THE UNIVERSITY OF CALGARY

Development of a Feature-based Intelligent Design System

by

Swatantra Yadav

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE

DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF MECHANICAL AND MANUFACTURING ENGINEERING

CALGARY, ALBERTA

DECEMBER, 1998

ABSTRACT

This research focuses on developing a feature-based intelligent design system for improving the efficiency of product modeling. In this system, features are used as primitives for modeling products. Features are represented at two different levels – class and instance levels, corresponding to standard product library and special product data respectively. Instance features are generated using class features as their templates. Both the qualitative descriptions and quantitative descriptions are preserved in the features. The product described by features is associated with a 3D geometric model for representing the product geometry. A knowledge-based system has been developed for automated generation of instance features. The reasoning efficiency has been improved by introducing an inference mechanism that can select only partial knowledge base and database. The system has been implemented using Smalltalk, C++, and 3D Studio MAX. An industrial application has been developed using this system for designing building products.

ACKNOWLEDGEMENTS

First of all, I would like to thank my supervisor, Dr. D. Xue, for his excellent guidance and untiring support during the past 2 years. It has been a learning experience for me working with him, not only in academics, but also in other aspects of life. I would like to thank Dr. D. H. Norrie for giving me encouragement and help in the project for the Gienow Products Ltd.

I would like to thank my examining committee members, Dr. D. Xue, Dr. D. H. Norrie, and Dr. R. Kremer, for the amount of time they have put into reading this thesis.

I would also like to thank the Faculty of Graduate Studies, the Department of Mechanical and Manufacturing Engineering, and the Gienow Products Ltd. for their generous financial support during my studies at University of Calgary.

Special thanks are due to Lynn Banach for helping me in a number of ways. I would like to acknowledge the help received from the support staff of the Department of Mechanical and Manufacturing Engineering including Dan Forre, Khee Teck Wong and Nick Vogt for solving the computer-related problems. I would like to thank Mr. Z. Feng for early work on the Feature Modeling System.

My parents and my uncle Shri Dayaram Yadav deserve a special thanks for giving me encouragement, support and advice.

TABLE OF CONTENTS

Chapter 1

Introduction

This chapter provides a brief introduction to the existing technologies in the area of computer-aided design and shortcomings of these technologies. The objective of this research is outlined. The structure of this thesis is given at the end of this chapter.

1.1 Prologue

Industries in the last four decades have endeavored to increasingly computerize their operations. In today's information age, industries are competing at the global level. They are hard pressed to satisfy the requirements of the customers who want the best products, at the lowest price and requiring short product development lead-time. The industries meet these demands by relying more and more on computer-based technologies.

The use of computers in industry leads to shorter product development time, lower inventory level, better equipment utilization, improved quality, increased flexibility, and low production costs. Industries use powerful Computer Aided Design (CAD) systems in modeling product geometry. Computer Aided Process Planning (CAPP), Computer Numerically Controlled (CNC) machining, automated inspection using coordinate measuring machines, and automated assembly by industrial robots have revolutionized the manufacturing process. The emphasis is on automation of the product development process in order to meet the high demands of the international marketplace. This research is an effort to develop an intelligent design system that is capable of automating the process of product modeling.

1.2 Background

Automation in product design and manufacturing aims at improvement of design efficiency, productivity, and product quality, and reduction of production costs. Few developments in the history of manufacturing have had greater impact than the advent of computers. The development of computer-based design systems started with the work of creating drawings on CRT, for the Sketchpad system, by Ivan Sutherland in 1962 [Sutherland63]. Thereafter, computers have been used in a broad range of applications in mechanical design and manufacturing.

