Implementation and Characteristics Of Rule-based System For The Finite Element Analysis

Hyo Jong Lee[1]

UUCS-89-002

Department of Computer Science
University of Utah
Salt Lake City, UT 84112 USA

August 28, 1990

[1]This was supported in part by DARPA (DAAK1184K0017 and N00014-88-K-0688). All opinions, findings, conclusions or recommendations expressed in this document are those of the author and do not necessarily reflect the views of the sponsoring agencies.

Contents

List of Figures

Abstract

It is well known that the analysis of process for the finite element method is tedious and error-prone steps. Considering the importance of the task of engineering analyses, such as structural analysis, heat transfer, fluid flow simulation, and electromagnetic potential, many researchers have tried to develop better and easier systems.

Meanwhile, expert systems have been developed in various areas, such as *DENDRAL*, *MYCIN*, and *XCON*. There are two main reasons for developing expert systems. First, an expert system can facilitate the dissemination of vital knowledge to a certain organization with a reasonable cost. Second, an expert system does not suffer from human problems such as confusion so that it can apply appropriate rules to the problem. It is obvious that development of an expert system for finite element mesh generation can save both time and money in the finite element analysis process.

A rule-based system for optimal finite element mesh generation, *EFEM* has been developed and implemented in powerful interactive solid modeler. Because required knowledge is translated into rules, it is not required to know detail information about the finite element analysis processes or computer science to test structural analysis. The implementation of the *EFEM* has been analyzed.

1 Introduction

The artificial intelligence technique has been widely used to reduce human beings' burden and increase the efficiency of works. knowledge engineers have developed rule-based systems to solve problems in various areas, such as prediction, diagnosis, design, planning, monitoring, debugging, instruction, and controlling systems. The artificial intelligence software has had great success since it moved from the laboratory into the real world. In fact the general tendency of software development goes from conventional programming techniques to artificial intelligence (AI) approaches that created expert systems. Because of ease and efficiency of AI approaches, the demand for expert systems is rapidly increasing. Pressman [Pre87] pointed out that using artificial intelligence techniques in software engineering has a great advantage. As an example, VAX family computer's line configurator *XCON* saves Digital Equipment Corporation $18 million annually [Wil86]. The ideal expert system can finish tasks thoroughly and correctly, because it does not feel fatigue and cannot become confused with facts if knowledge is provided by appropriate expertise and knowledge acquisition is done correctly. Therefore, users of the expert system are benefited by greater speed, fewer errors, reduced cognitive load, increased adaptability and robustness. For example, an efficient expert controller can adaptively govern the behavior of a problem domain system [FWL83]. In order to achieve this, the expert control system interprets the current situation, reasons the future, diagnoses the cause of expected problems, formulates a remedy and monitors its execution to ensure success.

As the success of an expert system has been demonstrated with results of the *XCON*, the expert system for the finite element analysis can save analysts both time and money. The prototype expert system, *EFEM* (Expert System for Finite Element Mesh generation) has been developed and implemented in this thesis. The knowledge required for the finite element analysis has been translated into the knowledge base of *EFEM* and the top-loop control program of the whole system, called *inference engine*, generates optimal meshes for the specified problem domain designed by a solid modeler. Thus, *EFEM* will disseminate vital knowledge of the finite element analysis into industry with reasonable cost.

2 Backgrounds for Expert Systems

In the previous section, the brief introduction to how the applications of expert systems had spread in various categories of problems was described. In order to understand the answer better, it is necessary to look at the difference between conventional programming and expert system techniques. Conventional programming deals with the problems that have known algorithms or procedures [Wil86]. On the other hand, expert systems can be defined as problem solving programs with computer models of expert reasoning to accomplish high level performance in a specified problem domain. In this technique the problem-solving approach is heuristic rather than algorithmic.

In the real world most problem-solving tasks involve uncertainty. For an example, it is very ambiguous to generate optimal meshes for an object, of which structural analysis is required to maintain enough strength. This uncertainty can be represented and resolved better in expert system programming than conventional programming techniques. In expert system development, knowledge is processed by an inference engine, while data are processed by looping in the conventional programming technique.

The second reason that knowledge engineers should develop expert systems actively is that expert systems can provide a means of codifying crucial knowledge in a certain field [FAM*86]. Knowledge limitations are often found in any organization. If the knowledge is absolutely necessary to achieve progress of the organization, it should be supplied to those groups within the organization that want to apply it in some way. In that case expert systems can be easily transported with affordable cost. The followings are a few examples to realize the importance of developing expert systems.

2.1 Successful Expert Systems

In the early days expert systems dealt with simple systems that had the heuristic skill to find goal states in playing games. However, recently many groups of researchers are trying to solve real-life problems with realistic models of reasoning rather than simply generalized problem solving.

DENDRAL began in 1965 to help solve the difficult problem of interpreting molecular structures from mass spectrographic information. Although algorithms existed to generate all possible molecular structures, the exhaustive search was extremely expensive. *DENDRAL* had the knowledge of expert chemists in rules who could search for satisfactory answers.

MYCIN was a successful expert system utilized for medical practice since the project began 1972. It gave advice on diagnosis and therapy for bacterial infectious diseases. Necessary medical knowledge was encoded in terms of production rules involving certainty factors, which helped doctors to accommodate probabilistic reasoning. It also gave the reasons for its decisions in terms of its rule. A rule acquisition system also allowed users to add new rules and change existing ones.

XCON is one of the most mature and successful expert system application since it configured a VAX-11/780 in 1980. Digital Equipment Corporation spent about 50 man-years to develop *XCON*. *XCON* saves DEC $18 million annually [Wil86]. It configures the VAX computer family system by checking that the order is complete and then determining the spatial arrangement of over 5000 different components. *XCON* uses a bottom-up approach that begins with knowledge about components like voltage, amperage, pin-type, and the number of ports, and tries to produce configuration within the constraints imposed by the properties of the components and relationships among them. New knowledge was added from time to time to handle a wider class of data and to introduce new subtasks. The maintenance of this system is relatively easier than that of conventional programming.

2.2 Knowledge Representation

To place experts' knowledge into an expert system, knowledge should be structured as natural by as possible. The most widely used schemes are semantic net, rule based representation, and frame based representation [Wat86].

2.2.1 Semantic NetIn the semantic net technique, knowledge is represented on a network structure of nodes and their links, called arcs. Objects, concepts, or events can be placed in nodes, while their relationships between nodes are defined in arcs. Although the semantic net scheme has advantages such as the ability to detect similarities in the meaning of sentences that are closely related but have different structures, it has the disadvantage of not showing the correct meaning of nodes. As an example, nodes labeled *edge* may have the meaning *class of all edges*, or *concept of edge*, or a *specified edge*. Thus it is realized that this scheme lacks a logical and heuristic adequacy [Jac86].

