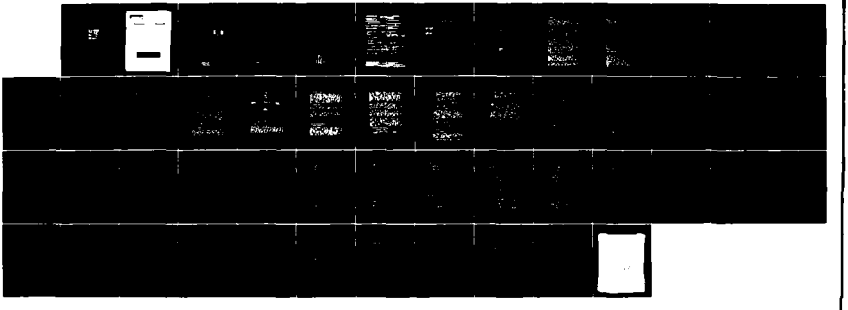
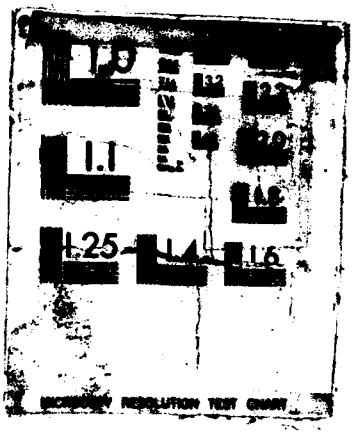
AD-A183 371 A FINITE ELEMENT EXPERT SYSTEM: AN INITIAL STUDY AND PROTOTYPE PROGRAM(U) CRANFIELD INST OF TECH (ENGLAND) AERODYNAMICS DIV A J MORRIS NOV 86 EOARD-TR-87-07 1/1

UNCLASSIFIED AFOSR-85-0306 F/G 12/9 NL




1.1
1.25
1.4
1.6
RESOLUTION TEST CHART

Professor A. J. Morris

November 1986

A FINITE ELEMENT EXPERT SYSTEM

An Initial Study and Prototype Program

EOARD-TR-87-02

(2)

Professor A. J. Morris

November 1986

A FINITE ELEMENT EXPERT SYSTEM

An Initial Study and Prototype Program

Contents

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

AD-A183 271

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: Unclassified	1b. RESTRICTIVE MARKINGS:
2a. SECURITY CLASSIFICATION AUTHORITY:	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE:	
4. PERFORMING ORGANIZATION REPORT NUMBER(S):	5. MONITORING ORGANIZATION REPORT NUMBER(S): EOARD-TR-87-03
6a. NAME OF PERFORMING ORGANIZATION: Cranfield Institute of Technology (6b. OFFICE SYMBOL (If applicable):)	7a. NAME OF MONITORING ORGANIZATION: European Office of Aerospace Research and Development
6c. ADDRESS (City, State, and ZIP Code): Cranfield, Bedford MK43 0AL UK	7b. ADDRESS (City, State, and ZIP Code): Box 14 FPO New York 09510-0200
8a. NAME OF FUNDING/SPONSORING ORGANIZATION: EOARD (8b. OFFICE SYMBOL (If applicable): LTS)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: AFOSR 85-0306
8c. ADDRESS (City, State, and ZIP Code): Box 14 FPO New York 09510-0200	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. 61102F; PROJECT NO. 2301; TASK NO. D1; WORK UNIT ACCESSION NO. 194

11. TITLE (Include Security Classification)
A Finite Element Expert System, An Initial Study and Prototype Program

12. PERSONAL AUTHOR(S)
Dr A J Morris

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Final Scientific	FROM 1-9-85 TO 31-8-86	November 1986	44

16. SUPPLEMENTARY NOTATION

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Artificial Intelligence; 0604 (TEST)
Computer Programming 0702
Structural Analysis; 1313 1402

19. ABSTRACT (Continue on reverse if necessary and identify by block number)
The report represents an initial study into the possibility of using Expert System concepts to aid users of finite element analysis systems. By concentrating on limited aspects of the problem a Lisp based prototype consult program has been constructed. This clearly demonstrates the advantages of adopting an Object Oriented approach in this area. The report suggests how this initial work may be further exploited to create a full scale Expert System. Keywords:

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT
☐ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS

21. ABSTRACT SECURITY CLASSIFICATION: Unclassified

22a. NAME OF RESPONSIBLE INDIVIDUAL: Lt Col. [illegible], USAF

22b. TELEPHONE (Include Area Code): (01) 409-4305

22c. OFFICE SYMBOL: LTS

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

EOARD-TR-87-0[illegible]

This report has been reviewed by the EOARD Information Office and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication.

[illegible], Lt Col, USAF
Chief, Structures and Structural Materials

[illegible], Lt Col, USAF
Chief Scientist

A FINITE ELEMENT EXPERT SYSTEM

An Initial Study and Prototype Program

A.J. MORRIS

College of Aeronautics,
Cranfield Institute of Technology,
CRANFIELD,
Bedford.
MK43 0AL

1. INTRODUCTION

The aim of the work presented in this report is directed at the creation of an Expert System which can act as a consultant to aid the finite element analyst. In principle the system must be capable of assisting the FE user to cover the full range of requirements from initial modelling, to result interpretation and error checking and should be capable of advising over the full range of problem types; statics, dynamics, composites, non-linear analysis, etc.

The incorporation of A.I. techniques into the engineering design process has progressed at an ever increasing pace in recent years. Conferences [1] have been devoted to this aspect of computer based design and at least one system [2] has been incorporated into the preliminary design process. Some of this expertise has been passed across to the finite element world with a view to providing consultancy programs for either assisting with the modelling process or generating a computer based 'users manual'. The main approach in this situation has involved the use of existing Expert System Shells either constructed from 'empted' programs such as EMYCIN [3] or specially constructed shells as SAVIOR [4].

This latter approach involving the use of existing shell like programs assumes that the knowledge and the computer systems designed to handle it can be completely separated. There is, however, evidence from human learning studies that the modelling of a domain contributes strongly to the problem solving ability and the flexibility of any resulting inferencing system. Johnson and Thompson [5] show that the consequences of inadequate knowledge representation in both human and computer problem solving gives rise to a performance which has the false appearance of success. Sloman [6] supports this concept and argues cogently against the advocates of particular formalisms for representing all kinds of knowledge. It can be demonstrated that first order predicate logic may have an important role in theorising about intelligent systems but is limited in scope and range. In the light of this type of argument it is suggested [6] that different formulations are useful for different purposes to cover the wide variety of types of expressional systems used by people in different fields such as mathematicians, scientists and engineers.

These arguments point away from the immediate use of a shell environment for any finite element consultant program. A better approach seems to lie in examining the nature of the problem and basing any system on the inherent structure of the knowledge. In this way any 'inference engine' can then be constructed to implement the casual sequences inherent in the skilled use of the rule base employed by an expert practitioner. This is the approach adopted herein.

The need for the application of A.I. methodologies to finite element analysis is growing in importance. First of all, because the FE systems are becoming complex, the non-finite element specialist finds them very difficult to use.

Secondly, because it is now widely recognised that the systems cannot be used as 'black boxes' and require a high level of experience if errors are to be avoided. As a result a general assessment engineer cannot examine the structural integrity of an item or component using the finite element method as a non-specialist without running the risk of making, potentially, catastrophic errors. A successful F.E. Expert System will return the FE method to the general engineer as a safe and reliable design assessment tool.

Because the finite eleme method has a vast range of applicability the starting point to the present work was fixed as the static analysis of linear structures. Having limited the scope of the proposed Experty System it was also found necessary to limit the range in order to build a prototype program. Prototyping has proved to be beneficial in other A.I. projects and was felt to be extremely important in F.E. analysis where there is a complex interaction between engineering judgement and firm mathematical knowledge. Thus a very limited prototype program has been constructed and is outlined below.

The main activity of this program is to select appropriate finite elements for specific structural configurations. It is written in LISP as initial studies indicated that the list compiling properties of the language could be used to advantage. The outcome of the work does, however, point strongly to some form of object oriented description language particularly with respect to procedual aspects. Within this framework the main influence and control processes could be efficiently exploited.