CAD systems allow the designers to design products without having to make costly and time consuming illustrations, models and prototypes. The functional performance of the product subjected to forces, fluctuations, and temperature variances, etc. can be simulated, analyzed, and tested accurately and effectively, at low costs. Rapid prototyping is a technique to produce prototypes of solid models using polymers or metal powder rapidly. CNC machines can be programmed to produce the components with complex geometry. Adaptive Control automatically adjusts the process parameters, such as force, feed rate, etc., to optimize production process. Optimal process-plans can be generated using CAPP. Group technology (GT) is a method to organize the parts with similar geometry in groups, thereby applying similar manufacturing process to the parts in the same group for improving manufacturing efficiency. Flexible manufacturing systems are used to integrate manufacturing cells into large units, which consist of several industrial robots serving a number of CNC machines, using central computers. Artificial Intelligence (AI) is a study of how to make computers to think, which at the moment humans do better [Rich91]. In design and manufacturing, AI is being used in the form of expert systems, neural networks, fuzzy logic, etc. to automate the design and manufacturing processes.

Among all these computer-based systems, CAD is first used to model the product data. The downstream product development data, such as manufacturing process, assembly process, etc., are generated from the CAD database. The CAD systems in industries today are primarily used for creating 2D and 3D product geometry. The CAD systems are also utilized as visual tools for generating shaded images and animated displays.

Previous research efforts in the area of CAD have mainly focused on geometric modeling of products. However, modeling of product geometry is only one design aspect conducted during detailed design stage. The real life design involves, among many other things, requirement identification, conceptualization, optimization of design parameters, modeling of geometry, engineering analysis, etc. The current CAD systems do not support these design activities fully.

Product design requires considerable human experience and decision making. Engineering designs are classified into two types: creative design and routine design. It is very difficult to automate the creative design process, due to the lack of understanding of the nature of creative design. However, many engineering designs are not exactly creative. In a routine design, the sequence of processes is well formulated, therefore being easily encoded by developing knowledge-based systems [Potts88, Mittal89, Ishii93].

In order to automate the process of product modeling, an intelligent system that has the capability of knowledge-based reasoning and decision making is required. Current CAD systems are not integrated with the knowledge-based systems, therefore the development of knowledge-based CAD must be conducted.

1.3 Problem Statements

Conventional CAD systems provide reliable functions for modeling design geometry. Some of these systems provide facility for defining quantitative relations among the geometric parameters. However, they still fall short of being complete design systems. The deficiencies of these systems are listed below:

- *Product modeling method is not efficient.*

  For instance, the object oriented programming concepts, such as abstraction and inheritance, are not well used in the present CAD systems. If similar products are defined as libraries and the actual product model is generated using the libraries as building blocks, the efficiency of product modeling can be improved considerably. In addition, if the product database libraries are organized in a hierarchical data structure of classes, the properties defined in a super-class can be inherited by its sub-classes automatically to improve the efficiency of modeling design libraries.

- *Non-geometric information cannot be modeled.*

  Existing CAD systems provide excellent geometric modeling functions. However, product geometry modeling is only a part of the design process. The whole design process includes conceptual design, detailed design, etc. Geometric model emerges only during late detailed design stage. Therefore, support of the whole design activities, especially the conceptual design, must be considered. The models generated during the early stages of design are usually represented as symbolic models. Therefore association of the symbolic model and geometric model is needed.

- *Product modeling process is not automated.*

  The conventional CAD systems are not intelligent, therefore being incapable of carrying out the intelligent design activities automatically. Design is a

process involving decision-making and reasoning. The knowledge of the designers should be implemented in the design system in order to enable the system to conduct design with minimum human interference.

1.4 Research Objective

The objective of this research is to develop a product modeling system that addresses the problems listed in Section 1.3. The motivation to achieve this objective is to develop a CAD system that supports all phases of the design process including conceptual design and detailed design. The system should provide efficient product modeling environment. Product geometry should be generated in symbolic form and this symbolic model should be associated with the geometric model. This design system should have a knowledge base and an inference engine that are used to automate the product modeling process. A special product modeling scheme should also be developed in this research for simple and lucid representation of product data and knowledge. The system should be composed of the following three modules.