2.2.2 Rule Based SystemRules provide a formal way of representing recommendations, directions, or strategies using IF *condition* THEN *action* statement. When the IF portion(*premise*), of a rule is matched by the facts in a forward chaining system, the action specified by the THEN portion is fired so that inference chains are produced continuously. The firing of the rule may add new knowledge into the knowledge base. This technique has two distinctive chaining mechanisms, forward chaining and backward chaining. In forward chaining, the search for new information proceeds in the direction of THEN portion, i.e., the system uses the premise to derive the information of THEN portion. In a backward chaining system, the system only executes rules that are relevant to establishing a goal part. It tries to prove the goal. If it cannot prove the goal directly, it establishes other facts, subgoals, which will prove the original goal. Rule system also provides an environment that is easy to debug and maintain.

2.2.3 Frame Based SystemFrames are data structures for representing stereotyped situations, which are grouped together. Each frame has multiple slots that hold attributes, and each slot may have local procedures attached to it. Procedures are executed when changes happen in the slot. There are three types of procedures based on when they are executed. They are *If-Added*, *If-Deleted*, and *If-Needed* and are executed when new information is added into the slot, when existing information is deleted from the slot, and when the slot is accessed for new information, respectively. This is very useful and powerful for problem domains where expectations about slot value play an important role in problem solving.

2.3 *FROBS*, a Building Tool

Once a knowledge engineer identifies problems and has clear concepts about the problem with its control mechanisms, Waterman [Wat86] pointed out that the most difficult task is to select tools to build expert systems. It is common to encounter several types of difficulties when trying to build an expert system. Some of them are the scarcity of resources, the limits of artificial intelligence technology, and the length of time [Wat86]. The first two are related to the building tool. If a knowledge engineer could select appropriate tools as needed, it means he/she has overcome most of the problems. Thus, the selection of expert system building tool is very important. The tool must help the knowledge engineer to map key concepts into formal representation using one or more representation schemes discussed in previous section.

In developing *EFEM*, *FROBS* (FR+OBS) [MKK87,Mue87a,Mue88,Mue87b] was selected to represent the knowledge base. *FROBS* was developed by the Utah PASS (Portable Artificial Intelligence Support System) group at the Computer Science Department of University of Utah. It was designed to take the best of the frame base system (FR) and the object oriented system (OBS) and combine them into a Lisp environment. It is written in Hewlett Packard Common Lisp and Portable Common Lisp Subset (*PCLS*) [SL86] that was developed by the same group. *PCLS* covers the subset of Common Lisp embedded in Portable Standard Lisp (*PSL*), the Lisp language used in the designing the interface of *Alpha_1*, *Shape Editor*. Thus, selecting *FROBS* provides a complete Lisp environment that is known as the most suitable for expert systems. Meanwhile, it is consistent with the geometric modeler. *FROBS* supports *methods* and allows users to do *rule-based programming* inside a *frame system*. A *module* of *FROBS* will be the basic building block of the expert system. Each *module* contains class FROBS and *methods* of the class that can manipulate a class. This will be discussed in more detail with examples in the next chapter. *FROBS* has distinct differences in comparison with other object systems. First, *methods* of *FROBS* can be invoked by using a function call style. Second, *FROBS* allows context based multiple inheritances. Third, it supports the frame world by allowing multiple valued slots. Finally, it can hide information completely from other users by using *private slots* and *methods*. Furthermore *FROBS* can continuously update the knowledge base that is important for reasoning, since the system also provides a rule system.

3 Implementation

3.1 System Overview

The development of an expert system for the finite element analysis is closely related to a few other systems. The expert system has been built on the top of the *Alpha_1* geometric modeler [Alp88b,Alp88a] being developed by the *Alpha_1* group at the Department of Computer Science, University of Utah, and will interact with a separate finite element analysis package, to get optimal meshes simultaneously. As was already discussed in the previous section, *FROBS* has been selected as a tool to build *EFEM*.

Figure 1 shows the integrated feature-level expert system for the finite element analysis. The thick solid boxes represent the parts of *EFEM* that are not implemented in a conventional programming technique.

It is important to represent models precisely in order to get the best analysis results. Thus, the geometric modeler-aided mesh generator has the advantage of being able to represent problem domains completely and uniquely and to provide user friendly interaction. Because the *Alpha_1* geometric modeler represents models by the boundary representation method, models can be kept in an evaluated form so that the finite element analysis package may be applied to them directly. Since the *Alpha_1* modeler utilizes the tensor product nonuniform rational B-spline representation, every surface has four boundaries, which is compatible with the input format of the finite element analysis package used.

Although the *Alpha_1* modeler has some disadvantages compared to the CSG tree based geometric modeler, it can represent a wider variety of complex models from two-dimensional to three-dimensional in terms of surfaces, such as airplanes, automobiles, and various mechanical parts. It also provides a high quality graphical capability that is very useful for postprocesses, such as displaying the deformation of a shape or strain energy distribution field along with an original shape. Because the *Alpha_1* geometric modeler has all integrated capabilities from preprocesses to postprocesses, *EFEM* interacts with them and generates optimal finite element meshes self-adaptively. Therefore, *EFEM* does not need to ask users for tedious information to analyze, although several packages intermingle in a complicated way as shown in Figure 1.

ADINA (Automatic Dynamic Incremental Nonlinear Analysis) [ADI81], from *ADINA* Engineering, Incorporated, is a well-known commercial finite element analysis package. *ADINA* cooperates with the *Alpha_1* geometric modeler in that *ADINA* can generate better results with the input of quadrilateral meshes. It also allows users to specify different orders of shape function in the same element such that the element can have side nodal points, which unavoidably occur in the process of subdivision or refinement of surfaces by the geometric modeler. As was discussed earlier, the stress of most material is linearly proportional to the energy. *ADINA* supports quadrilateral meshes that provide good accuracy with reasonable cost to analyze objects. The quadrilateral elements are also generated by the *Alpha_1* geometric modeler. The usage of the *ADINA* package is completely covered by *EFEM* such that a user does not even realize when *ADINA* is invoked and it is not