The remaining sections of this report described the overall problem at which the program is directed and the specific sub-program selected for prototyping. An outline description of the program architecture is presented though aspects related to usage are reported separately. Finally the report indicates future lines of progress which could be implemented with a view to generating a user friendly interface and expert advisor to major finite element systems.

2. OVERALL PROBLEM DEFINITION

2.1 Introduction

When the finite element method is considered and subsequently used for the analysis of a major structural item a range of problems confront the engineer. These start with overall aspects which relate to planning the job and continue through more technically specific matters such as meshing, element selection and conclude with the need to interprete the results with respect to the accuracy of stresses, displacements, frequencies, mode shapes etc. The questions associated with FE applications tend to be the same in nature for all types of application. Often the only change in moving from one application to another is the focus of the question. For example, similar questions must be answered if elements are being selected for static and dynamic analyses but the relative importance changes in the move from static to dynamic considerations. In attempting to construct an Expert System it

is, therefore, natural to define the problem in terms of the questions which must be answered by the analysis. In this section we look at the overall problem and in the next one we address the items specific to the prototype application.

2.2 Problem Description

2.2.1 Rationale

The problem of analysing a structure with the aid of an FE system is described below in terms of the main areas of interest together with an indication of some of the detail considerations which fall within these areas. As indicated earlier, the knowledge required for the resolution of these problem areas may be heuristic or mathematical in nature. Some attempt to classify the broad nature of the knowledge is given.

2.2.2 Classification (Outline)

1. Overall Considerations (Heuristic Knowledge)

   i) Planning requirements - staff, time-scales, etc.

   ii) Deciding problem type - static, dynamic, non-linear etc. or combination.

   iii) General suitability of FE for problem.

   iv) Solution requirements - hardware, software.

2. Modelling Problems (Heuristic/Mathematical Knowledge)

   i) Main structural features

   - shape (flat/curved), thickness of members, curved/straight boundaries, boundary constraints.

   ii) Materials

   - isotropic, anisotropic, composites (orthotropic).

   iii) Responses

   - linear, non-linear, dynamic, etc.

   iv) Loading

   - static, aerodynamic, transient, thermal, shock, etc.

v) Structural Inconsistencies

- joints, off-sets, discontinuities, stress raisers, contact problems, hinges, etc.

vi) Damping Type

vii) Structural/Loading Symmetry or Asymmetry

3. Element Selection (Heuristic/Mathematical Knowledge)

i) Structural regions

- regions with similar structural properties identified
  i.e. membranes, beams, shells, etc.

ii) Effect of material

- e.g. 'composite' elements.

iii) Nodal compatibility or M.P.C.'s required.

iv) Geometric properties accommodated

- curved edges, reinforcements, off-sets, etc.

v) Special problems

- e.g. shells.

4. Meshing Problems (Heuristic/Mathematical Knowledge)

i) Grid density requirements.

ii) Isoparametric shape restrictions.

5. Solution Problems (Mathematical Knowledge)

i) Potential numerical idealisation problems.

ii) Matrix inversion/solution requirements.

iii) Eigenvalue/Vector solvers.

iv) Non-linear routines.

v) Reduction methods.

vi) Sub-Structure considerations.

6. Results Interpretation
(Heuristic/Mathematical Knowledge)

i) Equilibrium checks

- nodal, overall.

ii) Consistency checks

- Stress jumps, displacement discontinuities, etc.

iii) The effects of structural inconsistencies.

It should be emphasised that this classification is not exhaustive and represents an outline of the type of knowledge and problems found in finite element applications. No attempt has been made to augment the task to include the application of pre- and post-processors, nor to the overall design problem which would require consideration being given to automated design methods.

3. CREATION OF FEES (finite element expert system)

3.1 Prototype Philosophy

Although it is tempting to use an established shell as the basis for a new finite element Expert System the arguments advanced in section 1 and the experience gained in the creation of FEASA [6] indicate that this may not be the best path. A better approach is to create an Expert System based on the structure of the finite element applications knowledge. However, the creation of an Expert System to handle the full finite element analysis problem outlined in Section 2 represents a formidable task. A more appropriate line of attack involves writing a prototype Expert System based on a limited sub-set of this knowledge. Specifically the items in 2.2.2 under Modelling Problem i, ii and Element Selection i-iv (inclusive) were included. In addition, it was decided that only statics problems should be considered at this, initial, stage.

Using such a limited sub-set of the overall FE knowledge base it was clear that no meta-level knowledge or control was required. But the augmentation of the program in the directions of generality would soon impose such a requirement. Thus a modular structure for the Expert System architecture was created. For convenience and portability this program was made specific to the Digital Equipment Corporations VAX range of computers.

No Expert System can be made independent of human input nor is it desirable that it should be independent. Computers are useful for certain operations and engineers are more effective in other, more creative, areas. Any Expert System should be able to participate in a dialogue with the user. In order to facilitate this man-machine interaction a range of FORTRAN routines have been created to allow the program to exploit the screen management facilities available under the VAX/VMS 4.2 operating facility. These routines also serve as the main control program for the Expert System as a whole.

In principle there is no need to limit the range of elements which can be included in FEES and, in consequence, there is no restraint on the system with respect to potential interacing FE programs. However, because special problems exist with certain complex elements, such as shells etc. it was decided for convenience to limit the current element library to simple elements. Because this is not a fundamental limitation the library can be augmented when deamed suitable.

3.2 FEES Knowledge/Inference Modules

3.2.2 Overview

When the finite element process is examined, it is found possible to describe the basic inference process as one of, essentially, accumulating constraints. To begin with the geometric properties of the structure starts a process of limiting the elements which can be used. The concept emmerging from this limiting process is that the structure to be analysed needs to be divided up into regions with similar geometric properties. If further limiting factors relating to the structure, such as inconsistencies, special features etc., are included it can be seen that there are a range of constraints which relate to the structural description. With these limits in place it is possible to evaluate appropriate elements for each region taking into account factors such as nodal conformability. Finally, with all constraints in place and the evaluation complete the process of selecting specific elements can be commenced.

Clearly, the human eye is more appropriate at deciding if a structure is curved or thin or has other important geometric properties. Thus the FEES program assumes that the user is able to supply this type of information and enters a dialogue in order to obtain it. But it does, currently, assume that the user may be unsure about the types of mechanical action the structure is performing. Thus, during the dialogue, the user is asked to place a level of certainty on such factors as membrane action, bending etc.

3.2.3 Module Overviews

1) Structural Descriptions Module

Purpose:

to describe the geometric shape of the structure and material properties on a region by region basis

Mode of Action:

i) Divides the structure into regions depending upon the basic spatial properties

ii) In each region defines structural properties (curvature, thickness, boundaries)

iii) Identifies special regions which relate to beams, nodes ets.

iv) Defines material properties in each region

v) For each region sets up an element identification 'vector' and region 'vector'.

These 'vectors' are predicate type descriptors encapsulating the properties of the entities under consideration thus:

REGION (( ), ( ), ( ), ( ))

ELEMENT (REGION, PROPERTY, MATERIAL, ...)

where

REGION: is the 'vector' describing the geometry of the structure and its boundaries

PROPERTY: is the 'vector' describing the action property of an element, i.e. membrane, rod, beam .....

MATERIAL: relates to the material constants and properties

Each of these defining terms may have a list of arguments like REGION, for example, MATERIAL, includes Poisson's ratio, elasticity constants etc.

2) Element Evaluation Module

Purpose:

to advise the system on the element properties required for each region such that the Selector Module can attempt to select appropriate elements.

Mode of Action:

i) Augments the ELEMENT list with further constraints on potential element types

ii) Examines each region to decide if special reasons exist which cut down the available choices, i.e. shape requirements (e.g. the incorporation of curved boundaries), grading requirements (e.g. local stress raisers requiring fine meshes), etc.

iii) Examines each region to see if adjacent regions impose constraints because elements have already been selected there (this may indicate use of M.P.C.'s)

iv) Special loading requirements may require specific elements or the structure may require special elements, e.g. presence of cracks.

