];
    char constraint[7][40];
    char element_type[7][40];
    char algor_name[10];
    char filename[40];
    char comment[10][80];
    struct TEXTRULE explain_rule[25];
};
```

Records of the system's past experience can be seen in *memory.dat*. Some members of the above data structure are used for reasoning process only and do not appear in the data file. The *TEXTRULE* is a structure which is used to store the explanation-statements and these statements are stored in *memrule.dat* together with the similarity rules.

3.3.2. The Similarity rules.

The similarity rules are written in the following format:

```
// rule 1
if ( ..............          &&
     ..............          &&
        .
        .
        .
     ..............                )  {
  ..................;
  rule[1]=TRUE;
  cf[1]=x.xx;
}
// text rule 1
strcpy(textrule[1].text[0]," .... ");
strcpy(textrule[1].text[1]," .... ");
        .
        .
        .
strcpy(textrule[1].text[q-2]," .... ");
strcpy(textrule[1].text[q-1],"cf=x.xx");
textrule[1].textnum=q;

        .
        .
        .

// rule i
if ( ..............          &&
     ..............          &&
        .
        .
        .
     ..............                )  {
  ..................;
  rule[i]=TRUE;
  cf[i]=x.xx;
}
// text rule i
strcpy(textrule[i].text[0]," .... ");
strcpy(textrule[i].text[1]," .... ");
        .
        .
        .
strcpy(textrule[i].text[r-2]," .... ");
strcpy(textrule[i].text[r-1],"cf=x.xx");
textrule[i].textnum=r;