1. *A feature-based design system:*

   The feature-based design system is used to generate the product descriptions. In this system, a product is described in terms of product building primitives called features. Features are described at two levels: class level and instance level, corresponding to generic library and special product data, respectively. Both qualitative descriptions and quantitative descriptions can be represented by these features. The efficiency of product modeling can be increased using the feature-based design system. This design system is implemented using an object oriented programming approach.

2. *A geometry representation system:*

   The design geometry representation system interprets the symbolic descriptions of the design model into geometric descriptions. To model the product geometry, a special representation syntax has been developed to succinctly describe product geometry in the symbolic form. The symbolic geometric data are extracted from the design module, and converted to 3D geometric model using a specially developed product geometry translation system.

3. *An intelligent system:*

   The intelligent system is used to automate the product modeling process using knowledge-based inference. A knowledge modeling platform is developed to encode and preserve the knowledge base, using a knowledge description language. During knowledge-based reasoning, only relevant knowledge base and database are selected for improving inference efficiency. The knowledge base is also modeled using an object oriented modeling approach.

The three modules described above form an integrated system where feature-based intelligent design can be conducted. The system architecture is shown in Figure 1.1.

1.5 Organization of this Thesis

This thesis consists of 9 chapters. Chapter 2 describes the detailed research background of this work. A study of the existing CAD tools is presented at the beginning of this chapter. The chapter introduces the concept of features and the feature modeling approach. Existing knowledge-based systems and their applications by other researchers are surveyed. Object oriented programming concepts used in the implementation of the system are discussed, with special

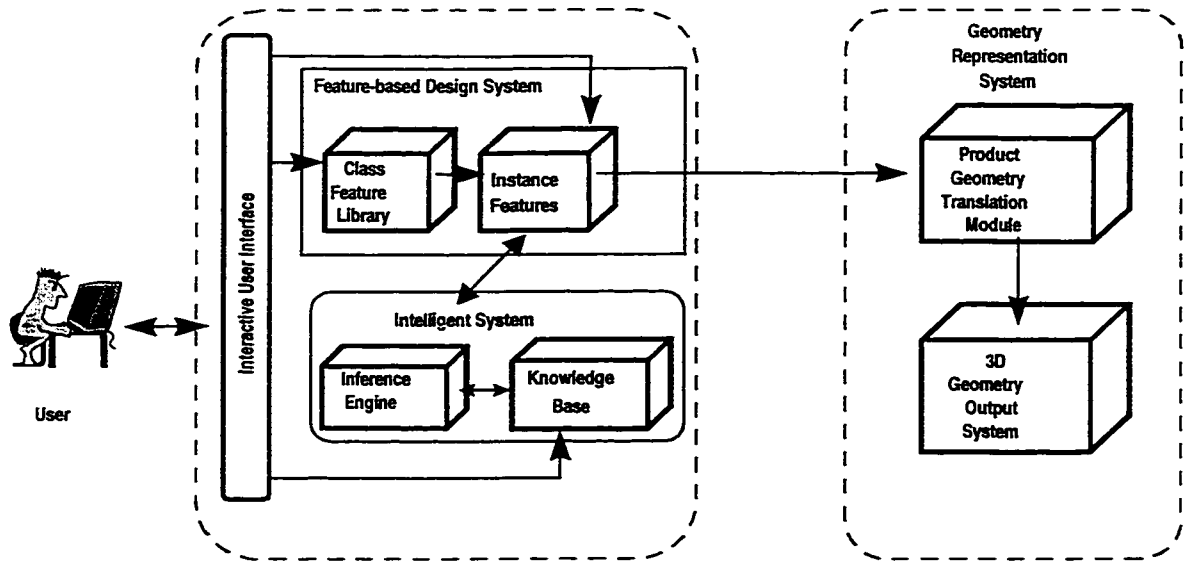


Figure 1.1: System architecture

focus on Smalltalk, an object-oriented programming language, which has been used in implementing the major portion of the system in this research. This work is a continuation of the research by Xue on development of an intelligent and integrated design system [Xue93, Xue94a, Xue94b, Xue96, Xue97]. Chapter 2 ends with the introduction of Xue's previous research.