The module then augments the ELEMENT list to incorporate these new constraints hence:

ELEMENT (REGION, PROPERTY, MATERIAL, NODAL CONSTRAINTS, SHAPE, ETC....)

where, as usual, the new augments may have a list of properties SIC;

SHAPE (BOUNDARY SHAPE, CURVATURE, GRADING, ETC.)

NODAL CONSTRAINTS
(REGION, ATTACHED ELEMENTS, GEOMETRIC CONSTRAINTS, ETC).

etc.....

Some of the new augments can take account of any special features which the analyst may feel are appropriate to element selection. For example, not using a 3-D membrane element in a Wing Box structure which is genuinely 3-D whilst the membrane element is really only 2-D

3) Element Selector Module

Purpose:

to use the accummulated information from the earlier modules to select elements for each region

Mode of Action

i) For each region unravels the ELEMENT 'vector' and attempts to match with the elements in the library.

4) Additional Modules

Although no additional modulus have currently been incorporated into FEES the above procedures could be extended to account for Meshing, Loads, etc.

3.2.4 Language

The structure of the modules outlined in 3.2.2 indicates a clear pattern to the way the knowledge is accummulated. If this is compared to the existing programming languages it is seen that both LISP and PROLOG resemble the above forms. The fact that one of the processes involves adding facts to an accumulating list of items would seem to point towards a LISP implimentation. But higher order inferencing which more advanced forms of the system would require point to a PROLOG or even a 'C' implimentation. However, the availability of DEC/LISP which has recently been introduced to all VAX machines decided the issue and the core part of FEES which impliments the modulus of section 3.2.2 and selects the elements is written in DEC/LISP.

As explained above, the control modulus which call on the element selector routines and interface with the user employ FORTRAN or VMS routines.

4. FEES Architecture

4.1 Main Components

The program is constructed in the usual modular form adopted by all modern software developments. The main components consist of three major modules described in 3.2.2; the structural description module, the element evaluation module and the element selector. Information flow and overall control are achieved through the control module whilst the user operates the system through the input/output module. A schematic outline is shown in Fig.1.

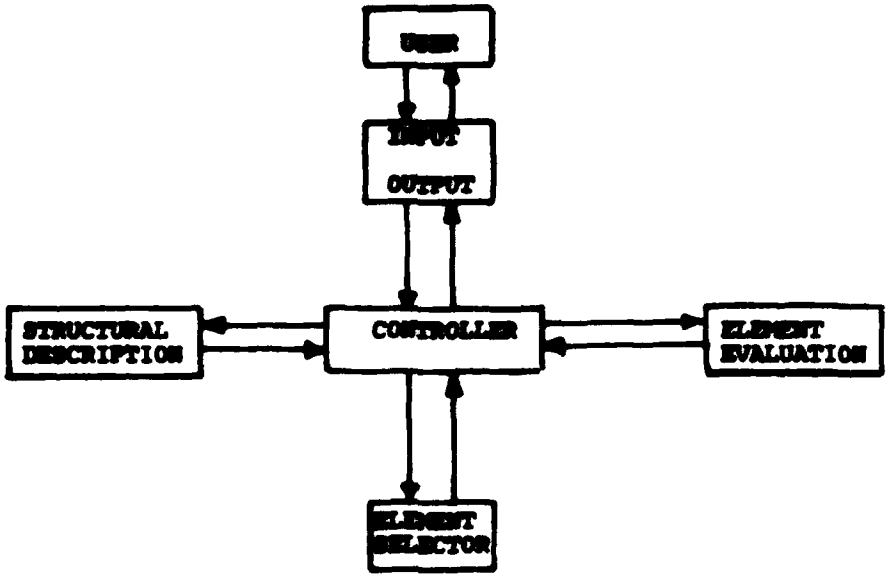


Fig.1

At the present time there is no specific database though a database and appropriate management system will be required in future as the data content of the system builds up.

4.2 Controller

The control module manages the flow of the program organising tasks such that each activity is carried out at its appropriate time. If the user asks for a task to be performed out of sequence the controller overrides and indicates what the correct procedure ought to be. It also ensures that concurrent programs run without interference with each other and keeps track of the overall process. The module progresses all of this in a manner invisible to the user and performs these tasks through the VMS operating system.

4.3 Input/Output Modules

The Input/Output Module currently forms the interface between the Controller/Inference modules and the user. This module uses the power of VMS to screen manage the input/output requirements and allow the Expert System to dialogue with the user. Some of the programs required to manipulate the data are written in FORTRAN which form the link with LISP. Although HELP facilities are supplied to the user via pre-written text, no attempt has yet been made to allow genuine explanation and tutorial facilities to be incorporated. However, experience with the LOKI [7] project indicates that a way forward does exist to introduce a natural language facility to improve the dialogue capability of the program.

4.4 Inference Modules

The three modules, Structural Description, Element Evaluator and Element Selector, represent the basic inference facilities of the existing program. The functions and construction of these modules are described in Section 3.2.2.

4.5 Usage

The program outlined above has been put into operation on the CoA VAX 11/750 operating under DEC V.M.S. The current element library contains a small set of element descriptions created for research purposes and an alternative set representing the LUSAS element library (LUSAS is a proprietry package).

5. CONCLUSIONS AND RECOMMENDATIONS

5.1 Lessons from FEES

The development and use of the prototype FEES program has indicated the soundness of the basic concepts of applying Expert System ideas to enhance the 'User-friendliness' and 'applicability' of Finite Element Systems. It has also made clear that no E.S. can be created which does not require a basic engineering knowledge of structural mechanics on behalf of the user. Nevertheless, E.S. technology can be employed in the near future to allow engineers with design experience only to approach and use systems such as MARC, PAFEC, ANSYS etc. without having to become finite element experts first.

In creating the program it was necessary to start the process of classifying and defining finite element 'surface' and 'deep' knowledge. This leads to the creation of the 'Inference' Modules which have a clearly defined structure of their own. As a result it is apparent that the handling of finite element knowledge can be done by some form of Object Oriented language. Such a language could be used to define the knowledge already described in the current 'Inference' Modulus and any future extensions covering more of the F.E. problem. In addition, an Object Oriented approach could be used to great effect in the Control Module as part of a Meta-level reasoning ability.

If an Object Oriented Approach similar to the C.M.L. [8] language is adopted than a link would be formed with existing Natural Language programs which could be employed to create a useful tutorial and explanation facility for FEES. Within such a tutorial model the 'User Assesor' program would be located which would asses the users general capability and knowledge and automatically adjust the level of explanation accordingly.

As seen in 3.2.2 the approach adopted in the development of FEES allows knowledge and constraints to be put together in a form which can be easily interpreted as instructions for a User Manual. Thus, the resulting E.S. program could output results in a form allowing a direct input to NASTRAN (say) and thereby dispence with major parts of the traditional Users Manual. This would also require the incorporation of graphics facilities which are easily achieved in this type of approach.

Finally, the structure of the prototype and any future successor are of such a simple form that it is possible to consider ways in which the computer itself could asses a given solution run. In consequence, a form of learning can be built into the system whereby previous runs are examined for patterns of usage. This would be done in quiet periods through the development of an interesting program called 'DREAMER'. This program would come into operation when FEES was not in use, would review previous runs, and attempt to assess any useage patterns which would then be employed to speed-up future consultations.

5.2 Recommended Future Developments

5.2.1 Extensions to FEES

The existing program can be extended in several directions without a radical re-organisation in the following areas:

i) Development of the knowledge base of FEES to cover the full finite element problem range indicated in Section 3.2.2. This would require extending the architecture of the program from that shown in Fig.1 to incorporate extra modules which would cater for meshing, results interpretation, etc.

ii) Interface with a graphics capability to allow screen manipulation of input and output data. Linking with pre-and post-processors would also be advantageous.

iii) Creation of a User Manual replacement module to allow for the easier use of NASTRAN, ANYSIS, etc.

iv) Linking of the program to a general finite element rational database to increase the range of potential problems.