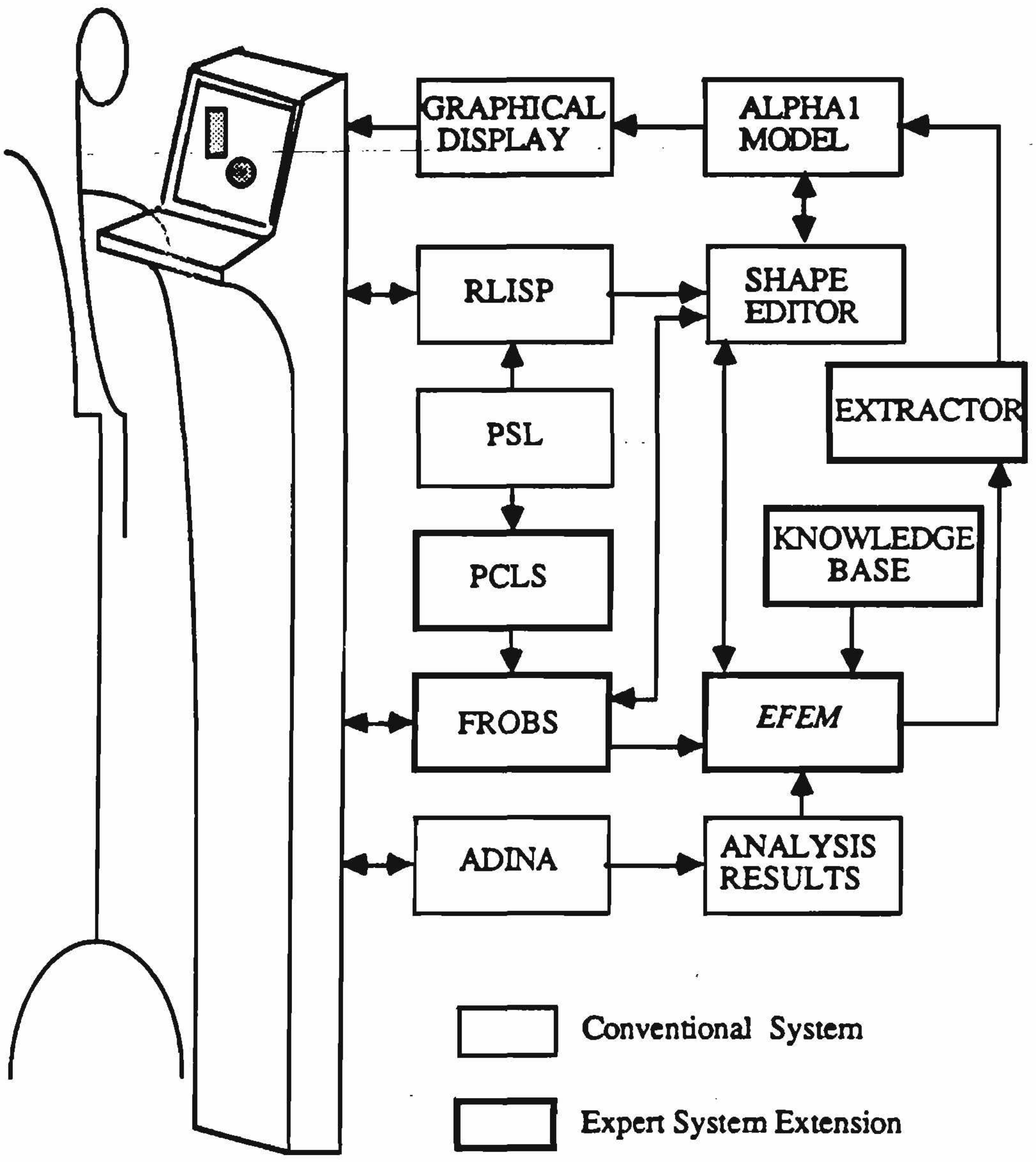


Figure 1: The Integrated Feature-Level Expert System

required for a user to know how to use the package.

The characteristics of the *FROBS*, expert system building tool, was briefly explained in the previous chapter. It was also discussed that *Shape Editor* and *FROBS* were built on the Portable Standard Lisp (*PSL*) and the Portable Common Lisp Subsets (PCLS), respectively. Because some symbol names of PCLS are conflicting with ones of *PSL*, all package symbols defined in the PSL package should be used with a package qualifier inside PCLS package, and vice versa. A user can access both the *Shape Editor* and the *FROBS* by calling them with their qualifier whenever it is necessary. More detail usage will be described in a later section.

3.2 User-friendliness

It is expected that two groups of people would use the *EFEM*. One group is made up of people who are familiar with finite element analysis but not with computer-aided geometry or expert systems. The other group is made up of people who are familiar with computer graphics or modeling objects but not with finite element analysis. An intelligent computer graphic hacker may not be knowledgeable in the area of finite element analysis. On the other hand, someone who knows finite element analysis may not be skilled in designing models.

The system rationale of *EFEM* discussed in the previous section shows that the working environment of *EFEM* is quite complex. Thus, one of the most important goals of *EFEM* is to provide a user-friendly system. The system should be easy enough to use for a technician who does not have the background of computer science or a computer scientist who is not skilled in the finite element analysis. That is, the information needed to use a good expert system should be as simple as possible.

Most early finite element analysis packages such as ADINA, NASTRAN [RHM72], and NON-SAP [Uni72] are difficult to use, although they have great power to solve various classes of problems. Some investigators [SDR88,WPWW87,Rud88,Ben88] have developed new techniques trying to reduce the difficulty in the input preparation of preprocessing for finite element models. However, there are still a number of problems with their systems.

Firstly, most packages have been designed based on menu driven interaction. However, the menus are four or five levels deep, making it very confusing for a novice to select the proper sequence of menu items. It is also easy to overlook incomplete attributes that must be defined for the analysis. The user realizes the problem when the analysis process encounters errors. Secondly, the mesh generation technique is a node-oriented method rather than an object-oriented method. For example, a move operator has to be used in order to model a dented square plate after a perfect square has been modeled. Obviously, it is desirable to extract nodes from a model in order to achieve better accuracy in speed. In this technique it is also necessary to label nodes or elements in order to specify referenced nodes or elements. To deal with labeling of nodes or elements is very tedious and error-prone when the object is big or complicated. Thirdly, it is very difficult to specify element sizes in case composite curves are used. In existing systems, if a boundary consists of multiple curves, the proper element size must be specified for each curve in order to build accurate boundary curve segments. It is a time consuming process to set an appropriate element number for every curve. Finally, extensive knowledge about the finite element analysis is required to generate optimal meshes. *SUPERTAB* [SDR88] utilizes the strain energy and deformation to solve this problem, but it is still necessary to modify nodes manually to generate reasonable optimal meshes.

Therefore, *EFEM* has been built such that a user does not need to specify difficult information. The primary inputs in *EFEM* are the geometric definition of an object model and analysis-dependent information. Once the analysis-dependent information has been specified, *EFEM* generates the optimal meshes by itself.

