Mechanical design rationalization using function description language

Lai, Kewei, Ph.D.

Northwestern University, 1988

NORTHWESTERN UNIVERSITY

MECHANICAL DESIGN RATIONALIZATION USING FUNCTION DESCRIPTION LANGUAGE

A DISSERTATION

SUBMITTED TO GRADUATE SCHOOL
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE

DOCTOR OF PHILOSOPHY

FIELD OF MECHANICAL ENGINEERING

by

KEWEI LAI

Evanston, Illinois

JUNE 1988

ABSTRACT

MECHANICAL DESIGN RATIONALIZATION USING FUNCTION DESCRIPTION LANGUAGE

Kewei Lai

In recognition of the clear need for improving the competitive position of U.S. industry and the availability of ever more sophisticated computer tools, research in the new engineering discipline termed design theory and methodolgy has been strongly promoted. In this dissertation the author presents a new design methodology for mechanical design, Decomposition-Description-Rationalization. This has been developed to help implement a new design philosophy, Design for Manufacturing. An automatic design system, Function Rationalization System (FRS) and A high-level design language, Function Description Language (FDL), have been written using the methodology. Mechanical design can be characterized in a way that closely resembles human cognitive processes. A user can develop product layouts which are both functionally optimal and economically manufacturable by consulting FRS at the conceptual design stage.

[illegible]
3/5/88.

AKNOWLEDGEMENTS

The author would like to extend his sincerest gratitude to his advisor, Professor William R. D. Wilson. for his constant encouragement and financial support which led to this dissertation. The author would like to thank Professor G. K. Krulee and Professor W. E. Schmitendorf for their outstanding contributions to this work. The author would also like to thank the Center For Manufacturing Engineering at Northwestern University for the use of its computer facilities.

The author would like to thank his parents for all the love and encouragement they gave to him through his academic years.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1 INTRODUCTION

Engineering design is a sequence of activities using a systematic methodology to synthesize something new or to arrange existing things in a new way to satisfy a recognized need of society [1]. Design usually starts with design goals, such as functional requirements, performance, cost, delivery time, etc., and proceeds from a rough conceptual design to a fully detailed final design for release to manufacturing. In the conceptual design, the fundamental operating principles are developed, alternative systems that could satisfy the specified need are explored and the layout of products is established, systems are partitioned into parts, sub-assemblies and assemblies, and materials from which to manufacture parts are selected. In the final design, the details of the system and components are specified, such as, kinematics, dimensions and tolerancing, positions, etc. The design produces a outcome, which is evaluated to decide whether it is adequate to meet the needs. If the evaluation uncovers deficiencies, then the design operation must be repeated. The information from the first design is fed back as input, together with new information that has been developed as a result of questions raised at the evaluation step. The iterative nature of design provides an opportunity to improve the design on the basis of a preceding outcome, which in turn leads to the search for the best possible outcome [1-5].

The manufacturing world wide is going through a major upheaval. The advent of inexpensive, high speed computers has provided extensive support for manufacturing, such as numerical control, computer numerical control; flexible manufacturing and computer integrated manufacturing systems; computer-automated process control; computer-automated process planning; robots for parts handling, welding, and spray painting; automatic material handling, etc. However, in almost all industries, product design is still carried on nearly independently of manufacturing. There is a great need for design methods which take advantage of the versatility of new manufacturing techonology, especially flexible manufacturing and flexible assembly systems [6].

In recognition of this emerging demand for new design strategies and the availability of ever-improving computer technology, there is an increasing interest in the new engineering discipline, Design Theory and Methodology. Design theory refers to systematic statements of principles and relationships which explain the design process and provide a useful methodology for design. Design methodology is the collection of procedures, tools and techniques which the designer may use in applying design theory to the process of design. Two sub-areas are considered central to this research, (1) Conceptual design and innovation and (2) Quantitative and systematic methods for design. In addition, three supporting disciplines and methodologies are considered critical to the future growth of the design field: (1) Intelligent and knowledge-based systems; (2) Information integration and management; and (3) Human interface aspects in design [6].

There are now extensive computer software tools available for detailed design. Computer graphics systems are becoming dominant tools for drafting and representation of geometry, components, and assemblies. Advanced computer analysis, simulation and optimization tools are also in wide use. However, the application of the computer in conceptual design is still quite limited. Conceptual design is a complex cognitive process by which a market need is transformed into a well-formed set of design specifications and functional structures are established. The cognitive process involved is still far beyond our understanding. There is no coherent body of principles and methods to guide this design process. Very few computer tools are available for the special needs of this stage, such as the establishment of the general configuration of a product [7-11].

Current research in the area of intelligent and knowledge-based systems, i.e. Artificial Intelligence (AI) has provided useful tools for mechanical design as related to manufacturing. One of the motivations for applying AI to mechanical design is to bring manufacturing knowledge into consideration early in the design process. Most importantly, research is being directed toward modeling the design process and developing theory and tools, such as expert systems, for specific designs or design related tasks.

Expert systems are just beginning to make their way into mechanical design. To date, the tools available only cover a small part of the intellectual tasks and problem-solving methods used in mechanical design: those that have a very limited choice of solutions, are easily broken down into independent sub-solutions, and have reliable data available for the knowledge base. Future development and integration of these technologies are still dependent on the understanding of the methodology of the mechanical design process.

In this dissertation research in the theory and methodology for the conceptual design of mechanical products is presented. Progress has been made in the following aspects, (1) a new design methodology, Design Rationalization, has been developed, (2) A high-level design language, Function Description Language (FDL), has been developed to allow the methodology to be carried out on computers, and (3) A trial version of the design system, Function Rationalization System (FRS), has been built up to test the utility of the new methodology and the FDL design language.

Except for a few innovative products, most mechanical designs in industry today result from an evolution process. New designs are usually based upon the 'best' features of previous designs. However, there are often cases where the structures taken from a previous design may have functions which are not needed in the new design. Even essential functions may not be properly assigned to the components. If the resulting redundancies are removed, and the remaining functions are re-distributed properly, the structure of a product will be simpler and much better organized. Therefore, a good design can be achieved by investigating the functions of the parts in an initial design in a systematical way. Based on this observation, a new design method, called 'Decomposition-Description-Rationalization', or Design Rationalization, has been developed.

Traditionally, conceptual design can only be done by human designers, because most of the information involved in this stage lacks quantitative expression. The automation of this activity requires the development of vocabularies and reasoning methods such that the abstract design concepts, specifications, and design steps can be represented in an operational form and be manipulated by computers as a counterpart of the human cognitive process. The use of a design language as a new tool for automatic conceptual design process has been explored. The research has resulted in a high-level design language, Function Description Language (FDL).

These two aspects have been integrated into an automatic design system, Function Rationalization System (FRS). The prototype of FRS is written in the C language and runs on a DEC VAX-11/785 with VMS operating system.

The organization of this dissertation is as follows. This chapter addresses the emerging needs due to the progress on manufacturing technologies for new design methods, especially the methods of carrying out conceptual designs on computers. It highlights the author's work in the area of modeling and automating the conceptual design of mechanical products. Chapter 2 presents a survey on the new design philosophy, Design for Manufacturing, the computer tools for conducting conceptual design, and the Expert Database technologies, which are used to develop the FRS system. Chapter 3 introduces our research work. The architecture of the Function Rationalization System (FRS) is presented. The rationale for developing Decomposition-Description-Rationalization and Function Description Language (FDL) is discussed. Chapter 4 introduces the features of the FRS database. Several important improvements are made to the relational data base model using Expert Database techniques. Chapter 5 discusses the design library and most important principles of design rationalization. In Chapter 6 an application example of the Design Rationalization is presented. Chapter 7 summarizes our work and proposes future research directions.

CHAPTER 2 CURRENT PROGRESS OF MECHANICAL DESIGN

2.1 A NEW DESIGN PHILOSOPHY - DESIGN FOR MANUFACTURING

With the rapidly growing number of products and product variations, shorter product life cycles and increased competition from foreign industries, computer aided manufacturing technologies, such as numerical control, robotics, and flexible automation in general, have been brought into wide use in industry today [12-15]. However, such efforts are often not as successful as expected, and it has been realized that mechanical design and manufacturing must be treated as a global strategy, involving both the product itself and the production systems. Greater attention must be paid to the design itself, since it is here that manufacturing costs are largely determined.

Traditionally, the designer does the design and then the manufacturing engineer tries to plan processes to fit the design. However, it is a common experience that a part can be much more easily made if it is slightly redesigned to allow the use of a different process, different tooling, or different materials [16,17]. Such changes become more difficult to implement as design proceeds. Therefore, a design should be evaluated for manufacturability at an early stage of design.

Assembly is usually the single most important process contributing to manufacturing costs. When productivity improvements are sought, one must determine whether the design of a product lends itself to assembly operations. This idea is due to Boothroyd et al and is well known as 'Design for Assembly' [18]. Boothroyd's approach for the first time provides a systematic way for testing the assemblability of a product design prior to its release to manufacturing. As a result, the number of parts in a product can be reduced and the ease of assembling the remaining parts can be increased [19-28].

There are many more factors determining production costs, such as materials, processing sequences, machine tools, and development of new processes. For example, forging can be replaced by heavy stamping and casting; stamping can be replaced by die-casting or plastic molding; cold forming parts may eliminate the need for machining operations. Various materials and processes compete with each other for the most economy in a product. To take the advantages of the new possibilities, methods of selecting materials and processes are needed. For the greatest saving, the process and material selection activity should occur prior to the time that final part drawings are generated. The material and process selection procedure should start immediately after initial product design sketches are available. For each known process, analysis will be made to determine whether it can be used to make the entire part. The material handling ability, the part shape and size, and batch quantity are major criteria for evaluating processes. Often one of these factors will completely eliminate a process from further consideration. Once preliminary planning has been made, the operation routines are prepared for the remaining processes. Final decisions can be made based upon the cost estimation of each combination of processes. This idea can be summarized as a simple philosophy, 'Design For Manufacturing'. Through eliminating difficult-to-make structures at early conceptual design stage, the product can be economically manufacturable.

Developing the methodology of 'Design For Manufacture' and implementing it into automatic design systems is described by Wilson and his co-workers [29-31]. This work has resulted in the family of computer programs, MAPS (Material And Process Selection). MAPS-1 [29] generates interactively a twelve digit code representing characteristics of the part. After eliminating incompatible process and material combinations, the program indicates how many combinations remain and on request will provide a list of them broken down into two categories, usual practice and unusual (or more costly) practice. If the number of combinations is too small (or none) the user may elect to investigate relaxing some of the design requirements or manufacturing the part in more than one step. The MAPS-2 [30] is an improved version of MAPS-1 with the addition of a ranking system, which is based on the use of figures of merit. These are quantitative measures of how well a particular material and process combination satisfies a qualitative criterion of excellence set up by the designer. In MAPS-3 [31] artificial intelligence techniques have been employed. The part geometry is divided into two categories: Primary geometry, which is defined as an envelope around the part and which will be generated by a primary process; Secondary surface features, which include all the geometry that can not be produced by a primary process and which will be generated by machining or other secondary operations. The designer inputs the description of primary part geometry in answer to a request from the system. After the primary process and material combination has been determined, a problem solver is used to establish secondary process sequences. The problem solver defines desired secondary surface features as goal states, and the transient geometry produced by the chosen primary process as the current state. It searches for qualified machining operations and connects them onto the goal as a search tree. An evaluation criterion is added onto the paths as a weighting factor. The optimal machining sequence which corresponds to the shortest path can be obtained.

The driving forces for the adoption of Design for Manufacturing also include such factors as less experienced personnel, rapid changes in technology, and advances in computer techniques. Significant improvements can be expected by applying the Design for Manufacturing approach, such as smaller number of parts, easy of assembly, avoidance of special tooling, shorter lead time, cheaper materials, and more flexibility. The idea of Design for Manufacturing has gained evér-increasing national acceptance [32-38].

2.2 COMPUTER AIDS FOR CONCEPTUAL DESIGN

In mechanical engineering the first step towards CAD took place in the 1960's with the introduction of computer-aided drafting systems [39,40]. Three dimensional modeling techniques have permitted the objects be viewed in a convenient way and most types of analysis be carried out. The widespread use and decreasing cost of computers have brought about a revolution in the practice of engineering design. Computers can currently aid design in many ways, such as,

- Finite element analysis systems

- Sculptured surface design and manipulation programs
- Volumetric modeling facilities
- Realistic image synthesis
- Computer-aided operation, process and motion planning
- Kinematic and dynamic simulation and analysis.
- Computer-aided NC part programming.
- Computer-aided material selection

However, current CAD systems do not possess the integrative ability to conduct conceptual design activities. Classical, algorithmically-based computer techniques are ill-suited to these problems. The new techniques of knowledge engineering or AI techniques, such as expert systems, logic programming, database management techniques, and so on, provide tools to create effective computer aids for solving such problems.

In general the design process can be described as a transformation from function specifications to a physical layout [2-4,41,42]. Although this general theory captures the similarities of different design processes at some high level of abstraction, it does not describe every design process precisely. It is conceivable that different types of design may require different approaches. In mechanical engineering, there is a wide spectrum of functions and many ways to realize a single function. The problems of how to define functions and how to find a unique transformation for defined functions still remains intractable [11,43,44]. Recently, some AI researchers have become interested in mechanical engineering design and mechanical engineers have also made their contributions. The effort concentrates on modeling the mechanical design process and developing rational, scientific design methods [3,7-11,41-50]

Mechanical design is the design of devices and systems of a mechanical nature - machines, structures, devices and instruments. Most new mechanical designs in industry today result from an evolutionary approach. New designs are usually based upon the 'best' parts, sub-assemblies, or ideas taken from previous designs. Design alternatives are generated and evaluated as a process of redesigning the prototype. However, only a few researchers has recognized this unique "Re-Design" feature of mechanical design. Simmons and Dixon state that design of mechanical parts and products differs from other designs in several fundamental ways: material selection, sensitivity to manufacturing issues, non-modularity, high coupling of form and function, and especially the role of geometry. They view design as a hierarchy of nested iterative processes of (1) decomposition and redecompostion, (2) specification and respecification, and (3) design and redesign [51,52]. Brown summarizes the design into four phases: Requirements Phase, Rough Design Phase, Design Phase, and Redesign Phase [8,9].