5.2.2 Object Oriented Language

As indicated elsewhere in this report the structure of the knowledge base indicates that an opportunity exists to exploit an Object Oriented Approach. A variety of O.O. Languages are available but there is some advantage in selecting one where the development of the Language itself can be blended into the development of the knowledge base and inference engine. Within this framework one possibility is to employ the Conceptual Modelling Language (C.M.L.) being developed in the LOKI [8] project within the European Economic Committees ESPRIT programmes. Developments could, therefore, take place in the following areas:

i) Exploitation of C.M.L. to enhance the Control Module facilities and to be the main procedure for describing the knowledge.

ii) The use of C.M.L. to create a more advanced set of reasoning facilities both at the Meta-level and lower levels of the program.

iii) Exploitation of C.M.L. to incorporate a limited set of natural language facilities into the Control Module.

5.2.3 M.M.I. Features

The current Man-Machine Interface facilities of the FEES program are limited to the current Input/Output Module which has a very restricted user interface capability. Two major developments are required in this area to allow the Expert System to be effectively used by the general (aeronautical) engineer:

i) Explanation and Tutorial facilities are required to allow the program to explain the reasons for

reaching a specific decision and to instruct the user. A simple set of 'canned' texts and HELP files is not adequate to this complex task.

ii) Coupled with the Explanation/Tutorial facilities a 'User Assessor' is required to allow the program to cope with users who are more ignorant of the finite element method and its limitation than they realise.

5.2.4 Dreamer

As indicated in Section 5.1 it is possible to construct a program which can have a limited 'learning' capability built-up from runs of the FEES system. The concept uses the idea that each run of the system is logged in a file and is then available for analysis. This analysis is carried out by a new program called Dreamer which examines the data for patterns of usage. These patterns can then be used to prompt users towards likely solutions to meshing or other problems when a new structure is being set-up for F.E. analysis.

It is felt that most design offices do have their own range of structural analysis problems which have a high degree of similarity. As a result many problems will have been previously confronted and the F.E. analyst often starts the solution process using this 'in-house' knowledge. Dreamer is intended to follow the same philosophy and, thus, speed-up the inferencing process. The use of quiet periods on the computer for solution analysis implies that the Expert System will appear to 'learn' finite element knowledge when not in use and each consultation should seem to represent an improvement on the previous one.

REFERENCES

1. Mason Conference 'IKBS in Design and Manufacture' at the British Association for the Advancement of Science, sponsored by the Institution of Mechanical Engineers, 4th September, 1986, Bristol University, Bristol, U.K.

2. J. Alsina, J.P. Fielding and A.J. Morris
ADROIT: An Aid, ESPRIT Technical Week, Brussels, Belgium, October 1985.

3. J. Bennet, L. Creasy, R. Englemore and R. Melish
SACON: A Knowledge - Based Consultant for Structural Analysis, Stamford University Computer Science Dept., Rept. No.STAN-CS-78-699, September 1978.

4. I.C. Taig
Private Communication, BAe. Warton Division, Warton, Lancashire, U.K.

5. P.E. Johnson and W.B. Thompson
Strolling Down The Garden Path: Error Prone Tasks in Expert Problem Solving.
7th I.J.C.A.I., 1981, p.p.215-217

6. A. Sloman
Why We Need Many Knowledge Representation Formalisms. Res. and Dev. in Expert System. (Ed.) M.A. Blower, Proc 4th Conf. Bi. Computer Society Specialist Group in E.S. December 1984.

7. ESPRIT PROJECT 107
LOKI: A Logic Oriented Approach to Knowledge and Databases Supporting Natural User Interfaces.

8. Research Centre of Crete
Conceptual Modelling Language. LOKI Interim Progress Report Number 4 on C.M.L., March 1986.

APPENDIX A

USER'S GUIDE

A.1 INTRODUCTION

The FEES program described in this appendix is a prototype program that selects the best elements according to a specified set of requirements. The user selects the FE-package to be used and then specifies the problem to be modelled. FEES takes the element description data and the problem specification and selects the elements which best satisfied these.

Great emphasis has been made on making FEES easy to use. The data input is free-format and it is performed interactively from the terminal. The system uses the Screen Management Guidelines (SMG) available in VAX/VMS in order to provide a "friendly" environment to work with. The presentation of the solutions are clear and self explanatory and enables the user to interpret the results without delay. The user has a set of commands which allow him to control the consultation.

A.1.1 Facilities in FEES

The user will find the following facilities of great benefit when using the system:

1. Flexible data input by the user. The data input is performed interactively via menu windows. For each menu a set of specific instructions is displayed in a separate window. Further help can be obtained with the HELP command. The options done in a menu are always highlighted. For menu questions, the system displays a small counter window which indicates the number of questions, the number answered, and the number to answer.

2. Solution order. A typical consultation with FEES consists in defining the FE-package file name and defining the problem to be modelled. The program then selects the best elements and displays the solution.

3. Error diagnostics. FEES checks all user input and the correct evaluation of the different options and issues the appropriate error message when an error occurs.

4. User interface facility. At any stage during the consultation session, FEES can be instructed by the use through a set of simple commands to get further explanation on the current menu, go back to a previous menu etc.

A.1.2 Solutions

FEES has been developed to provide solutions for the element selection problem within FE analysis and it is hoped to extend the present prototype to cover a much larger part of this field. As a prototype it has been verified by reference to worked examples, it is relatively easy to learn, use and apply. In presenting the results we have tried to provide the explanations which lie behind the reasons for rejecting or selecting a particular element.

A.2 USER PROCEDURES

This section describes the procedures the user must follow in order to employ the FEES system together with the procedures for checking input data and correcting input errors. The order in which these are given approximates the order in which a FEES consultation proceeds.

A.2.1 Logging in

Make sure that the terminal is on and press the RETURN key one or more times until the system prompts to you for your user name, type in your user name and press the RETURN key. The system then prompts you for your password, enter your password and press the RETURN key.

A.2.2 Accessing the FEES program

After successfully logging in, the default DCL "$" prompt will be shown on the left margin indicating that the computer is ready to accept your commands. Type the FEES command and press the RETURN key. This command will start up the FEES system by loading the required files (this usually takes a minute or two depending on the load on the machine) and displaying the FEES main menu on a window positioned at about the centre of the screen (this will be the position from which FEES will display all the question and text menus) together with an instructions window positioned at the bottom left hand corner of the screen (this will be the position which FEES uses to display the instructions for every question or text menu).

NOTE

At the DCL level all commands given to the computer must be followed by pressing the RETURN key to indicate that you have finished typing the command. Within FEES you tell the system that you finished with a menu when you press CTRL/Z, or a function key command, see below.

A.2.3 Consulting FEES

The consultation session starts by asking the user to choose one of the options displayed in the main menu. These are outlined below.

A.2.3.1 OPTION 1. Element selection - This option allows the reading of the element data from a file which has been created on a previous consultation or edited separately. A further menu is presented here, whose main purpose is to define the element data to be used by FEES in order to model the problem. These options are:

OPTION 1. Lists the file names or tables which contain the element data in the format needed by FEES. Selecting one of the names restores element data, if a new name is used, a new table name is created.

OPTION 2. This option lists the element names known to FEES.

OPTION 3. Enables the user to view the element data in terms of its property names and property values.

OPTION 4. This option enables the user to add (delete or modify) information to (from) an old element.

OPTION 5. With this option new elements can be added to FEES.

OPTION 6. When creating or modifying an element the property names and values defined must be known to FEES thus, this option list the valid names and values.

OPTION 7. Returns the user to the main menu.

A.2.3.2 OPTION 2. Problem definition - With this option FEES is told the problem to model by answering a series of questions. The systems displays previous regions which may be selected if it is necessary to restore and modify a previous region. If one of the names is selected or a new one is typed, a further menu is displayed with a series of questions to be answered in order to define the problem. For each question several answers are possible, by specifying how certain (or

important) the user is regarding the options the system may employ these in order to satisfy the user requirements. This value should be given as a percentage, no answer or 0 being don't know, 100 being for certainty, 50 for not too sure etc.