Chapter 3 enumerates the functional requirements for the feature-based intelligent design and architecture of the feature-based intelligent design system. The system details are presented in Chapters 4, 5 and 6.

Chapter 4 of the thesis gives details of the feature-based design module. The structure of features and their applications in modeling products are discussed, with special focus on representation of qualitative and quantitative relations. The properties of hierarchy and inheritance, based upon object oriented programming, are also presented. The interface environment of the implemented system is also described.

Geometry representation is an important aspect of a CAD system. A special syntax has been developed for describing product geometry in the feature definitions. This representation scheme is introduced in Chapter 5. The chapter describes the syntax for generating 2D and 3D geometric primitives and applying geometric operations to them. Generation of 3D solid model is also given in this chapter.

Chapter 6 describes the intelligent system that is used for automatic product modeling. In this chapter, three issues are discussed. They are (1) the knowledge base modeling, (2) the database modeling, and (3) the knowledge-based inference.

Chapter 7 discusses the issues in the implementation of the feature-based intelligent design system.

Chapter 8 presents an industrial application for modeling building products using the developed feature-based intelligent design system.

Chapter 9 gives conclusions. The future work is also discussed in this chapter.

Chapter 2

Research Background

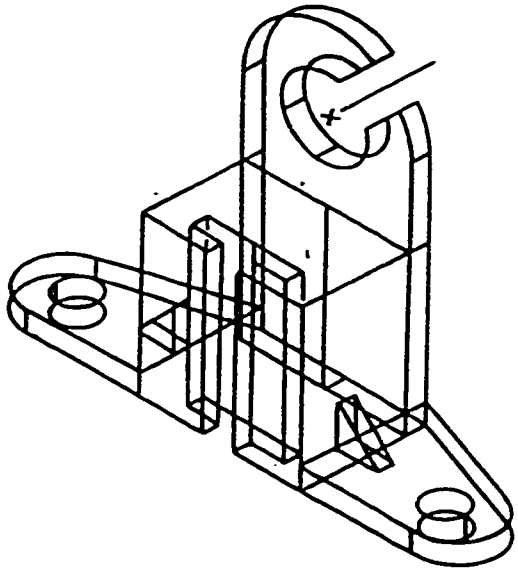
This chapter presents a general review of relevant technologies and research activities for the research introduced in this thesis. The areas covered in this survey include: CAD, feature modeling, knowledge-based intelligent design, and Object Oriented Programming (OOP). The capabilities and limitations of the conventional CAD tools are also discussed.

2.1 Conventional CAD Systems

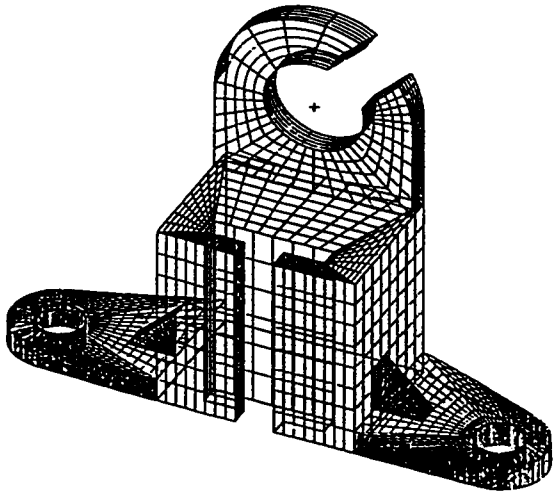
Conventional CAD systems primarily provide three functions: (1) geometric modeling, (2) computer graphics, and (3) design. These systems take the microscopic view of design where a design is represented and understood by its geometry. This section gives a survey of the geometric modeling functions in the conventional CAD systems. In the conventional CAD systems, geometric information is organized by databases such as relational databases, hierarchical database, and network database [Shenoy83]. Three types of geometric models are usually used. They are: (1) wireframe model, (2) surface model, and (3) solid model, as shown in Figure 2.1. These three types of geometric models are described in the following sections.