Although expert systems are considered the major AI tools for engineering design, to date not many systems are practically in use. XCON(originally named R1) by DEC, which configures computers, seems to be the only commercially successful system [53]. Recent examples in the mechanical design domain include ACOLADE by Allen, R.H. et al at University of Houston, an assistant for composite laminated design [54], and PRIDE by Mittal, S. at Xerox Palo Alto Research Center, a system creates design for paper handling machines [47]. Dixon et al are trying to build up a domain independent design system for various design tasks, such as injection molding part design and V-belt pulley design [49,50,55]. More coverage in this topics can be found in the referencés [56-61].

The advantages of using language in design processes has long been recognized. Language, and drawings, are the most natural media for a designer to express design intentions and to interact with a design system. Language is able to provide a sophisticated combination of convenience, flexibility, expressive power, and reasoning media. Efforts have been made to use high-level languages (formal language, natural language, or something in between) directly in design automation systems. Progress has already been made in certain design domains. For example, VHDL (VHSIC) Hardware Design Language has been adopted as an industry-wide standard for electronic circuits design [62,63]. The necessity of developing such languages for mechanical design has been addressed by several researcher. Ullman et al suggest developing a machine design language capable of representing specifications, components, processes, etc [11]. In the ASME report [6] it is pointed out that the construction of a design language, complete with axioms and rules for transformations in manipulation of design models, is essential to express and manipulate the abstract elements in the design process [6]. The Design Specialists and Plans Language (DSPL) for designing air-cylinders developed by Brown [64,65] is one of the few such languages ever implemented.

2.3 KNOWLEDGE MANAGEMENT UTILITIES FOR DESIGN

The development of computer aids for conceptual design will benefit from an effective and intelligent database management system. The CAD community has recognized the need to integrate a variety of CAD packages around a common database. However, early experience with the use of database management systems in CAD has shown that they lack many features necessary to an engineering environment, such as handling of complex objects, handling of unstructured data of variable length, handling of graphical data, logical design of semantically rich engineering database, design-rule checking and consistency constraint enforcement, and flexible and powerful data models and modifications [66-74].

As an emerging research field, Expert Databases represent the confluence of research in database, logic, and artificial intelligence [75]. Compared with expert systems, expert databases are superior in their ability to efficiently search and exploit large amounts of data and in their ability to reason about a whole class of objects rather than about only a single instance [76]. Compared with conventional databases, expert databases are rich in data models, especially abstract data types for knowledge representation. The common approaches to building up expert database are the incorporation of alternative data models into relational databases and the merging of database system technology and artificial intelligence technology [77-81].

Our research includes an augmented relational database developed using the techniques of Expert Databases. The major extensions made on traditional relational database are logic programming and virtual relations.

A logic program is a set of clauses of the form

$$p_0 <- p_1, \ldots, p_n.$$

Each p_i is called a literal and has the form $p(t_1,\ldots,t_m)$, where p is a predicate symbol and $t_1, \ldots, t_m$ are terms. Terms may be constants, variables, or functions. The term

p_0 is called the head or conclusion, and p_1 through p_n form the body or conditions or the clause. A clause with an empty set of conditions is always true, and it is called an assertion. A clause with an empty head is interpreted as a goal, which the system tries to solve using the principles of resolution [82-87].

A logic program can be considered a natural extension of the relational database model because many relational tuples can be expressed as an assertion, or a predicate of the form $p(t_1, \ldots, t_m)$. Thus, it serves as both a database definition language and as a practical high-level queryl anguage. The main contribution of logic programming to databases is the incorporation of deductive information [88].

Virtual relations, representing a type of derived data, are defined in terms of existing relations. Virtual relations can be defined most commonly using the syntax of logic programming [89,90]. A user view of the database is then the collection of base and virtual relations. Virtual relations can be defined by means of data abstraction techniques, which hide the original underlying base relations. This means that a user generally cannot tell which relations are base and which are virtual. Internally, however, only the virtual scheme is stored with the database and not the relation itself.

CHAPTER 3 FUNCTION DESCRIPTION AND RATIONALIZATION

Research work in the area of automation of conceptual design has resulted in a prototype of an automated design system - Function Rationalization System (FRS). This has been built up based on our new design methodology, Decomposition-Description-Rationalization, or Design Rationalization. A design language, Function Description Language (FDL), has been developed and used in the FRS system to carry out this design methodology on computers.

3.1 OVERVIEW OF THE FRS SYSTEM

The architecture of the Function Rationalization System (FRS) is shown in Figure 3.1. The main components of FRS include an FDL parser, a product data base, an analysis program, and a library of design rules. The current version of FRS is written in the C language and runs at satisfactory speed on a DEC VAX-11/785 minicomputer.

In operation, the FDL parser takes the product description provided by the user, checks it for syntax and semantic errors, converts it into a set of relation tables and stores it in the relational database. The analysis program, on command from the user, applies rules selected from the library of design rules to the whole product or particular modules or components. It detects design

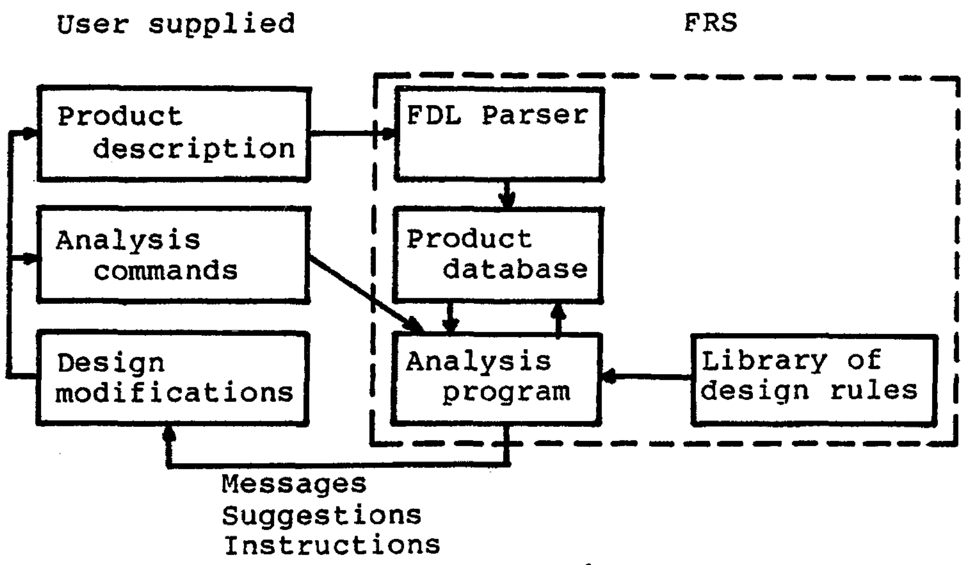


Figure 3.1 Function Rationalization System

deficiencies such as redundant functions and/or structures in the product and outputs diagnostic messages and suggestions for improving the design. FRS works basically in an advisory way. It only provides recommendations. The designer may reject the suggestions, because he knows the functional purpose of a special feature. He may accept the suggestion and modify the design accordingly. This rationalization process may be repeated recursively until satisfactory results are achieved. In addition, statistical data will be given to help the designer to compare the new design with the old ones, such as the total number of parts, the number of part types, and the degree of function concentration (average number of functions performed by each part).

The library provides a set of basic rules for general users. Advanced users can define their own rules and add them to the library, so that the system can be adapted to a user's particular environment.

3.2 DECOMPOSITION-DESCRIPTION-RATIONALIZATION METHOD

As it was mentioned in Section 2.2 most new mechanical designs in industry today take an evolutionary approach. New designs are usually based upon the 'best' parts, 'best' sub-assemblies, or 'best' ideas taken from previous designs. There are often cases where the structures taken from a previous design may have parts or functions which are not needed in the new design. Even essential functions may not be properly assigned to the components. If this redundancy is removed, and remaining functions are re-distributed properly, the structure of the design will be simpler and better organized. Therefore, a formal design review can lead to both the enhancement of product performance and the reduction of production cost. This review process should take place at critical stages in the initial design process. It can be carried out in the following steps,

- Decompose a primary design into functional modules and components
- Describe their functions in a systematic way
- Identify redundancies, overlapping, or mis-assignment of functions with respect to these descriptions
- Improve design by eliminating the redundancy, and rearranging remaining functions and further improvements may be achieved by prompting the designer for new ideas.

This can be characterized as the 'Decomposition-Description-Rationalization' approach, or 'Design Rationalization'.

3.3 FUNCTION DESCRIPTION LANGUAGE

The description of functions of parts and relationships between components is the essence of this work. A human engineer uses sentences to specify functions of parts and the functional relationships between parts. Using language allows the construction of an infinite number of sentences from a finite number of syntax rules. The vocabulary of the language can be extended and this, in turn, allows the specification of various structures and a wide spectrum of functions in mechanical products. And best of all, in-depth information about a design can be deduced from function description sentences.

For instance the sentence,

'part-X holds part-Y'

describes the functional relationship between part 'X' and part 'Y'

From this sentence, a structural hierarchy can be inferred

```
          module-?
          /      \
         /  hold  \
        /          \
       X            Y
```

Where 'module' is defined as the ancestor of parts 'x' and 'y'. This hierarchy implies that the part 'X' and part 'Y' should have the same ancestor in an ideal design. The actual structures, however, may take another form, as shown below,

```
        module-1              module-2
          /   \                 /   \
         X     ...             Y     ...
```

In general, inconsistency between the hierarchy deduced from function description sentence and the actual structural hierarchy, will indicate deficiencies in structure, such as redundancy or mis-assignment of functions. From that point, a series of design rules can be applied.

The current natural language processing techniques still have many problems. On the other hand, the formal languages in software engineering are able to describe logic precisely, but their vocabulary is quite limited. A special language, called Function Description Language

(FDL) has been developed as a compromise between these two aspects. The sentences of FDL resemble English sentences closely, but without inflection rules on nouns and verbs. This makes function description sentences quite readable and reduces the difficulty of implementation tremendously. At the same time, FDL allows the user to define his own vocabulary freely which greatly enhances its depictive power. An additional advantage is that by allowing an expandable vocabulary, FDL can easily adapt to different domains. If the terminology of English grammar is used, the syntax rules of FDL Function Description sentence can be represented as,

```
sentence    : element_name v_o_phrase
            ;

v_o_phrase : verb_phrase obj_phrase
            ;

verb_phrase: verb noun_phrase
            ;

obj_phrase :
            | prep_phrase obj_phrase
            ;

prep_phrase: preposition noun_phrase
            | 'to' verb noun_phrase
            ;

noun_phrase: element_name
            | element_name function_name
            ;
```

The simplest FDL sentence has only three words, a subject, a verb, and an object. Such as

```
top hold dial_capsule
```

The recursive rule allows designers to write complicated sentences, which include one or more preposition phrases. Such as,

```
bearing connect dial_shell to magnet_bar
bracket support compass on cover through top
```

FDL allows the user to express design concepts and design intentions in ways much like they are used to. It also allows the user to interact with the design system much like they might discuss it with their team workers. The complete syntax specification of FDL is given in Appendix A.

CHAPTER 4 PRODUCT DATABASE

A relational data base (RDB) has been chosen as the basic structure for the product data base in FRS due to its simplicity, flexibility, and inherent inferencing capability based on relational algebra. The RDB has been augmented in two main ways: (1) Hierarchical data structures can be handled integrally on the top of the relational data base, and (2) Multiple reasoning mechanisms have been incorporated into the system in addition to relational algebra. In the latter area virtual relations are used to deduce relations that are not explicitly stored, recursive queries can be processed, and 'IF ...THEN ... ELSE ...' statements are used to gain a better control over reasoning processes. This feature also allows the incorporation of data modification operations so that certain types of design tasks can be actually performed. As a result, the FRS database possesses many advanced features of expert systems.

4.1 HIERARCHICAL DESIGN ENTITIES

Mechanical design deals with complex objects, which are often best represented in terms of hierarchical structure of entities, as shown in Figure 4.1. However, the basic data structures supported in relational database are records or sets of homogeneous records [90]. The implementation of a complex object as a collection of tuples and relations, is a big issue in CAD-oriented database [91-93].

```
                  part_description
                   /      |        \
                  /       |         \
      parameter     geometry       material
                         /      \
           surface_1 ..    surface_k
             /     \
    dimensions    manufacture
```

Figure 4.1 Hierarchical Description of a Part

FRS provides two predefined data structures for design entity description, Module Declaration and Component Declaration.

4.1.1 Module Declaration

A module is a group of elements which are connected together, (mechanically or otherwise) to perform particular common functions. A module declaration includes a list of elements and a group of function description sentences. The format of a module declaration is,

```
module name
  { element
      element_name element_type * quantity;
      ...
    function
      FDL sentence;
      ...
  };
```

Here is an example,

```
module compass
  { element
      compensator;
      case;
      dial;
    function
      dial point direction;
      compensator compensate environment magnetic_field;
      case adjust compass position;
  };
```

An element can be another module, or an individual part in the conventional sense. In addition, some abstract items, such as 'magnetic field' in the example, can also be included in the specification, if they must be considered in order to fully describe a design. The item 'magnetic field' is not included in the elements list, because it considered as an external entity.