3.3 Internal Representation and Rules

EFEM reads a modeled object with a list format so that a user calls a main function **efem** to read geometric information such that `(efem geometric-type list-of-geometry)`. If the geometry entity is a point or a curve, `list-of-geometry` should be a nested list to accept four points or four curves forming a quadrilateral element. As an example, input for a thin I-shaped aluminum plate that will be discussed in a later section is shown as

```
    (efem '2d (list (list psl:P1 psl:P2 psl:P3 psl:P4)
                    (list psl:P3 psl:P4 psl:P5 psl:P6)))
```

Two regions of planar domain were specified by lists of points. Because the points were defined in PSL, they are referred to by a package name qualifier, `psl:`. Once *EFEM* recognizes the geometry, it stores the information into the basic building block of *FROBS*, a module that was mentioned in the previous section. A module consists of a class and all of its associated methods. *EFEM* has three distinct classes, *Geometry*, *Region*, and *Analysis*, based on its functionality.

Class Geometry copies geometric information from *Shape edit* and decides which boundary type is appropriate such as top, bottom, left, and right. *Class Region* has slots that are related with nongeometric information such as material properties, constraints, and loading information. Once it gets all required information, it creates new symbols for *PSL* packages. *Class Analysis* is one that controls top-loop level analysis sequence. It has slots about symmetric properties, criterion surfaces, and information about the next job state. Definition of *Class Geometry*, *Class Region*, and *Class Analysis* look like:

```
;; define class of geometry
(def-class geometry NIL
 :slots (type typeOk            ; type of geometry
         heading dname id        ; Adina related slots
         dim geometryLeng        ; geometry dimension
         pa pb pc pd interPt     ; vertices
         stop doneGeom           ; control slots
```

```
        edge1 edge2 typeEdge1 ; edges of region
        surface               ; region surface
        (convex T)            ; the region is convex initially
        (boundingBox NIL)     ; bounding box slot
        (interPtTrial 0)      ; intersection point
        (pmlist NIL)          ; vertex type based on location
        (checkId 0)           ; tag for recognizing four points
        (trial NIL)           ; # of the trial of finding edge
        (knownPts NIL)        ; list of known points
        (tempFinPts NIL)      ; temporary list of known points
        (consistOk 'Unknown)  ; tag for consistency of edges
        (finPts NIL)          ; list of final points
        (finBdy NIL)          ; list of final boundaries
        (dispBdy 'Unknown)))  ; tag for displaying option

;; define class of region.
(def-class region nil
 :slots (id           ; region id
        topBdy        ; boundary slots for top,
        bottomBdy     ;                   bottom,
        leftBdy       ;                   left, and
        rightBdy      ;                   right
        bdyOk         ; tag for checking the proper boundary list
        meshDensity   ; mesh density type
        tTobDim       ; dimension of boundary from top to bottom
        lTorDim       ; dimension of boundary from left to right
        topN          ; neighbor information of top boundary
        botN          ;                      of bottom boundary
        leftN         ;                      of left boundary
        rightN        ;                      of right boundary
        material      ; material identification
        young         ; material property, Young's module
        poisson       ;                   Poisson's ratio,
        thickness     ;                   thickness of material
        loadVec       ; loading information, a given loading vector
        cornerL       ;                     corner set of loading
        bdyL          ;                     boundary set of loading
        loading       ;                     final loading value
        topCons       ; displacement constraint, top constraint,
        bottomCons    ;                          bottom,
        leftCons      ;                          left, and
        rightCons     ;                          right
        tConsSet      ; given constraint set onto top boundary,
        bConsSet      ;                            bottom boundary,
        lConsSet      ;                            left boundary, and
        rConsSet      ;                            right boundary
        startAnal     ; tag for ready state of start analysis
        )
 :mv (regionL))       ; final region list

;; define class of analysis.
(def-class analysis nil
 :slots (genUniform      ; generate uniform mesh
        orgDomain        ; original domain
        unifMesh         ; uniform mesh
        symXaxis         ; symmetricP with respect to X axis
        symYaxis         ; symmetricP with respect to Y axis
        symZaxis         ; symmetricP with respect to Z axis
        dispUniform      ; display uniform mesh
        femInFile        ; fem input file name
        TEMPSLOT         ; temporary for editing input file
        adinaOutFile     ; adina output file
        evalUniform      ; evaluate uniform
        highest          ; the highest strain energy from evaluation
        dispScale        ; display scale
        drawDeform       ; name of deformation file
        critScale        ; scaling factor for drawing criterion
        buildCritSrf     ; build a criterion surface
        dispCritSrf      ; tag for displaying criterion srf
        workDomain       ; domain currently working with
        subdivCrit       ; subdivision critical value for optimal
        newDomain        ; domain for nearoptimal mesh
        femInFileNO      ; fem input file for near optimal
        TEMPSLOT2        ; temporary
        adinaOutFileNO   ; adina output file name for optimal
        evalNO           ; tag for evaluation of optimal mesh
        noMesh           ; name of near optimal mesh
        nextJob          ; next job state
        rerun            ; rerun for a new loading vector
        render           ; render option
        renderObj        ; a rendering object
        tryAnother       ; generate another near optimal mesh
        ))
```

Defining a class does not build an actual instance of the class. An actual instance is created by the method of the *Class Geometry* that looks like:

```
;; make an instance of point or curve related geometry class.
(def-method ({class geometry} formGeomPtCrv)
 (name type heading dname dim geometryLeng pa pb pc pd)
 (let ((Instance (new-instance* $self)))
  (setf (type Instance) type)
  (setf (heading Instance) heading)
  (setf (dname Instance) dname)
  (setf (dim Instance) dim)
  (setf (id Instance) name)
  (setf (geometryLeng Instance) geometryLeng)
  (setf (pa Instance) pa)
  (setf (pb Instance) pb)
  (setf (pc Instance) pc)
  (setf (pd Instance) pd)
  Instance))

;; make an instance of surface related geometry class.
(def-method ({class geometry} formGeomSrf)
 (name type heading dname dim geometryLeng)
 (let ((Instance (new-instance* $self)))
  (setf (type Instance) type)
  (setf (heading Instance) heading)
  (setf (dname Instance) dname)
  (setf (dim Instance) dim)
  (setf (geometryLeng Instance) geometryLeng)
  (setf (doneGeom Instance) 'Unknown)
  (setf (id Instance) name)
  Instance))

;; other methods related with geometry class.
(def-method ({class geometry} incCheckId)()
 (assert-val $self 'checkld (1+ (checkId $self))))

(def-method ({class geometry} copyKnown)(Order)
 (assert-val $self 'knownPts (nth Order **FinPtsList)))

(def-method ({class geometry} getSharedPts)(Pa Pb Pc Pd KnownPList)
 (assert-val $self 'edgel (memberPt Pa Pb Pc Pd KnownPList)))
```

```
;;; Rule for generating uniform meshes for two dimension.
(def-rule generateUniformMeshFor2D
 :type ((?region region)(?analysis analysis)(?geometry geometry))
 :local ()
 ;; if geometry dimesion is 2D, analysis is begun, and uniform mesh
 ;;    is not generated yet,
 :prem ((dim ?geometry 2d)
        (startAnal ?region yes)
        (genUniform ?analysis notYet))
 ;; then generate uniform mesh by calling gen2DUniformMesh and fill
 ;;    the slot such that the uniform mesh is generated.
 :conc (atomic
        (printRule "Generating uniform mesh for 2D..~%")
        (gen2DUniformMesh (regionL (regClass 0)))
        (assert-val ?analysis 'genUniform 'Done)))
```