        .
        .
        .
```

```
// rule n
if ( ..............        &&
     ..............        &&
        .
        .
        .
     ..............             )   {
   ..................;
   rule[n]=TRUE;
   cf[n]=x.xx;
}
   // text rule n
   strcpy(textrule[n].text[0]," .... ");
   strcpy(textrule[n].text[1]," .... ");
          .
          .
          .
   strcpy(textrule[n].text[s-2]," .... ");
   strcpy(textrule[n].text[s-1],"cf=x.xx");
   textrule[n].textnum=s;
```

A certainty factor (cf) with a value of *x.xx* is attached to each similarity rule. Although the rules above show AND (&&) relationships, the rule premises could also include OR relationships. As shown above, the explanation-statements are written below associated rules. The similarity rules are stored in the file *memrule.dat*.

3.3.3. **Editing The Knowledge Base.**

(1). <u>The Domain Knowledge</u>.

Editing the domain knowledge (past cases) is not normally required. The only situation where it is necessary to edit the domain knowledge is when the similarity parameters of the cases need modification. However this requires some modification in the code of the memory module (file *memory.cxx*). The domain knowledge is stored in the file *memory.dat*.

Each past case also includes the FE model which is written in NASTRAN convention. Instead of a complete FE data, each case only stores the name of the file where the data is stored. All FE files are placed in the directory *FE_dat* which is the child directory of ESSO working directory.

(2). The similarity rules.

The similarity rules are stored in *memrule.dat*. Although not normally recommendable the user can change the contents of the rules and certainty factor values.

3.4. THE KNOWLEDGE BASE OF THE RUN-TIME AND RESULT INTERPRETATION MODULES.

Both the run-time and result interpretation modules use procedural rules as the knowledge base. The structure of the rules (IF-THEN rules) are similar to the rules in the conventional module. Editing process also carried out in a similar way. The knowledge base of the run-time interpretation module is stored in *runtrule.dat* while the explanation-statements are in *runtexpl.dat*. The knowledge base of the result interpretation module is stored in *reslrule.dat*, while the explanation statements are in *reslexpl.dat*.

CHAPTER 4:

THE COMPUTER PROGRAMS OF ESSO

This prototype is called ESSO (Expert System for Structural Optimization) and it's task to give guidance to the designers who wish to employ optimization. The task of ESSO is to guide the users in setting up an optimization, monitoring the optimization run, and interpreting the results or any failure associated with an optimization run. For the trial implementation, the STARS optimization package is used as the optimization tool. The ESSO set-up application produces setup files, which are called driver files, to run the STARS optimization system. These files are CDF (command data) and CONS (design specification) files. The third file required by STARS, a file contains finite element model, is to be prepared by the user.

At present, the prototype can handle static and dynamic problems. Any set of constraints associated with both types of analysis can be handled by the system. Currently, the prototype deals only with isotropic materials.

The modules available in the prototype are,

(1) the design input module,
(2) the memory module,
(3) the conventional module,
(4) the element module,
(5) the variable module,
(6) the constraint module,
(7) the driver-files module,
(8) the run-time module,
(9) the result interpretation module, and
(10) the modification module.

Modules (1) to (7) are set-up modules with the task of setting up the optimization. The last three modules are part of the system's back end. For data processing, the modification module makes use the object structures of the design input module, element module, variable module, and constraint module.

The prototype is written in C++ in object oriented approaches. Each module is written as a class with its own task. Some modules are defined with only one class while others have a parent and several child classes. Objects are created during executions and defined as global objects (similar in concept to global variables) which can be accessed by every module in the prototype.

The set-up modules are compiled and linked into an executable file called *Set-up*, the run-time interpretation module into *Run-time*, the result interpretation module into *Result*, and the modification module into *Modify*. As discussed in chapters one and two of this manual, ESSO is run through a user interface based on XView. This user interface is written in C. All the programming was carried out in the SUN SPARC II workstation.

In this chapter, the computer program of ESSO and its user interface is presented.

4.1. THE DESIGN INPUT MODULE.

The task of the design input module is to obtain information relating to the design requirements such as the analysis requirements, constraint requirements, number of design variables, and general information of the FE model. This module consists of one class, Design, which has the parameters for defining the design requirement and the operations/functions relating to obtaining the design requirement. The class Design defines the object *design*. All design specifications are stored in this object and these can be used by other modules through object passing. This object deals only with a general design requirement. More detail specifications are dealt by other modules. For example, with regard to the constraint requirement, object *design* only stores type of constraints (such as stress, displacement) while details of the constraint values are handled by the module constraint.

The data for this module, in the form of a design specification, is supplied by the user and stored in *design*. Data in this object is required by the memory, conventional, variable, element, constraint, driver-files, and modification modules. Data transfer is performed by sending object *design* to those modules.

The design input module consists of one file, that is **design.cxx**.

4.2. THE MEMORY MODULE.

Following the execution of the design input module, the memory module is invoked. This contains the reasoner and the knowledge base for the past experience knowledge. The detail description of the mechanism inside this module has been discussed in chapter five. If there is a **similar** case found then the execution proceeds to the variable and element module, otherwise the conventional module is executed.

The information required by the memory module is supplied by object design in the form of the design requirements. If **similar** cases are found then those cases are proposed for use. The output of this module, definition of the optimization set-up of the proposed case is used by the driver-files module. During its execution, the memory module needs data from several files, **s_case.dat**, **d_case.dat**, and **sd_case.dat** which stores records of system's experience, and **memrule.dat** which contains the similarity rules. Reasons on past cases selection are recorded in files which are produced during the module run.

The memory module consists of the following files,

- **memory.cxx**, the memory module program,
- **memrule.dat**, which contains the similarity rules,
- **s_case.dat**, which contains static past cases,
- **d_case.dat**, which contains dynamic past cases,
- **sd_case.dat**, which contains past cases with combination of static and dynamic analysis.

4.3. THE CONVENTIONAL MODULE.

The conventional module contains the reasoner and the knowledge base for the a priori knowledge. Similar to the memory module, the information required by the conventional module is supplied by object design. The output of this module, definition of the optimization set-up, is sent to the driver-files module. During execution, this module needs data from **method.dat**, which contains the names of available algorithms, and the following files, **fsd_pn.dat**, **fsd_opc.dat**, **fsd.dat**, **opc.dat**, and **pn.dat**, each stores information about the algorithm capabilities.

The files which constitute this module are,

- **convent.cxx**, the inference-engine of conventional module,
- **method.dat**, which contains the algorithm names,
- **fsd.dat**, which contains the characteristics of fully stressed design algorithm,
- **opc.dat**, which contains the characteristics of optimality criteria algorithm,
- **pn.dat**, which contains the characteristics of pseudo newton algorithm,
- **fsd_pn.dat**, which contains the characteristics of the combination of fully stressed design and pseudo newton.
- **fsd_opc.dat**, which contains the characteristics of the combination of fully stressed design and optimality criteria.
- **convrule.dat**, which contains the IF-THEN rules,
- **convexpl.dat**, which contains explanation statements associated with the IF-THEN rules.

4.4. THE ELEMENT MODULE.

The task of the element module is to deal with those operations related to the finite element model such as elements numbering and displaying elements characteristics. The details of the FE model are supplied by the user. This module has a base class, Element, which itself has three child classes, Plate_membrane, Beam, and Axial_rod. The class Beam has four child classes, those are I-beam, channel-beam, T-beam, and rectangular_beam. The classes under the class Element contain information about element characteristics and relations between design variable properties and analysis properties. Any element type used in the FE model is associated to an appropriate element class and entitles to the characteristics belong to that class. For example, a CBEAM (a NASTRAN element type) element type will be associated to the Beam class, and if the beam used is having a rectangular cross section then it will be associated further to the rectangular_beam class. The output of this module is an FE data and element characteristics stored in various element objects.

Other information required by this module is the design requirement which are concerned with the general FE model and supplied by object *design*. The output of this module is required by the variable and driver-files modules.

The element module consists of the following files,

- **element.cxx,**
- **beam.cxx,**
- **rod.cxx,**
- **plate_membrane.cxx.**

4.5. THE VARIABLE MODULE.

The variable module has the task of dealing with operations relating to variable formulation. This module consists of one class, Variable. Object *variable* is created from this class. The *variable* asks the user to supply variable specifications such as finite element types and element numbers in each design variable and stores them in this object. It also advises the users if incompatible element types (having different analysis properties) are grouped in the same design variable, for example if the rod elements is put under the same design variable as the plate elements. This module needs information relating to the element characteristics.

In order for the variable module to run, it needs information relating to the design requirement which is

supplied by object *design*, and the elements characteristics supplied by various element objects. The user also has to supply variable specifications. The object variable is required by the driver-files module.

This module contains two files, those are,

- **var.cxx**, which handles variable operations,
- **el_infer.cxx**, which examined whether the grouping of elements in a design variable is appropriate.

4.6. THE CONSTRAINT MODULE.

The operations relating to details of the design constraints are handled by the constraint module. This module has a base class, Constraint. This class has several child classes which deal with more specific type of constraints namely, Static_constraint, Normal_modes_constraint, Buckling_constraint, and Dynamic_constraint. Depending upon the analysis and constraint requirements, the appropriate objects are created. For example, if a static analysis with stress constraint is required, then object *static_constraint* is activated.

This module needs detail information which is related to constraint specification supplied by the user. Some of the information is supplied by the object *design*. The products of this module, various constraint objects, are required by the driver-files module.

This module consists of the following files,

- **constr.cxx**,
- **static.cxx**,
- **buckling.cxx**,
- **dynamic.cxx**.

4.7. THE DRIVER-FILES MODULE.

The task of the driver-files module is to produce files which are used to run the optimization software. In this prototype, the driver-files module produces files for running the STARS optimization package, namely command data file (CDF) and design specification file (CONS). We may note that STARS is being used to represent a general structural optimization system and the prototype is not written in a STARS specific manner. The ideas and concepts being trialed here are generally applicable to any system which employs driver files. The CDF consists of a set of instructions for running optimization which includes the

optimization method to use, number of iterations, and convergence criteria. The CONS defines the variable set-up and constraint set. The driver files module consists of two sub-modules. The first sub-module, the **cdf** produces the CDF file, **essocdf.dat**, while the second sub-module, the **cons** produces the CONS file, **essocons.dat**. The information required by the driver files module to produce these two files is obtained from the modules executed previously. For different optimization packages, the driver files module has to be modified accordingly.

The driver files module consists of these files,

- **cdf.cxx**, which produces the STARS command data file,
- **cons.cxx**, which produces the STARS design file.

4.8. THE RUN-TIME MODULE.

The run-time module is part of the backend module. With the driver files and FE model (prepared by the user) available, the optimization system can be run. This will be suspended after several iterations to check whether the optimization is converging. This check is performed with the aid of an extrapolation routine in the run-time module. If convergence is predicted the optimization run is resumed otherwise the module searches for explanations and proposed actions to follow. The proposed actions are printed in the file **action.dat**.

The run-time module consists of the following files,

- **es_run.cxx**, which controls the run-time module,
- **constype.cxx**, which check if a constraint type is required in the design requirement,
- **runtime.cxx**, the inference-engine,
- **runtrule.dat**, which contains the IF-THEN rules,
- **runtexpl.dat**, which contains the explanation-statements associated with the IF-THEN rules.

4.9. THE RESULT INTERPRETATION MODULE.

If a point is reached where the convergence criteria selected by the user are satisfied (or the maximum number of iterations is reached) the result interpretation module checks the optimization results to ensure that this point is indeed optimal. For satisfactory results the current design problem and its optimization set up is appended to the case memory in the memory module. If this module regards the optimization result as unsatisfactory it offers explanation of the reasons of failure and proposed actions

to follow. The proposed actions are printed in the file **action.dat**.

The result interpretation module consists of the following files,

- **es_res.cxx**, which controls the run-time module,
- **result.cxx**, the inference-engine,
- **constype.cxx**, which check if a constraint type is required in the design requirement,
- **case_upd.cxx**, which updates the case memory of the memory module with the current design problem,
- **reslrule.dat**, which contains the IF-THEN rules,
- **reslexpl.dat**, which contains the explanation-statements associated with the IF-THEN rules.

4.10. THE MODIFICATION MODULE.

The modification module has the task of modifying the optimization set-up if required. The modification is based on the actions, stored in **action.dat**, proposed by the run-time module or the result interpretation module. The modification module also needs design data and the optimization set up produced by the front-end. For design/constraint data processing, the modification module makes use of the objects structures of the design input module, the variable module, the element module, and the constraint module. The memory and conventional modules are used for algorithm data processing. The modification instructions are sent to the driver files module. After modification, the new data is sent to (modified) **buffer.dat**.

The modification module consists of the following files,

- **act_read.cxx**, which reads the proposed actions stored in **action.dat**,
- **des_read.cxx**, which reads the optimization set-up and design data stored in **buffer.dat**,
- **mod.cxx**, which decides on what to do with the proposed actions.

4.11. SEQUENCE OF MODULES EXECUTION.

ESSO execution is started by running *set-up*. The first step the users have to do is to supply the design requirements. The design input module asks the user to supply,

- type of analysis,
- the constraint set,
- FE data which includes the number of nodes, number of elements, element types, and number of load cases,
- number of design variables.

Based on the above information, the memory module searches for a similar case in its data base and if one is found the associated optimization set up solution is proposed to the user. The reason for proposing a case is also presented. Having accepted the solution the execution moves to the element specification catered for by the element module.

The conventional module is invoked if the memory module can not find similar cases. As with the memory module, the conventional module searches for an optimization solution using the information provided during the design input stage. This is performed by a reasoning process employing a-priori knowledge. The user can query the solution proposed by this module. Having accepted the solution the execution moves to the design variables specification.

To specify the design variables the user needs to define the element types for every design variable. If incompatible element types are grouped in one design variable the user will be warned and asked to make modifications. If everything goes well then the user is asked to specify the element numbers for each design variable.

Depending upon the type of analysis used the system asks the user to supply the associated set of constraints. For example, if a static analysis is required then the user will be asked to supply constraints such as displacement, stress, or strain, and not natural frequency. For an analysis type the user has to supply one or more constraint types associated with that analysis. Most constraints require the users to supply upper and lower value of the constraint and gauge limits for each design variable.

At this stage the system runs the driver-files module. If required by the user the driver files can be displayed. With the driver files and FE model available (the NASTRAN FE model prepared by the user) the optimization system can start running. At this point the set-up phase is completed. In addition to the driver files, set-up phase also produces a file called **buffer.dat**. This file contains the design information which is required by the backend module. The file which controls the set-up phase is **es_f.cxx**. For producing the **buffer.dat**, **es_f.cxx** calls **buffsave.cxx**.

Figure 4-1 shows a flow chart of the procedure in setting-up an optimization job as implemented in the set-up phase. It shows the sequence of modules execution but does not show the modules interactions which can occur at any stage during the optimization process.

The performance of the optimization run is monitored by the run-time module which is part of the system's backend. As mentioned in (4.8) this is performed by suspending the run after three iterations to check whether the optimization is converging. If convergence is predicted the run is resumed. The optimization results are interpreted by the result interpretation module. If the optimization results are satisfactory the ESSO session stops. In the case where the results are not satisfactory or the optimization run is predicted not to be heading for convergence, the system will search for explanations and proposed actions to follow. Based on the proposed actions, the modification module modifies the optimization set up for another optimization run.

Figure 4-1 shows the sequence of modules execution but does not show the modules interactions which can occur at any stage during the optimization process. The data flow diagram among various modules is shown in figure 4-2.

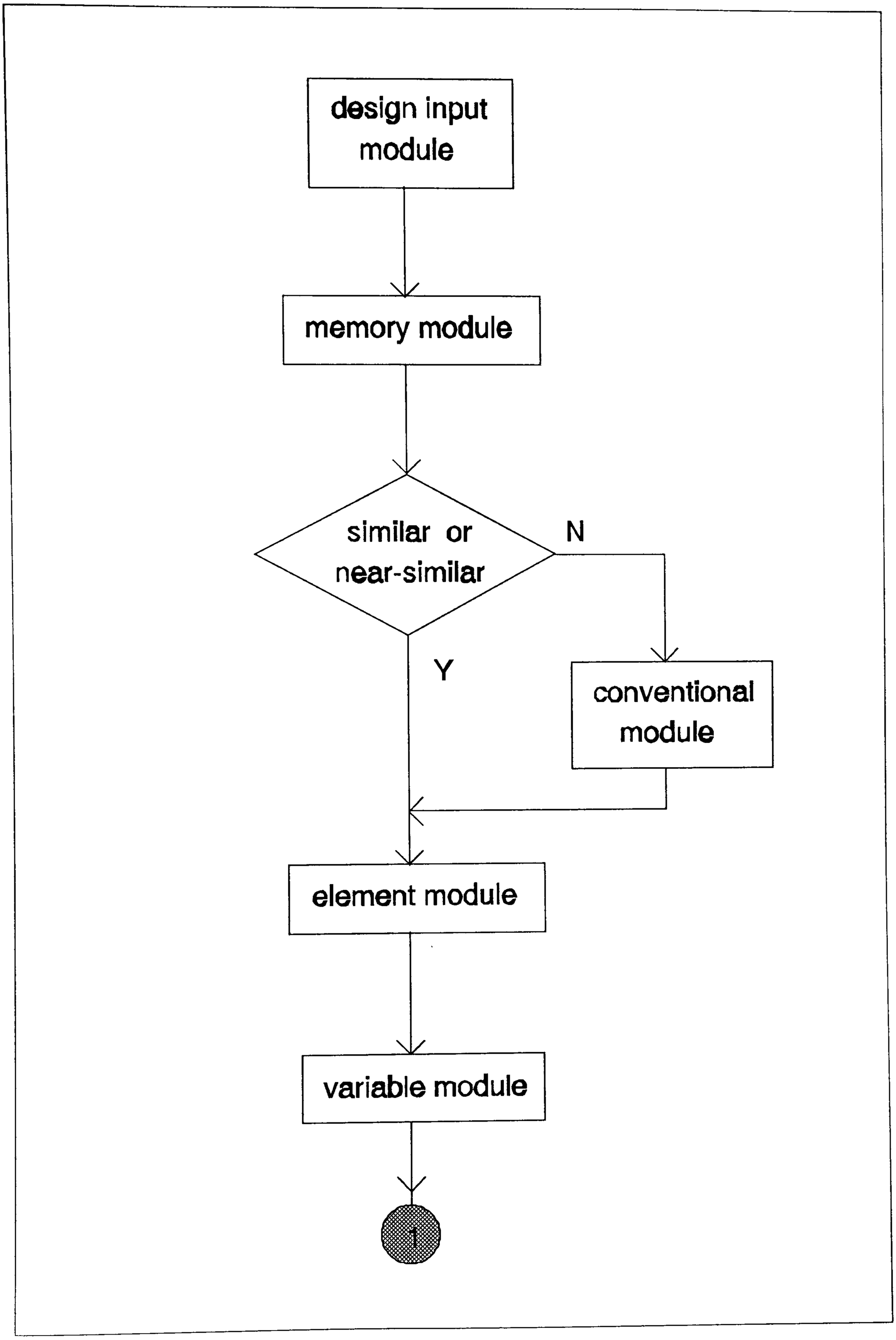


Figure 4-1 (continued on next page)

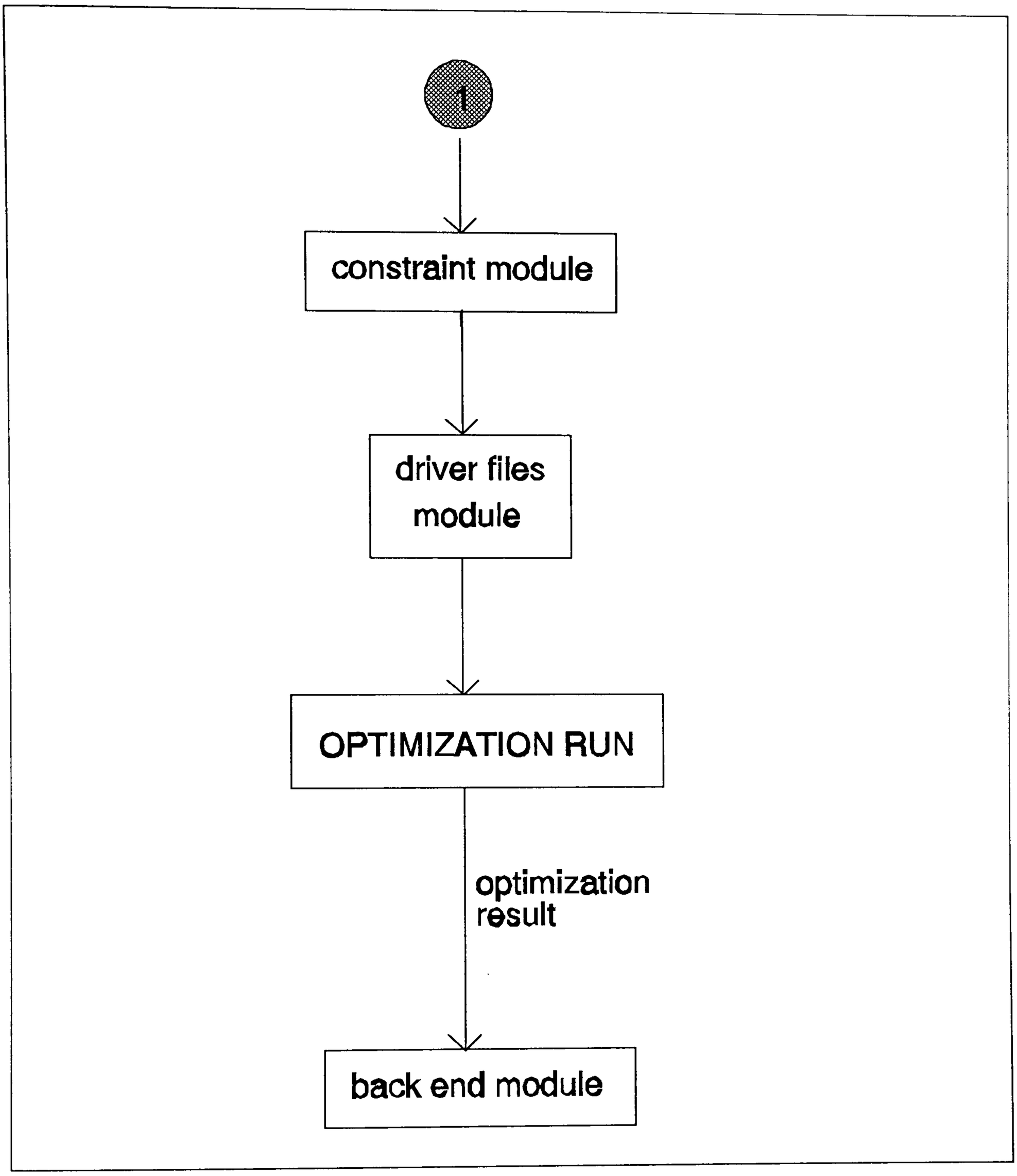


Figure 4-1 The sequence of module execution.

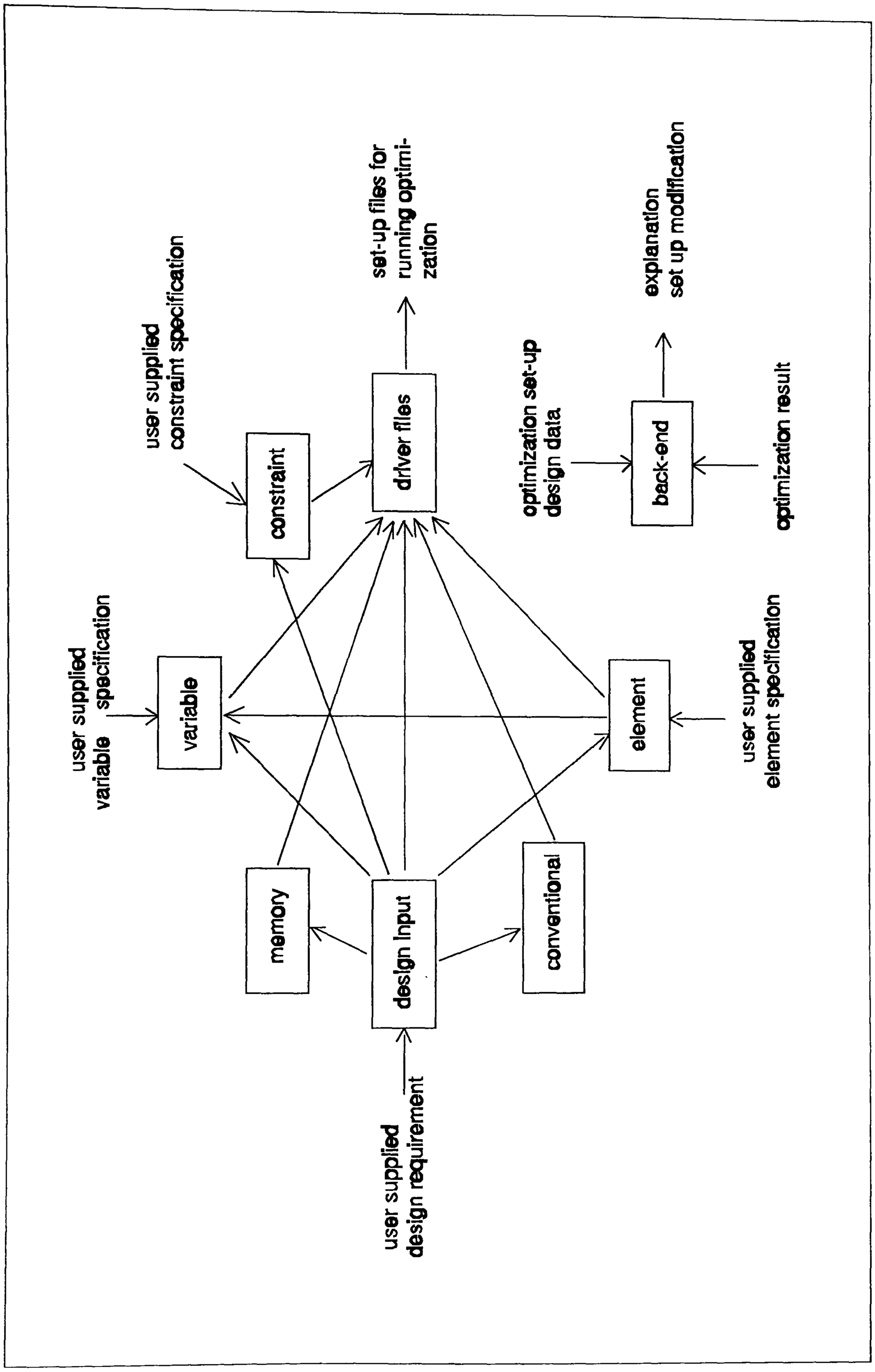


Figure 4-2 The data flow diagram.

CHAPTER 5:

INSTALLATION PROCEDURE

The ESSO programs are provided in two 3.5" floppy disks. *Disk1* contains all ESSO programs and data. *Disk2* contains FE structural models in NASTRAN format associated with past design cases. Except for the ESSO user interface program which is in C language, all other programs are in C++. Currently the provided disks are in MS-DOS format. It is possible that the programs are stored in different format (UNIX for example) and media (CD-ROM, tapes, etc.). The programs, however, are intended to run in UNIX.

All programs are provided as source codes, hence the computer needs to have C and C++ compiler. Furthermore the computer should have XView toolkit under the X Window System installed. This is required for ESSO user interface. The installation procedure described in this chapter applied to SUN SPARC-II workstation. For other systems, the user must consult the computer manager.

5.1. TRANSFERRING ESSO FILES FROM STORING-MEDIA TO COMPUTER.

To install the programs, a directory called **esso** and another subdirectory under **esso** which is called **FEdat** should be created. All the files in *disk1* should be copied to the directory **esso**. This is done by inserting the *disk1* into the floppy disk drive and making **esso** as the active directory. Then type the command,