The definition of a module is converted into a set of relation tables. In addition, directories are set up to keep track of the hierarchies. Here are the storage tables used for module declaration.

Table name	Attribute list
element	(module_name, element_name)
element_type	(module_name, element_name, element_type)
quantity	(module_name, element_name, quantity)
function	(module_name, FDL_sentence)

4.1.2 Storage Of FDL Sentences

The underlying data storages of FRS are relational tables, in which the number of attributes is fixed. However, the number of sentence components in FDL can vary greatly. There is no way to know how many attributes are needed for storing various sentences. To overcome this problem, a special scheme for storing FDL sentences has been developed. Each FDL sentence, no matter how many words it contains, is stored as a special attribute 'sentence' in relation tables. When some of the sentence elements, say 'subject' or 'verb', are referenced, a secondary parser (It is called 'secondary', in order to distinguish it from the FDL parser used for initial syntax checks) is invoked to retrieve the sentence element and pass it to the application program. From the user's view point, sentence components are treated just like ordinary attributes in any relation table.

4.1.3 Component Declaration

Component declaration is a lower-level data structure for describing a single part. It includes additional information on a part such as geometry, dimensions, material, and processes. The format is,

```
component name
  { geometry
      shape_name(feature_name = dimension,
                 ....);
      ...
    material
      material_name;
    manufacture
      feature_name manufacturing_process;
      ...
  };
```

Here is an example,

```
component gear
  { geometry
      gear_head(d = 0.875, h=0.125);
      shaft( d =0.0625, length = 0.75);
    material
      plastic;
    manufacture
      inject_mold;
  };
```

The items, 'geometry', 'material', in this declaration are optional. At the beginning, some of these items may not be defined. As the design proceeds, more and more details will be worked out and be added. For example, if two parts are combined, the new features and dimensions need to be specified. The material and related manufacturing process may also be changed and should be specified. The final design should provide a complete description of both functional and manufacturing issues.

The physical storage for the component declaration includes the following tables,

Table name	Attribute list
parameter	(component, parameter)
geometry	(component, geometry)
dimensions	(component, geometry, dimension, value)
material	(component, material)
manufacture	(component, geometry, manufacture)

4.1.4 Generic Entity

An entity in these declaration can be uniquely specified by the notation

$value_1$: ... :$value_k$

Such as,

gear:shaft:length

The nomenclature 'generic entity' is used to refer to a class of entities. For example, all the geometric features of the component 'gear' can be called by using the notation 'gear:geometry'. Using the notation of generic entities allows operations on a whole class of objects as discussed in the next section.

These extensions allow useful concepts from other paradigms, such as the hierarchical model, to be incorporated into the relational database while retaining the high-level user interfaces and the underlying clarity of a relational model.

4.2 OPERATIONS

Design requires that design entities be manipulated at the highest possible level of abstraction. That is, each entity can be treated as a single unit in terms of higher level operations without knowing explicitly its underlying data structure. This has been realized in FRS by defining a set of operations on generic entities and implementing them using more primitive features provided by the conventional RDB. These are discussed in more detail below.

4.2.1 Compare With A Value

A Generic entity can be compared with a value.

$v_1: \ldots :v_k$:N BOOLEAN_OPTR V

where 'V' is a given value. BOOLEAN_OPTR may be <, <=, ==, =, >=, or >. 'N' is an attribute name. When the operation is executed, each value of attribute 'N' will retrieved and compared with the given value 'V'.

4.2.2 Compare Two Generic Entities

$x_1: \ldots :x_k$:N1 OPTR $y_1: \ldots :y_m$:N2

There are two cases:

(a) Set Comparison, represented using operators '<=>' or '<>'. The values of 'N1' and the values of 'N2' will be compared on the base of set equality.

(b) Value Comparison, defined using boolean operators '==', '=', etc. Each value of the attribute 'N1' will be compared with every values of the attribute 'N2' on an individual base.

4.2.3 Generic Comparison

A 'generic entity' stands for one class of objects, the exact number of which may not be known a priori. Operations on generic entities can be defined in a general form using the notation,

OPTR(generic_entity)

This is called 'generic comparison'. When it is applied, each value of the attribute 'N' will be compared with every other in a combinatorial way. 'OPTR' is a boolean operator.

4.2.4 Assignment Operations

(a) A value can be assigned to a attribute using the statement,

$x_1: \ldots :x_k:N1 = V$

(b) Two entities can be set equal,

$x_1: \ldots :x_k:N1 = y_1: \ldots :y_k:N2$

where the two attributes 'N1' and 'N2' must be identical. As a result, the attribute 'N1' will take all the values of attribute 'N2'.

4.3 REASONING MECHANISM

In addition to relational algebra, several deductive mechanisms have been added to the database. All operations discussed below apply to the relation tables as well as module declaration and component declaration. That is, the whole database can be handled by the reasoning mechanism in a consistent way.

4.3.1 'IF conditions THEN actions1 ELSE actions2' Statement

'If_then_else' statements are incorporated into FRS to represent rule-based knowledge and hierarchical design logic, and to gain better control over reasoning processes.

(a) Condition Specification And Linguistic Operator

The condition specification in the 'if_then_else' statement includes the operations discussed in Section 4.2. In addition, a class of special operators, called 'Linguistic Operator', are defined using the format,

table_name(attribute$_1$=entity$_1$,...,attribute$_k$=entity$_k$)

or

!table_name(attribute$_1$=entity$_1$,...,attribute$_k$=entity$_k$)

Where 'table_name' can be any relation table defined by the user. The linguistic operator can be used in a variety of ways. Physically it checks whether some entries are in a relation table. Logically it epitomizes the meaning of the entries in the relational table as the name of the table. For example, a relation table 'fixed_ connection' contains the following words.

fixed_relation:

x
fasten
support
latch
connect

The name of the table, 'fixed_connection', can be considered as an abstraction of the meanings of the words, 'fasten', 'support', 'latch', and so on.

(b) Action Specification

The action specification in the 'if_then_else' statement may be a combination of the three types, message, assignment statement, and 'if_then' statement. Messages may contain names of entities in an easily readable form. Assignment statement allows direct manipulation on data entities. The nested 'IF ... THEN ... ELSE ...' statements allows hierarchical logic and a better control over reasoning processes.

4.3.2 Virtual Relation

A virtual relation is defined by deduction rules and elementary facts in the RDB. A DEDUCTION RULE has the form

Q:- P_1, P_2, ..., P_k
WHERE
  modifications;

where Q is called Virtual Relation, and
P_1, ..., P_k are constituent relations,
'Modifications' are additional constraints on the attributes involved.

If the name of a virtual relation appears on both sides of a deduction rule, it is recursive, such as,

```
  similar(x, y):-
     table(x,y);

  similar(x, y):-
     similar(x, y=z),
    table(x=z, y);
```

For each recursive relation, there must be at least two parallel definitions, one of which must be non-recursive and specifies the termination condition of the recursion. Recursion may be simple transitive closure or a more complex form [83]. In the current system only simple recursion is supported.

In FRS virtual relations are used for two main purposes:

(a) Deduction About Data

Virtual relations can be used to deduce facts and relations which are not explicitly stored in the database. For example, function description sentences are stored as a relation table,

```
    function(subject, verb, object)
```

and the elements in a module are stored in another table,

```
    element(module, element)
```

If it is necessary to find out to which module the 'subject' belongs and to which module the 'object' belongs, a virtual relation 'ancestor' can be defined as,

```
ancestor(a1, subject, a2, object):-
   element(module=a2, element=object),
   element(module=a1, element=subject),
   function(subject, object);
```

The definition says that if the 'subject' of a sentence happens to be the attribute 'element' in the table 'element', then the attribute 'module' in the same tuple is defined as the ancestor of the 'subject'.

(b) Semantic Abstraction

Virtual relations provide a powerful tool for handling what are called 'abstraction ladders' by semanticists. For example, if there is a virtual relation 'connection' defined as,

```
   connection(x):-
     movable_connection(x);
   connection(x):-
     fixed_connection(x);
```

The definition says that a word 'x' can be considered as 'connection' if it is either in the table 'movable_connection' or in the table 'fixed_connection'. The table 'fixed_connection' is given above and the table 'movable_connection' is defined as followings,

Table: movable_connection

x
attach
pivot

According to the definition, the meanings of all the words in these two tables can be considered in the category of 'connection'. It thus can be considered as a further abstraction of the 'fixed_connection' and 'movable_connection'.

The combination of linguistic operators and virtual relations provides FRS with even greater flexibility in handling conceptual information, such as synonyms and antonyms.

CHAPTER 5   LIBRARY OF DESIGN RULES

FRS provides a set of major principles for design rationalization. Some of them are summarized from common engineering practice, and the others are developed particularly from the features of the FDL language. All analysis rules are represented in an easily readable format. By the use of appropriate analysis commands, the design rules can be selectively applied with different emphasis at various levels, from a single component to a complex assembly consisting of many modules. This allows high level concepts and operations on objects to be defined and design methodologies to be enforced in a flexible way. This chapter will discuss some of these principles. Appendix B lists the analysis rules in the Design Library of the FRS system.

5.1 ANALYSIS COMMAND AND DESIGN RULE

5.1.1 Analysis Command

Analysis commands causes the system to apply a particular set of design rules to a particular set of design entities. The format is,

```
Analyze
   entity-1,
   ...
   entity-k
Using
```

```
     rule-1,
     ...
     rule-m;
```

The entity may be a particular module, component, or attribute, or a generic entity. Here is an examples,

```
   Analyze
      compensator:gear1,
      compensator:gear2
   Using
      parametric_design;
```

5.1.2 Design Rule

Design rules may generally be defined in the format,

```
  rule name
     { virtual_relation_definition;
       ...
       IF condition THEN action1 ELSE action2;
       ...
     };
```

The action specification in the 'if_then_else' statement may be a combination of the following four types,

(a) Message. This may be diagnostic messages, comments, suggestions, or instructions.

(b) Assignment. This assigns a value to an attribute, allowing FRS to do some design tasks directly.

(c) 'if_then_else' statement. This allows further investigation of the design using a locally defined rule.

(d) Analysis command. This applies another set of rules from the design rule library. A combination of 'if_then_else' statements and analysis commands can allow a wide variety of hierarchical design investigations.

The use of pseudo variables in design rules, analogous to that of dummy arguments in programming languages, allows the names of objects to be integrated with output messages into an easily readable form. Such as,

```
rule parametric_design
   { if $1:geometry == $2:geometry and
        $1:dimensions != $2:dimensions
     then
        message(Can $1 and $2 be redesigned using
           parametric design principle ?);
   }
```

The pseudo variables $1 and $2 are logically equivalent to the entities set by the analysis command. Thus the command

```
ANALYZE
   compensator:gear1, compensator:gear2
USING
   parametric_design;
```

will cause '$1' to be substituted by 'gear1' and '$2' by 'gear2'.

An analysis command can be applied to a whole class of object by using generic entity notation. Such as,

```
ANALYZE
   compensator:element
USING
   parametric_design;
```

The name 'element' here is not a single part, but refers to all the elements in the module 'compensator'.

5.2 THE PRINCIPLES OF DESIGN RATIONALIZATION

5.2.1 Principle Of Modularization

A very important principle of Design Rationalization is to make the functional hierarchy match the structural hierarchy. A mixture of hierarchies will cause extra complexity both in structure and in manufacturing. For example, two brackets 'A' and 'B' are used to support a rotating shaft. If 'A' and 'B' belongs to two different modules, extra caution is needed in machining and assembly in order that the two brackets are properly aligned. Therefore this should be avoided. This idea has been implemented as the rule 'modularization'.

```
rule modularization
  { ancestor(a1, e1, v, a2, e2):-
      element(module=a1, element=e1),
      element(module=a2, element=e2),
      function(sub=e1, verb = v, obj=e2)
    where
      ancestor:a1 = ancestor:a2;

    if ancestor(e1 = $1) then
      "-- $1 $ancestor:v $ancestor:e2 from another module
      $ancestor:a2, can it be modified by modularization
      principle ?";
  };
```

In this rule a virtual relation 'ancestor' is defined. This is used to determine whether the 'subject' and the 'object' of a sentence belong to the same module. If not, the structure is considered to violate the principle of modularization.

5.2.2 Elimination Of Direct Chain

An important feature of FDL is that the complexity of sentence patterns indicates the complexity of the actual structure. Therefore, by looking at sentence patterns, we can locate potential problems in structures. For example, the sentence,

'part-X connect part-Y to part-Z through part-W'

has a pattern which can be summarized as follows,

subject verb object preposition prop_object ....

This pattern can be identified using the following rule,

```
rule direct_chain
  { if function(sub= $1) and function:prepl = _ then
    "-- Can $1 be eliminated and
     Can $function:obj $function:verb $function:pobjl
       directly ?
     Can $1 be combined with $function:obj or
       $function:pobjl ?";
  };
```

The rule says that if a sentence has preposition phrases, it may have a direct chain structure, which helps to identify complicated nesting in a design. A variation of the principle is as follows,

```
rule direct_restrict
  { if fixed_connection(x = $1:function:verb) and
       ( function:prepl == by or
         function:prepl == through )
    then
    "-- Can $1:function:pobjl be eliminated and
        $1 $function:verb $function:obj directly ?";
  };
```

This rule can be used to identify parts which are connected together permanently. Other chains, including movable connections and force transfer are not included.