Figure 2: A Sample Rule Generates Uniform Meshes

One of most powerful feature of *FROBS* is that *FROBS* provides its own forward chaining rule system built in. Since forward chaining rules are data driven, rules that satisfy current knowledge base can be triggered and trigger another rule simultaneously. Thus, *EFEM* completes the whole analysis cycle by one assertion of a slot value. The rules of *FROBS* consist mainly of *premise* and *conclusion* such as the general rule system, which was discussed in the knowledge representation section, Figure 2 shows the rules that generate the uniform mesh for two-dimensional objects. Its premise checks three conditions: if the dimension type is two-dimensional; if the region is created properly, and if the uniform mesh is not ever created. If all three conditions are satisfied, the rule invokes a function that generates uniform mesh and updates the knowledge base as the uniform mesh is created.

3.4 Control Flow of *EFEM*

Since a rule is translation of logical thought, it is important to the logical sequence of *EFEM* to write correct rules. The primary control flow of *EFEM* is extracting boundaries from the designed model, building regions with the boundaries, setting regional dimension and adjacency information, generating and analyzing the uniform mesh, and generating optimal meshes. After executing one cycle of these stages, *EFEM* may regenerate the next optimal meshes as the user wants. The sequence of control flow is shown Figure 3 and the main functionality of each part will be explained later. *EFEM* may generate optimal meshes with different critical values of subdivision of criterion surface or restart another analysis with new loading information.

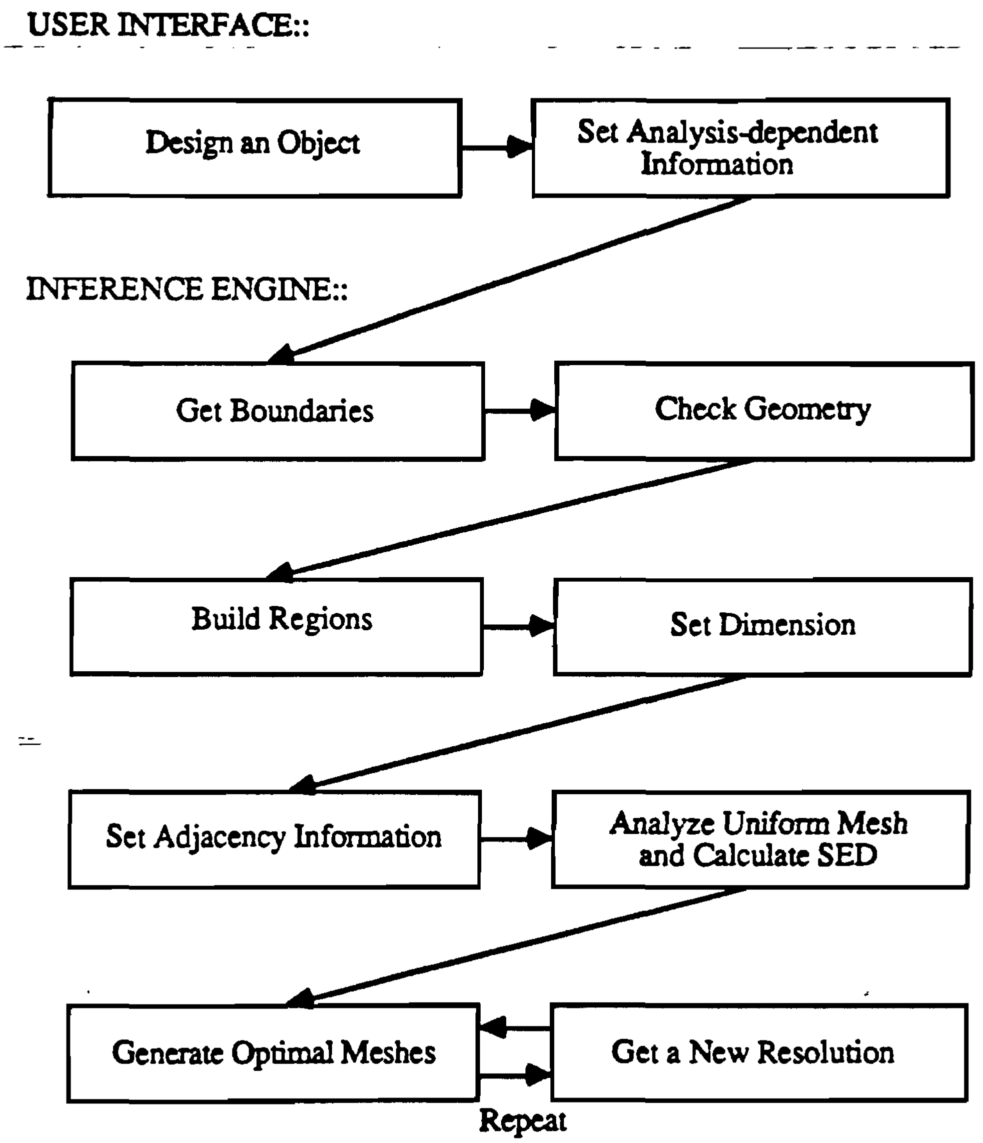


Figure 3: The Process Sequence of *EFEM*

3.4.1 Checking Input GeometrySince the *Alpha_1* modeler utilizes the tensor product nonuniform rational B-spline representation, every surface must have four boundaries in order to be compatible with the input format. It is possible that a user may input erroneous geometry. *EFEM* checks the validity of input geometry before it starts the analysis process to prevent wasting computer time. Although most difficulties of executing *EFEM* have been avoided, there are still two potential problems for a user because the input geometry must be a satisfactory format for both *ADINA* and *Alpha_1* that utilizes the tensor product nonuniform rational B-spline representation: a region must consist of four vertices to form a quadrilateral region and the quadrilateral region must be convex. The former error can be detected by counting the number of vertices and this portion only is checked in shell analysis cases because it is assumed that the shell surface has been designed in an appropriate way. In order to determine the type of polygon such as convex or concave, the bounding box of the input geometry must first be calculated by a rule in following pseudo code:

```
IF   there is any class geometry
     and the type of the geometry is convex
     and the bounding box of the geometry is not calculated
THEN calculate the bounding box by calling a function, boundingBox
     and fill the slot of a bounding box
```

The bounding box of a region is useful to get a diagonal intersection point of the region and to find boundary attribute types. The rule for calculating diagonal intersection point is triggered as soon as the bounding box slot of a region is defined and gets the intersection point of the diagonals of the bounding box. The pseudo code of the rule looks like:

```
IF   there is any class geometry
     and the type of the geometry is convex
     and the bounding box of the geometry is defined
     and the diagonal intersection point is not calculated
THEN calculate the diagonal intersection point by calling a
       function getInterPt
     and fill the slot of the diagonal intersection point
```

The two rules discussed above control the top-loop sequence, while the actual process such as calculation of a bounding box or an intersection point are done by Lisp functions.