A.2.3.3 OPTION 3. Element selection - Here the program evaluates the answers given above by selecting the elements which best match the requirements. The screen is cleared and the message "LISP WORKING." is displayed on the top left hand corner of the screen (this might take a few minutes depending on the number of elements and regions used, and how busy the machine is). When finished, the main menu is re-displayed with the option to view the valid solutions.

A.2.3.4 OPTION 4. Restart a consultation - This option allows the user to restart a previously SAVEd session so as to modify or continue the session. The system displays the names which can be restored. The name typed must correspond to one of the names shown.

A.2.3.5 OPTION 5. Solutions - This option allows the solution to be vieved. The region names being used are displayed for viewing the solutions individually. The combined solutions are shown under the name "REGIONS".

1. RULE 1. Eliminates the elements which have nothing in common with the important requirements. i.e., All the answers greater 50 percent.

2. RULE 2. Eliminates the elements which lacked some important requirements. i.e., The answers greater 50 percent.

3. RULE 3. Eliminates the elements which had all the essential requirements but missed some inessential ones. i.e., All the answers below and including 50%. The don't knows and 0 certainty are not taken into consideration.

The final solution for a region is shown as a list of elements which have satisfied all the important and essential requirements.

For multiple regions the valid elements and the ones eliminated by RULE 3 are collected and RULE 4 rule is used. This rule finds the bar and membrane elements and tries to match the nodes at the boundaries. If it succeeds it shows the results as a list of bars with the corresponding membrane(s). Elements which do not belong to these two types are displayed as a separate list.

A.2.3.6 OPTION 6. Logoff from FEES - This option exits the user from FEES. The screen is cleared and the following message displayed

```
End of consultation. Back to LISP.
Type (EXIT) to exit LISP or (RUN) to start a new session.
Lisp>
```

To the LISP prompt "Lisp>" type the desired action.

A.2.4 Answering questions

The actions required in order to "make the system do things" are follow:

1. Name Question. The system presents the question to the user with the appropriate instructions, the name given must be alphanumeric and according to the instructions given. Names are typically used to identify an element or a file.

2. Numeric Questions. The system presents the question(s) to the user, the answer given must be within a valid range if not, the system prints an error message notifying what went wrong.

3. Menu Options. The answer given should correspond to one of the options displayed in the menu. If the answer does not match any of the options then an error message is displayed and the correct option should be typed.

4. Yes/No Questions. This type of question expects a yes or no for an answer if not, the system prints an error message. For some questions a default answer is included in the question, this is taken if no answer is given.

A.2.5 Finalizing the element selection

If no element is found to satisfy all the important and inessential requirements then, use can be made of one of the elements given in the list provided by RULE 3 or the requirements modified, or a different F.E. package chosen. On the other hand if the evaluation is found to be satisfactory, a new session can be started with the restart option or logout from FEES with the main menu options 4 or 6 respectively.

A.2.6 Requesting help during a consultation session

At any point during a consultation session the user can find the commands available to him through the INDEX command or alternatively by hitting the "PF4" key.

A.2.7 Procedures for checking and correcting input data

According to the type of question the system checks that the value given corresponds to a valid input format. Numeric questions should be within a valid range, for yes/no questions the answers given must be YES or NO (Y or N for short). Finally, there are menu type of questions in which the answer given must be one or more of the choices displayed, when there is only one answer this is made explicit in the wording of the question. The correction of input data is made easy by having the system display the error thus, the user can edit his answer.

A.3 OPERATING INSTRUCTIONS

The instructions described in this section are carried out in order to consult the FEES system, assuming that all the software (LISP version 2 and the FEES suite programs, text, and data) and hardware (e.g. VAX-11/VMS version 4, terminals such as the VT100 or compatible) are available. The instructions described here include: format and content of each input, input checks that are made by the system, action taken if an error is found.

A.3.1 Looking at error messages

When answering questions or using commands, FEES has a wide range of error messages in order to provide objective information about the error. The system displays the message on a separate window positioned at the bottom right hand corner, this window is removed when the answer given is accepted. Some of these are:

1. Outside of range. This error occurs when the answer given does not lie within a valid range.

2. Incorrect choice. When the answer(s) given does not match any of the options in the menu.

3. Invalid answer. This error is displayed when the system does not understand the answer or it is invalid within the present state of evaluation of the session.

A.3.2 Using the FEES commands

The system provides four basic commands which can be issued at any time during the consultation by using the appropriate PF key or by typing the command and pressing CTRL/Z. These are (PF key and word):

1. PF1 or RPM
   This command returns the user to the previous menu without processing any of the answers given in the current menu. It is useful if the user does not want to make a choice.

2. PF2 or HELP
   This command produces help text on the current menu in the form of definitions and/or example answers.

3. PF3 or SAVE
   This command saves the state of the consultation and logs the user off from FEES returning to LISP.

4. PF4 or INDEX
   The INDEX command produces this help text on the commands available.

A.3.3 Instructions for answering questions

Every question and text menu in FEES displays a set of instructions aimed at helping the user answer and read a piece of text. The main types of display are:

1. Menu Questions. These type of questions display an *enumerated selection of* possible answers from which the user can select the option number and attach a certainty of his answer. Depending on the question single or multiple options can be chosen.

2. Menu Options. Menus of this type present an enumerated sequence of the possible choices and the answer given should correspond to one of the options.

3. Yes/No Questions Questions of these type are used to confirm an action to be taken step before proceeding. Possible answers are: Y, YES, or N, NO. For this type of question there is no approximation i.e., answers like "not really" or "unknown" are not permitted.

4. Text Menus. These type of menus are used to display information to the user.

A.3.4 Logging out

When the session at the terminal is final, the option 6 is used in the FEES main menu to exit returning to LISP (the prompt "Lisp>" is displayed) then, LISP macro EXIT is used to return to DCL command level (i.e., the DCL prompt "$" is displayed) then the DCL command LOGOUT ends the session. The system responds by displaying the user

name, date and time when logged out.

APPENDIX B

PROGRAMMER'S GUIDE

FEES is a prototype program for element selection. This appendix is aimed at a programmer who wishes to understand FEES in order to extend it or modify it. Familiarity with LISP, FORTRAN, and VAX/VMS Screen Management Guidelines (SMG) is assumed. The appendix is concerned with the overall structure of the program, details are explained in the program comments.

B.1 OVERVIEW OF FEES

There are two major components to the FEES system. These are the Man-Machine-Interface (MMI) programs and the element selection programs.

B.1.1 The Man-Machine-Interface

The MMI is used to interact with the user by obtaining all the required input and displaying the results found. The input from the user is translated into a suitable form for the element selection program, the results produced by the latter are then displayed to the user.

B.1.2 The element selector

The element selector reads the element description data and "picks" the elements which best satisfy the user requirements. A further selection is made according by matching the nodes between elements.

B.1.3 The FEES source files

The FEES system is made up of a set of LISP and FORTRAN programs, and a set of text and data files. These are:

B.1.3.1 FORTRAN files -

1. ELEMENT_DATA.FOR
   Reads from and writes the element data to a file.

2. ERROR.FOR
   Used to display the different error messages.

3. EXTERNAL.FOR
   External routines used within LISP to write results.

4. FILES.FOR
   Controls the user interaction.

5. GET_FE_FILE.FOR
   Requests the FE-package name to be used by the element selecter, it also provides various facilities for viewing and modifying the element description data.

6. GET_REGIONS.FOR
   Requests one or more region names to be used to identify the user's problem.

7. GET_TRUTH.FOR
   Asks the user various questions in order to categorized the problem.

8. HELP.FOR
   Displays help text on a question.

9. SESSION.FOR
   Saves or restores a consultation session.

10. SMG_ROUTINES.FOR
    Various routines are defined here which use the SMG VAX/VMS facility to request input and display results in the form of windows.

11. SOLUTION.FOR
    Routine to retrieve and display the results.

12. UTILITIES.FOR
    General purpose routines for string handling, input checking, initialization etc.

B.1.3.2 LISP files -

1. ELEMENT_SELECTION.LSP
   The controller for element selection.

2. EXTERNAL.LSP
   Definitions of the external FORTRAN routines used within LISP.

3. FE.LSP
   The controller for the interaction with FEES.

4. LOAD.LSP
   This file is used to load the different LISP source files required by FEES, and starts up the session.

5. MATCH_NODES.LSP
   The controller to match nodes between different regions.

B.1.3.3 TEXT files - A set of files is used by the FORTRAN procedures to prompt the user, it also provides the necessary instructions and help text to answer a question and the text used in the error messages.