5.2.3 Elimination Of Implied Chain

Simplifying sentence patterns means simplifying the actual structure. If all the sentence have a simple pattern, such as 'x verb y', the overall structure will be much simpler. However, problems may still exist. For instance, the two sentences, 'x v1 y' and 'y v2 z' are independent in syntax, but if 'v1' and 'v2' are synonums a chain is implied by these two sentences. Considering two sentences, 'x secure y', and 'y hold z'. If the meanings of 'secure' and 'hold' are considered similar, then the structure raises the question 'Can x hold z directly?'. The associated rule is written as,

```
rule implied_chain
  { chain(x, v1, y, v2, z):-
      function(sub=x, verb=v1, obj=y),
      function(sub=y, verb=v2, obj=z)

    if chain(x=$1) and
       ( similar_verb( x1 = $chain:v1,
                       x2 = $chain:v2) or
         $chain:v1 == $chain:v2 )
    then
    "-- There is a chain between $1, $chain:y, $chain:z,
     Improvement can be made:
       1. combing all three together
       2. combing two of three, eliminating the third";
  };
```

The virtual relation 'chain' locates all chains. Then the verbs are checked to see whether they are similar or identical. If such is the case, modification is suggested.

As it was pointed out in Chapter 4, FDL allows the user to set up his own dictionary, including tables of synonyms and antonyms. Here, the relation table 'similar_verb' refers to such a table which contains pairs of verbs which are considered as synonyms.

5.2.4 Principle Of Function Integration

A smaller number of parts usually means simpler structure and lower cost. One way to reduce the number of parts is through the integration of elements which perform identical or similar functions on the same object. The resultant complexity in the new parts may be overcome by employing one of the new processes which enable complex parts to be produced. This is called 'The Principle of Function Integration' and is implemented as,

```
rule integration
  { if ==($1:function:obj) and
       ( similar_verb( x1 = $1:function:verb,
                       x2 = $2:function:verb) or
       ==($:function:verb) )
    then
    "-- $list
     These elements perform similar function on same
    element $1:function:obj, can they be integrated ?";
  };
```

5.2.5 Principle Of Function Distribution

Sometimes, although two elements share some common functions, they may not be integrated because one of them may carry multiple functions and can not be simply eliminated. However, elimination may still be achieved by distributing those functions to several other elements. This idea can be implemented as the rule 'distribution',

```
rule distribution
  { if !=($1:function:verb) and
        $1:function:verb == $2:function:verb and
        $1:function:obj == $2:function:obj
    then
   "-- $list have common function $1:function:verb on the
        same object $1:function:obj, but they cannot be
        integrated simply because they have other functions.
        However, improvement can be made by redistributing
        function $1:function:verb between them.";
  };
```

The rules 'integration' and 'distribution' can be combined into one,

```
rule rationalization
  { if ==($1:function:verb) and ==($1:function:obj)
    then
    "-- $list
        These elements perform identical function on same
        element $1:function:obj, can they be integrated ?"
    else
      { if $1:function:verb == $2:function:verb and
           $1:function:obj == $2:function:obj   then
        "-- $list have common function $1:function:verb on
            the same object $1:function:obj, but they
            cannot be integrated simply because they have
            other functions. However, improvement can be
            made by redistributing function
            $1:function:verb between them."
      };
  };
```

5.2.6 Principle Of Parametric Design

There are many factors which should also be considered in establishing the layout of a product. For example, in engineering practice, if several parts have similar features and most dimensions of these feature are common, a generic part can be designed. A family of parts can be derived from that single design by changing certain parameters. This is known as the 'Principle of Parametric Design' and can be applied in conjunction with Group Technology for more benefits [94,95]. Similar parts should be identified and grouped together to take advantages of their similarities in manufacturing and design. One implementation of this rule is thus,

```
rule parametric_design
  { if ==($1:geometry) and
       !=($1:dimensions)
    then
   "-- $list, these parts share common geometric features,
      but different dimensions, different dimensions,
      Can they be re-designed using parametric design
      principle ?";
  };
```

The following is a modified version of the rule. It can even list out which dimensions should be used as parameters in the new part family.

```
rule parametric_design
  { if ==($1:geometry) and
       !=($1:dimensions)
    then
    "-- $list, these parts share common geometric features,
        but differ in dimensions, the following dimensions
       can be chosen as parameters of the new part family",
     { if $1:dimensions != $1:dimensions
        then
           "$1:dimensions";
     };
  };
```

5.2.7 Principle Of Dimension Standardization

If several similar machined features, e.g. holes, are present in a part, their dimensions should be the same if possible and should compatible with standard tool sizes. This will reduce the time needed for tool changing during processing and reduce the cost of tooling. This is called the 'Principle of Dimension Standardization'.

```
rule dimension_standardization
  { if !=($1:dimensions) and
       similar_feature( x = $1, y = $2)
    then
      "-- $list
          Can these geometric features use common
          dimensions ?";
  };
```

where 'similar_feature' is a user-defined table, which lists out in pairs those geometries which are considered similar.

5.2.8 Principle Of Direct Assembly

Traditionally, assembly problems are dealt with in the production stage, but assembly efficiency is restricted by the structures of the product. Therefore, good assembly should be designed into a product rather than planned at the time of production. In Design Rationalization, assembly can also be investigated with respect to the function of components. The following are some rules about assembly.

```
rule direct_assembly
  { if assembly( x = $1:function:verb) and
        $1:function:prepl != _
    then
    "-- Can $1 be eliminated and
        $1:function:obj $1:function:verb $1:function:pobjl
        directly ?";
  };
```

This rule says that if the verb of a sentence is considered as 'assembly' function and its preposition phrase is not empty, then it represents a triple assembly and improvement can be made by assembling two of the elements directly.

5.2.9 Principle Of Fastener-Less Assembly

A large amount of time in assembly is devoted to handling fasteners, such as bolts, nuts, and screws. Whenever possible, replacing these removable connections with permanent connections or some other easier ways are strongly recommended. Such ideas can be implemented as rule 'Fastener-Less'.

There are several ways to check whether an element is a fastener: to check whether the verb in the sentence performs 'assembly' function; or to check whether the name of the subject is a fastener, such as a screw; or to check whether two objects are assembled together by a fastener.

```
rule fastener_less
  { if fastener( x = $1 ) or
       assembly( x = $1:function:verb) or
       fastener( x = $1:function:pobj1 )
    then
    "-- Can $1:function:pobj1 be eliminated and
        $1 and $function:obj be assembled directly ?";
  };
```

5.2.10 Miscellaneous Rules

There are many other well-proven engineering practices that can be incorporated into the FRS system. For example, reducing the number of parts is always a major concern in simplification, since the smaller the part number, the simpler a design. This idea is implemented as the rule 'Elimination',

```
rule elimination
  { if function( sub = $1)
    then
      "Can $1 be eliminated, if so, how can function
      $1:function:verb on $1:function:obj be preserved ?";
  };
```

If several similar features, say holes, are generated by different processes, cost may be reduced by machining these features by the same process if possible. The following rule confirms this practice.

```
rule less_machining
  { if ==($1:manufacture:geometry) and
       !=($1:manufacture:process)
    then
    "-- $list are similar features,
        can they be machined by the same manufacturing
        process ?";
  };
```

Cost may be reduced by using cheaper materials.

```
rule material_cost
  { if expensive(x = $1:material)
    then
       "--Can $1 be made by cheaper material than
       $1:material ?";
  };
```

where the table 'expensive' is user defined, which lists materials that are considered expensive.

A very important consideration in developing the FRS was to help designers to do a creative job. In addition to the rules described above, the system will pose questions or suggestions to stimulate the designer to develop creative ideas. Here are some typical questions posed by the FRS system,

- Can this part assume another shape ?
- Can the surface quality requirements, i.e. tolerance and roughness, of certain geometric features be relaxed ?
- What alternative material can be used for this part ? If so, what is the new manufacturing process ?
- Can this part be redesigned using 'Divide-and-Conquer' principle

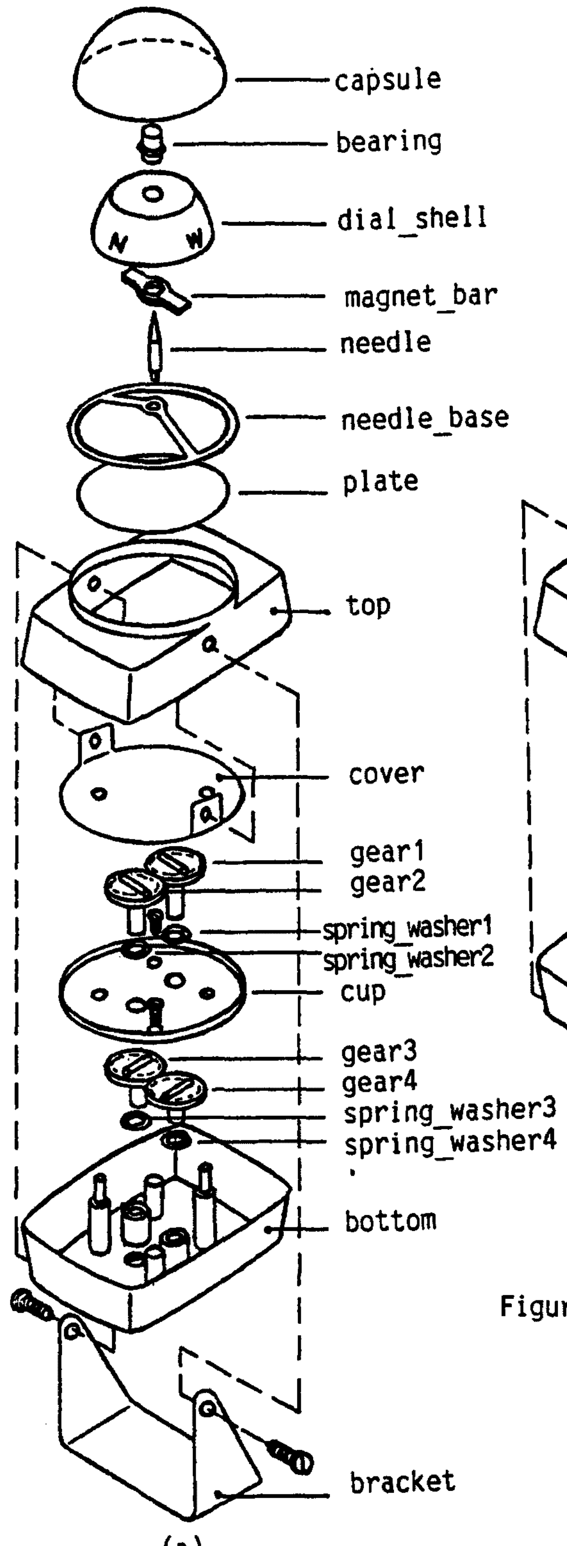
CHAPTER 6 AN APPLICATION EXAMPLE

In this chapter an example is discussed in detail to show how FRS can be applied to helping layout configurations of mechanical products. Figure 6.1 is the sketch of a compass which has been analyzed successfully using FRS. The compass, which is used popularly in cars and boats, is a product of the Airguide Instrument Co. at Chicago. Figure 6.1a shows the original design and Figure 6.1b shows the modified or improved design. When compared with the initial design the improved design has about 30% less parts and the number of part types is reduced by 15%. It is estimated that this will result in a cost reduction of about 30%.

6.1 DESCRIPTION OF THE INITIAL DESIGN

The initial design of the compass can be decomposed into three modules: dial_capsule, case, and compensator. Each of them consists of a number of components. The following is the description of the initial design.

```
module compass
  { element
      case;
      compensator;
      dial_capsule;
    function
      case adjust compass position;
      compensator compensate environment
          magnetic_field;
```