After the diagonal intersection is calculated, four vertices of input geometry are compared with the point and a plus-minus list (`pmList`) is formed based on the relative locations of four vertices that represent top left, top right, bottom left, and bottom right. For example, if the X and Y coordinates of a vertex are greater than the ones of a diagonal intersection point, the `pmList` becomes plus-plus and the vertex should be a top right point. A rule `checkConvex` simply checks the plus-minus slot of a region. If a `pmList` contains only unique combinations, it means that four vertices are distributed in the right positions. However, if the pmList contains the same combinations, it means that the polygon is concave because the diagonal intersection point is not inside the bounding box. If the boundaries of the first region are set properly, the other boundaries of later regions can be easily arranged. When input geometry is composed of the arbitrary number of multiple regions, it is necessary to define a rule that triggers *EFEM* to scan the next regions until all regions have final vertex lists that are consistent with each other. The rule in pseudo code looks like:

```
IF   there is more than one class of geometry
     and the final edges of the first geometry are defined
     and the bounding box of the next geometry is not defined
THEN continue to define boundaries of the next geometry
```

3.4.2 Getting Boundaries for RegionsIf geometry has been input correctly, boundaries are extracted from the defined object. Each boundary will have a unique identification number along with geometric information. If an object is defined by points, a boundary will be defined for every two points after a curve is created. If the line class entity has been utilized for the defined object, a boundary list will be constructed directly from curves. However, a line and an arc entity will generate a curve. If a surface defines an object, the four boundaries will be extracted from the surface directly.

Each boundary has its own type, such as top, bottom, left, and right. The orientation of each boundary must be consistent throughout the whole geometric definition. One of the important tasks of *EFEM* is to decide an arbitrary consistent boundary set because it is difficult to find the consistent boundary type in complex geometry. Since the four vertices of each region are listed in order in the slot of a final point list during the stage of checking geometry, the boundary can be easily extracted from the final points such that the first two points, the last two points, the first and the third point, and the second and fourth point decide top, bottom, left, and right boundaries, respectively. The boundary extracting rules can be coded as follows:

```
IF   there is a class geometry
     and every point is arranged correctly
     and the boundary of the geometry is not built yet
THEN build boundary list by calling a function, getBdyFromObj
     and fill the slot of a final boundary
```

Finally, a simple rule can show a user the boundary configuration drawing if all boundaries are set correctly, such as:

```
IF   there is a class geometry
     and the boundary of the geometry is defined
     and boundaries are not displayed yet
THEN display boundaries by calling a function, dispAllBoundaries
     and fill the slot of displayed boundary with 'done'
```

When a user needs to specify information that is related to boundaries such as loading vectors and displacement constraints, the drawing can help a user to easily set the correct information.

In the prototype implementation it was assumed that an object was a list of convex polygons. Once every boundary is set properly, the creation method of *Class Region* builds an instance of every region with the boundary.

3.4.3 Setting a Dimension for Each BoundaryThe dimension is the number of elements in the row and column direction of the uniform meshes in the analysis. Since the optimal meshes will eventually be generated from the initial uniform mesh, the size of uniform meshes is not a main issue here. However, it is desirable to set dimensions that form square-shaped elements as much as possible, because the strain energy distribution is calculated based on the initial uniform meshes and *EFEM* uses the SED for generation of optimal meshes. Although the second optimal meshes can be generated in the next cycle, the criterion surface with initial SED will be used again. Thus, the knowledge bases that tweak the region shape have been designed such that the size will be chosen that makes element shapes close to squares. This allows a user to get reasonable variation of SED field results with initial uniform meshes.

For an example, if the given geometry consists of three regions shown in (a) of Figure 4, the single dimension, 4 × 4 will create uniform meshes shown in (b) of the same figure. Because *EFEM* checks the given geometry first, it will generate multiple dimension size, such as 8 × 4, 4 × 4 and 4 × 12 for three different regions, and the resulting meshes are shown in (c) of the same figure.

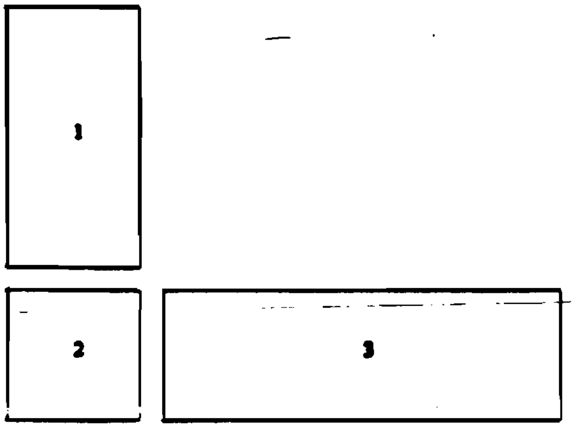
FROBS provides the **known** clause in rule premises so that a rule can fill slots interactively. A mesh density can be selected interactively by a rule using the **known** clause in the following pseudo code:

```
IF   there is a class region
     and the boundaries are defined correctly
     and a mesh density is selected in interactive mode
THEN fill the slot of a mesh density
```

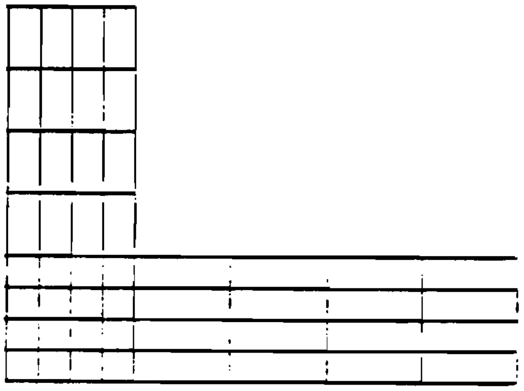
To avoid difficulty of setting appropriate dimensions, three different mesh densities, coarse, normal, and dense, are provided and the number of each density can be changed by a user. In some cases, it is necessary to set it to an irregular size. Therefore, *EFEM* also allows a user to set to arbitrary dimensions. After mesh density is set to appropriate dimension sizes, all region lists can be filled as follows:

```
IF   there is a class geometry and a class region
     and the bounding box of the geometry is defined
     and mesh density of the region is defined
THEN calculate mesh size by calling a function, getDim
     and fill the slot of dimensions of height and width
```