```
mcopy a:*.*
```

in **esso** directory. This will copy all files from *disk1* into esso directory. This command is only possible if the computer has MTOOLS installed which emulates DOS-like commands for floppy drives. Different way of transferring the files are required if these files are stored in different format and media. And this should be consulted from the relevant computer manual.

The files in *disk2* should be copied to directory FEdat. This is done by making **FEdat** as the active directory. Insert *disk2* to the floppy drive and enter the command,

```
mcopy a:*.*
```

in **FEdat** directory. All files in *disk2* will be copied to the directory **FE_dat**. Again, if the files are stored in different format, different way of transferring the files are required.

5.2. COMPILING AND LINKING.

As the files provided for installation are in the form of source codes (in C and C++), it is necessary to compile and link these programs into executable files. ESSO executable files consist of three application programs and a user interface program. The application programs are,

(i) **es_f**, for setting-up optimization,
(ii) **es_run**, for monitoring the optimization run, and,
(iii) **es_res**, for interpreting the optimization results,
(iv) **es_mod**, for set-up modification.

The user interface program is **esso**. The compiling and linking process to obtain the above executable files is described in this section.

5.2.1. COMPILING AND LINKING OF ES_F.

The compiling process for obtaining **es_f** includes all files in the modules,

- the design input module,
- the memory module,
- the conventional module,
- the variable module,
- the element module,
- the constraint module,
- the driver-files module,

and files associated with the front end controller. The command for compiling and linking these files using C++ compiler is,

CC -o es_f es_f.cxx

and entered from **esso** directory.

5.2.2. COMPILING AND LINKING OF ES_RUN.

The **es_run** is obtained from all files associated with the run-time module which is part of the backend module. The command for compiling and linking is,

CC -o es_run es_run.cxx -lm

and entered from **esso** directory. The parameter **-lm** is required to include C++ mathematical routines which is not available in the standard header file **stdlib.h**.

5.2.3. COMPILING AND LINKING OF ES_RES.

The **es_res** is obtained from all files associated with the result interpretation module which is part of the backend module. The command for compiling and linking is,

CC -o es_res es_res.cxx

and entered from **esso** directory.

5.2.4. COMPILING AND LINKING OF ES_MOD.

The **es_mod** is obtained from all files associated with the modification module which is part of the backend module. The command for compiling and linking is,

CC -o es_mod es_mod.cxx

and entered from **esso** directory.

5.2.5. COMPILING AND LINKING OF ESSO.

The user interface program, **esso**, was written in C language (**esso.c**) and based on XView. This requires that XView toolkit under the X Window System installed. The command for compiling and linking (in SUN SPARC II) is,

cc -o esso esso.c -I/applics/vol1/openwin3/include
-lxview -lolgx -lX11

entered from esso sub-directory.

Another C file is **xl_proc.c** which is required for invoking MSC/XL. The command for compiling and linking is,

cc -o xl_proc xl_proc.c -I/applics/vol1/openwin3/include
-lxview -lolgx -lX11

entered from esso sub-directory. The executable **xl_proc** is called by **esso** when MSC/XL is needed.

CHAPTER 6:

EXAMPLES

In this chapter two examples of ESSO sessions are presented. The first example is concerned with a static case optimization. The second example deals with a dynamic problem.

6.1. STATIC CASE EXAMPLE.

6.1.1. PROBLEM DESCRIPTION.

A wing structure (cantilever-wing) is designed to carry a static load which is assumed to be applied at the wing tip. The structure consists of upper and lower panels, front and rear spars, and wing ribs. The length of the wing is 3600 mm (half-span). The root-chord length is 2400 mm with a maximum thickness of 600 mm. The tip-chord length is 800 mm with a maximum thickness of 200 mm. The finite element model of the wing is shown in figure 6-1 and consists of 90 nodal points and 112 elements. CQUAD4 membrane elements are used for the wing panels and ribs. For front and rear spars CSHEAR elements are used. The FE model in NASTRAN format is shown in table 6-1.

The tensile strength of the material is 250.0 MPa while the compressive strength is -120.0 MPa. The other material data used is:

Modulus Young = 79000.0 MPa
poisson's ratio = 0.33
mass density = 2.8E-6 Kg/mm^3

The structure is to be designed to be of minimum weight subject to limits on stresses given above and on the lateral displacement of 25 mm at nodes 81 to 85 (wing-tip). The gauge limits are as follows, thickness of the structural elements should be not less than 0.6 mm. It is specified that maximum thickness of upper and lower panel is 10 mm, the maximum thickness of spars is 3.5 mm, and the maximum thickness of ribs is 2.5 mm.

6.1.2. ESSO SESSION.

ESSO starts the session by asking the user to supply information relating to,
- type of analysis, in this case statics,
- type of constraints, stress and displacement.

The user must now supply a general description of the finite element model of the structure, which consists of,

- the number of nodal/grid points, which is 90,
- the element types, which in this case are CQUAD4 and CSHEAR,
- the number of elements, which is 112,
- and the number of load cases, in this case is 1.

Ideally the system should draw this information directly from the FE (NASTRAN) file, but at its current stage ESSO cannot do this. The user now informs the system of the number of design variables required, 6 in this example. It is up to the user to define the set up of the design variables which link specific finite elements to a design variable. For example if the panel of a wing bay (between two ribs) is desired to have a uniform thickness, then all finite elements in that bay are grouped in one design variable.

Based on the above information, ESSO searches its memory using the reasoning process described earlier and finds one "similar" case, that is,

case index number 3 (from static cases in memory),
number of analysis = 1, statics,
number of constraint type is 2, stress and displacement,
number of nodal points = 24,
number of elements = 26,
number of element types = 1, CQUAD4,
number of design variables = 5,
name of solution algorithm = Pseudo Newton,
maximum number of solution iteration = 4,
convergence criteria = dual gap.

The finite element model for the above case is shown in figure 6-2. The certainty factor of this case (indicating the degree to which this solution algorithm can solve the current problem successfully) is 0.381. The same analysis and constraint types give positive contribution to similarity, though the number of nodal points, elements, and design variables of this past case are smaller than the current design problem. This lack of total agreement does not become a kill off factor because the analysis type is static and ESSO believes there is a reasonable chance of success in using this case. The resulting algorithm proposed as a result of this process is the pseudo newton with maximum iteration numbers of 4. It is assumed that the user agrees with this proposal.

The next step is design variables set up. The user informs ESSO of the element types in each design variable (there are six design variables in the present case), and ESSO will check whether the elements grouping is appropriate. The element types of the design variables are,

variable-1 = CQUAD4,
variable-2 = CQUAD4,
variable-3 = CQUAD4,
variable-4 = CQUAD4,
variable-5 = CSHEAR,
variable-6 = CQUAD4.

The above grouping is accepted by ESSO and the user then has to supply the element numbers for each of these design variables,

variable-1 = 33,34,49,50,81,82,97,98 (the first upper and lower panel bays near the wing root)
variable-2 = 35,36,51,52,83,84,99,100 (the second upper and lower panel bays)
variable-3 = 37-48, 53-64 (the rest of the upper panel)
variable-4 = 85-96, 101-112 (the rest of the lower panel)
variable-5 = 65-80 (the front and rear spars)
variable-6 = 113-144 (the ribs).

The gauge and stress constraints are now supplied:

Gauges:

1. lower-limit = 0.6
   upper-limit = 10.0
   element = 33-64, 81-112 (upper and lower panels)

2. lower-limit = 0.6
   upper-limit = 3.5
   element = 65-80 (front and rear spars)

3. lower-limit = 0.6
   upper-limit = 2.5
   element = 113-144 (ribs)

Stress: lower-limit = -120.0
upper-limit = 250.0
for elements 33-144.

ESSO also asks the user to supply the displacement constraint:

lower-limit = -25.0
upper-limit = 25.0, to Y-direction
at nodes = 81-85.

At this stage ESSO has performed the necessary reasoning to construct a solution strategy and the user has supplied the description of the design problem. It is now a relatively straightforward for ESSO to create appropriate input data sets for the structural optimization, thus, ESSO now informs the user that the front end session is completed and the STARS driver files "essocdf.dat" (see table 6-2) and "essocons.dat" (see table 6-3) are produced. ESSO also requests the user to run STARS. When executed, the STARS run is suspended after 3 iterations. This suspension is invoked to allow ESSO to examine the progress of the optimizer and asses the likely success of the current run to converge to an optimum.

ESSO employs a number of parameters to predict convergence or otherwise as described earlier. In the present case the iteration history up to the third iteration (can be read from the STARS result file "essorslt.dat") is shown below.

iteration	feasible weight	dual weight
0	127.9	48.3
1	109.5	77.0
2	106.5	99.6
3	106.2	104.8

Based on the above information, the run-time application predicts that the optimization will reach convergence at maximum iteration (maximum iteration=4). In fact the above data shows that the optimization run has produced a very good design at iteration 3.

The STARS run is resumed with one more iteration to complete. After the optimization stops at fourth iteration, the result application is invoked to check whether this design can be regarded as optimum. This ESSO application uses dual weight of the last iteration (iteration 4) and the most non-feasible constraint (if any) together with any violated limit value. In the present case we find that at the optimum,

- feasible weight = 106.2, dual weight = 106.1,

- the "violated" constraint is displacement at node 83 equal to 25.0 mm with the limit value of 25.0 mm.

Based on this data, ESSO concludes that the design at iteration 4 can indeed be regarded as optimum because the constraints are satisfied while the dual gap is satisfied.

Because the optimization produces an optimum design, the ESSO memory is updated with this design problem.

6.2. DYNAMIC CASE EXAMPLE.

6.2.1. PROBLEM DESCRIPTION.

The same structure as in the first case (6.1) is used again for this second design case. Instead of the static design requirement, now the structure is desired to have a minimum weight while maintaining its lowest natural frequency above 2.50 cycles/second. This is a dynamic case with frequency constraint. The FE model and the set up of the design variables of the first case are retained. The same material is also used with similar gauge limits on elements. The FE model is similar to the one in figure 6-1 omitting the static forces. Table 6-4 shows the NASTRAN data of the model.

6.2.2. ESSO SESSION.

Similar to the first case, ESSO starts the session by asking the user to supply information relating to,

- type of analysis, which in this case is dynamics,
- type of constraint, which is natural frequency.

This is followed by the general description of the finite element model of the structure, and this consists of,

- number of nodal/grid points, which is 90,
- element types, which in this case are CQUAD4 and CSHEAR,
- number of elements, which is 112,
- and number of load cases, which in this case is 1.

The user then must inform the system on the number of design variables required, which we have selected to be 6.

Based on the above information, ESSO suggests the following as a suitable "similar" case,

```
case index number 1 (from dynamic cases in memory),
number of analysis = 1, dynamics,
number of constraint type is 1, natural frequency,
number of nodal points = 90,
number of elements = 148,
number of element types = 3, CQUAD4, CSHEAR, CONROD,
number of design variables = 8,
name of solution algorithm = Pseudo Newton,
maximum number of solution iteration = 7,
convergence criteria = dual gap
```

The finite element model of the above case can be seen in figure 6-3. The certainty factor of this case is 0.473. In addition to the front and rear spars, the structure in the "similar" proposed case has a middle spar. We now assume that the user is satisfied that there is a good chance of success in using this case to supply the optimization solution strategy for the present example. The algorithm proposed is the pseudo newton with maximum iteration numbers of 7. It is assumed that the user prefers to increase the maximum iteration numbers to 10.

The next step is design variables set up. The element types associated with the design variables are,

```
variable-1 = CQUAD4,
variable-2 = CQUAD4,
variable-3 = CQUAD4,
variable-4 = CQUAD4,
variable-5 = CSHEAR,
variable-6 = CQUAD4,
```

and the element within each design variable,

```
variable-1 = 33,34,49,50,81,82,97,98 (the first upper and
             lower panel bays near the wing root)
variable-2 = 35,36,51,52,83,84,99,100 (the second upper
             and lower panel bays)
variable-3 = 37-48, 53-64 (the rest of the upper panel)
variable-4 = 85-96, 101-112 (the rest of the lower panel)
variable-5 = 65-80 (the front and rear spars)
variable-6 = 113-144 (the ribs).
```

Similar to the first case, there is no fix-element.

The gauges limit are specified as:

```
1. lower-limit = 0.6
   upper-limit = 10.0
   element     = 33-64, 81-112 (upper and lower panels)

2. lower-limit = 0.6
   upper-limit = 3.5
   element     = 65-80 (front and rear spars)

2. lower-limit = 0.6
   upper-limit = 2.5
   element     = 113-144 (ribs)
```

Next, ESSO asks the user to supply the lower limit value of the frequency and the associated mode number. In this case,