2.1.1 Wireframe Geometric Model

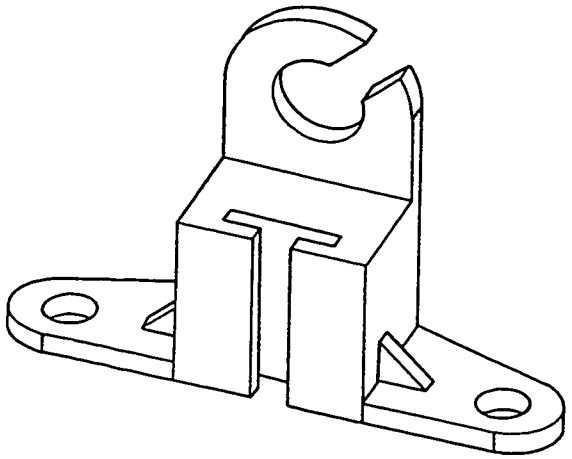
Wireframe geometric model is the simplest geometric model. It can be seen as a natural extension of traditional methods of drafting. Typical wireframe entities are analytic entities, such as points, lines, arcs, circles, ellipses, parabolas and hyperbolas, and synthetic entities, such as cubic splines, Bezier curves, and B-splines [Lee85]. Mathematical representation of curves in CAD systems is in form of parametric equations.



Wireframe model



Surface model



Solid model

Figure 2.1: Three types of geometric models [Zeid91]

The wireframe model requires much less computation time and memory than the surface and solid geometric models in modeling product geometry [Zeid91]. The wireframe model has many disadvantages, such as the ambiguity of geometry interpretation.

2.1.2 Surface Geometric Model

The surface model is an improvement of the wireframe model by providing product surface information. Surface entities used in these systems are analytic entities and synthetic entities [Mortenson85]. Analytic entities include plane surface, ruled surface, surface of revolution, and tabulated cylinder. Synthetic entities include bicubic hermite spline surface, B-spline surface, rectangular and triangular Bezier patches, Coon patches, and Gordon surface [Forest68].

The surface model provides better visualization functions such as hidden line/surface removal, shading, coloring, etc. The surface model is less ambiguous for representing product geometry compared with the wireframe model. Although the surface model and solid model have similar appearance, there is a fundamental difference between the two – the surface model doesn't store any topological information on the geometric elements.