(a)



(b)



(c)

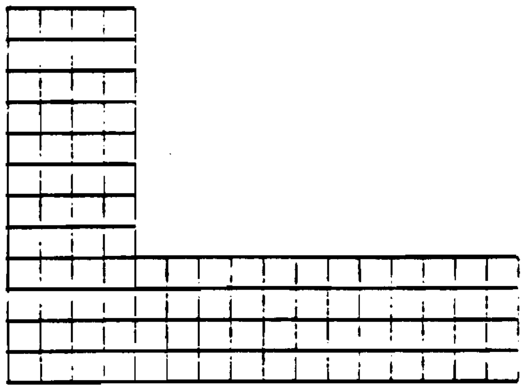


Figure 4: A Case that Needs Multiple Dimension Sizes (a) A Given Problem Domain (b) Meshes by a Single Dimension (c) Meshes by Multiple Dimension

3.4.4 Setting Adjacency InformationIn this level, *EFEM* sets adjacency relationships into the slots of region attributes. This can be done by rules that scan every region sequentially in order to find adjacent boundaries. For example, a rule that finds top neighbors can be coded based on the following pseudo code:

```
IF   there is a class geometry and a class region
     and the region is the first one
     and the top neighbor of the region is not defined
THEN find top neighbors by calling a function getTopNeighbor
     and fill the slot of a top neighbor
```

For every boundary, if there is a shared boundary edge, *EFEM* will insert the region identification into its neighbor slot. At this level, slots of each region related to analysis dependent information and material are furnished by user's input triggering rules that have known clauses. For example, the Young's module or Poisson's ratio can be specified by a rule that may look like:

```
IF   there is a class region
     and material name is selected in interactive mode
THEN fill the slots of a material name, Young's module,
       and Poisson's ratio
```

3.4.5 Analyzing Uniform MeshesUniform meshes are generated with the information of boundary and mesh size described in the above steps. The uniform mesh generation rule may look like:

```
IF   there is a class geometry, a class region, and
       a class analysis
     and analysis is begun
     and uniform mesh is not defined yet
THEN generate uniform meshes by calling a function, genUniformMesh
     and fill the slot of uniform meshes
```

The rules actually call *PSL* functions like most other rules in order to subdivide a surface uniformly. The uniform meshes may or may not be optimized initially. At this stage, simple rules are triggered to extract nodal data of the uniform meshes generated and invoke the finite element analysis package, *ADINA* automatically. The rules can be defined by the following pseudo codes:

```
IF   there is a class analysis and a class geometry
     and uniform mesh is generated
     and analysis input file is not extracted
THEN write nodal information into a file by calling a
```

```
     function, writeFemIn
     and fill the slot of an input file name

IF   there is a class analysis
     and nodal data are extracted for an analysis
THEN call an analysis package
     and fill the slot of an output file name
```

EFEM will evaluate the analysis result in terms of number of elements, number of nodes, total degree of freedom, SED, and maximal stress value for each axis. This interpretation may be compared with the analysis result of optimal meshes later in order to see the efficiency of the optimal meshes. The evaluation rule can be coded shown in the following pseudo code:

```
IF   there is a class analysis
     and the evaluation of mesh is not done yet
THEN evaluate mesh by calling a function, execFemEval
     and fill the slot of an evaluation tag
```

In this step, the distorted shape is also drawn in *Alpha_1* data format so that the uniform mesh can be visualized with the distorted shape.

3.4.6 Generating Optimal MeshesThe main philosopy about generating optimal meshes is based on the strain energy distribution (*SED*) [SGA80,YFRC87]. A uniform and coarse mesh is synthesized with the variation of the *SED* used to generate the optimal meshes. In this approach, *SED* or displacement will be important criteria in the placement of key nodal points, although *SED* has been used here. A criterion surface can be built by adding SED into the original geometry with a rule as follows:

```
IF   there is a class analysis
     and the problem domain of analysis is defined
     and next job is not 'quit'
THEN build a criterion surface by calling buildCriterion
     and fill the slot of a criterion surface
```

Domain geometry will be subdivided and refined based on the variation of *SED* in this level. A user may need to input the criterion value for subdivision. If the variation of the *SED* is larger than the subdivision criterion value, *EFEM* will continue to subdivide the problem domain until the SED is less than the subdivision critical value. New domains with optimal nodes are then generated for two-dimensional and shell objects. The rule invoking an *Alpha_1* function can be coded from:

```
IF   there is a class analysis and a class geometry
```

```
      and the problem domain of the analysis is defined
      and the new domain is not defined yet
THEN  generate the new domain by calling a function, getNewDomain
      and fill the slot of the new domain
```

This step leads *EFEM* to generate optimal meshes. In this stage, a user may control the criterion value to get better results. The synthesized optimal meshes are displayed and analyzed for the efficiency of the mesh as was described earlier regarding uniform mesh for every iteration. At this level an analysis cycle is completed, and other analyses can be performed to generate new optimal meshes or to test a new loading vector, if it is necessary. Any change of criterion values or loading vectors cause *EFEM* to trigger some rules that control the execution such as changing loading vectors, changing a subdivision critical value, and rerunning for the new loads. These rules simply update the related slots so that rules that are affected by the slots are triggered. For example, the pseudo code of the rerunning for the new loads can be:

```
IF    there is a class analysis and a class region
      and the rerun slot of analysis is specified by a
        new load option
THEN  ask a new load to a user
      and fill the slot of a load
```

4 Conclusion

It has been pointed that the problem of conventional finite element analysis package is still a time-consuming and error-prone step. Furthermore, it requires that a user have background knowledge of both computer-aided geometry and finite element analysis.

In this paper rule-based system for finite element analysis (*EFEM*) has been developed to overcome the difficulties of structural analysis processes and to spread the finite element analysis expertise. *EFEM* has been implemented as a hybrid of an interactive geometric modeler, a finite element analysis system, and an object oriented frame language. In order to avoid the possibility of user's error, every top-loop control sequence is translated into a rule base that consists of over 80 rules. The expert's knowledge of the finite element is implemented as three different kinds of classes: a geometry class, a region class, and an analysis class based on class characteristics.

4.1 Methodology

As the forward chaining rules were triggered by changes of related slot values, the triggering of the first rule updates the knowledge base. It triggers other rules as well. Most rules are triggered once in every cycle of analysis, since the rules define the control of sequential top-loop. The average total number of triggered rules for most analysis is 75. The rule system provided an environment that made it easy to maintain the program, to debug the problem, and to change tasks, if necessary. To analyze an object, a user only needs to load geometric information into *EFEM*. Then *EFEM* asks object-dependent information such as loading vectors and displacement constraints and generates optimal meshes self-adaptably, while the rules of *EFEM* handle the correct analysis sequences.