```
mode number = 1
natural frequency = 2.50.
```

At this stage ESSO inform the user that the front end session is completed and the STARS driver files "essocdf.dat" (see table 6-5) and "essocons.dat" (see table 6-6) are produced. ESSO also requests the user to run STARS. As with the previous "statics" case the run is suspended after 3 iterations.

The iteration history for these initial iterations is:

iteration	feasible weight	dual weight
0	127.9	35.9
1	80.4	35.9
2	107.9	35.9
3	168.7	35.9

Based on the above information, the run-time application predicts that the optimization will not converge to the optimum at maximum iteration (maximum iteration=10).

At this point, the optimizer has been able to raise the natural frequency of the structure from the starting value of 1.54 Hz to 2.0 Hz. However, ESSO believes that the optimum will not be reached because there is insufficient scope to make changes due to the design variable set up. ESSO, thus, proposes that number of design variables be increased. In response to this the number of design variables is increased to 17.

The element types associated with the design variables are,

variable-1 = CQUAD4, the first bay of upper and lower panels,
variable-2 = CQUAD4, the second bay of upper and lower panels,
variable-3 = CQUAD4, the third bay of upper and lower panels,
variable-4 = CQUAD4, the fourth bay of upper and lower panels,
variable-5 = CQUAD4, the fifth bay of upper and lower panels,
variable-6 = CQUAD4, the sixth bay of upper and lower panels,
variable-7 = CQUAD4, the seventh bay of upper and lower panels
variable-8 = CQUAD4, the eighth bay of upper and lower panels,
variable-9 = CQUAD4, the ribs,
variable-10 = CSHEAR, the first bay of front and rear spars,

variable-11 = CSHEAR, the second bay of front and rear spars,
variable-12 = CSHEAR, the third bay of front and rear spars,
variable-13 = CSHEAR, the fourth bay of front and rear spars,
variable-14 = CSHEAR, the fifth bay of front and rear spars,
variable-15 = CSHEAR, the sixth bay of front and rear spars,
variable-16 = CSHEAR, the seventh bay of front and rear spars,
variable-17 = CSHEAR, the eighth bay of front and rear spars.

The associated elements within each design variable can be seen in table 6-7.

Having completed the modification, ESSO produces "essocdf.dat" and "essocons.dat". The "essocons.dat" is shown in table 6-7. ESSO then asks the user to run STARS.

After 3 iterations STARS run is suspended. The ESSO's run-time application asks the user to supply data on feasible and dual weights from iteration 0 to iteration 3 which is shown below,

iteration	feasible weight	dual weight
0	127.9	35.9
1	72.1	34.7
2	71.6	48.3
3	76.8	56.4

Based on the above information, the run-time application predicts that the optimization will reach convergence at maximum iteration (maximum iteration=10) and asks the user to resume the optimization run.

The optimization run stops at maximum iteration and the ESSO's result application is called to check the results. It asks the user to supply the feasible and dual weights of the last iteration (iteration 10) and also the most non-feasible constraint (if any) together with the violated limit value. From the "essorslt.dat",

- feasible weight = 78.3, dual weight = 66.0,

- the "violated" constraint is the natural frequency of 2.39 Hz which is still below the limit value of 2.5 Hz.

From the data above ESSO concludes that the design is not satisfactory, but the constraint relatively small, under five percent. At this stage, ESSO asks the user to supply the history of frequency during optimization run which is as follow,

f0	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10
1.54	1.73	2.05	2.29	2.42	2.36	2.42	2.38	2.41	2.40	2.39

The data above shows the frequency at every iteration from initial value, f0, to the last iteration, f10. Based only on the above data, in fact the optimization is converging to a design point where the natural frequency of the structure is close to 2.4 Hz (still infeasible). From this fact ESSO concludes that the problem is caused by the algorithm itself (Pseudo Newton). To solve this problem (an infeasible design) ESSO suggests the user to increase slightly the constraint value. This is accepted by increasing the natural frequency to 2.65 Hz.

With this slight modification, the STARS is re-run. The optimization data after ten iteration is

- feasible weight = 95.9, dual weight = 72.5,
- the natural frequency is 2.53 Hz which is above the original limit value of 2.50 Hz.

The data above shows that the design is not yet optimum but it is a useful feasible design (with respect to the original design requirement) and probably not that far away from the optimum point. The frequency history as shown below,

f0	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10
1.54	1.75	2.09	2.37	2.56	2.48	2.56	2.48	2.54	2.48	2.53

shows that the optimization is converging to a design point with the frequency close to 2.5 Hz. This supports the fact points out by ESSO that the Pseudo Newton method of STARS could not cope well with this dynamic problem.

Finally ESSO updates its memory by appending this case and asking the user to put comments on the running of the optimization. The comments might look like this,

- natural frequency requirement of at least 2.5 Hz.
- initial frequency of structure (iter-0) = 1.54 Hz.

- needs effective material distribution through appropriate design variable set-up
- optimization tends to point to a slightly infeasible design
- the trick is to slightly increase the constraint to 2.65 Hz.

Table 6-1: The FE data of the Static Case Example.

```
==========================================================
TITLE = WING STATIC-CASE STRESS-DISPLACEMENT CONSTRAINT
SUBCASE=1
LOAD=1
BEGIN BULK
GRID            1                0.     300.       0.          123456
GRID            2                0.     270.     600.          123456
GRID            3                0. 221.052    -600.          123456
GRID            4                0. 142.104   -1200.          123456
GRID            5                0.     140.    1200.          123456
GRID            6                0.-142.105   -1200.          123456
GRID            7                0.    -200.    1200.          123456
GRID            8                0.-221.052    -600.          123456
GRID            9                0.    -300.     600.          123456
GRID           10                0.    -300.       0.          123456
GRID           11              450.     275.       0.             456
GRID           12              450.    247.5     550.             456
GRID           13              450. 202.631    -550.             456
GRID           14              450. 130.813    1100.             456
GRID           15              450. 130.262   -1100.             456
GRID           16              450.-130.263   -1100.             456
GRID           17              450.-186.875    1100.             456
GRID           18              450.-202.632    -550.             456
GRID           19              450.    -275.     550.             456
GRID           20              450.    -275.       0.             456
GRID           21         900.0001     250.       0.             456
GRID           22         900.0001     225.     500.             456
GRID           23         900.0001   184.21    -500.             456
GRID           24         900.0001 121.625    1000.             456
GRID           25         900.0001 118.421   -1000.             456
GRID           26         900.0001-118.421   -1000.             456
GRID           27         900.0001 -173.75    1000.             456
GRID           28         900.0001-184.211    -500.             456
GRID           29         900.0001    -250.     500.             456
GRID           30         900.0001    -250.       0.             456
GRID           31             1350.     225.       0.             456
GRID           32             1350.    202.5     450.             456
GRID           33             1350. 165.789    -450.             456
GRID           34             1350. 112.438900.0001             456
GRID           35             1350. 106.579-900.000             456
GRID           36             1350.-106.579-900.000             456
GRID           37             1350.-160.625900.0001             456
GRID           38             1350. -165.79    -450.             456
GRID           39             1350.    -225.     450.             456
GRID           40             1350.    -225.       0.             456
----------------------------------------------------------
```

(continued)

Table 6-1

GRID	41	1800.	200.	0.	456
GRID	42	1800.	180.	400.	456
GRID	43	1800.	147.368	-400.	456
GRID	44	1800.	103.25	800.0001	456
GRID	45	1800.	94.73701	-800.000	456
GRID	46	1800.	-94.7375	-800.000	456
GRID	47	1800.	-147.369	-400.	456
GRID	48	1800.	-147.5	800.0001	456
GRID	49	1800.	-200.	400.	456
GRID	50	1800.	-200.	0.	456
GRID	51	2250.	175.	0.	456
GRID	52	2250.	157.5	350.	456
GRID	53	2250.	128.948	-350.	456
GRID	54	2250.	94.06251	700.0001	456
GRID	55	2250.	82.89521	-700.000	456
GRID	56	2250.	-82.8956	-700.000	456
GRID	57	2250.	-128.948	-350.	456
GRID	58	2250.	-134.375	700.0001	456
GRID	59	2250.	-175.	350.	456
GRID	60	2250.	-175.	0.	456
GRID	61	2700.	150.	0.	456
GRID	62	2700.	135.	300.	456
GRID	63	2700.	110.527	-300.	456
GRID	64	2700.	84.87501	600.	456
GRID	65	2700.	71.0535	-600.	456
GRID	66	2700.	-71.0537	-600.	456
GRID	67	2700.	-110.527	-300.	456
GRID	68	2700.	-121.25	600.	456
GRID	69	2700.	-150.	300.	456
GRID	70	2700.	-150.	0.	456
GRID	71	3150.	125.	0.	456
GRID	72	3150.	112.5	250.	456
GRID	73	3150.	92.10591	-250.	456
GRID	74	3150.	75.68751	500.	456
GRID	75	3150.	59.21181	-500.	456
GRID	76	3150.	-59.2119	-500.	456
GRID	77	3150.	-92.1059	-250.	456
GRID	78	3150.	-108.125	500.	456
GRID	79	3150.	-125.	250.	456
GRID	80	3150.	-125.	0.	456
GRID	81	3600.	100.	0.	456
GRID	82	3600.	90.00001	200.	456
GRID	83	3600.	73.68501	-200.	456
GRID	84	3600.	66.50001	400.	456
GRID	85	3600.	47.37001	-400.	456

(continued)

Table 6-1

```
GRID            86                3600.-47.3700   -400.                   456
GRID            87                3600.-73.6850   -200.                   456
GRID            88                3600.-95.0000    400.                   456
GRID            89                3600.   -100.    200.                   456
GRID            90                3600.   -100.      0.                   456
MAT1             1  79000.                 0.33  2.80-6
PSHELL          33        1       1.
PSHEAR          65        1       1.
FORCE            1       81            5000.0      0.0      1.0      0.0
FORCE            1       82           10000.0      0.0      1.0      0.0
FORCE            1       83           12000.0      0.0      1.0      0.0
FORCE            1       84           10000.0      0.0      1.0      0.0
FORCE            1       85            5000.0      0.0      1.0      0.0
CQUAD4          33       33        5       14       12        2       0.
CQUAD4          34       33        2       12       11        1       0.
CQUAD4          35       33       14       24       22       12       0.
CQUAD4          36       33       12       22       21       11       0.
CQUAD4          37       33       24       34       32       22       0.
CQUAD4          38       33       22       32       31       21       0.
CQUAD4          39       33       34       44       42       32       0.
CQUAD4          40       33       32       42       41       31       0.
CQUAD4          41       33       44       54       52       42       0.
CQUAD4          42       33       42       52       51       41       0.
CQUAD4          43       33       54       64       62       52       0.
CQUAD4          44       33       52       62       61       51       0.
CQUAD4          45       33       64       74       72       62       0.
CQUAD4          46       33       62       72       71       61       0.
CQUAD4          47       33       74       84       82       72       0.
CQUAD4          48       33       72       82       81       71       0.
CQUAD4          49       33        1       11       13        3       0.
CQUAD4          50       33        3       13       15        4       0.
CQUAD4          51       33       11       21       23       13       0.
CQUAD4          52       33       13       23       25       15       0.
CQUAD4          53       33       21       31       33       23       0.
CQUAD4          54       33       23       33       35       25       0.
CQUAD4          55       33       31       41       43       33       0.
CQUAD4          56       33       33       43       45       35       0.
CQUAD4          57       33       41       51       53       43       0.
CQUAD4          58       33       43       53       55       45       0.
CQUAD4          59       33       51       61       63       53       0.
CQUAD4          60       33       53       63       65       55       0.
CQUAD4          61       33       61       71       73       63       0.
CQUAD4          62       33       63       73       75       65       0.
CQUAD4          63       33       71       81       83       73       0.
CQUAD4          64       33       73       83       85       75       0.
```

(continued)

Table 6-1

CQUAD4	81	33	6	16	18	8	0.
CQUAD4	82	33	8	18	20	10	0.
CQUAD4	83	33	16	26	28	18	0.
CQUAD4	84	33	18	28	30	20	0.
CQUAD4	85	33	26	36	38	28	0.
CQUAD4	86	33	28	38	40	30	0.
CQUAD4	87	33	36	46	47	38	0.
CQUAD4	88	33	38	47	50	40	0.
CQUAD4	89	33	46	56	57	47	0.
CQUAD4	90	33	47	57	60	50	0.
CQUAD4	91	33	56	66	67	57	0.
CQUAD4	92	33	57	67	70	60	0.
CQUAD4	93	33	66	76	77	67	0.
CQUAD4	94	33	67	77	80	70	0.
CQUAD4	95	33	76	86	87	77	0.
CQUAD4	96	33	77	87	90	80	0.
CQUAD4	97	33	10	20	19	9	0.
CQUAD4	98	33	9	19	17	7	0.
CQUAD4	99	33	20	30	29	19	0.
CQUAD4	100	33	19	29	27	17	0.
CQUAD4	101	33	29	39	37	27	0.
CQUAD4	102	33	30	40	39	29	0.
CQUAD4	103	33	40	50	49	39	0.
CQUAD4	104	33	39	49	48	37	0.
CQUAD4	105	33	50	60	59	49	0.
CQUAD4	106	33	49	59	58	48	0.
CQUAD4	107	33	60	70	69	59	0.
CQUAD4	108	33	59	69	68	58	0.
CQUAD4	109	33	70	80	79	69	0.
CQUAD4	110	33	69	79	78	68	0.
CQUAD4	111	33	80	90	89	79	0.
CQUAD4	112	33	79	89	88	78	0.
CQUAD4	113	33	17	14	12	19	0.
CQUAD4	114	33	19	12	11	20	0.
CQUAD4	115	33	20	11	13	18	0.
CQUAD4	116	33	18	13	15	16	0.
CQUAD4	117	33	27	24	22	29	0.
CQUAD4	118	33	29	22	21	30	0.
CQUAD4	119	33	30	21	23	28	0.
CQUAD4	120	33	28	23	25	26	0.
CQUAD4	121	33	37	34	32	39	0.
CQUAD4	122	33	39	32	31	40	0.
CQUAD4	123	33	40	31	33	38	0.
CQUAD4	124	33	38	33	35	36	0.

(continued)

Table 6-1

```
--------------------------------------------------------------------------------
CQUAD4          125          33          48          44          42          49          0.
CQUAD4          126          33          49          42          41          50          0.
CQUAD4          127          33          50          41          43          47          0.
CQUAD4          128          33          47          43          45          46          0.
CQUAD4          129          33          58          54          52          59          0.
CQUAD4          130          33          59          52          51          60          0.
CQUAD4          131          33          60          51          53          57          0.
CQUAD4          132          33          57          53          55          56          0.
CQUAD4          133          33          68          64          62          69          0.
CQUAD4          134          33          69          62          61          70          0.
CQUAD4          135          33          70          61          63          67          0.
CQUAD4          136          33          67          63          65          66          0.
CQUAD4          137          33          78          74          72          79          0.
CQUAD4          138          33          79          72          71          80          0.
CQUAD4          139          33          80          71          73          77          0.
CQUAD4          140          33          77          73          75          76          0.
CQUAD4          141          33          88          84          82          89          0.
CQUAD4          142          33          89          82          81          90          0.
CQUAD4          143          33          90          81          83          87          0.
CQUAD4          144          33          87          83          85          86          0.
CSHEAR           65          65           5          14          17           7
CSHEAR           66          65          14          24          27          17
CSHEAR           67          65          24          34          37          27
CSHEAR           68          65          34          44          48          37
CSHEAR           69          65          44          54          58          48
CSHEAR           70          65          54          64          68          58
CSHEAR           71          65          64          74          78          68
CSHEAR           72          65          74          84          88          78
CSHEAR           73          65           4          15          16           6
CSHEAR           74          65          15          25          26          16
CSHEAR           75          65          25          35          36          26
CSHEAR           76          65          35          45          46          36
CSHEAR           77          65          45          55          56          46
CSHEAR           78          65          55          65          66          56
CSHEAR           79          65          65          75          76          66
CSHEAR           80          65          75          85          86          76
ENDDATA
================================================================================
```

(end of table)

Table 6-2: The command data file of the Static Case Example.