2.1.3 Solid Geometric Model

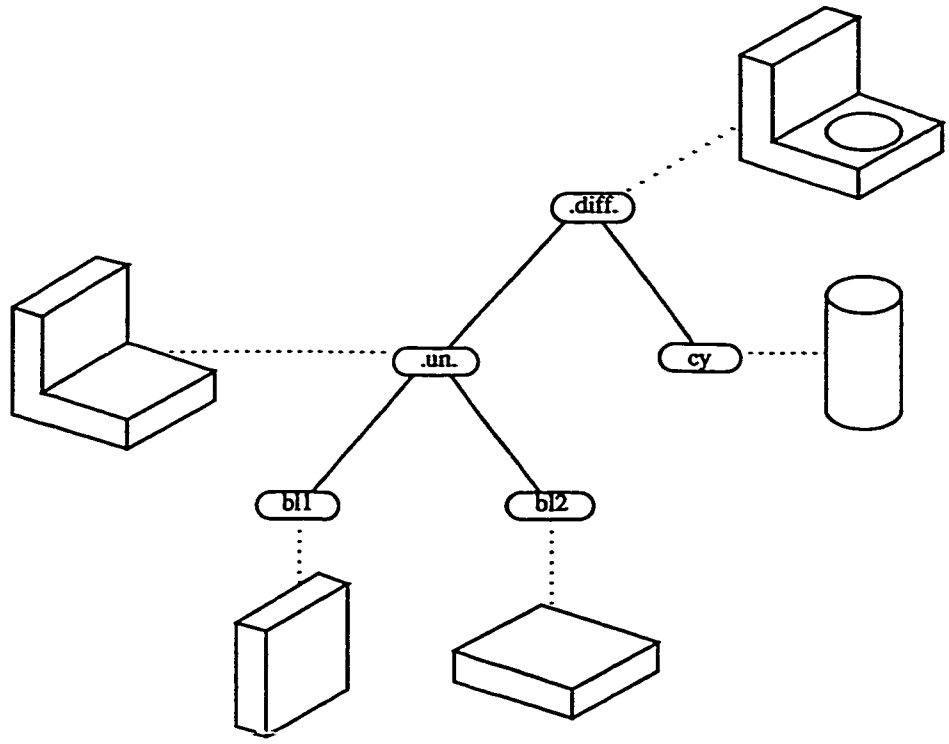
The solid geometric model provides the complete information of the 3D product geometry, such as topological relations among geometric elements including points, lines/curves, surfaces, therefore is better than the wireframe model and the surface model [Casale85]. Solid model based CAD tools were not accepted by design engineers until early 1980s due to the limitations on the speed of computations. But these limitations were fading away with rapid advances in computer software and hardware development [Krouse85]. Using this approach, the completeness of information in solid models makes CAD/CAM

systems suitable for automation and integration of design and manufacturing tasks such as interference analysis, Computer-Aided Process Planning (CAPP), machine vision, Finite Element Analysis (FEA), and mass property analysis [Tan86].

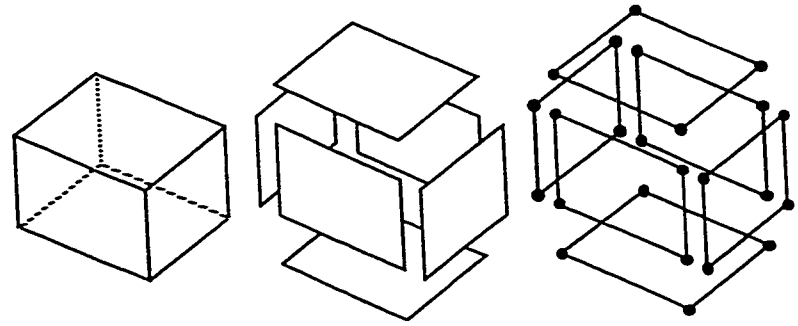
The solid model is built up using primitives, such as blocks, cylinders, cones, spheres, etc. In addition, geometric primitives can also be created by extruding and rotating 2D wireframe entities, such as circles and rectangles [Tan87]. Two or more primitives can be combined to form a compound solid by Boolean operations such as union and subtraction.

In the world of the solid model, space is divided into two regions: interior and exterior, separated by solid boundaries. A solid model of an object is defined mathematically as a point set S in three-dimensional euclidean space (E^3) [Eastman79]. With this theory as the background, there are many representation schemes to define solid models. These schemes are: half space model, boundary representation (B-rep) model, constructive solid geometry (CSG) model, sweeping, analytic solid modeling, cell decomposition, spatial occupancy enumeration, and octree encoding [Dave95, Zeid91]. Among these schemes, CSG and B-rep are used most often. Examples of CSG and B-rep are shown in Figure 2.2.

CSG models are developed based upon the notion that a physical object can be divided into a set of primitives that are combined in a certain way using Boolean operations to form the object [Putnam86]. Unlike B-rep models, topological information about intersecting edges and faces is not stored, but calculated when required. A CSG model is represented by a tree of nodes, as illustrated in Figure 2.2. The nodes at the bottom of this tree are geometric primitives, while the other nodes represent Boolean operations. The CSG representation scheme is concise and powerful with minimum storage



CSG model



B-rep model

Figure 2.2: CSG and B-rep solid models [Shah95]

requirements [Johnson86]. The biggest shortcoming of the CSG model is that it takes considerable time to derive necessary information such as the volume of the object or the visibility of surfaces.