1. file.MENU
   The menu files are based on the philosophy that a question consists of two parts, the question and the instructions on how to answer. Thus, when FEES prompts the user it creates two windows: one instruction window and one menu window with a "reading side" where the user can type an answer or command or close the windows. Each file contains the instructions title and text (3 lines) followed by the menu title which names the "parent" menu and text (100 lines maximum).

2. file.HELP
   Similarly, the help files consists of an instructions part and a text part. Each menu has a help text file associated with it.

3. ERROR.MESSAGES
   This is a single file which stores the text (3 lines) used when an input has occurred. The error messages are displayed on a separated window to the above.

B.1.3.4 DATA files - Two files are used to store the element property names and property values.

1. PROPERTY.DATA
   This file stores the property names and values known to the system. It is used to provide a guideline to the user when defining or modifying an element.

2. QUESTIONS.DATA
This file stores the property names and values associated with the questions used by the system in order to define the problem. The order in which they are stored relates to the order of the questions as specified in the file "QUESTIONS.MENU".

3. file.DAT
Stores the element description data to be used by the element selector program.

B.2 LISP FILES

Here we outline the overall function of the LISP files, since a more detailed description can be found in the source files comments.

B.2.1 The ELEMENT SELECTION.LSP file

Two important facilities are provided by this file:

1. A method for retrieving and storing the element description data.

2. A method of selecting an element given some requirements, providing some explanation when eliminating or selecting an element.

B.2.1.1 Reading the element data -

READ-ELEMENT-DATA (FILENAME)
This procedure is used to read the data from the file FILENAME. The element data must be stored in the following way:

1. The first record indicates the number of elements in the file.

2. For each element description the following order is taken: the element name, the number of properties, then for each property there is associated the property name and its property value(s). With the last record of the element description having the property STORE-SCORE with value 0.

The data is read in a procedural manner and the element names are stored in the list ELEMENT-NAMES, and the following two procedures are used to organise the data.

PUT (ELEMENT-NAME, PROPERTY-NAME, PROPERTIES).

This procedure is the PEES property list builder by associating each element with a set of property names and values.

ORDER-PROPERTY-LIST (PROPERTY-LIST)
This procedure takes a property list as input and generates a new property list based on the old one but re-ordered so that the property identifiers and their values are in alphabetical order.

B.2.1.2 Element selection -

ELEMENT-SELECTION (REGIONS, FILENAME)
For each region the hard and soft requirements are obtained and the elements are scored accordingly. Three rules are used in order to eliminate the elements which do not satisfied some or all of the requirements. The results are written to a file for later consultation. The following are the main procedures used:

1. READ-REQUIRED-ELEMENT-DATA (FILENAME PRIORITY I-PRIORITY)
   This routine reads required element description data as specified by the user. FILENAME represents the region name and it is used to write the questions (property names) asked to the file "FILENAME_QUESTIONS.MEMORY". PRIORITY is a string which denotes the certainty of the answers given i.e., currently 100% for answers between 51 and 100 certainty value, and 50% for answers between 1 and 50. The I-PRIORITY is the integer value of PRIORITY.

2. SCORE-EACH-ELEMENT
   Given the information contained in modules READ-ELEMENT-DATA and READ-REQUIRED-ELEMENT-DATA this module compares the data in the element property list (plist) and the required data property list and writes the score in the element property list under the property identity SCORE-STORE.

3. RULE-1 (FILENAME)
   This is an element selection rule which states that if an element in the database does not match any of the requirements specified, we must ignore it for the rest of the run. The scores of each element are checked and if they are zero that element is removed from the list ELEMENT-NAMES. The rejected elements are stored in a list called RULE-1-DISCARD. FILENAME is the name of the region. The rejected elements are written to the file "FILENAME.RULE1" and the answers given to the file "FILENAME_ANSWERS.MEMORY".

4. RULE-2 (FILENAME)
   This is an element selection rule which states that if an element in the database does not match all of the essential requirements specified, we must ignore it for the rest of this run. The scores of each element are checked and if they are less than MAXR (maximum score for a given requirement) that element is removed from the list ELEMENT-NAMES. The rejected elements are stored in a list called RULE-2-DISCARD

together with the number of missing properties. FILENAME is the name of the region. The rejected elements are written to the file "FILENAME.RULE2" and the answers given to the file "FILENAME_ANSWERS.MEMORY".

5. RULE-3 (FILENAME)
This is an element selection rule which states that if an element in the database does not match all the requirements specified, we must ignore it for the rest of this run. The scores of each element are checked and if they are less than MAXR that element is removed from the list ELEMENT-NAMES. The rejected elements are stored in a list called RULE-3-DISCARD together with the number of missing properties. The rejected elements are written to the file "FILENAME.RULE3" and the answers given to the file "FILENAME_ANSWERS.MEMORY".

6. SCORE-EACH-ELEMENT
Given the information read by modules READ-ELEMENT-DATA and READ-REQUIRED-ELEMENT-DATA this module compares the data in the element plist and the required data plist and writes the score in the element plist under the property name SCORE-STORE.

7. OUTPUT-ELEMENT-NAMES (FILENAME N I)
This routine writes the valid elements to a file FILENAME (region name) after RULE-1 and RULE-2 and then after RULE-3, creating the files "FILENAME_1.OUT" and "FILENAME_2.OUT" respectively.

8. Finally, the results (the valid element names) are written to the file "FILENAME.SOLUTIONS" where FILENAME denotes the region name.

B.2.2 The MATCH NODES.LSP file

MATCH-NODE-BETWEEN-ELEMENTS (REGIONS)
When there are more than two regions, a further reduction in the number of elements is made by taking the list of elements eliminated by RULE-3 and the valid elements for each region and then matching the nodes at the boundaries. The following are the main procedures used:

1. READ-REGIONS-ELEMENT-NAMES
This procedure reads the element names for the different REGIONS from the file FILENAME.RULE3 and FILENAME_2.OUT where FILENAME is the name of a region in REGIONS, and puts them in the list ELEMENT-NAMES.

2. RULE-4 (FILENAME)
This is an element selection rule which states that if two elements of type bar and membrane in the database are not compatible at the nodes they must be ignored for the rest of

the run. Other elements which are not immediatly seen as incompatible are stored in LIST-OF-OTHER. The rejected elements are stored in a list called RULE-4-DISCARD i.e., bar elements for which there are no matching membrane elements. FILENAME is the name of the file "REGIONS". The rejected elements are written to the file "REGIONS.RULE4".

3. OUTPUT-REGIONS-ELEMENT-NAMES (FILENAME)
The list ELEMENT-NAMES of valid elements is written to the file "REGIONS.OUT".

4. Finally, the results (the list of valid elements and the elements for which RULE-4 does not apply) are written to the file "REGIONS.SOLUTIONS".

B.2.3 The EXTERNAL.LSP file

Because of the nature of LISP, explicit declarations of the external routines used have to be made, see VAX-LISP user manual. These are (equivalent format):

1. FEES (REGIONS, FILENAME, ORDER-FLAG)
This routine gets the input from the user and it then displays the solution file. REGIONS is a string of the form "REG REG1 ... etc", FILENAME is a string of the form "LUSAS.DAT", and ORDER-FLAG is an integer which indicates the state of the consultation i.e., exit, solution, or evaluation.

2. INTERPRETER_1 (FILENAME)
This routine reads the solutions data files and formats them into a readable form explaining the meaning of the data. FILENAME is a string denoting the region name. This routine creates the solution file "REGION.SOLUTIONS" for each region.

3. INTERPRETER_2 (FILENAME)
This routine reads the solutions data files and formats them into a readable form explaining the meaning of the data. FILENAME is a string denoting all the regions used. This routine creates the solution file "REGIONS.SOLUTIONS".