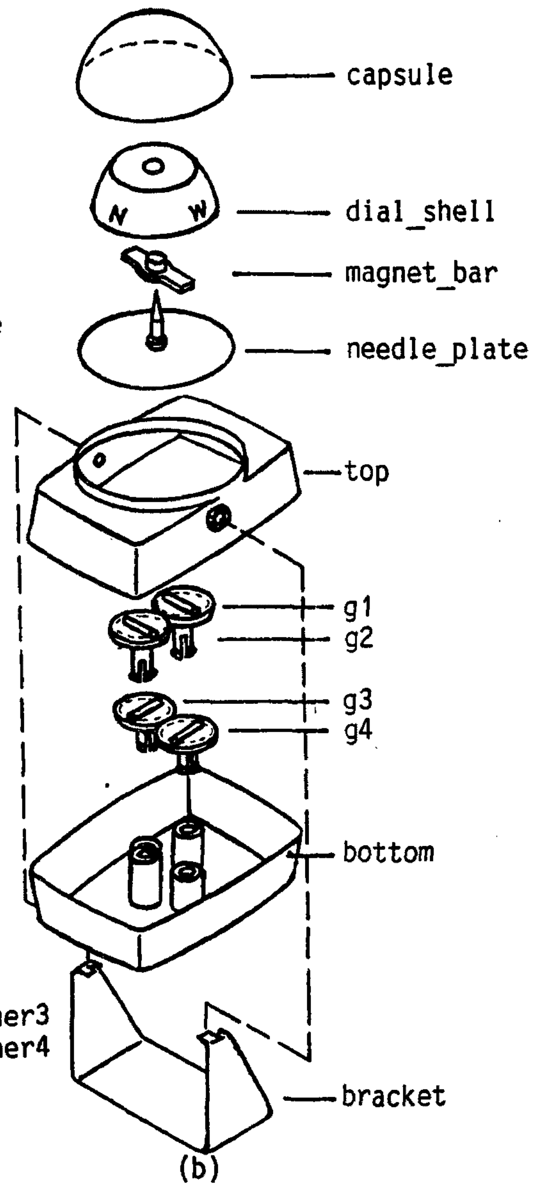


Figure 6.1 The Initial Design(a) and Modified Design(b) of a Compass

```
        dial_capsule display direction;
    };


module compensator
    { element
        cover;
        cup;
        spring_washer1;
        spring_washer2;
        spring_washer3;
        spring_washer4;
        gear1
          { geometry
              gear_head(d = 0.875,
                        h = 0.125);
              shaft( d = 0.0625,
                     l = 0.75);
          };
        gear2
          { geometry
              gear_head(d = 0.875,
                        h = 0.125);
              shaft( d = 0.0625,
                     l = 0.1875);
          };
        gear3
          { geometry
              gear_head(d = 0.875,
                        h = 0.125);
              shaft(d = 0.0625,
                    l = 0.5);
          };
        gear4
          { geometry
              gear_head(d = 0.875,
                        h = 0.125);
              shaft( d = 0.0625,
                     l = 0.1875);
          };

      function
        cup hold spring_washer1;
        cup hold spring_washer2;
        spring_washer1 restrict gear1
              vertical_motion;
        spring_washer2 restrict gear2
              vertical_motion;
        spring_washer1 damp gear1 rotation;
        spring_washer2 damp gear2 rotation;
        cover constrain gear1;
        cover constrain gear2;
        cup constrain gear3;
        cup constrain gear4;
        spring_washer3 restrict gear3
            vertical_motion;
        spring_washer4 restrict gear4
            vertical_motion;
        spring_washer3 damp gear3 rotation;
        spring_washer4 damp gear4 rotation;
    ];


module case
    [ element
        top;
        bottom;
        bracket;
      function
        top hold dial_capsule;
        bracket connect cover by screw;
        bracket support compass on cover
               through top;
        bracket adjust compass position
               on cover;
        bottom fix cover;
        top connect bottom through cover;
        bottom hold cup;
        bottom hold spring_washer3;
        bottom hold spring_washer4;
        bottom hold gear3;
        bottom hold gear4;
    ];


module dial_capsule
    [ element
        capsule;
        plate;
        needle_base;
        dial_shell;
        bearing;
        magnet_bar;
        needle;
      function
        capsule contain liquid;
        needle pivot bearing;
        needle_base hold needle;
        plate constrain needle_base;
        bearing connect dial_shell
        to magnet_bar.
};
```

6.2 OUTPUT MESSAGES OF FRS

A number of different rules were applied to the compass by using appropriate analysis commands. A detailed list of input and output of FRS for analyzing the compass is given in Appendix C.

In this section a number of typical commands and outputs will be discussed in detail.

(a) Apply Rule Modularization

Here are some output of the messages by applying rule Modularization,

```
- bottom hold cup from another module
  compensator, can they be modified by
  modularization principle ?

- bottom hold gear3 from another
  module compensator, can they be
  modified by modularization principle?

- bottom hold spring_washer3 from
  another module compensator, can they
  be modified by modularization
  principle ?

- bottom hold cover from another module
  compensator, can it be modified by
  modularization principle ?
```

As shown in Figure 6.1a, the cup is put in the bottom first, washers and gears are put into the cup in sequence, and then the cover is fixed by screws onto the bottom. Using the module case to support several individual parts in another module compensator is identified as a potential problem. Improvements can be made by making the design modular. For example, the module 'compensator' can be made self-contained and can be mounted into the module 'case' simply.

(b) Apply Rule Direct Chain

Here are some output of the messages by applying the rule Direct chain,

- Can top connect bottom directly ?
- Can bracket support compass directly ?
- Can bracket adjust compass position directly ?

In the initial design, the connection between 'top' and 'bottom' in module 'case' is complicated. 'Top' is connected to the 'cover' of another module 'compensator' by a screw, and the 'cover' is connect to the 'bottom' by another screw. The system identifies this nesting and suggests a direct connection between the 'top' and the 'bottom'.

(c) Apply Rule Implied_chain

Here are some output messages generated by applying the rule Implied Chain,

- There is a chain between plate 'constrain needle_base' and 'needle_base hold_needle', can they be_simplified ?

In the Figure 6.1a plate 'constrain' the needle_base, and needle_base 'hold' needle. This is identified as an implied chain because 'constrain' and 'hold' are considered synonyms.

(d) Apply Rule Direct Assembly

Here are some output of the messages by applying the rule Direct assembly,

```
- Can bearing be eliminated and
  dial_shell and magnet_bar be
  assembled together directly ?
```

In the original design, the 'bearing' functions like a rivet. It connects dial_shell and magnet_bar together. Riveting is a time-consuming operation. An improvement is suggested by connecting the magnetic bar and dial_shell directly.

(e) Apply Rule Parametric Design

Here are some output of the messages by applying the rule Parametric design,

```
- compensator:gear1, compensator:gear2
  shares common geometric feature, but
  different dimensions, Can they be
  re-designed using parametric design
  principle ?

- compensator:gear1, compensator:gear3
  shares common geometric feature, but
  different dimensions, Can they be
  re-designed using parametric design
  principle ?
```

In the initial design, there are four gears, each carrying a small magnet. The detailed drawings of these gears are in Figure 6.2. The rule 'parametric design' identifies that gear1, gear2, and gear3 have common feature but different dimensions and suggests a re-design.

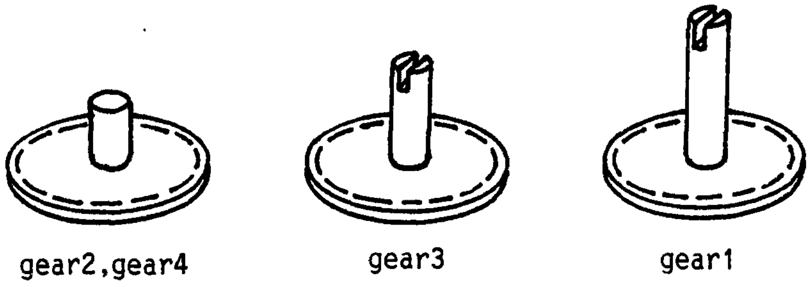


Figure 6.2 The Initial Design of Gears

(f) Apply Rule Fastener Less Assembly

Here are some output messages generated by applying the rule Fastener less,

- Can screw be eliminated and bracket
  connect bottom directly ?
  If so, how bracket adjust compass
  position on cover ?

The rule found out that there are screws in the design performing the fastening function and elimination is suggested.

6.3 DESCRIPTION OF MODIFIED DESIGN

The following is the description of the modified design of the compass.

```
module compass
  { element
       case;
       dial_capsule;
     function
       dial_capsule display direction;
       case compensate environment
          magnetic_field;
       case adjust compass position;
  };


module dial
  { element
       needle_plate;
         { geometry
             needle;
             plate;
         };
       dial_shell;
       magnet_bar
         { geometry
             bearing_hole;
           manufacture
             powder_metallurgy;
         };
     function
       needle_plate pivot magnet_bar;
       magnet_bar point direction;
       magnet connect dial_shell;
  };

module case
  { element
       g1: gear(shaft:length = 0.835);
       g2: gear(shaft:length = 0.835);
       g3: gear(shaft:length = 0.36);
       g4: gear(shaft:length = 0.36);
       bracket;
       bottom;
       top;
     function
       bracket support top to adjust
         compass position;
       bottom connect top by adhesive;
       bottom hold g1;
       bottom hold g2;
       bottom hold g3;
```

```
        bottom hold g4;
        bottom restrict g1 vertical_motion;
        bottom restrict g2 vertical_motion;
        bottom restrict g3 vertical_motion;
        bottom restrict g4 vertical_motion;
        bottom damp g1 rotation;
        bottom damp g2 rotation;
        bottom damp g3 rotation;
        bottom damp g4 rotation;
   };
```

There are several major improvements in the new design. The number of modules is reduced while all the essential functions of the compass are preserved. In the module 'case', 'top' is now snapped onto 'bottom' directly. The 'bracket' is connected to the 'top' also by snapping. There are no screws in the new design. The spring washers, cups, and covers are also eliminated. The gears are mounted in the bottom directly. The split-end of the gear shafts restrict the both rotation and vertical movement of the gears (Figure 6.3). The bearing is integrated with the magnetic bar. The resultant complex geometry can be generated by using the powder-metallurgy technique. As a result, the dial-shell can simply be pressed onto the magnet bar.

FRS supports parametric design also by allowing generic components. In the original design, four gears have most their dimensions in common, and a generic component 'gear' is created in the new design. The declaration of the generic component 'gear' is given below. G1, g2, G3, and G4 are its variations.

```
component gear(shaft:length)
  { geometry
      gear_head( d=0.5, h=0.1);
      shaft( d =0.0625, length = 0.1875);
      slot( width = 0.02, length = 0.2)
        { damp gear rotation };
      rim_edge( diameter = 0.07 )
        { restrict gear vertical_motion };
    material
      plastic;
    manufacture
      inject_mold;
  }
```

In the module 'case', the components g1, g2, g3, and g4 are now treated as instantiations of the generic component 'gear' and their designs can be derived by substituting the value of the parameter 'shaft:length'.

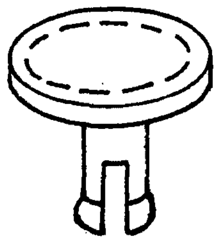


Figure 6.3 Modified Design of Gear

CHAPTER 7 SUMMARY AND FURTHER WORK

7.1 ACHIEVEMENTS AND LIMITATIONS

FDL is the first mechanical design language which allows computer layout configurations of mechanical products at the conceptual design stage. A number of test products have been analyzed using FRS [96,97]. In general these trials have yielded useful results though new users tend to need help in developing useful product descriptions and analysis commands.

FRS provides a flexible way to express abstract design concepts. FRS provides a logical way to integrate design activities, such that functional specifications and manufacturing constraints can be considered simultaneously. FRS also provides a natural and versatile way for human/machine interaction. By allowing users to define their own vocabulary and parsing rules, FRS can be readily adapted to the users' particular needs.

FRS enables designers to express design intentions and design concepts at higher abstract levels. As a result, mechanical design can be carried out by computers in a way that resembles human cognitive processes closer than ever before.

FRS, however, was not intended to be a fully "automatic" design system, which is still far beyond our ability at this moment. Therefore, the quality of designs generated with FRS still largely depends on the factors crucial in any design environment, the knowledge and the creativity of engineers.

7.2 FUTURE WORK

What we have done so far indicates that a design language is an effective way of approaching design automation. However, this framework can be improved in many aspects,

(1) The FRS system has been used in our Computer-integrated Manufacturing classes for several quarters. It is found that for users with little experience in design, guidelines are needed to help in decomposing a product and writing function sentences.

(2) More design rules need to be developed involving different aspects of design, such as, manufacturing considerations, common mechanical design practices, and the particular features of FDL.

(3) The scope of the system should be extended. At a higher level, it should be able to deal with function of modules, and at a lower level, it should be able to deal with features of the parts. In order to do this, a direct connection between 3-D geometric representation and

high-level functional description is needed.

In conceptual analyses, many relations are reviewed, such as kinematics and force balance. FRS can also be used in these analyses.

(4) To increase the power of FDL, more flexible and more extensive data models are needed for the diverse types of data present in a CAD/CAM environment.

(5) FRS allows the user to define his own vocabulary and parsing rules. Therefore, the maintenance of the integrity and the consistency are crucial to the future system.

In general, the author believes that further work in improving FDL and developing design languages for machine design will be useful in a variety of ways. In addition to providing a series of software tools which will be of direct benefit to the industrial designer, it will help further the course of design automation. It will improve our understanding of the complex processes involved in design, and will eventually lead to a sound science base for the 'art' of design.

REFERENCES

1. Dieter, G.E., "Engineering Design - A Materials and Processing Approach", McGraw-Hill Inc., NY, 1983.

2. Dixon, J.R., "Design Engineering: Inventiveness, Analysis, and Decision Making", McGraw-Hill, Inc., New York, 1966

3. Dixon, J. R., "Artificial Intelligence and Design: A Mechanical Engineering View", Proc. 5th AAAI, 1986, 872-877.

4. Shigley, Joseph E.,"Mechanical Engineering Design", 3rd, McGraw-Hill, Inc., New York, 1977.

5. Wolfe, R.N., et al, "Solid Modelling for Production Design", IBM. J. Res. Develop., Vol. 31, No. 3, May 1987, 277-295.

6. ASME, "Goals and Priorities for Research in Engineering Design - A Report to the Design Research Community", Dearborn, MI, 1986.

7. Allen, R.H., et al, "Using Hybrid Expert System Approaches for Engineering Applications", Engineering with Computers, Vol. 2, No. 2, Mar. 1987, 95-110.

8. Brown, D.C., "Capturing Mechanical Design Knowledge", Proc. 1985 ASME Computer in Engineering Conf., Boston, MA, Aug. 1985.

9. Brown, D. C. and Chandrasekaran, B., "Knowledge and Control for a Mechanical Design Expert System", IEEE Computer, Vol. 19, No. 7, July 1986, 92-100.

10. Mostow, J., "Toward Better Models of The Design Process", The AI Magazine, Vol. 17, No. 1, Spring 1985, 44-57.

11. Ullman, D.G. and Diettrich, T.A., "Mechanical Design Methodology: Implications on Future Developments of Computer- Aided Design and Knowledge-Based Systems", Engineering with Computers, Vol. 2, No. 1, Jan. 1987, 21-29.

12. "Flexible Manufacturing systems Handbook", The Charles Stark Draper Laboratory, Inc., Noyes Pub., Park Ridge, NJ, 1984.

13. Groover, Mikell P. and Emory W. Zimmers, Jr., "CAD/CAM Computer-Aided Design and Manufacturing", Prentice-Hall, Inc., Englewood Cliffs, NJ, 1984.