```
==========================================================================
START
COMMAND DATA PNEWTON
C
      SET NSIV=1;
      SET MXIT=4;
      SET APRT=YES;
      SET ACTG=YES;
      SET IFLM=YES;
      DO NSIN;
      DO OPTIN;
C
      DO ANALYSIS;
      DO INSPECT;
      DO OPTX;
C     PSEUDO NEWTON ITERATIONS
L010: DO ITER;
      DO QNEWTON;
      DO OPTX;
      IF(ENDC.EQ.YES) GOTO L031;
      IF(NOIT.EQ.3) DO HISTORY;
      IF(NOIT.EQ.3) SUSPEND;
      IF(ENDI.EQ.YES) GOTO L040;
      GOTO L010;
L031: DISPLAY CONVERGENCE TO THE LOWER BOUND WEIGHT;
      GOTO L050;
L032: DISPLAY CONVERGENCE TO DESIGN VARIABLES;
      GOTO L050;
L033: DISPLAY CONVERGENCE TO FEASIBLE VS ACTUAL WEIGHT;
      GOTO L050;
L034: DISPLAY CONVERGENCE TO FEASIBLE WEIGHT;
      GOTO L050;
L040: DISPLAY MAX. NUMBER OF ITERATIONS REACHED;
C
L050: SET APRT=YES;
      DO ANALYSIS;
      DO INSPECT;
      DO HIST;
      STOP;
      END;
C     ANALYSIS PROCEDURE;
      PROC OPTX;
      DO ANALYSIS;
      DO ACTSET;
      DO DERV;
      DO CONV;
      DO INSPECT;
      END OPTX;
ENDATA
==========================================================================
```

Table 6-3: The design/constraint file of the Static Case Example.

```
================================================================================
C234567890123456789012345678901234567890123456789012345678901234567890
DESIGN              1         33
DESIGN              1         34
DESIGN              1         49
DESIGN              1         50
DESIGN              1         81
DESIGN              1         82
DESIGN              1         97
DESIGN              1         98
DESIGN              2         35
DESIGN              2         36
DESIGN              2         51
DESIGN              2         52
DESIGN              2         83
DESIGN              2         84
DESIGN              2         99
DESIGN              2         100
DESIGN              3         37
DESIGN              3         38
DESIGN              3         39
DESIGN              3         40
DESIGN              3         41
DESIGN              3         42
DESIGN              3         43
DESIGN              3         44
DESIGN              3         45
DESIGN              3         46
DESIGN              3         47
DESIGN              3         48
DESIGN              3         53
DESIGN              3         54
DESIGN              3         55
DESIGN              3         56
DESIGN              3         57
DESIGN              3         58
DESIGN              3         59
DESIGN              3         60
DESIGN              3         61
DESIGN              3         62
DESIGN              3         63
DESIGN              3         64
DESIGN              4         85
DESIGN              4         86
DESIGN              4         87
DESIGN              4         88
DESIGN              4         89
DESIGN              4         90
--------------------------------------------------------------------------------
```

(continued)

Table 6-3

DESIGN	4	91
DESIGN	4	92
DESIGN	4	93
DESIGN	4	94
DESIGN	4	95
DESIGN	4	96
DESIGN	4	101
DESIGN	4	102
DESIGN	4	103
DESIGN	4	104
DESIGN	4	105
DESIGN	4	106
DESIGN	4	107
DESIGN	4	108
DESIGN	4	109
DESIGN	4	110
DESIGN	4	111
DESIGN	4	112
DESIGN	5	65
DESIGN	5	66
DESIGN	5	67
DESIGN	5	68
DESIGN	5	69
DESIGN	5	70
DESIGN	5	71
DESIGN	5	72
DESIGN	5	73
DESIGN	5	74
DESIGN	5	75
DESIGN	5	76
DESIGN	5	77
DESIGN	5	78
DESIGN	5	79
DESIGN	5	80
DESIGN	6	113
DESIGN	6	114
DESIGN	6	115
DESIGN	6	116
DESIGN	6	117
DESIGN	6	118
DESIGN	6	119
DESIGN	6	120
DESIGN	6	121
DESIGN	6	122
DESIGN	6	123

(continued)

Table 6-3

DESIGN	6	124	
DESIGN	6	125	
DESIGN	6	126	
DESIGN	6	127	
DESIGN	6	128	
DESIGN	6	129	
DESIGN	6	130	
DESIGN	6	131	
DESIGN	6	132	
DESIGN	6	133	
DESIGN	6	134	
DESIGN	6	135	
DESIGN	6	136	
DESIGN	6	137	
DESIGN	6	138	
DESIGN	6	139	
DESIGN	6	140	
DESIGN	6	141	
DESIGN	6	142	
DESIGN	6	143	
DESIGN	6	144	
GAUGE	0.600000	10.0000	33
GAUGE	0.600000	10.0000	34
GAUGE	0.600000	10.0000	35
GAUGE	0.600000	10.0000	36
GAUGE	0.600000	10.0000	37
GAUGE	0.600000	10.0000	38
GAUGE	0.600000	10.0000	39
GAUGE	0.600000	10.0000	40
GAUGE	0.600000	10.0000	41
GAUGE	0.600000	10.0000	42
GAUGE	0.600000	10.0000	43
GAUGE	0.600000	10.0000	44
GAUGE	0.600000	10.0000	45
GAUGE	0.600000	10.0000	46
GAUGE	0.600000	10.0000	47
GAUGE	0.600000	10.0000	48
GAUGE	0.600000	10.0000	49
GAUGE	0.600000	10.0000	50
GAUGE	0.600000	10.0000	51
GAUGE	0.600000	10.0000	52
GAUGE	0.600000	10.0000	53
GAUGE	0.600000	10.0000	54
GAUGE	0.600000	10.0000	55
GAUGE	0.600000	10.0000	56
GAUGE	0.600000	10.0000	57

(continued)

Table 6-3

GAUGE	0.600000	10.0000	58
GAUGE	0.600000	10.0000	59
GAUGE	0.600000	10.0000	60
GAUGE	0.600000	10.0000	61
GAUGE	0.600000	10.0000	62
GAUGE	0.600000	10.0000	63
GAUGE	0.600000	10.0000	64
GAUGE	0.600000	10.0000	81
GAUGE	0.600000	10.0000	82
GAUGE	0.600000	10.0000	83
GAUGE	0.600000	10.0000	84
GAUGE	0.600000	10.0000	85
GAUGE	0.600000	10.0000	86
GAUGE	0.600000	10.0000	87
GAUGE	0.600000	10.0000	88
GAUGE	0.600000	10.0000	89
GAUGE	0.600000	10.0000	90
GAUGE	0.600000	10.0000	91
GAUGE	0.600000	10.0000	92
GAUGE	0.600000	10.0000	93
GAUGE	0.600000	10.0000	94
GAUGE	0.600000	10.0000	95
GAUGE	0.600000	10.0000	96
GAUGE	0.600000	10.0000	97
GAUGE	0.600000	10.0000	98
GAUGE	0.600000	10.0000	99
GAUGE	0.600000	10.0000	100
GAUGE	0.600000	10.0000	101
GAUGE	0.600000	10.0000	102
GAUGE	0.600000	10.0000	103
GAUGE	0.600000	10.0000	104
GAUGE	0.600000	10.0000	105
GAUGE	0.600000	10.0000	106
GAUGE	0.600000	10.0000	107
GAUGE	0.600000	10.0000	108
GAUGE	0.600000	10.0000	109
GAUGE	0.600000	10.0000	110
GAUGE	0.600000	10.0000	111
GAUGE	0.600000	10.0000	112
GAUGE	0.600000	3.50000	65
GAUGE	0.600000	3.50000	66
GAUGE	0.600000	3.50000	67
GAUGE	0.600000	3.50000	68
GAUGE	0.600000	3.50000	69
GAUGE	0.600000	3.50000	70
GAUGE	0.600000	3.50000	71

(continued)

Table 6-3

GAUGE	0.600000	3.50000	72
GAUGE	0.600000	3.50000	73
GAUGE	0.600000	3.50000	74
GAUGE	0.600000	3.50000	75
GAUGE	0.600000	3.50000	76
GAUGE	0.600000	3.50000	77
GAUGE	0.600000	3.50000	78
GAUGE	0.600000	3.50000	79
GAUGE	0.600000	3.50000	80
GAUGE	0.600000	2.50000	113
GAUGE	0.600000	2.50000	114
GAUGE	0.600000	2.50000	115
GAUGE	0.600000	2.50000	116
GAUGE	0.600000	2.50000	117
GAUGE	0.600000	2.50000	118
GAUGE	0.600000	2.50000	119
GAUGE	0.600000	2.50000	120
GAUGE	0.600000	2.50000	121
GAUGE	0.600000	2.50000	122
GAUGE	0.600000	2.50000	123
GAUGE	0.600000	2.50000	124
GAUGE	0.600000	2.50000	125
GAUGE	0.600000	2.50000	126
GAUGE	0.600000	2.50000	127
GAUGE	0.600000	2.50000	128
GAUGE	0.600000	2.50000	129
GAUGE	0.600000	2.50000	130
GAUGE	0.600000	2.50000	131
GAUGE	0.600000	2.50000	132
GAUGE	0.600000	2.50000	133
GAUGE	0.600000	2.50000	134
GAUGE	0.600000	2.50000	135
GAUGE	0.600000	2.50000	136
GAUGE	0.600000	2.50000	137
GAUGE	0.600000	2.50000	138
GAUGE	0.600000	2.50000	139
GAUGE	0.600000	2.50000	140
GAUGE	0.600000	2.50000	141
GAUGE	0.600000	2.50000	142
GAUGE	0.600000	2.50000	143
GAUGE	0.600000	2.50000	144
STRESS	-120.000	250.000	33
STRESS	-120.000	250.000	34
STRESS	-120.000	250.000	35
STRESS	-120.000	250.000	36

(continued)

Table 6-3

STRESS	-120.000	250.000	37
STRESS	-120.000	250.000	38
STRESS	-120.000	250.000	39
STRESS	-120.000	250.000	40
STRESS	-120.000	250.000	41
STRESS	-120.000	250.000	42
STRESS	-120.000	250.000	43
STRESS	-120.000	250.000	44
STRESS	-120.000	250.000	45
STRESS	-120.000	250.000	46
STRESS	-120.000	250.000	47
STRESS	-120.000	250.000	48
STRESS	-120.000	250.000	49
STRESS	-120.000	250.000	50
STRESS	-120.000	250.000	51
STRESS	-120.000	250.000	52
STRESS	-120.000	250.000	53
STRESS	-120.000	250.000	54
STRESS	-120.000	250.000	55
STRESS	-120.000	250.000	56
STRESS	-120.000	250.000	57
STRESS	-120.000	250.000	58
STRESS	-120.000	250.000	59
STRESS	-120.000	250.000	60
STRESS	-120.000	250.000	61
STRESS	-120.000	250.000	62
STRESS	-120.000	250.000	63
STRESS	-120.000	250.000	64
STRESS	-120.000	250.000	65
STRESS	-120.000	250.000	66
STRESS	-120.000	250.000	67
STRESS	-120.000	250.000	68
STRESS	-120.000	250.000	69
STRESS	-120.000	250.000	70
STRESS	-120.000	250.000	71
STRESS	-120.000	250.000	72
STRESS	-120.000	250.000	73
STRESS	-120.000	250.000	74
STRESS	-120.000	250.000	75
STRESS	-120.000	250.000	76
STRESS	-120.000	250.000	77
STRESS	-120.000	250.000	78
STRESS	-120.000	250.000	79
STRESS	-120.000	250.000	80
STRESS	-120.000	250.000	81

(continued)

Table 6-3

STRESS	-120.000	250.000	82
STRESS	-120.000	250.000	83
STRESS	-120.000	250.000	84
STRESS	-120.000	250.000	85
STRESS	-120.000	250.000	86
STRESS	-120.000	250.000	87
STRESS	-120.000	250.000	88
STRESS	-120.000	250.000	89
STRESS	-120.000	250.000	90
STRESS	-120.000	250.000	91
STRESS	-120.000	250.000	92
STRESS	-120.000	250.000	93
STRESS	-120.000	250.000	94
STRESS	-120.000	250.000	95
STRESS	-120.000	250.000	96
STRESS	-120.000	250.000	97
STRESS	-120.000	250.000	98
STRESS	-120.000	250.000	99
STRESS	-120.000	250.000	100
STRESS	-120.000	250.000	101
STRESS	-120.000	250.000	102
STRESS	-120.000	250.000	103
STRESS	-120.000	250.000	104
STRESS	-120.000	250.000	105
STRESS	-120.000	250.000	106
STRESS	-120.000	250.000	107
STRESS	-120.000	250.000	108
STRESS	-120.000	250.000	109
STRESS	-120.000	250.000	110
STRESS	-120.000	250.000	111
STRESS	-120.000	250.000	112
STRESS	-120.000	250.000	113
STRESS	-120.000	250.000	114
STRESS	-120.000	250.000	115
STRESS	-120.000	250.000	116
STRESS	-120.000	250.000	117
STRESS	-120.000	250.000	118
STRESS	-120.000	250.000	119
STRESS	-120.000	250.000	120
STRESS	-120.000	250.000	121
STRESS	-120.000	250.000	122
STRESS	-120.000	250.000	123
STRESS	-120.000	250.000	124
STRESS	-120.000	250.000	125
STRESS	-120.000	250.000	126

(continued)

Table 6-3

STRESS	-120.000	250.000	127
STRESS	-120.000	250.000	128
STRESS	-120.000	250.000	129
STRESS	-120.000	250.000	130
STRESS	-120.000	250.000	131
STRESS	-120.000	250.000	132
STRESS	-120.000	250.000	133
STRESS	-120.000	250.000	134
STRESS	-120.000	250.000	135
STRESS	-120.000	250.000	136
STRESS	-120.000	250.000	137
STRESS	-120.000	250.000	138
STRESS	-120.000	250.000	139
STRESS	-120.000	250.000	140
STRESS	-120.000	250.000	141
STRESS	-120.000	250.000	142
STRESS	-120.000	250.000	143
STRESS	-120.000	250.000	144
DISP-Y	-25.0000	25.0000	81
DISP-Y	-25.0000	25.0000	82
DISP-Y	-25.0000	25.0000	83
DISP-Y	-25.0000	25.0000	84
DISP-Y	-25.0000	25.0000	85
ENDATA			

(end of table)

Table 6-4: The FE data of the Dynamic Case Example.