Meanwhile, the control of the top-loop is easily governed by the rule based system; the algorithmic approach has a great advantage in solving some low-level processes such as calculating a bounding box, subdividing a B-spline surface, or drawing a uniform mesh. Thus, the algorithmic approach to generate optimal meshes was adapted to handle low-level processes in this system. A great number of rules has been reduced because most low-level processes were handled by Lisp functions.

The first analysis with uniform mesh calculates deformation and generates a criterion surface that is the result of synthesizing the strain energy distribution (SED) with the problem in domain geometry, such that the strain energy value takes place in the fourth component of the geometry field. This criterion surface is recursively subdivided into four subsurfaces until the SED over subsurfaces is less than the specified critical value. The optimal mesh was constructed for the final state of subdivided criterion surfaces.

4.2 Comparison to Existing Systems

It was tedious and error prone to use early finite element analysis packages because of their poor interfaces. Recently some packages such as *NAVGRAPH* [Ben88] and *SUPERTAB* [SDR88] tried to solve the difficulties of early systems. However, their interfaces have disadvantages such as deeply nested menu items or a node-oriented mesh generation approach rather than an object-oriented one. While other systems require extensive knowledge to generate optimal meshes, *EFEM* can handle many tedious steps easily. However, most general purpose analysis packages have a wider selection of problem domains such as beam, truss, thick shell and fluid flow besides the two-dimensional plate and shell, while *EFEM* supports the latter domains only.

4.3 Future Work

EFEM shows that the rule based system approach to the finite element analysis, the preprocessor especially, is very appropriate and helpful to many users. Considering the performance of *EFEM*, however, it would be good to improve the system in terms of speed. Although the computer time ratio of *EFEM* and *ADINA* is not high, the computer time of *EFEM* could be reduced more by rearranging nodal points by finding the minimal bandwidth. *EFEM* would be more powerful if the problem domain was larger as mentioned in a previous section. The beam or truss structure is often found in industry. The expert system approach introduced in this thesis may be also applied to other nonstructural areas such as heat transfer or fluid flow problems. Another possible improvement of this system for future work is to build the capability to combine uniform meshes and optimal meshes, if necessary. If the maximum strain energy does not occur where the boundary curve is a higher order than the linear, the optimal mesh has poor accuracy in boundary approximation. This occurred in the example of a spoon analysis. Although very dense meshes were formed in the joint areas of the bowl and the handle, very coarse meshes were also formed inside the bowl so that the boundary approximation was not good. Other future work is to apply the same technique to the volumetric analysis.

4.4 Summary

This approach demonstrates that an expert system can be applied to generate optimal mesh in an efficient and intelligent way. Since analysis through an expert system does not require technical knowledge about the finite element analysis process, the implemented rule base generates optimal mesh with few interactions. After the first optimal mesh has been generated, other optimal mesh is easily generated by changing the critical value of the subdivision. Furthermore, the rule system demonstrated that the different analyses on the same object can be done easily by controlling rules. Considering the difficulties of existing systems, it is expected that the knowledge based system would contribute substantial benefits to industrial areas.

References

[ADI81] ADINA Engineering. *ADINA User's Manual.* ADINA Engineering, Inc., Sept. 1981.

[Alp88a] Alpha_1 Group. *Alpha_1 System Manual.* University of Utah, January 1988.

[Alp88b] Alpha_1 Group. *Alpha_1 User's Manual.* University of Utah, January 1988.

[Ben88] Benzley. *NAVGRAPH User's Manual.* Brigham Young University, Provo, August 1988.

[FAM*86] M. Freiling, J. Alexander, S. Messick, S. Rehfuss, and S. Shulman. Starting a Knowledge Engineering Project: A Step-by-Step Approach. *The AI Magazine*, 6(3):150–164, 1986.

[FWL83] Hayes-Roth Frederick, D. A. Waterman, and D. B. Lenat. *Building Expert Systems.* Addison-Wesley Publishing Company, Inc., Wokingham, England, 1983.

[Jac86] Peter Jackson. *Introduction to Expert Systems.* Addison-Wesley Publishing Company, Inc., Wokingham, England, 1986.

[MKK87] Eric Muehle, Robert Kessler, and Jed Krohnfeldt. *Efficient Structures for Knowledge-based Applications.* Technical Report, University of Utah, Sept. 1987.

[Mue87a] Eric Muehle. *FROBS: A Merger of Two Knowledge Representation Paradigms.* Master's thesis, University of Utah, December 1987.

[Mue87b] Eric Muehle. *FROBS Manual.* University of Utah, February 1987. Utah PASS Project OpNote 87-09.

[Mue88] Eric Muehle. *FROBS User Guide.* University of Utah, March 1988. Utah PASS Project OpNote 87-05.

[Pre87] Roger S. Pressman. *Software Engineering - A Practitioner's Approach.* McGraw-Hill Book Company, New York New York, 1987.

[RHM72] Ed. R. H. MacNeal. *The NASTRAN Theoretical Manual.* National Aeronautical and Space Administration, April 1972.

[Rud88] B W Rudd. Impacting the Design Process Using Solid Modelling and Automated Finite Element Mesh Generation. *Computer-Aided Design*, 20(4):212–216, May 1988.

[SDR88] SDRC. *I-DEAS Supertab, Engineering Analysis User's Manual.* Structural Dynamics Research Corporation, 1988.

[SGA80] Mark S. Shephard, Richard H. Gallagher, and John F. Abel. The Synthesis of Near-Optimum Finite Element Meshes with Interactive Computer Graphics. *International Journal for Numerical Methods in Engineering*, 15:1021–1039, 1980.

[SL86] S. Shebs and S. Loosemore. *Portable Common Lisp Subset User's Guide.* University of Utah, May 1986.

[Uni72] University of California. *NONSAP - A Structural Analysis Program for Static and Dynamic Response of Non-linear Systems.* Technical Report, University of California, Berkley, March 1972.

[Wat86] Donald A. Waterman. *A Guide to Expert Systems.* Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, 1986.

[Wil86] C. Williams. Expert Systems, Knowledge Engineering, and AI Tools - An Overview. *IEEE Expert*, 1(4):66–70, 1986.

[WPWW87] P. Ward, D. Patel, A. Wakeling, and R. Weeks. Application of Structural Optimization Using Finite Elements. *Computer-Aided Design*, 19(3):148–156, April 1987.

[YFRC87] Wu-chien J. Yen, Russell D. Fish, Richard F. Riesenfeld, and Elaine Cohen. *An Algorithmic Approach Toward Near-Optimum Finite Element Mesh Generation*. Technical Report, Dept. of Computer Science, Univ. of Utah, June 1987. (27 Pages).