14. Harvany, J., Newman, W., and Sabin, M., "World Survey of CAD", CAD, Vol. 9, No. 2, Apr. 1977, 79-98.

15. Hatvany, J., "Computer-Aided Manufacture", CAD, Vol. 16, No. 3, May 1984, 161-165.

16. Eary, D. F., "Process Selection -- Key to Profit: Part 1 -- Six Steps to Process Analysis", The Tool and Manufacturing Enginner, Vol. 56, No. 4, Apr. 1966, 40-4

17. Eary, D. F., "Process Selection -- Key to Profit: Part 2 -- Process Analysis", The Tool and Manufacturing Engg., Vol. 56, No. 5, May 1966, 80-83.

18. Boothroyd, G., and Redford, A. H., "Mechanized Assembly", McGraw-Hill Publishing Co. Ltd., Maidenhead, Berkshire, England, 1966.

19. Baldwin, S. P., "How to Make Sure of Easy Assembly", The Tool and Manufacturing Engr., vol. 56, No. 5, May 1966, 76-78.

20. Boothroyd, G., and Dewhurst, P., "Design for Assembly - A Designer's Handbook", Dept. of Mech. Engr., Univ. of Mass., Amherst, MA, 1983.

21. Boothroyd, G., and Dewhurst, P., "Design for Assembly: Selecting the Right Method", Machine Design, Vol. 21, No. 11, Nov. 1983, 94-98.

22. Boothroyd, G., and Dewhurst, P., "Design for Assembly: Manual Assembly", Machine Design, Vol. 21, No. 12, Dec. 1983, 140-145.

23. Boothroyd, G., and Dewhurst, P., "Design for Assembly: Automatic Assembly", Machine Design, Vol. 21, No. 1, Jan. 1984, 87-92.

24. Boothroyd, G., and Dewhurst, P., "Design for Assembly: Robots", Machine Design, Vol. 21, No. 12, Feb. 1984, 72-76.

25. Bradyhouse, Richard G., "Design for Assembly and Value Engineering Helping You Design Your Product for Easy Assembly", Proc. 1984 SAVE Conf., 14-23.

26. Jakiela, M., Papalambros, P., and Ulsoy, A. G., "Programming Optimal Suggestions in The Design Concept Phase: Application to the Boothroyd Assembly Charts", J. Mech. Trans. Auto. Design, Vol. 107, Jun. 1985, 285-291.

27. Lund, T. and S. Kahler, "Product Design for Automatic Assembly", Programmable Assembly, W. B. Heginbotham (ed), IFS(Pub.) Ltd., UK., Springer-Verlay, 1984, 53-81.

28. Dargie, P. P., K. Parmeshwar, and W. R. D. Wilson, "MAPS-1: Computer-Aided Design System for Preliminary Material and Manufacturing Process Selection", ASME Trans. J. Mech. Design, Vol. 104, 1982, 126-136.

29. Miaw, D. C. and Wilson, W. R. D., "Use of Figures of Merit in Computer-Aided Material and Manufacturing Process Selection", J. Mechnical Design, 1982, 806-815.

30. Lai, K. and Wilson, W. R. D., "Computer Aided Material Selection and Process Planning", Proc. 13-th NAMRC, Berkeley, CA., May 1985, 505-508.

31. Bittence, John C., "When Computers Select Materials", Materials Engineering, Vol. 100, No. 1, Jan. 1983, 38-42.

32. Bittence, John C., "Property Data Bases are Coming Your Way", Material Engineering, Aug. 1984, 40-45.

33. Chao, Nieh-Hua, "The Application of A Knowledge-Based System to Design For Manufacture", 1985 IEEE Intl. Conf. Robotics and Automation, IEEE Computer Society Press, Silver Spring, MD, 182-185.

34. Dwivedi, S. N., and Klein, B. R., "Design for Manufacturability Makes Dollars and Sense", CIM Review, Vol. 3, No. 1, Spring 1986, 53-59.

35. Harrington, J.Jr., "Understanding the Manufacturing Process: Key to Successful CAD/CAM Implementation", Marcel Dekker, Inc., New York, 1984.

36. Kilhoffer, A. R. and Kempf, K. G., "Designing for Manufacturability in Riveted Joints", Proc. 5th AAAI,

1986, 820-824.

37. Mcalpine, G., "CAD Based - Manufacture of Parts", Proc. 1985 IEEE Int. Conf. Robotics and Automation, IEEE Computer Press, Silver Spring, MD., 1985, 389-390.

38. Requicha, Aristides A. G., "Representations for Rigid Solids: Theory, Methods, and Systems", Computing Surveys, Vol. 12, No. 3, May 1980, 437-464.

39. Requicha, A. A. G. and Voelcker, H. B., "Solid Modeling: A Historical Summary and Contemporary Assessment", IEEE CG&A, Vol. 2, No. 3, Mar. 1982, 9-23.

40. Asimov, M., "Introduction to Design", Prentice Hall Pub. Inc., Englewood Cliffs, NJ, 1962.

41. Eastman, C. M., "Recent Developments in Representation in the Science of Design", Proc. 18th Design Automation Conf., 1981, 13-21.

42. Yoshikawa, H., "General Design Theory and a CAD System", Man-Machine communication in CAD/CAM, North-Holland, 1981.

43. Yoshikawa, Hiroyuki, "Automation of Thinking in Design", Computer Applications in Production and Engineering, E. A. Warman (ed), North-Holland Pub. Co., Amsterdam, Holland, 1983. 405-417.

44. Freeman, P., and Newell, A., "A Model for Functional Reasoning in Design", Proc. 2nd Int. conf. AI, London, England, 1971. 621-633.

45. Fulton, R. E., "A Framework for Innovation", CIME, Vol. 5, No. 2, Mar. 1987, 26-40.

46. Mistree, F. and Muster, D., "The Decision Support Problem Techniques for Design", Proc. 1986 ASEE, 1986, 117-125.

47. Mitchell, T., Steinberg, L. and Shulman, J., "A Knowledge-Based Approach To Design", IEEE Trans, Pattern Analysis and Machine Intelligence, Vol. 7, No. 5, Sep. 1985, 502-510.

48. Mittal, S. and Araya, A., "A Knowledge-Based Framework for Design", Proc. 5th AAAI, 1986, 856-865.

49. Nadler, Gerald, "Systems Methodology and Design", Mechanical Engineering, Vol. 108, No. 9, Sep. 1986, 84-88.

50. Dixon, J. R., et al, K., "Dominic I: Progresss Toward Domain Indepenedence in Design By Iterative Redesign", Engineering with Computers, Vol. 2, No. 1, Jan. 1987, 137-145.

51. Dixon, J. R., et al, "Expert Systems for Mechanical Design: Examples of Symbolic Representations of Design Geometries", Engineering with Computers, Vol. 2, No. 2, Mar. 1987, 1-10.

52. McDermott, J., "Rl: A Rule-based Configurer of Computer Systems", AI Magazine, Vol. 19, No. 1, Jan. 1982, 39-88.

53. Allen, R.H., "Design Guidelines for Expert Systems", Applications of Artificial Intelligence to Engineering Problems, Boarnet, Sriram, D. and Adey, R., (eds), Berlin, Heidelberg, New York, Springer, 651-658.

54. Vaghul, M., et al, "Expert Systems in a CAD Environment: Injection Molding Part Design as an Example", Proc. 1985 ASME Computers in Mechanical Engineering Conf., Boston, MA, Aug. 1985.

55. Proc. 1985 ASME Computer in Engineering Conf., Boston, MA, Aug. 1985.

56. Chandrasekaran, B., "Generic Tasks in Knowledge-Based Reasoning: Characterizing and Designing Expert Systems at the Right Level of Abstraction", Proc. IEEE Intl. Conf. on AI Applications, Dec. 1985.

57. Dym, C.L.,(ed), "Applications of Knowledge-Based Systems to Engineering Analysis and Design", ASME, New York, 1985.

58. Gero, J. S., "Bibliography of Books on Artificial Intelligence with Particular Reference to Expert Systems and Knowledge Engineering", CAD, Vol. 17, No. 9, Nov 1985, 463-464.

59. "Research in Progress on Knowledge-Based Engineering Systems", IEEE Software, Vol. 3, No. 3, Mar. 1986, 48-60.

60. Wright, P.K. and Bourne, D.A., "Manufacturing Intelligence", Addison-Wesley Pub. Inc., Reading, MA, 1988.

61. Agrawal, V. D., "The Linguistics of Design and Test", IEEE Design & Text, Vol. 1, No. 4, Apr. 1986, 8.

62. Nash, J.D., "Bibliography of Hardware Description Languages", ACM SIGDA Newsletters, Vol. 14, No. 1, Feb., 1984, 18-37.

63. Brown, D. C. and Chandrasekaran, B., "An Approach to Expert Systems for Mechanical Design", Proc. 1983 Conf. Trends. Applications Computer, 173-180.

64. Brown, D.C. and Chandrasakeran, B., "Expert Systems for a Class of Mechanical Design Activity", Proc. IFIP WG5.2 Working Conf. on Knowledge Engineering in Computer Aided Design, Budapest, Hungary, 1984.

65. Buchmann, Alejandro P., "Current Trends in CAD Data base", CAD, Vol. 16, No. 1, Jan. 1984, 123-126.

66. Eastman, C. M., "System Facilities for CAD Databses", Proc. 17th Design Automation conf., Jun. 1980, 50-56.

67. Eastman, C.M., "Database Facilities for Engineering Design", Proc. IEEE, Vo. 69, No. 10, Oct. 1981, 1249-1263.

68. Guttman, A. and Stonebraker, M., "Using a Relational Database Management System for Computer Aided Design Data", IEEE Database Engineering Bulletin, Vol. 5, No. 2, June 1982, 21-28.

69. Hardwick, M., "Extending the Relational Data Model for Design Applications", Proc. 21st DAC, 1984.

70. Hartzband, David J., "Enhancing Knowledge Representation in Engineering Databases", IEEE Computer, Vol. 18, No. 9, Sep. 1985, 39-48.

71. Kellogg, C., "Intelligent Assistants for Knowledge and Information Resource Management", Proc. 8th IJCAI, 1983, 170-173.

72. Kellogg, C., "From data Management to Knowledge Management", IEEE Computer, Vol. 19, No. 9, Sep. 1986, 75-84.

73. Rehak, D. H., and Howrad, H. C, "Interfacing Expert Systems with Design Database in Integrated CAD Systems", CAD, Vol. 17, No. 9, Sep. 1985, 443-454.

74. Kerschberg, L., (ed) "Expert Database Systems", Menlo Park, Ca, Cummings Pub., 1985.

75. Bic, Lubomir and Jonathan P. Gilbert, "Learning from AI: New Trends in Database Technology", IEEE Computer, Vol. 19, No. 3, Mar. 1986, 44-54.

76. Codd, E.F., "Extending the Database Relational Model to Capture More Meaning", ACM Trans. Database Systems, Vol. 4, No. 4, Dec. 1979, 397-434.

77. Fuchi, Kazuhiro, "Aiming for Knowledge Information Processing Systems", Logic Programming and its Applications, (ed) Michel van Caneghem and David H. D. Warren, Ablex Publishing Corporation, Norwood, NJ, 1986, 279-305.

78. Haskin, R.L. and Lorie, R., "On Extending the Functions of a Relational Database System", Proc. 1982 ACM SIGMOD Intl. Conf. Management of Data, June 1982, 207-212.

79. Smith, J.M. and Smith, D.C., "Data Abstractions: Aggregation and Generalization", ACM Trans. Database Systems, Vol. 2, No. 2, Jun. 1977, 105-133.

80. Wiederhold, G., "Knowledge and Database Management", IEEE Software, Vol. 1, No. 1, Jan. 1984, 63-73.

81. Lloyd, J.W., "Fundations of Logic Programming", Springer-Verlag Pub., Heidelberg, 1984.

82. Dahl, V., "On Database Systems Development Through Logic", ACM Trans. Database Systems, Vol. 7, No. 1, Mar. 1982, 102-123.

83. Gallaire, H. and Minker, J., (eds.), "Logic and Data Bases", Plenum Press, New York, 1978.

84. Gallaire, H. et al (Eds), Advances in Data Base Theory, Vol. 1, Plenum Press, New York, 1981.

85. Gallaire, H., Minker, J., and Nicolas, J.-M., "Logic and Databases: A Deductive Approach", ACM Computing Surveys, Vol. 16, No. 1, Jan. 1984, 153-185.

86. Korth, H, F., "Extending the Scope of Relational Language", IEEE Software, Vol. 3, No. 1, Jan. 1986, 19-28.

87. Ullman, J. D., "Implementation of Logical Query Languages for Databases", ACM Trans. Database Systems, Vol. 10, No. 3, Sep. 1985, 289-321.

88. Henschen, L. and Naqvi, S., "On Compiling Queries in Recursive First-Order Databases", J. ACM, Vol. 31, No. 1, Jan. 1984, 47-85.

89. Zaniolo, C., "The Representation and Deductive Retrieval of Complex Objects", Proc. VLDB 85, Stockholm, Aug. 1985, 458-469.

90. Date, C.J., "An Introduction to Database Systems", (3rd)., Addison-Wesley, Reading, MA, 1981.

91. Bancilhon, F. and Khoshafian, S., "A Calculus for Complex Objects", Proc. ACM SIGACT/SIGMOD Symposium on Principles of Database Systems, Cambridge, MA, 1986.

92. Lorie, R. A. and Plouffe, W., "Complex Objects and Their Use in Design Transactions", Proc. 1983 IEEE Annual Meeting - Database Week, 1983, 115-121.