```
==========================================================================
TITLE = WING DYNAMIC-CASE FREQUENCY CONSTRAINT
BEGIN BULK
GRID            1              0.     300.       0.          123456
GRID            2              0.     270.     600.          123456
GRID            3              0. 221.052    -600.          123456
GRID            4              0. 142.104   -1200.          123456
GRID            5              0.     140.    1200.          123456
GRID            6              0.-142.105   -1200.          123456
GRID            7              0.    -200.    1200.          123456
GRID            8              0.-221.052    -600.          123456
GRID            9              0.    -300.     600.          123456
GRID           10              0.    -300.       0.          123456
GRID           11            450.     275.       0.             456
GRID           12            450.    247.5     550.             456
GRID           13            450. 202.631    -550.             456
GRID           14            450. 130.813    1100.             456
GRID           15            450. 130.262   -1100.             456
GRID           16            450.-130.263   -1100.             456
GRID           17            450.-186.875    1100.             456
GRID           18            450.-202.632    -550.             456
GRID           19            450.    -275.     550.             456
GRID           20            450.    -275.       0.             456
GRID           21        900.0001     250.       0.             456
GRID           22        900.0001     225.     500.             456
GRID           23        900.0001   184.21    -500.             456
GRID           24        900.0001 121.625    1000.             456
GRID           25        900.0001 118.421   -1000.             456
GRID           26        900.0001-118.421   -1000.             456
GRID           27        900.0001 -173.75    1000.             456
GRID           28        900.0001-184.211    -500.             456
GRID           29        900.0001    -250.     500.             456
GRID           30        900.0001    -250.       0.             456
GRID           31            1350.     225.       0.             456
GRID           32            1350.    202.5     450.             456
GRID           33            1350. 165.789    -450.             456
GRID           34            1350. 112.438900.0001             456
GRID           35            1350. 106.579-900.000             456
GRID           36            1350.-106.579-900.000             456
GRID           37            1350.-160.625900.0001             456
GRID           38            1350. -165.79    -450.             456
GRID           39            1350.    -225.     450.             456
GRID           40            1350.    -225.       0.             456
GRID           41            1800.     200.       0.             456
GRID           42            1800.     180.     400.             456
--------------------------------------------------------------------------
```

(continued)

Table 6-4

```
GRID          43          1800. 147.368   -400.                    456
GRID          44          1800.  103.25800.0001                    456
GRID          45          1800.94.73701-800.000                    456
GRID          46          1800.-94.7375-800.000                    456
GRID          47          1800.-147.369   -400.                    456
GRID          48          1800.  -147.5800.0001                    456
GRID          49          1800.    -200.    400.                    456
GRID          50          1800.    -200.      0.                    456
GRID          51          2250.     175.      0.                    456
GRID          52          2250.    157.5    350.                    456
GRID          53          2250. 128.948   -350.                    456
GRID          54          2250.94.06251700.0001                    456
GRID          55          2250.82.89521-700.000                    456
GRID          56          2250.-82.8956-700.000                    456
GRID          57          2250.-128.948   -350.                    456
GRID          58          2250.-134.375700.0001                    456
GRID          59          2250.   -175.    350.                    456
GRID          60          2250.   -175.      0.                    456
GRID          61          2700.    150.      0.                    456
GRID          62          2700.    135.    300.                    456
GRID          63          2700. 110.527   -300.                    456
GRID          64          2700.84.87501    600.                    456
GRID          65          2700. 71.0535   -600.                    456
GRID          66          2700.-71.0537   -600.                    456
GRID          67          2700.-110.527   -300.                    456
GRID          68          2700. -121.25    600.                    456
GRID          69          2700.   -150.    300.                    456
GRID          70          2700.   -150.      0.                    456
GRID          71          3150.    125.      0.                    456
GRID          72          3150.   112.5    250.                    456
GRID          73          3150.92.10591   -250.                    456
GRID          74          3150.75.68751    500.                    456
GRID          75          3150.59.21181   -500.                    456
GRID          76          3150.-59.2119   -500.                    456
GRID          77          3150.-92.1059   -250.                    456
GRID          78          3150.-108.125    500.                    456
GRID          79          3150.   -125.    250.                    456
GRID          80          3150.   -125.      0.                    456
GRID          81          3600.    100.      0.                    456
GRID          82          3600.90.00001    200.                    456
GRID          83          3600.73.68501   -200.                    456
GRID          84          3600.66.50001    400.                    456
GRID          85          3600.47.37001   -400.                    456
```

(continued)

Table 6-4

```
---------------------------------------------------------------------------
GRID           86              3600.-47.3700   -400.                   456
GRID           87              3600.-73.6850   -200.                   456
GRID           88              3600.-95.0000    400.                   456
GRID           89              3600.   -100.    200.                   456
GRID           90              3600.   -100.      0.                   456
MAT1            1  79000.                0.33  2.80-6
PSHELL         33        1        1.
PSHEAR         65        1        1.
CQUAD4         33       33        5       14       12        2       0.
CQUAD4         34       33        2       12       11        1       0.
CQUAD4         35       33       14       24       22       12       0.
CQUAD4         36       33       12       22       21       11       0.
CQUAD4         37       33       24       34       32       22       0.
CQUAD4         38       33       22       32       31       21       0.
CQUAD4         39       33       34       44       42       32       0.
CQUAD4         40       33       32       42       41       31       0.
CQUAD4         41       33       44       54       52       42       0.
CQUAD4         42       33       42       52       51       41       0.
CQUAD4         43       33       54       64       62       52       0.
CQUAD4         44       33       52       62       61       51       0.
CQUAD4         45       33       64       74       72       62       0.
CQUAD4         46       33       62       72       71       61       0.
CQUAD4         47       33       74       84       82       72       0.
CQUAD4         48       33       72       82       81       71       0.
CQUAD4         49       33        1       11       13        3       0.
CQUAD4         50       33        3       13       15        4       0.
CQUAD4         51       33       11       21       23       13       0.
CQUAD4         52       33       13       23       25       15       0.
CQUAD4         53       33       21       31       33       23       0.
CQUAD4         54       33       23       33       35       25       0.
CQUAD4         55       33       31       41       43       33       0.
CQUAD4         56       33       33       43       45       35       0.
CQUAD4         57       33       41       51       53       43       0.
CQUAD4         58       33       43       53       55       45       0.
CQUAD4         59       33       51       61       63       53       0.
CQUAD4         60       33       53       63       65       55       0.
CQUAD4         61       33       61       71       73       63       0.
CQUAD4         62       33       63       73       75       65       0.
CQUAD4         63       33       71       81       83       73       0.
CQUAD4         64       33       73       83       85       75       0.
CQUAD4         81       33        6       16       18        8       0.
CQUAD4         82       33        8       18       20       10       0.
CQUAD4         83       33       16       26       28       18       0.
CQUAD4         84       33       18       28       30       20       0.
CQUAD4         85       33       26       36       38       28       0.
CQUAD4         86       33       28       38       40       30       0.
CQUAD4         87       33       36       46       47       38       0.
---------------------------------------------------------------------------
```

(continued)

Table 6-4

CQUAD4	88	33	38	47	50	40	0.
CQUAD4	89	33	46	56	57	47	0.
CQUAD4	90	33	47	57	60	50	0.
CQUAD4	91	33	56	66	67	57	0.
CQUAD4	92	33	57	67	70	60	0.
CQUAD4	93	33	66	76	77	67	0.
CQUAD4	94	33	67	77	80	70	0.
CQUAD4	95	33	76	86	87	77	0.
CQUAD4	96	33	77	87	90	80	0.
CQUAD4	97	33	10	20	19	9	0.
CQUAD4	98	33	9	19	17	7	0.
CQUAD4	99	33	20	30	29	19	0.
CQUAD4	100	33	19	29	27	17	0.
CQUAD4	101	33	29	39	37	27	0.
CQUAD4	102	33	30	40	39	29	0.
CQUAD4	103	33	40	50	49	39	0.
CQUAD4	104	33	39	49	48	37	0.
CQUAD4	105	33	50	60	59	49	0.
CQUAD4	106	33	49	59	58	48	0.
CQUAD4	107	33	60	70	69	59	0.
CQUAD4	108	33	59	69	68	58	0.
CQUAD4	109	33	70	80	79	69	0.
CQUAD4	110	33	69	79	78	68	0.
CQUAD4	111	33	80	90	89	79	0.
CQUAD4	112	33	79	89	88	78	0.
CQUAD4	113	33	17	14	12	19	0.
CQUAD4	114	33	19	12	11	20	0.
CQUAD4	115	33	20	11	13	18	0.
CQUAD4	116	33	18	13	15	16	0.
CQUAD4	117	33	27	24	22	29	0.
CQUAD4	118	33	29	22	21	30	0.
CQUAD4	119	33	30	21	23	28	0.
CQUAD4	120	33	28	23	25	26	0.
CQUAD4	121	33	37	34	32	39	0.
CQUAD4	122	33	39	32	31	40	0.
CQUAD4	123	33	40	31	33	38	0.
CQUAD4	124	33	38	33	35	36	0.
CQUAD4	125	33	48	44	42	49	0.
CQUAD4	126	33	49	42	41	50	0.
CQUAD4	127	33	50	41	43	47	0.
CQUAD4	128	33	47	43	45	46	0.
CQUAD4	129	33	58	54	52	59	0.
CQUAD4	130	33	59	52	51	60	0.

(continued)

Table 6-4

```
--------------------------------------------------------------------------------
CQUAD4          131          33          60          51          53          57          0.
CQUAD4          132          33          57          53          55          56          0.
CQUAD4          133          33          68          64          62          69          0.
CQUAD4          134          33          69          62          61          70          0.
CQUAD4          135          33          70          61          63          67          0.
CQUAD4          136          33          67          63          65          66          0.
CQUAD4          137          33          78          74          72          79          0.
CQUAD4          138          33          79          72          71          80          0.
CQUAD4          139          33          80          71          73          77          0.
CQUAD4          140          33          77          73          75          76          0.
CQUAD4          141          33          88          84          82          89          0.
CQUAD4          142          33          89          82          81          90          0.
CQUAD4          143          33          90          81          83          87          0.
CQUAD4          144          33          87          83          85          86          0.
CSHEAR           65          65           5          14          17           7
CSHEAR           66          65          14          24          27          17
CSHEAR           67          65          24          34          37          27
CSHEAR           68          65          34          44          48          37
CSHEAR           69          65          44          54          58          48
CSHEAR           70          65          54          64          68          58
CSHEAR           71          65          64          74          78          68
CSHEAR           72          65          74          84          88          78
CSHEAR           73          65           4          15          16           6
CSHEAR           74          65          15          25          26          16
CSHEAR           75          65          25          35          36          26
CSHEAR           76          65          35          45          46          36
CSHEAR           77          65          45          55          56          46
CSHEAR           78          65          55          65          66          56
CSHEAR           79          65          65          75          76          66
CSHEAR           80          65          75          85          86          76
ENDDATA
================================================================================
```

(end of table)

Table 6-5: The command data file of the Dynamic Case Example.

```
==========================================================
START
COMMAND DATA PNEWTON
C
      SET NSIV=1;
      SET MASS=STR;
      SET MODE=1;
      SET MXIT=10;
      SET APRT=YES;
      SET ACTG=YES;
      SET IFLM=YES;
      DO NSIN;
      DO OPTIN;
C
      DO ANALYSIS;
      DO INSPECT;
      DO OPTX;
C     PSEUDO NEWTON ITERATIONS
L010: DO ITER;
      DO QNEWTON;
      DO OPTX;
      IF(ENDC.EQ.YES) GOTO L031;
      IF(NOIT.EQ.3) DO HISTORY;
      IF(NOIT.EQ.3) SUSPEND;
      IF(ENDI.EQ.YES) GOTO L040;
      GOTO L010;
L031: DISPLAY CONVERGENCE TO THE LOWER BOUND WEIGHT;
      GOTO L050;
L032: DISPLAY CONVERGENCE TO DESIGN VARIABLES;
      GOTO L050;
L033: DISPLAY CONVERGENCE TO FEASIBLE VS ACTUAL WEIGHT;
      GOTO L050;
L034: DISPLAY CONVERGENCE TO FEASIBLE WEIGHT;
      GOTO L050;
L040: DISPLAY MAX. NUMBER OF ITERATIONS REACHED;
C
L050: SET APRT=YES;
      DO ANALYSIS;
      DO INSPECT;
      DO HIST;
      STOP;
      END;
C     ANALYSIS PROCEDURE;
      PROC OPTX;
      DO ANALYSIS;
      DO ACTSET;
      DO DERV;
      DO CONV;
      DO INSPECT;
      END OPTX;
ENDATA
==========================================================
```

Table 6-6: The design/constraint file of the Dynamic Case Example.