A B-rep model is composed of faces, edges, and vertices of the object linked together in a way that topological consistency is maintained. In this scheme, the solid is bounded by a set of faces. These faces are regions or subsets of closed and orientable surfaces. A closed surface is one that is continuous without breaks. An orientable surface is one that is possible to distinguish the two sides of the surface. Euler operations are used to check the validity of B-rep models by counting the numbers of faces, edges, vertices, and so on [Wilson85]. All kinds of polyhedral and curved objects can be represented in an unambiguous way using B-rep models. Wireframe models can be derived from the B-rep models. The major disadvantage of B-rep model is that it requires large amounts of storage for representing a CAD database.

2.2 Feature Modeling

The conventional CAD systems are primarily used for modeling 3D and 2D product geometric information. Clearly modeling of design geometry is only part of the activities during the whole design process, which consists of conceptual design, embodiment design and detailed design. Geometric modeling serves for documenting design result during the last design stage. To support the designers during whole design stages, new CAD systems must be developed.

According to Shah [Shah95], geometric models have the following deficiencies:

1. The data available in the geometric models are at too low level. The geometric elements are organized only according to their topological

relations. To represent the functions behind geometry, the relations among geometric elements for particular purposes, including design and manufacturing, should also be defined.

2. The geometric model fails to capture the design intent and cannot be used to generate a manufacturing process, since geometric data are described at low level. To improve product modeling, the non-geometric information, such as design functions and manufacturing processes to produce geometry, should also be defined in the product model.

To solve the problems introduced above, the concept of feature, which is described by a collection of relevant geometric elements for a particular design or manufacturing purpose, has been introduced [Grayer76].

There is no single universally accepted definition for features. Researchers have given various overlapping definitions of features according to their perspectives and implementation needs. According to Shah [Shah95], a feature

- is a physical constituent of a part,
- is mappable to generic shape,
- has engineering significance, and
- has predicable properties.

A feature model provides combination of the following information [Shah95]:

- Generic shape (topology and/or geometry)
- Dimension parameters
- Constraints (parameters and/or relations)
- Default values of parameters

- Location (parameters and/or methods)
- Orientation (parameters and/or methods)
- Inheritance rules
- Tolerances
- Construction procedures
- Validation rules
- Non-geometric attributes

From the conventional geometry centered interpretation, a feature is a geometry for a special purpose such as a design functional performance or a manufacturing process [Hovart96]. Since these types of features are part of the descriptions in the conventional geometric model, they are referred to as form features or shape features. Another similar definition is given by Vickers [Vickers88] that a feature is a geometric abstraction represented by a collection of geometric and topological descriptions with commonly recognized names, shapes, and/or functions. Many other similar feature definitions can be found in references [Dixon88, Vickers88, Gindy89]. An example product and its composing features are illustrated in Figure 2.3.

In the Standard for the Transfer and Exchange of Product Model Data (STEP), form features are classified into the following categories: passages, depressions, protrusions, transactions, area features, and deformations. Some other classifications suggested by references [Shah88a, Shah89] are: precision features, technological features, material features, assembly features, cost features, etc.