93. Patrick, V., Khoshafian, S., and Copeland, G., "Implementation Techniques of Complex Objects", Proc. 12th Intl. Conf. VLDB, Kyoto, Aug. 1986. 101-110.

94. Gallagher, C.C. and Knight, W.A., "Group Technology", Butterworths, London, 1973.

95. Wilson, R. and Henry, R., "Introduction to Group Technology in Manufacturing and Engineering", SME, Dearborn, MI, 1980.

96. Lai, K., "Mechanical Design Simplification Using Function Description Language", Proc. 15th North American Manufacturing Research Conference(NAMRC), May 1987, 615-620.

97. Lai, K. Lai and W.R.D. Wilson, "FDL - A Language for Function Analysis and Rationalization in Mechanical Design", Proc. 1987 ASME Intl. Conf. Computers in Engineering, New York, Aug. 1987, 87-94.

APPENDIX 1 GRAMMER OF FUNCTION DESCRIPTION LANGUAGE

```
FDL          : table_manipulation
             | component_declaration
             | module_declaration
             | vitual_relation_definition
             | analysis_command
             | design_rule
             ;

name, value  : string
             ;

table_manipulation
             : DEFVERB string ';'
             | DEFPREP string ';'
             | DEFPROC string ';'
             | DEFTAB  string '(' attributes ')' ';'
             | ADDTAB  string '(' values ')' ';'
             ;

attributes   : name
             | attributes ',' name
             ;

values       : values ',' value
             | value
             ;

component_declaration
             : COMPONENT head component_body ';'
             ;

head         : name '(' parameters ')'
             ;

parameters   :
             | parameters ',' entity
             | entity
             ;

entity       : entity ':' name
             ;

component_body
             : '{' geomtry_specification
                   material_specification
                   manufacture_specification
               '}'
             ;
```

```
geomtry_specification
                : GEOMETRY surfaces ';'
                ;

surfaces        : surfaces ';' surface
                | surface
                ;

surface         : name '(' dimensions ')' usage
                ;

dimensions      :
                | dimensions ',' dimension
                | dimension
                ;

dimension       : feature '=' value
                ;

usage           :
                | '{' v_o_phrases '}'
                ;

v_o_phrases     : v_o_phrases ';' v_o_phrase
                | v_o_phrase
                ;

material_specification
                :
                | MATERIAL name ';'
                ;

manufacture_specification
                :
                | MANUFACTURE processes ';'
                ;

processes       : processes ';' process
                | process
                ;

process         : verb_phrase
                ;

module_declaration
                : MODULE name module_body ';'
                ;
```

```
module_body : '{' element_list
                 function_specfication
              '}'
            ;

element_list
            : ELEMENT element_specifications ';'
            ;

element_specifications
            | element_specifications ';' element_specification
            | element_specification
            ;

element_specification
            : name quantifier
            | component_declaration quantifier
            | component_instantiation quantifier
            ;

component_instantiation
            : name ':' component_type '(' parameter_assignments ')'
            ;

component_type
            : string
            ;

quantifier  :
            | '*' string
            ;

function_specfication
            : FUNCTION sentences ';'
            ;

sentences   : sentences ';' sentence
            | sentence
            ;

sentence    : string v_o_phrase
            ;

v_o_phrase  : verb_phrase obj_phrase
            | verb_phrase adv_phrase
            ;

noun_phrase : noun_phrase string
            | string
            ;
```

```
obj_phrase   :
             | prep_phrase obj_phrase
             ;

prep_phrase : PREP noun_phrase
             ;

adv_phrase   : 'to' VERB noun_phrase
             ;

verb_phrase : VERB noun_phrase
             ;

vitual_relation_definition
             : relation_name '(' attributes ')' ':-'
                 component_relations
                 modification ';'
             ;

component_relations
             : component_relations ',' component_relation
             | component_relation
             ;

component_relation
             : name '(' attribute_assignments ')'
             ;

attribute_assignments
             : attribute_assignments ','
               attribute_assignment
             | attribute_assignment
             ;

attribute_assignment
             : name '=' name
             | name
             ;

modification:
             | WHERE constraints
             ;

constraints : '(' constraints ')'
             | constraints AND constraints
             | constraints OR  constraints
             | NOT constraints
             | constraint
             ;
```

```
constraint   : entity rel_optr entity_derivation
             ;

entity_derivation
             : entity math_optr value
             | '-' entity
             | entity
             | value
             ;

math_optr    : '+'
             | '-'
             | '*'
             | '/'
             ;

rel_optr     : '=='
             | '!='
             | '<='
             | '<'
             | '>='
             | '>'
             | '<=>'        /* set equality   */
             | '<<'         /* set enclousure */
             | '>>'
             ;

analysis_command
             : ANALYZE entities USING rules ';'
             ;

entities     : entities ',' entity
             | entity
             ;

rules        : rules ',' rule
             | rule
             ;

rule         : name
             | rule_definition
             ;

design_rule  : RULE name rule_definition ';'
             ;
```

```
rule_definition
             : '{' virtual_relation_definitions
                   if_then_sentences
               '}'
             ;

if_then_sentences
             : if_then_sentences ';' if_then_sentence
             | if_then_sentence
             ;

if_then_sentence
             : IF conditions THEN actions
             | IF conditions THEN actions ELSE actions
             ;

conditions   : '(' conditions ')'
             | conditions AND conditions
             | conditions OR conditions
             | NOT conditions
             | condition
             ;

condition    : entity rel_optr entity_derivation
             | ling_optr '(' attribute_assignments ')'
             ;

ling_optr    : string
             | '!' string
             ;

actions      : actions ',' action
             | action
             ;

action       :
             | MSG
             | '{' analysis_command '}'
             | '{' if_then_sentence '}'
             ;
```

APPENDIX B DESIGN RULES

```
rule modularization
  { ancestor(a1, e1, v, a2, e2):-
      element(module=a1, element=e1),
      element(module=a2, element=e2),
      function(sub=e1, verb = v, obj=e2)
    where
      ancestor:a1 != ancestor:a2;

    if ancestor(e1 = $1)
    then
"-- $1 $ancestor:v $ancestor:e2 from another module
    $ancestor:a2, can it be modified by modularization
    principle ?";
  };

rule integration
  { if ==($1:function:obj) and
       similar_function( x1 = $1:function:verb,
                         x2 = $2:function:verb)
    then
"-- $list, They perform identical function on same
    element $1:function:obj, can they be integerated ?",
"-- If $1 is eliminated, how $1:function:obj be
    $1:function:verb ?",
"-- If $2 is eliminated, how $2:function:obj be
    $2:function:verb ?";
  };

rule distribution
{ if !=($1:function:verb) and
     $1:function:verb == $2:function:verb and
     $1:function:obj == $2:function:obj
  then
"-- $list have common function $1:function:verb on the same
    object $1:function:obj, but they cannot be integrated
    simply because they have other functions.  However,
    improvement can be made by redistributing function
    $1:function:verb between them.";
};

rule rationalization
{ if ==($1:function:verb) and ==($1:function:obj)
  then
"-- $list
    They perform identical function on same element
    $1:function:obj, can they be integerated ?"
  else
```

```
      { if $1:function:verb == $2:function:verb and
            $1:function:obj == $2:function:obj
        then
        "-- $list have common function $1:function:verb on
              the same object $1:function:obj, but they cannot
              be integrated simply because they have other
              functions.  However, improvement can be made by
              redistributing function $1:function:verb between
              them."
    };
};

rule parametric_design
{ if ==($1:geometry) and
      !=($1:dimensions)
  then
"-- $list shares common geometric feature, but different
    dimensions, Can they be re-designed using
    parameterization principle ?";
};

rule standard_dimension
{ if !=($1:dimensions) and similar( x = $1, y = $2)
  then
"-- $list
   Can these geometric features using common dimensions?";
};

rule fastener_less
{ if assembly( x = $1:function:verb) and
      fastener( x = $1:function:pobj1 )
  then
"-- Can $1:function:pobj1 be eliminated and
    $1 and $function:obj be assembled directly ?";
};

rule less_machining
{ if ==($1:manufacture:geometry) and
      !=($1:manufacture:process)
  then
"-- $list are similar features,
  can they be machined by the same manufacturing process?";
};

rule direct_chain
{ if function(sub= $1) and function:prep1 != _ then
"-- Can $1 be eliminated and
 -- Can $function:obj $function:verb $function:pobj1
    directly ?
 -- Can $1 be combined with $function:obj or
    $function:pobjl ?";
};

rule direct_assembly
{ if assembly( x = $1:function:verb) and
     $1:efunction:prepl != _
  then
"-- Can $1 be eliminated and
    $1:function:obj $1:function:verb $1:function:pobjl
    directly ?";
};

rule direct_restrict
{ if restrict(x = $1:function:verb) and
     ( function:prepl == by or
       function:prepl == through )
  then
"-- Can $1:function:pobjl be eliminated and
    $1 $function:verb $function:obj directly ?";
};

rule implied_chain
{ chain(x, v1, y, v2, z):-
    function(sub=x, verb=vl, obj=y),
    function(sub=y, verb=v2, obj=z)
  where
    chain:vl == chain:v2;

  if chain(x=$1) and !transfer( x = chain:vl) then
"-- There is a chaining between $1 $chain:vl $chain:y and
    then $chain:z, Improvement can be made:
    1. combining all three together
    2. combining two of three, elimenating the third";
};

rule material_cost
{ if expensive(x = $1:material)
  then
" -- Can $1 be made by cheaper material than $1:material ?";
};

rule elimination
{ if function( sub = $1)
  then
"-- Can $1 be eliminated, if so,  how can function
    $1:function:verb on $1:function:obj be preserved ?";
};
```

```
rule permanent_assembly
  { if assembly( x = $1:function:verb) and
        $1:efunction:prepl != _
    then
"-- Can $1 be eliminated and $1:function:obj and
    $1:function:pobjl be assembled replace by some other
    permanent connection such as adhesive ? ";
  };

rule force_chain
{ trans_chain(x, vl, y, v2, z):-
    function(sub=x, verb=vl, obj=y),
    function(sub=y, verb=v2, obj=z);

  if trans_chain( y = $1) and
     !transfer( x = trans_chain:v2) and
     transfer( x = trans_chain:vl )
  then
"-- To which part $1 transfer force ?
    How the force transered by $trans_chain:x on it can be
    balanced ?";
};

rule direct_restrict
{ if restrict(x = $1:function:verb) and
     ( function:prepl == by or
       function:prepl == through )
  then
"-- Can $1:function:pobjl be eliminated and
     $1 $function:verb $function:obj directly ?";
};
```

APPENDIX C OUTPUT MESSAGES OF FRS ON THE INITIAL DESIGN OF COMPASS

COMMAND:

```
  analyze
    compensator:element,
    case:element,
    dial_capsule:element
  using
    modularization;
```

OUTPUT:

```
Analyze objects ...
(1) compensator      :element
Using rules
(1) modularization

Analyze objects ...
(2) case             :element
Using rules
(1) modularization

-- bottom hold cup from another module compensator,
   can it be modified by modularization principle ?

-- bottom hold gear3 from another module compensator,
   can it be modified by modularization principle ?

-- bottom hold gear4 from another module compensator,
   can it be modified by modularization principle ?

-- bottom hold spring_washer3 from another module
   compensator,
   can it be modified by modularization principle ?

-- bottom hold spring_washer4 from another module
   compensator,
   can it be modified by modularization principle ?

-- bottom support cover from another module compensator,
   can it be modified by modularization principle ?

-- bracket connect cover from another module compensator,
   can it be modified by modularization principle ?

-- top hold dial_capsule from another module compass,
   can it be modified by modularization principle ?
```

COMMAND:

```
  analyze
    compensator:element,
    case:element,
    dial_capsule:element
  using
    parametric_design;
```

OUTPUT:

```
Analyze objects ...
(1) compensator      :element
Using rules
(1) parametric_design

-- compensator:gear1, compensator:gear2, they shares common
   geometric feature, but different dimensions, Can they be
   re-designed using parametric design principle ?

-- compensator:gear1, compensator:gear3, they shares common
   geometric feature, but different dimensions, Can they be
   re-designed using parametric design principle ?

-- compensator:gear1, compensator:gear4, they shares common
   geometric feature, but different dimensions, Can they be
   re-designed using parametric design principle ?

-- compensator:gear2, compensator:gear3, they shares common
   geometric feature, but different dimensions, Can they be
   re-designed using parametric design principle ?

-- compensator:gear3, compensator:gear4, they shares common
   geometric feature, but different dimensions, Can they be
   re-designed using parametric design principle ?
```

COMMAND:

```
  analyze
    compensator:element,
    case:element,
    dial_capsule:element
  using
    direction_chain;
```

OUTPUT:

```
Analyze objects ...
(1) case                :element
Using rules
(1) direct_chain

-- Can bracket be eliminated ?
    Can compass adjust
        cover directly ?
    Can bracket be combined with compass or
        cover ?

-- Can bracket be eliminated ?
    Can cover connect
        screw directly ?
    Can bracket be combined with cover or
        screw ?


Analyze objects ...
(1) dial_capsule        :element
Using rules
(1) direct_chain

-- Can bearing be eliminated ?
    Can dial_shell connect
        magnet_bar directly ?
    Can bearing be combined with dial_shell or
        magnet_bar ?
```

COMMAND:

```
  analyze
    compensator:element,
    case:element,
    dial_capsule:element
  using
    implied_chain;
```