```
================================================================================
C234567890123456789012345678901234567890123456789012345678901234567890
DESIGN              1         33
DESIGN              1         34
DESIGN              1         49
DESIGN              1         50
DESIGN              1         81
DESIGN              1         82
DESIGN              1         97
DESIGN              1         98
DESIGN              2         35
DESIGN              2         36
DESIGN              2         51
DESIGN              2         52
DESIGN              2         83
DESIGN              2         84
DESIGN              2         99
DESIGN              2         100
DESIGN              3         37
DESIGN              3         38
DESIGN              3         39
DESIGN              3         40
DESIGN              3         41
DESIGN              3         42
DESIGN              3         43
DESIGN              3         44
DESIGN              3         45
DESIGN              3         46
DESIGN              3         47
DESIGN              3         48
DESIGN              3         53
DESIGN              3         54
DESIGN              3         55
DESIGN              3         56
DESIGN              3         57
DESIGN              3         58
DESIGN              3         59
DESIGN              3         60
DESIGN              3         61
DESIGN              3         62
DESIGN              3         63
DESIGN              3         64
DESIGN              4         85
DESIGN              4         86
DESIGN              4         87
--------------------------------------------------------------------------------
```

(continued)

Table 6-6

DESIGN	4	88
DESIGN	4	89
DESIGN	4	90
DESIGN	4	91
DESIGN	4	92
DESIGN	4	93
DESIGN	4	94
DESIGN	4	95
DESIGN	4	96
DESIGN	4	101
DESIGN	4	102
DESIGN	4	103
DESIGN	4	104
DESIGN	4	105
DESIGN	4	106
DESIGN	4	107
DESIGN	4	108
DESIGN	4	109
DESIGN	4	110
DESIGN	4	111
DESIGN	4	112
DESIGN	5	65
DESIGN	5	66
DESIGN	5	67
DESIGN	5	68
DESIGN	5	69
DESIGN	5	70
DESIGN	5	71
DESIGN	5	72
DESIGN	5	73
DESIGN	5	74
DESIGN	5	75
DESIGN	5	76
DESIGN	5	77
DESIGN	5	78
DESIGN	5	79
DESIGN	5	80
DESIGN	6	113
DESIGN	6	114
DESIGN	6	115
DESIGN	6	116
DESIGN	6	117
DESIGN	6	118
DESIGN	6	119
DESIGN	6	120
DESIGN	6	121

(continued)

Table 6-6

DESIGN	6	122	
DESIGN	6	123	
DESIGN	6	124	
DESIGN	6	125	
DESIGN	6	126	
DESIGN	6	127	
DESIGN	6	128	
DESIGN	6	129	
DESIGN	6	130	
DESIGN	6	131	
DESIGN	6	132	
DESIGN	6	133	
DESIGN	6	134	
DESIGN	6	135	
DESIGN	6	136	
DESIGN	6	137	
DESIGN	6	138	
DESIGN	6	139	
DESIGN	6	140	
DESIGN	6	141	
DESIGN	6	142	
DESIGN	6	143	
DESIGN	6	144	
GAUGE	0.600000	10.0000	33
GAUGE	0.600000	10.0000	34
GAUGE	0.600000	10.0000	35
GAUGE	0.600000	10.0000	36
GAUGE	0.600000	10.0000	37
GAUGE	0.600000	10.0000	38
GAUGE	0.600000	10.0000	39
GAUGE	0.600000	10.0000	40
GAUGE	0.600000	10.0000	41
GAUGE	0.600000	10.0000	42
GAUGE	0.600000	10.0000	43
GAUGE	0.600000	10.0000	44
GAUGE	0.600000	10.0000	45
GAUGE	0.600000	10.0000	46
GAUGE	0.600000	10.0000	47
GAUGE	0.600000	10.0000	48
GAUGE	0.600000	10.0000	49
GAUGE	0.600000	10.0000	50
GAUGE	0.600000	10.0000	51
GAUGE	0.600000	10.0000	52
GAUGE	0.600000	10.0000	53
GAUGE	0.600000	10.0000	54

(continued)

Table 6-6

GAUGE	0.600000	10.0000	55
GAUGE	0.600000	10.0000	56
GAUGE	0.600000	10.0000	57
GAUGE	0.600000	10.0000	58
GAUGE	0.600000	10.0000	59
GAUGE	0.600000	10.0000	60
GAUGE	0.600000	10.0000	61
GAUGE	0.600000	10.0000	62
GAUGE	0.600000	10.0000	63
GAUGE	0.600000	10.0000	64
GAUGE	0.600000	10.0000	81
GAUGE	0.600000	10.0000	82
GAUGE	0.600000	10.0000	83
GAUGE	0.600000	10.0000	84
GAUGE	0.600000	10.0000	85
GAUGE	0.600000	10.0000	86
GAUGE	0.600000	10.0000	87
GAUGE	0.600000	10.0000	88
GAUGE	0.600000	10.0000	89
GAUGE	0.600000	10.0000	90
GAUGE	0.600000	10.0000	91
GAUGE	0.600000	10.0000	92
GAUGE	0.600000	10.0000	93
GAUGE	0.600000	10.0000	94
GAUGE	0.600000	10.0000	95
GAUGE	0.600000	10.0000	96
GAUGE	0.600000	10.0000	97
GAUGE	0.600000	10.0000	98
GAUGE	0.600000	10.0000	99
GAUGE	0.600000	10.0000	100
GAUGE	0.600000	10.0000	101
GAUGE	0.600000	10.0000	102
GAUGE	0.600000	10.0000	103
GAUGE	0.600000	10.0000	104
GAUGE	0.600000	10.0000	105
GAUGE	0.600000	10.0000	106
GAUGE	0.600000	10.0000	107
GAUGE	0.600000	10.0000	108
GAUGE	0.600000	10.0000	109
GAUGE	0.600000	10.0000	110
GAUGE	0.600000	10.0000	111
GAUGE	0.600000	10.0000	112
GAUGE	0.600000	3.50000	65
GAUGE	0.600000	3.50000	66
GAUGE	0.600000	3.50000	67

(continued)

Table 6-6

```
-------------------------------------------------------------------
GAUGE          0.600000   3.50000          68
GAUGE          0.600000   3.50000          69
GAUGE          0.600000   3.50000          70
GAUGE          0.600000   3.50000          71
GAUGE          0.600000   3.50000          72
GAUGE          0.600000   3.50000          73
GAUGE          0.600000   3.50000          74
GAUGE          0.600000   3.50000          75
GAUGE          0.600000   3.50000          76
GAUGE          0.600000   3.50000          77
GAUGE          0.600000   3.50000          78
GAUGE          0.600000   3.50000          79
GAUGE          0.600000   3.50000          80
GAUGE          0.600000   2.50000         113
GAUGE          0.600000   2.50000         114
GAUGE          0.600000   2.50000         115
GAUGE          0.600000   2.50000         116
GAUGE          0.600000   2.50000         117
GAUGE          0.600000   2.50000         118
GAUGE          0.600000   2.50000         119
GAUGE          0.600000   2.50000         120
GAUGE          0.600000   2.50000         121
GAUGE          0.600000   2.50000         122
GAUGE          0.600000   2.50000         123
GAUGE          0.600000   2.50000         124
GAUGE          0.600000   2.50000         125
GAUGE          0.600000   2.50000         126
GAUGE          0.600000   2.50000         127
GAUGE          0.600000   2.50000         128
GAUGE          0.600000   2.50000         129
GAUGE          0.600000   2.50000         130
GAUGE          0.600000   2.50000         131
GAUGE          0.600000   2.50000         132
GAUGE          0.600000   2.50000         133
GAUGE          0.600000   2.50000         134
GAUGE          0.600000   2.50000         135
GAUGE          0.600000   2.50000         136
GAUGE          0.600000   2.50000         137
GAUGE          0.600000   2.50000         138
GAUGE          0.600000   2.50000         139
GAUGE          0.600000   2.50000         140
GAUGE          0.600000   2.50000         141
GAUGE          0.600000   2.50000         142
GAUGE          0.600000   2.50000         143
GAUGE          0.600000   2.50000         144
NFREQ                 1      2.50
ACTNFREQ              1
ENDATA
===================================================================
```

(end of table)

Table 6-7: The modified design/constraint file of the Dynamic Case Example.

```
================================================================================
C234567890123456789012345678901234567890123456789012345678901234567890
DESIGN                1          33
DESIGN                1          34
DESIGN                1          49
DESIGN                1          50
DESIGN                1          81
DESIGN                1          82
DESIGN                1          97
DESIGN                1          98
DESIGN                2          35
DESIGN                2          36
DESIGN                2          51
DESIGN                2          52
DESIGN                2          83
DESIGN                2          84
DESIGN                2          99
DESIGN                2          100
DESIGN                3          37
DESIGN                3          38
DESIGN                3          53
DESIGN                3          54
DESIGN                3          85
DESIGN                3          86
DESIGN                3          101
DESIGN                3          102
DESIGN                4          39
DESIGN                4          40
DESIGN                4          55
DESIGN                4          56
DESIGN                4          87
DESIGN                4          88
DESIGN                4          103
DESIGN                4          104
DESIGN                5          41
DESIGN                5          42
DESIGN                5          57
DESIGN                5          58
DESIGN                5          89
DESIGN                5          90
DESIGN                5          105
DESIGN                5          106
DESIGN                6          43
DESIGN                6          44
DESIGN                6          59
DESIGN                6          60
--------------------------------------------------------------------------------
```

(continued)

Table 6-7

DESIGN	6	91
DESIGN	6	92
DESIGN	6	107
DESIGN	6	108
DESIGN	7	45
DESIGN	7	46
DESIGN	7	61
DESIGN	7	62
DESIGN	7	93
DESIGN	7	94
DESIGN	7	109
DESIGN	7	110
DESIGN	8	47
DESIGN	8	48
DESIGN	8	63
DESIGN	8	64
DESIGN	8	95
DESIGN	8	96
DESIGN	8	111
DESIGN	8	112
DESIGN	9	113
DESIGN	9	114
DESIGN	9	115
DESIGN	9	116
DESIGN	9	117
DESIGN	9	118
DESIGN	9	119
DESIGN	9	120
DESIGN	9	121
DESIGN	9	122
DESIGN	9	123
DESIGN	9	124
DESIGN	9	125
DESIGN	9	126
DESIGN	9	127
DESIGN	9	128
DESIGN	9	129
DESIGN	9	130
DESIGN	9	131
DESIGN	9	132
DESIGN	9	133
DESIGN	9	134
DESIGN	9	135
DESIGN	9	136
DESIGN	9	137

(continued)

Table 6-7

DESIGN	9	138	
DESIGN	9	139	
DESIGN	9	140	
DESIGN	9	141	
DESIGN	9	142	
DESIGN	9	143	
DESIGN	9	144	
DESIGN	10	65	
DESIGN	10	73	
DESIGN	11	66	
DESIGN	11	74	
DESIGN	12	67	
DESIGN	12	75	
DESIGN	13	68	
DESIGN	13	76	
DESIGN	14	69	
DESIGN	14	77	
DESIGN	15	70	
DESIGN	15	78	
DESIGN	16	71	
DESIGN	16	79	
DESIGN	17	72	
DESIGN	17	80	
GAUGE	0.600000	10.0000	33
GAUGE	0.600000	10.0000	34
GAUGE	0.600000	10.0000	35
GAUGE	0.600000	10.0000	36
GAUGE	0.600000	10.0000	37
GAUGE	0.600000	10.0000	38
GAUGE	0.600000	10.0000	39
GAUGE	0.600000	10.0000	40
GAUGE	0.600000	10.0000	41
GAUGE	0.600000	10.0000	42
GAUGE	0.600000	10.0000	43
GAUGE	0.600000	10.0000	44
GAUGE	0.600000	10.0000	45
GAUGE	0.600000	10.0000	46
GAUGE	0.600000	10.0000	47
GAUGE	0.600000	10.0000	48
GAUGE	0.600000	10.0000	49
GAUGE	0.600000	10.0000	50
GAUGE	0.600000	10.0000	51
GAUGE	0.600000	10.0000	52
GAUGE	0.600000	10.0000	53

(continued)

Table 6-7

GAUGE	0.600000	10.0000	54
GAUGE	0.600000	10.0000	55
GAUGE	0.600000	10.0000	56
GAUGE	0.600000	10.0000	57
GAUGE	0.600000	10.0000	58
GAUGE	0.600000	10.0000	59
GAUGE	0.600000	10.0000	60
GAUGE	0.600000	10.0000	61
GAUGE	0.600000	10.0000	62
GAUGE	0.600000	10.0000	63
GAUGE	0.600000	10.0000	64
GAUGE	0.600000	10.0000	81
GAUGE	0.600000	10.0000	82
GAUGE	0.600000	10.0000	83
GAUGE	0.600000	10.0000	84
GAUGE	0.600000	10.0000	85
GAUGE	0.600000	10.0000	86
GAUGE	0.600000	10.0000	87
GAUGE	0.600000	10.0000	88
GAUGE	0.600000	10.0000	89
GAUGE	0.600000	10.0000	90
GAUGE	0.600000	10.0000	91
GAUGE	0.600000	10.0000	92
GAUGE	0.600000	10.0000	93
GAUGE	0.600000	10.0000	94
GAUGE	0.600000	10.0000	95
GAUGE	0.600000	10.0000	96
GAUGE	0.600000	10.0000	97
GAUGE	0.600000	10.0000	98
GAUGE	0.600000	10.0000	99
GAUGE	0.600000	10.0000	100
GAUGE	0.600000	10.0000	101
GAUGE	0.600000	10.0000	102
GAUGE	0.600000	10.0000	103
GAUGE	0.600000	10.0000	104
GAUGE	0.600000	10.0000	105
GAUGE	0.600000	10.0000	106
GAUGE	0.600000	10.0000	107
GAUGE	0.600000	10.0000	108
GAUGE	0.600000	10.0000	109
GAUGE	0.600000	10.0000	110
GAUGE	0.600000	10.0000	111
GAUGE	0.600000	10.0000	112
GAUGE	0.600000	3.50000	65
GAUGE	0.600000	3.50000	66
GAUGE	0.600000	3.50000	67

(continued)

Table 6-7