4. MEMORIZE (FILENAME, MEMORY-MODE, ORDER-NUMBER)
This routine collects all the output from a selection run and stores it in the memory file. Each piece of data is tagged with date, time , quantity and type information. FILENAME is the region name, MEMORY-MODE is a string number, and the ORDER-NUMBER is an integer.

B.2.4 The FE.LSP file

This is the controller for the interaction with FEES. It initializes some data and calls the external procedure FEES, according to the user specification it evaluates the ELEMENT-SELECTION or exits. After evaluation the external procedure FEES is called in order to observe the results, re-evaluate the ansvers, or exit from FEES.

B.3 FORTRAN FILES

Several files have been written in FORTRAN to build the MMI side of the system using the SMG facilities. Here we outline each file, with a more detail description in the source files comments.

B.3.1 The ELEMENT DATA.FOR file

This file defines the routine ELEMENT_DATA which reads from or writes to a file the element names, property names and property values.

FORMAT

ELEMENT_DATA (IO, IN_FILE, NO_OF_ELEMENTS, ELEMENT_NAME, NO_OF_PROPERTIES, PROPERTY_NAMES, PROPERTY_VALUES)

ARGUMENTS

IO = integer to indicate read or write.

IN_FILE = input string; indicates the name of the FE-package name.

NO_OF_ELEMENTS = output integer; indicates the number of elements in the file (maximum 100).

ELEMENT_NAME = output string vector; stores the elemenet names.

NO_OF_PROPERTIES = output integer vector; stores the number of properties for each element (maximum 20).

PROPERTY_NAMES = output 2D string array; stores the property name for an element.

PROPERTY_VALUES = output 2D string array; stores the property values for an element property.

B.3.2 The ERROR.FOR file

This file defines the routine ERROR which prints the error message by looking for the error type in the file DIR_ERROR:ERROR.MESSAGES. It rings the bell three times before printing the error message.

FORMAT

ERROR (PB, VD3, ERROR_TYPE)

ARGUMENTS

PB = input integer; denotes the pasteboard identification.

VD3 = input integer; denotes the virtual display for the error window.

ERROR_TYPE = input string; denotes the error type.

B.3.3 The EXTERNAL.FOR

This file defines the three routines used within LISP to format the results.

1. INTERPRETER_1. This routine creates a solution file of the for "REGION.SOLUTIONS" for each region REGION. It interprets the results of rules 1, 2, and 3 and gives the final solution as a list of valid elements.

   FORMAT

   INTERPRETER1 (REGION)

   PARAMETERS

   REGION = input string; specifies the region name.

2. INTERPRETER_2. This routine interprets the results for two or more regions by writting the solution to the file "REGIONS.SOLUTIONS". (The name REGIONS is used).

   FORMAT

   INTERPRETER2 (REGIONS)

   PARAMETERS

   REGIONS = input string; specifies the list of regions.

3. MEMORIZE. This routine writes to a file the location and time at which the questions and answers were processed.

   FORMAT

   MEMORIZE (FILENAME, MEMORY_NODE, ORDER_CODE)

   PARAMETERS

   FILENAME = input string; specifies the region name.

MEMORY_NODE = input string; specifies the filename to which questions ("REG_QUESTIONS.MEMORIZE") and answers ("REG_ANSWERS.MEMORIZE") are stored.

ORDER_CODE = input integer; specifies solution order.

B.3.4 The FEES.FOR file

This file defines the procedure FEES which controls the interaction with the user. The procedure is called from LISP and it is used to obtain and display results to the user. FEES erases the screen, creates the instructions, menu, and error windows then, displays the main menu text. The procedure actions in this menu are:

OPTION 1. Calls the routine GET_FE_FILE.

OPTION 2. Calls the routine GET_REGIONS.

OPTION 3. Exits from this procedure and returns to LISP, allowing the element selection to be performed.

OPTION 4. Calls the routine SESSION.

OPTION 5. Calls the routine SOLUTIONS.

OPTION 6. Exists from this procedure and returns to LISP.

FORMAT

FEES (REGIONS, FE_FILE, ORDER_FLAG)

ARGUMENTS

REGIONS = input/output string; specifies the regions used.

FE_FILE = input/output string; specifies the FE package used.

ORDER_FLAG = input/output integer; specifies the current state of the consultation. Valid values are as follows:

0 output integer; if consultation is to be terminated.

1 input/output integer; if FEES is to request the user the FE_FILE and the REGIONS.

2 input integer; specifies that the element selection has been completed and that the solutions can be viewed.

B.3.5 The GET FE FILE file

This file defines the routine GET_FE_FILE to request the table name

from the user in order to read the element description data in terms of the number of elements, property names and values, it also allows the user to define new elements, or modify existing ones.

FORMAT

GET_FE_FILE (PB_INFO, FE_FILE, OLD_FILE, OPTION_DONE, COMMAND_FLAG)

ARGUMENTS

PB_INFO = input integer vector containing the display information.

FE_FILE = output string denoting a previous or a new FE-file.

OLD_FILE = output string denoting a previous FE-file.

OPTION_DONE = output integer vector; indicates the options selected by the user.

COMMAND_FLAG = output integer; indicates the successful completion of this routine.

B.3.6 The GET REGIONS file

This routine asks for a REGION_NAME then it calls GET_TRUTH and saves the answers given in the file "REGION_NAME.REGION".

FORMAT

GET_REGIONS (PB_INFO, REGIONS, REGION_FLAG, COMMAND_FLAG)

ARGUMENTS

PB_INFO = input integer vector with the display information.

REGIONS = output string denoting the list of regions selected by the user.

REGION_FLAG = output integer vector denoting the state of each region (maximum of 10).

COMMAND_FLAG = output integer used to denote any command given by the user.

B.3.7 The GET TRUTH.FOR file

This file defines the routine GET_TRUTH which asks several questions to the user in order to define the problem. On successful completion it creates two files REG_100.REQ and REG_50.REQ where REG is the region name and the 100 and 50 denote the certainty of the answers given i.e., all the answers between 51 and 100 are stored in the file

REG_100.REQ while the answers between 1 and 50 are stored in the file REG_50.REQ. 0 answers and no answers are taken as don't knows. (For the property OUTPUT any answer given defaults to 50% certainty).

FORMAT

GET_TRUTH (PB_INFO, REGION_NAME, NO_OF_QUESTIONS, LINK_LINE, REPLY, COMMAND_FLAG, TRUTH_FLAG)

ARGUMENTS

PB_INFO = input integer vector containing the display information.

REGION_NAME = input string denoting the region name.

NO_OF_QUESTIONS = input/output integer denoting the number of questions.

LINK_LINE = input/output integer vector denoting the position of the questions in the text file.

REPLY = input/output string vector denoting the answers given by the user.

COMMAND_FLAG = output integer; denotes any user command.

TRUTH_FLAG = output integer; denotes the successful completion of this routine.

B.3.8 The HELP.FOR file

This file defines the HELP routine which prints help text on the current menu. It creates a smaller window than the FEES menu window which contains the 'help' text and is deleted returning to the menu window.

FORMAT

HELP (PB_INFO, IN_FILE)

ARGUMENTS

PB_INFO = input integer vector; contains the display information.

IN_FILE = input string; specifies the menu file (the question) for which help is required.

B.3.9 The SOLUTIONS.FOR file

This file defines the routine SOLUTIONS used to view the individual regions solutions. It displays the regions for selection, when more

than one region exists the name "REGIONS" is also included. The user selects one, and the procedure VIEW_TEXT is called.

FORMAT

SOLUTIONS (PB_INFO, REGIONS)

ARGUMENTS

PB_INFO = input integer vector; contains the display information.

REGIONS = input string; specifies the region name(s).

B.3.10 The UTILITIES.FOR file

This file defines various general purpose routines for string handling, input checking etc. The routines are:

1. CHECK. This routine counts the number of characters according to their type.

   FORMAT

   CHECK (STRING, L, NO_OF_BLANKS, NO_OF_DECIMALS, NO_OF_ILLEGALS, NO_OF_LETTERS, NO_OF_MINUS, NO_OF_NUMBERS)

   ARGUMENTS

   STRING = input string.