OUTPUT:

```
Analyze objects ...
(1) case                :element
Using rules
(1) implied_chain
```

```
-- There is a chaining between bottom hold cup and then
   spring_wahser1, improvement can be made:
   1. combiningg all three together
   2. combiningg two of three, elimenating the third

-- There is a chaining between bottom hold cup and then
   spring_wahser2, improvement can be made:
   1. combining all three together
   2. combining two of three, elimenating the third
```

COMMAND:

```
  analyze
    compensator:element,
    case:element,
    dial_capsule:element
  using
    direct_assembly;
```

OUTPUT:

```
Analyze objects ...
(1) compensator:element
Using rules
(1) direct_assembly

-- can top connect bracket directly ?

-- can top and bracket be combined ?

-- can top hold bottom directly ?

-- can top and bottom be combined ?


Analyze objects ...
(3) dial_capsule    :element
Using rules
(1) direct_assembly

-- Can bearing be eliminated and
   Can dial_shell connect magnet_bar directly ?
   Can bearing be combined with dial_shell or magnet_bar ?
```

```
COMMAND:

  analyze
    compensator:element
  using
    direct_restrict;

OUTPUT:

Analyze objects ...
(1) compensator:element
Using rules
(1) direct_restrict

-- Can spring_washes be eliminated and cover restrict
   g1 directly ?

-- Can spring_washes be eliminated and cover restrict
   g3 directly ?

-- Can spring_washes be eliminated and cup restrict g2
   directly ?

-- Can spring_washes be eliminated and cup restrict g4
   directly ?


COMMAND:

  analyze
    case:element
  using
    fastener_less

OUTPUT:

Analyze objects ...
(1) case            :element
Using rules
(1) fastener_less

-- Can screw be eliminated and
   bracket and cover be assembled directly ?
```

```
COMMAND:

  analyze
    compensator:element,
    case:element,
    dial_capsule:element
  using
    elimination;

OUTPUT:

Analyze objects ...
(1) compensator      :element
Using rules
(1) elimination

Can cover be eliminated, if so,
  how can function constrain on gear1 be preserved ?

Can cover be eliminated, if so,
  how can function constrain on gear2 be preserved ?

Can cup be eliminated, if so,
  how can function constrain on gear3 be preserved ?

Can cup be eliminated, if so,
  how can function constrain on gear4 be preserved ?

Can cup be eliminated, if so,
  how can function hold on spring_wahser1 be preserved ?

Can cup be eliminated, if so,
  how can function hold on spring_wahser2 be preserved ?

Can spring_washer1 be eliminated, if so,
  how can function damp on gear1 be preserved ?

Can spring_washer1 be eliminated, if so,
  how can function restrict on gear1 be preserved ?

Can spring_washer2 be eliminated, if so,
  how can function damp on gear2 be preserved ?

Can spring_washer2 be eliminated, if so,
  how can function restrict on gear2 be preserved ?

Can spring_washer3 be eliminated, if so,
  how can function damp on gear3 be preserved ?
```

```
Can spring_washer3 be eliminated, if so,
  how can function restrict on gear3 be preserved ?

Can spring_washer4 be eliminated, if so,
  how can function damp on gear4 be preserved ?

Can spring_washer4 be eliminated, if so,
  how can function restrict on gear4 be preserved ?


Analyze objects ...
(2) case            :element
Using rules
(1) elimination

Can bottom be eliminated, if so,
  how can function hold on cup be preserved ?

Can bottom be eliminated, if so,
  how can function hold on gear3 be preserved ?

Can bottom be eliminated, if so,
  how can function hold on gear4 be preserved ?

Can bottom be eliminated, if so,
  how can function hold on spring_washer3 be preserved ?

Can bottom be eliminated, if so,
  how can function hold on spring_washer4 be preserved ?

Can bottom be eliminated, if so,
  how can function support on cover be preserved ?

Can bracket be eliminated, if so,
  how can function adjust on compass be preserved ?

Can bracket be eliminated, if so,
  how can function connect on cover be preserved ?

Can top be eliminated, if so,
  how can function hold on dial_capsule be preserved ?


Analyze objects ...
(3) dial_capsule    :element
Using rules
(1) elimination
```

```
Can bearing be eliminated, if so,
  how can function connect on dial_shell be preserved ?

Can needle be eliminated, if so,
  how can function pivot on bearing be preserved ?

Can needle_base be eliminated, if so,
  how can function hold on needle be preserved ?

Can plate be eliminated, if so,
  how can function fix on needle_base be preserved ?
```

APPENDIX D USER'S GUIDE TO FRS

Following is a hand out used in the "Computer Integrated Manufacturing" class for lab work.

USER'S GUIDE FOR FUNCTION RATIONALIZATION SYSTEM(FRS)

FRS is a program that allows a designer to review the layout of a design. You are required to first create a data file that contains a description of the individual parts in the assembly as well as the function of each component in relation to others. When FDL is invoked using this data file, it gives you an output that confirms these descriptions. The second part, FDL2, will use another data file that contains the commands to analyse the design. The output of this part is in form of suggestions to improve the design.

The second part of the assignment is to implement some of the suggestions made by FDL2 and redesign the assembly. The objective of the redesign process is to reduce the number of parts and/or reduce the cost and/or simplify processing depending on the set of rules you use in your input file. You should run FDL and FDL2 on your new design also.

The input to FRS is supplied in three parts: Design Entity Description, Analysis Statement, and Analysis Rule.

1. HOW TO WRITE DESIGN ENTITY DESCRIPTIONS

Since most objects in mechanical design are hierarchically structured, two pre-defined hierarchical data structures are provided by FDL for describing design entities,viz; Component and Module.

A component is a single machine element. It can be defined using the format,

```
component name(parameters)
  { geometry
      surface_name
        (dimension = value, ...  );

    material
      material_name;
    manufacture
      process_name surface_name;
      ...
  };
```

Any of these items can be optional, depending the purpose of your use.

Example:

```
gear ()
  { geometry
      gear_head(d = 0.875, h=0.125);
      shaft(d = 0.0625, length = 0.5);
    material
      plastic;
    manufacture
      inject_mold gear;
  };
```

Module is a higher-level structure consisting of a group of components or modules. Its description includes a list of elements and function specification.
The list of elements may consist of components or modules. The function specification may provide a behavioral abstraction of the structural relationship, or, an assembly relationship between the elements of the module or between elements in other modules. The format for a module is,

```
module name
  { element
      name type quantity;
      ....

    function
      function_decription_sentence;
      ...
  };
```

Here is an example,

```
module case
  { element
      top component;
      bottom component;
      bracket component;
      bolt * 3;
```

```
    function
      top hold dial;
      bracket support compass to adjust compass position
      bracket support compass by cover;
      bracket connect top
      bottom hold cup;
      bottom hold gear3;
      bottom hold gear4;
  };
```

2. HOW TO WRITE FUNCTION DESCRIPTION SENTENCE

As it was shown in the example, the structure of function description sentences can be very flexible. The sample sentences are given in the examples given above. Notice that there are no inflection rules on tenses of verbs and on the number of nouns.

3. HOW TO WRITE ANALYSIS STATEMENT

The Analysis statement specifies the design entities to be investigated and the rules involved. Its format is,

```
analyze
  entity-1,
  ...
  entity-k using
  rule-1,
  ...
  rule-m;
```

An analysis rule may be applied to a particular design entity, such as,

```
analyze
  bottom using
  material_cost;
```

A rule can also applied to a class of design entities, such as,

```
analyze
  compensator:component using
  parameteric;
```

Where 'compensator:component' is called 'generic attribute'. It stands for all the components in the module 'componsator'. The system will search for all components and apply the rule to to each one of them.

When writing analysis commands, the entity must be uniquely specified. For example,

```
analyze
  cylinder using
  rule1;
```

is incorrect, because cylinder is an element and its ancestor's name ('module') should also be included in the name.

4. ANALYSIS RULE

For current lab assignment, the following rules are available in the FRS design library.

(1) MODULARIZATION

Each Functional Module should be assigned distinct tasks. Using several parts from different modules to perform a common function will result in additional complexity in manufacture. This .rule will check cross-module assembly relations.

(2) SIMILARITY

If several components perform identical functions on same components, some of them may be combined and some of them may be elimimated and its function can be ditributed to others. This rules performs this check.

(3) DIRECT_CHAIN

It identifies a relation of pattern 'x verb y to z' and suggests appropriate improvements.

(4) IMPLIED_CHAIN

It identifies indirect chaining relation between several components, such as 'x verb y' and 'y verb z'. Improvement may be made to eliminate one of them.

(5) PARAMETRIC

If there are several parts in a product which are similar in geometry, a generic part can be designed with common geometric features. By varying certain dimensions, a family of parts can be derived from that generic part. This rule will identify these components.

(6) STANDARD DIMENSION

If several similar machined features, e.g. holes, are present in a part, they should be the same size if possible. The advantage of doing so is obvious. To use this rule, the user must create a data table

x	y
hole1	hole2
hole1	hole2
hole2	hole3
slot1	slot2
slot1	slot3
slot2	slot3

x	y
hole1	hole2
hole1	hole3
hole2	hole3

where hole1, hole2 and hole3 are considered to be similar and the three slots are also considered to be similar.

When rule 'similar' is used, it must be written as,

```
analyze
  hydraulic:cylinder:geometry using
  standard_dim;
```

where the feature to be analyzed is 'geometry' of an element, hydraulic:cylinder in this case.

(7) FASTENER_LESS

A large amount of time in assembly is devoted to handling fasteners, such as bolts, nuts, cap screws, spring retainers, locking devices, and keys. Whenever possible, replace these removable connections with permanent connections such as adhesives. This rule will identify the components which are considered as fasteners. In the FDL libarary this is a data table

(8) LESS_PROCESS

If a component has several similar geometric features, such as holes machined by different processes, it is suggested that you use the least number of processes to achieve the same results. This rule allows you to identify these features and processes.

(9) MATERIAL_COST

To use this rule, the user needs to set up a table 'expensive', which stores all the materials which he thinks are expensive. Then the system will check all the materials against the table.

5. HOW TO RUN THE FRS SYSTEM

The FRS is divided into two parts, the first part takes the design entity description and stores it into the database, then the second part will be invoked to do the analysis. The step by step instructions are given below:

(1) Edit two files, one is for design entity description only, say 'ABC.DATA', and another is for analysis commands only, say 'ABC.RULE'. It is suggested that only lowercase letters be used, because the capital letter is recognized as an independent character.

(2) Type in command 'assign ABC.DAT sys$input'

(3) Type in command 'FDL', it generates a series of data tables.

(5) Type in command 'assign ABC.RULE sys$input'

(6) Type in command 'FDL2', it displays analysis messages on the function descriptions.

In FDL the following tables are referred to,

```
restrict(x)
fasten(x)
expensive(x)
similar(x)
```

Where tables 'restrict' and 'fasten' are defined in the system library. Before running FDL2, copy these two files into your directory. Tables "expensive" and "similar" can be created by the user as described in earlier sections. In addition, you should add these commands in your Design Entity Description file, if the corresponding rules are to be applied:

```
defrel fasten(x);
defrel restrict(x);
defrel similar(x);
defrel expensive(x);
```

so the table names can be stored into the system directory.

6. VERBS, PREPOSITIONS, AND PROCESSES IN FRS

The following are system defined vocabulary and cannot be changed by the user:

VERBS	DRIVE
	FIX
	GUIDE
	KEEP
	PUSH
	ADJUST
	CAST
	MOVE
	COMPENSATE
	SEAL
	SUPPORT
	HOLD
	CONNECT
	LINK
	PIVOT
	SENSE
	ROTATE
	RESTRICT
PREPOSITIONS	FROM
	AT
	BY
	IN
	ON
	OFF
	ONTO
	TO
	THROUGH
PROCESS	CAST
	PRESS
	INJECT_MOLD
	GRIND
	DRILL
	TURN
	FINISH_TURN
	BORE
	REAM
	FORGE
	MILL

The following files have been created for use but can be modified by the user if required.

```
file name RESTRICT.

          keep
          fix
          connect
          restrict
          constrain

file name FASTEN.

          fix
          screw
          clamp
```

VITA

NAME : KEWEI LAI

BIRTH : March 15, 1946. Shanghai, China

EDUCATION:

Ph.D. in Mechanical Engineering, June 1988
Northwestern University, Evanston, Illinois, U.S.A

M.S. in Mechanical Engineering, June 1984
Northwestern University, Evanston, Illinois, U.S.A

B.S. in Automotive Engineering, December, 1968
Tsinghua University, Peking, China

WORK EXPERIENCE:

SENIOR LECTURER, 1973-1982
Mechanical Engineering Department,
Qinghai Engineering and Agricultural Institute
Xining, China,

ENGINEER, 1968-1973
Qinghai Auto Parts Manufacture Inc.
Xining, China

PUBLICATIONS

1. Kewei Lai and Jyhwen Wang, "A Computational Geometry Approach To Geometric Tolerancing", To appear at 16th North American Manufacturing Research Conference (NAMRC), May 1988.
2. Kewei Lai, "An Extended Relational Database For Conceptual Analysis of Mechanical Design", Proc. 1987 ASME Design Automation Conference, Boston, MA, Sep. 1987
3. Kewei Lai and W.R.D. Wilson, "FDL - A Language for Function Analysis and Rationalization in Mechanical Design", Proc. 1987 ASME Intl. Conf. Computers in Engineering, New York, Aug. 1987, 87-94.
4. Kewei Lai, "Mechanical Design Simplification Using Function Description Language", Proc. 15th North American Manufacturing Research Conference(NAMRC), May 1987, pp. 615-620
5. Kewei Lai and W.R.D. Wilson, "Computer Aided Material Selection and Process Planning", Proc. 13th North American Manufacturing Research Conference(NAMRC), May 1985, pp. 505-508.