```
-------------------------------------------------------------------
GAUGE          0.600000   3.50000        68
GAUGE          0.600000   3.50000        69
GAUGE          0.600000   3.50000        70
GAUGE          0.600000   3.50000        71
GAUGE          0.600000   3.50000        72
GAUGE          0.600000   3.50000        73
GAUGE          0.600000   3.50000        74
GAUGE          0.600000   3.50000        75
GAUGE          0.600000   3.50000        76
GAUGE          0.600000   3.50000        77
GAUGE          0.600000   3.50000        78
GAUGE          0.600000   3.50000        79
GAUGE          0.600000   3.50000        80
GAUGE          0.600000   2.50000       113
GAUGE          0.600000   2.50000       114
GAUGE          0.600000   2.50000       115
GAUGE          0.600000   2.50000       116
GAUGE          0.600000   2.50000       117
GAUGE          0.600000   2.50000       118
GAUGE          0.600000   2.50000       119
GAUGE          0.600000   2.50000       120
GAUGE          0.600000   2.50000       121
GAUGE          0.600000   2.50000       122
GAUGE          0.600000   2.50000       123
GAUGE          0.600000   2.50000       124
GAUGE          0.600000   2.50000       125
GAUGE          0.600000   2.50000       126
GAUGE          0.600000   2.50000       127
GAUGE          0.600000   2.50000       128
GAUGE          0.600000   2.50000       129
GAUGE          0.600000   2.50000       130
GAUGE          0.600000   2.50000       131
GAUGE          0.600000   2.50000       132
GAUGE          0.600000   2.50000       133
GAUGE          0.600000   2.50000       134
GAUGE          0.600000   2.50000       135
GAUGE          0.600000   2.50000       136
GAUGE          0.600000   2.50000       137
GAUGE          0.600000   2.50000       138
GAUGE          0.600000   2.50000       139
GAUGE          0.600000   2.50000       140
GAUGE          0.600000   2.50000       141
GAUGE          0.600000   2.50000       142
GAUGE          0.600000   2.50000       143
GAUGE          0.600000   2.50000       144
NFREQ                 1   2.50000
ACTNFREQ              1
ENDATA
===================================================================
```

(end of table)

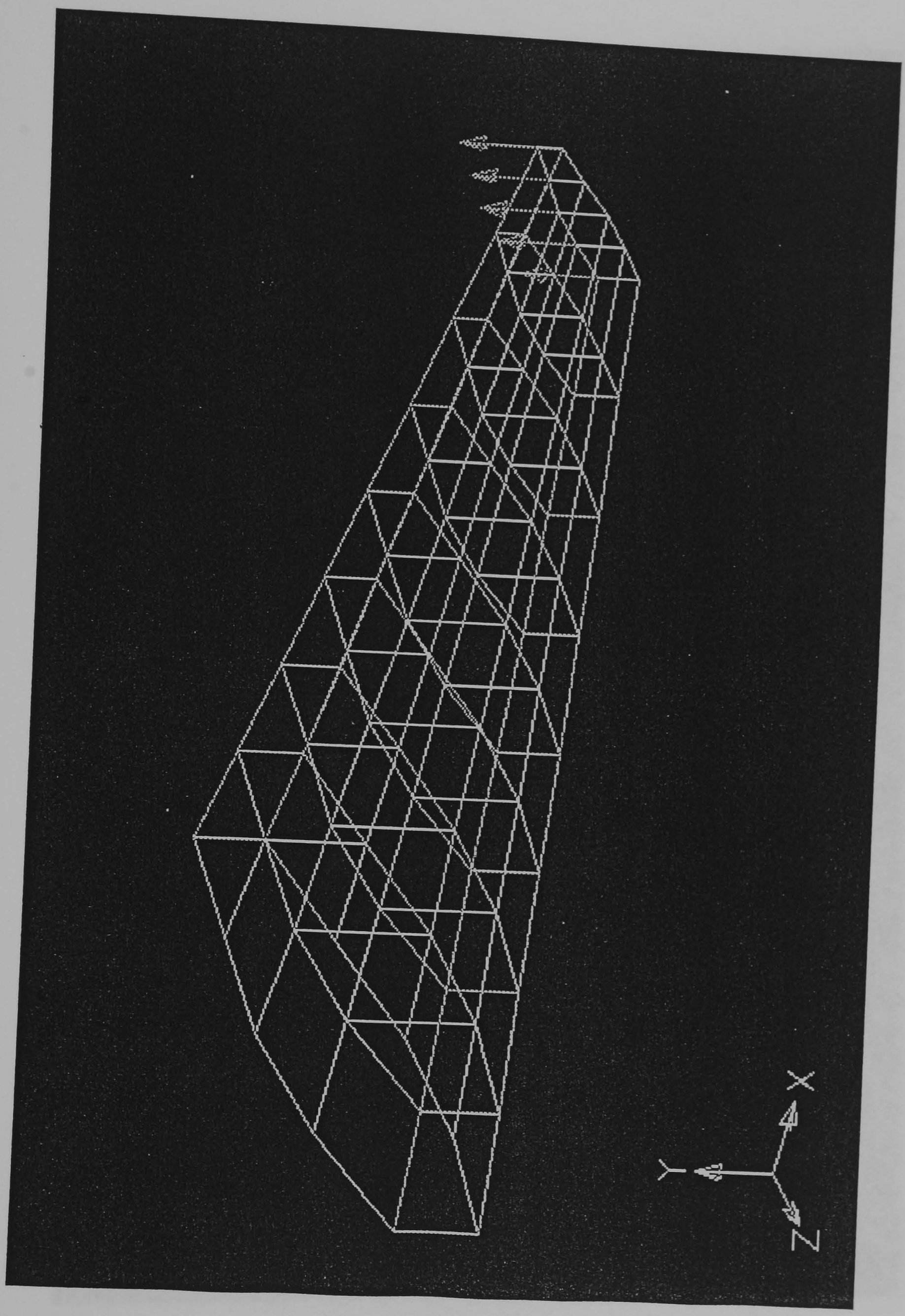


Figure 6-1 The FE model of the static case example.

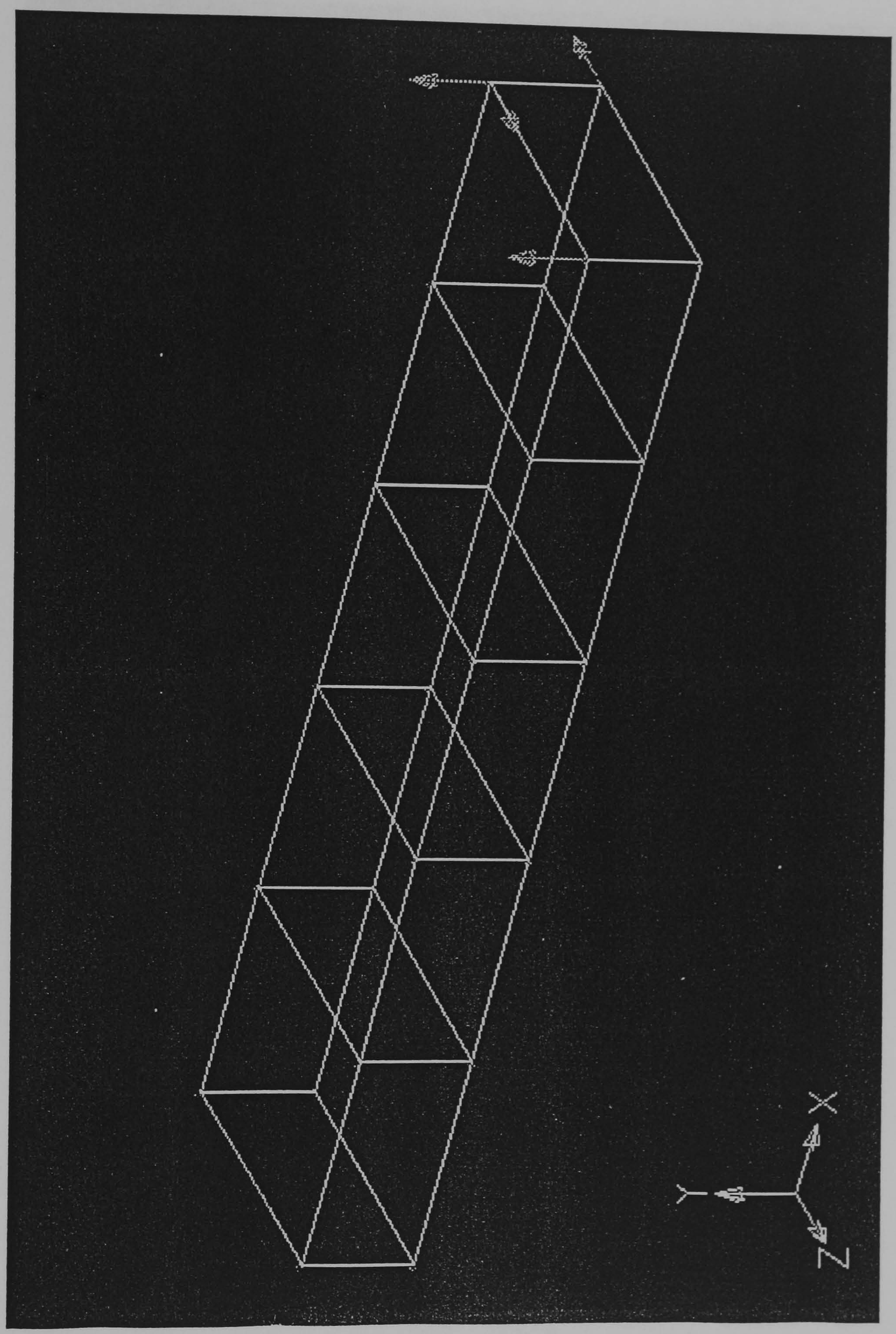


Figure 6-2 The FE model of the similar static past case.

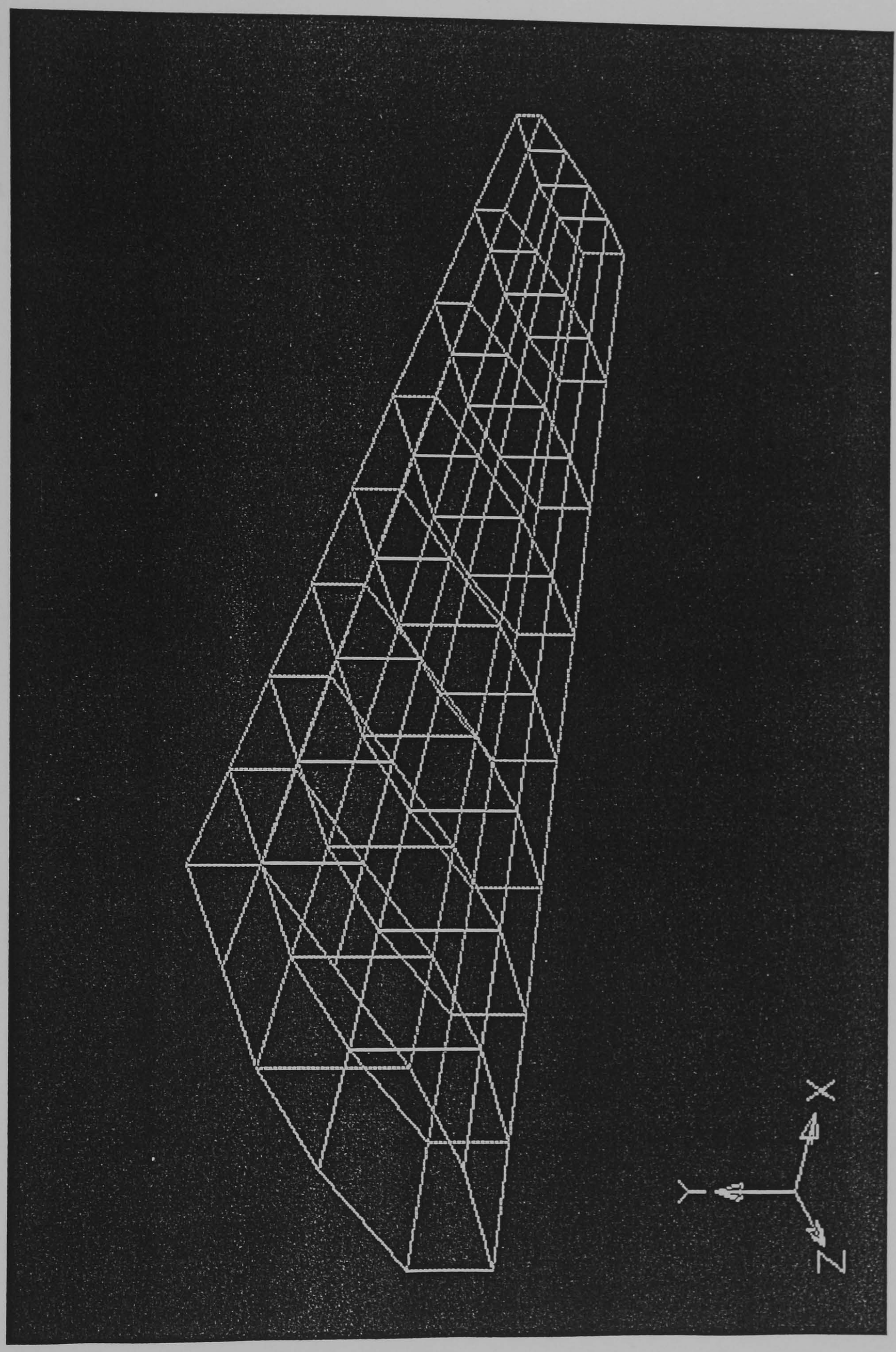


Figure 6-3 The FE model of the similar dynamic past case.

REFERENCES.

Heller, D. 1990.
XView Programming Manual. An OPEN LOOK Toolkit for X11.
O'Reilly & Associates, Inc. USA.

Kuntjoro, W. 1994.
Expert System for Structural Optimization Exploiting Past Experience and A-priori Knowledge.
PhD Thesis, College of Aeronautics, Cranfield University, UK.