   L = output integer; denotes the length of the string.

   The rest of the arguments are output integers, their name denotes their type.

2. CHECK_NAME. This routine checks that a string is alphanumeric.

   FORMAT

   CHECK_NAME (STRING, REASK)

   ARGUMENTS

   STRING = input string.

   REASK = output integer; denotes if the string is alphanumeric or no.

3. CHECK_YESNO. This routine checks that a string is a yes or no answer.

   FORMAT

CHECK_YESNO (STRING, REASK)

ARGUMENTS

STRING = input string.

REASK = output integer; denotes if the string is one of: YES, Y or NO, N.

4. CLEAR. This routine initializes a string vector to empty.

FORMAT

CLEAR (LINE, LINES)

ARGUMENTS

LINE = input string vector.

LINES = input integer; denotes the number of lines in the vector.

5. COUNT ITEMS. This routine counts the number of items in a string (assume one or blanks between them), and returns the items in a vector and the number of items.

FORMAT

COUNT_ITEMS (STRING, LINE, LINES)

ARGUMENTS

STRING = input string.

LINE = output string vector.

LINES = output integer.

6. CSORT. This routine sorts a list of words in dictionary order using a simple insertion sort technique.

FORMAT

CSORT (LINE, LINES)

ARGUMENTS

LINE = input/output string vector.

LINES = input/output integer.

7. C TO I. This routine converts a string number to integer.

FORMAT

C_TO_I (STRING, N)

ARGUMENTS

STRING = input string.

N = output integer.

8. DIR. This routine formats a string vector in a "directory" form.

FORMAT

DIR (WIDTH, COUNT, LINE, LINES)

ARGUMENTS

WIDTH = input integer; specifies the line width.

COUNT = input integer; specifies the number of items per line.

LINE = input/output string vector with the individual/arranged items.

LINES = input/output integer representing the number of lines.

9. GET_FILES. This routine returns in a vector form the files found according to a certain specification.

FORMAT

GET_FILES (IN_FILE, LINE, LINES)

ARGUMENTS

IN_FILE = input string representing the filename(s) to look for.

LINE = output string vector representing the files found.

LINES = output integer representing the number of files found.

10. I_TO_C. This routine converts an integer to a string.

FORMAT

I_TO_C (N, STRING, L)

ARGUMENTS

N = input integer.

STRING = output string.

L = output integer representing the length of STRING.

11. LOCATE. This routine finds the position of an item in a string.

FORMAT

LOCATE (STRING, N, TAIL, L)

ARGUMENTS

STRING = input string.

N = input integer representing the position of the item to be found.

TAIL = output string representing the truncated string with the "head" as the item.

L = output integer representing the length of the item (0 if not found).

12. MEMBER. This routine looks for an element in a string and returns its position.

FORMAT

MEMBER (STRING, LIST, N)

ARGUMENTS

STRING = input string representing the item to be found.

LIST = input string.

N = output integer representing the position of the item (0 if not found).

13. TRIM OFF. This routine trims off the trailing and leading edge blanks from a string.

FORMAT

TRIM_OFF (STRING, L)

ARGUMENTS

STRING = input/output string.

L = output integer representing the "true" length of the string.

B.3.11 The SMG ROUTINES.FOR file

This file defines various routines which are used to handle the input/output with the user.

1. HIGHLIGHT_OPTION. This routine is used to highlight the options of a menu.

   FORMAT

   HIGHLIGHT_OPTION (VD, LINE, LINES, OPTION_DONE, NO_OF_OPTIONS)

   ARGUMENTS

   VD = input integer; specifies the display identification.

   LINE = input string vector; represents the menu lines.

   LINES = input integer; represents the number of lines in the menu.

   OPTION_DONE = input integer vector; represents the options to highlight.

   NO_OF_OPTIONS = input integer; represents the number of options in the menu.

2. KEYSTROKE. This routine reads and prints a keystroke at a given position in the screen. It updates the column position according to the key type. Refreshes the screen if CTRL/U or CTRL/R are used, deletes a character if the DELETE key is used, and converts lowercase letters to uppercase.

   FORMAT

   KEYSTROKE (PB, KB, VD, ROW, COLUMN, KEY, KEY_TYPE)

   ARGUMENTS

   PB = input integer; specifies the pasteboard identification.

   KB = input integer; specifies the keyboard identification.

   VD = input integer; specifies the read display identification.

   ROW = input integer; represents the current cursor row position.

   COLUMN = input/output integer; represents the current/updated column position of the cursor.

   KEY = output integer; represents the ASCII code number of the key.

KEY_TYPE = output integer; represents a code used to identify the key.

3. PRINT_DISPLAY. This routine clears a specified window and prints a given text to that window.

FORMAT

PRINT_DISPLAY (VD, LINE, LINES)

ARGUMENTS

VD = input integer; specifies the display identification.

LINE = input string vector; stores the lines to print.

LINES = input integer; denotes the number of lines to print.

4. READ_DISPLAY. This routine creates a display of given width and with the same height as the menu display to read the answers and/or commands from the user. The menu and reading displays are scrolled when necessary. For menu questions (each question is defined in the text file by an asterisk in the first column) an answer display counter is created. The bell is rung once to attract the user's attention.

FORMAT

READ_DISPLAY (PB, KB, VD, LINES, LINE, ROW_START, COLUMN_START, WIDTH, NO_OF_QUESTIONS, LINK_LINE, REPLY, COMMAND_FLAG)

ARGUMENTS

PB = input integer; specifies the pasteboard identification.

KB = input integer; specifies the keyboard identification.

VD = input integer; specifies the menu display identification.

LINES = input integer; represents the number of lines in the menu.

LINE = input string vector; represents the text lines in the menu.

ROW_START = input integer; represents the row position for the read display.

COLUMN_START = input integer; represents the column position for the read display.

WIDTH = input integer; denotes the width of the read display.

NO_OF_QUESTIONS = output integer; denotes the number of questions.

LINK_LINE = output integer vector; represents the position of the questions in the text file.

REPLY = output string vector; stores the answers/commands given by the user.

COMMAND_FLAG = output integer; identifies any given command.

5. SCROLL_DISPLAY. This routine scrolls a display, when the UP-ARROW, DOWN-ARROW, LINE-FEED or RETURN key are used.

FORMAT

SCROLL_DISPLAY (TYPE, DIRECTION, VD, LINES, LINE, IFLAG, ROW_OLD, ROW, ROW_FILE)

ARGUMENTS

TYPE = input integer. Possible values are: 0 to scroll the display, and 1 to read from the display while scrolling.

DIRECTION = input integer. Possible values are: -1 to scroll up the display by one row, and 1 to scroll down the display by one row.

VD = input integer; specifies the display identification.

LINES = input integer; represents the number of lines.

LINE = input string vector; represents the text lines.

IFLAG = input integer vector; indicates the position of non-empty lines.

ROW_OLD = input integer; represents the previous row position.

ROW = input integer; represents the new row position.

ROW_FILE = output integer; represents the cursor position within the file.

6. TEMPLATE. This routine reads and prints a file to the screen in two parts: part 1 reads and prints the title and text for the instructions while part 2 reads and prints the title and text for the menu or help window.

FORMAT

TEMPLATE (IN_FILE, VD1, VD2, LINES, LINE)

ARGUMENTS

IN_FILE = input string; indicates the filename to be read.

VD1 = input integer; specifies the menu display identification.

VD2 = input integer; specifies the instructions display identification.

LINES = output integer; represents the number of lines in the menu.

LINE = output string vector; represents the menu text lines.

7. VIEWTEXT. This routine allows a text to be viewed.

FORMAT

VIEW_TEXT (PB_INFO, IN_FILE, LINE, LINES, COMMAND_FLAG)

ARGUMENTS

PB_INFO = input integer vector; specifies the display information.

IN_FILE = input string; indicates the filename.

LINE = input string vector; represents the menu text lines to be viewed.

LINES = output integer; represents the number of lines in the text.

COMMAND_FLAG = output integer; indicates the command code if one was used.

END

DATE
FILMED

9-87