In the manufacturing oriented feature definitions, a feature is considered as the partial product geometry to be produced by a certain manufacturing

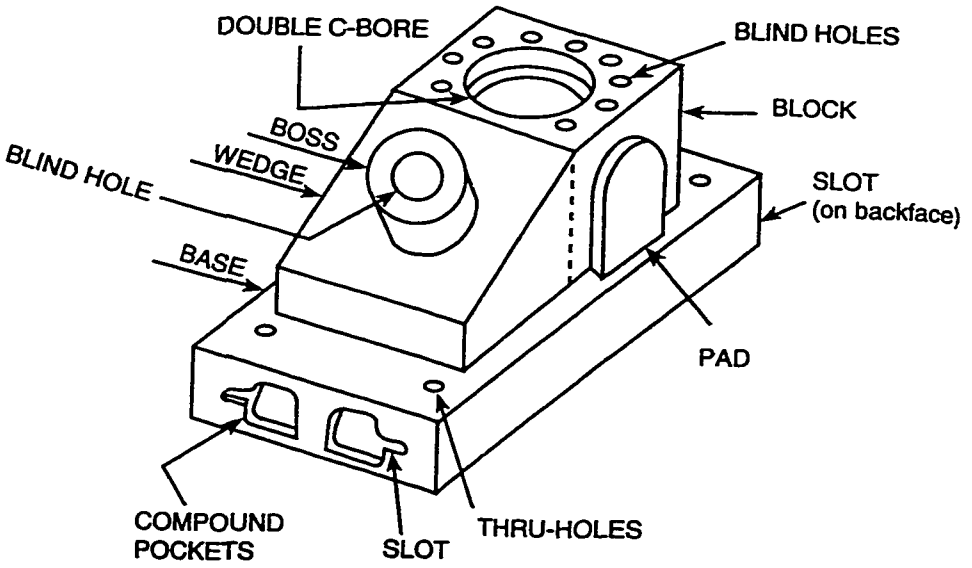


Figure 2.3: A product and its composing features [Shah95]

process [Shah86]. There are two approaches for modeling features: feature recognition and feature based design.

2.2.1 Feature Recognition

Feature recognition methods extract the manufacturing geometry from the CAD database for production process planning. Feature recognition is carried out either manually by users or automatically by computer programs.

2.2.1.1 Interactive Feature Recognition

In this approach as shown in Figure 2.4, first a geometric model is created using a geometric modeler. The user then picks relevant geometric elements from this geometric model using an interactive user-interface system to define the features to be manufactured.

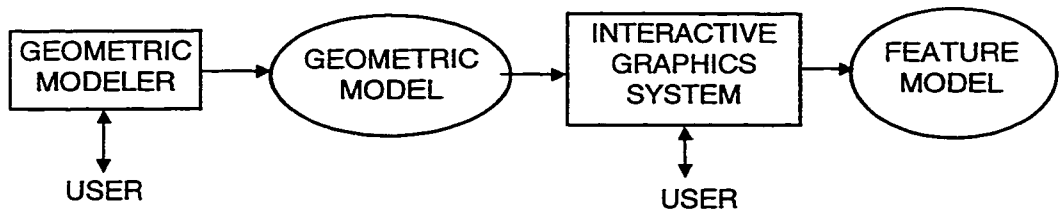


Figure 2.4: Interactive feature recognition approach [Shah95]

Geometric features can be extracted from B-rep model and CSG model. These two types of solid models have already been discussed in Section 2.1.3. When a B-rep model is used to define a geometric feature, the geometric entities (vertices, edges and faces) are picked up interactively by the user, grouped and associated with a feature name [Nau86]. The method to define geometric features from a CSG model was proposed by developing a data structure called VGraphs [Marisa88]. From the CSG graph, interior nodes (representing Boolean operations) and leaf nodes (representing geometric primitives) are selected and associated with the feature name.

2.2.1.2 Automatic Feature Recognition

In the automatic feature recognition approach, a computer program is used to identify the geometric features from the geometric model. The computer program consists of two parts: a feature recognition module and a feature extraction module. The feature recognition module identifies the geometric features by comparing the geometric model with the feature definitions. The feature extraction module uses the identified geometric elements to form the geometric features. The process of feature recognition/extraction is illustrated in Figure 2.5.

Shah classified automatic feature recognition methods into machining region recognition method and pre-defined feature recognition method [Shah95].

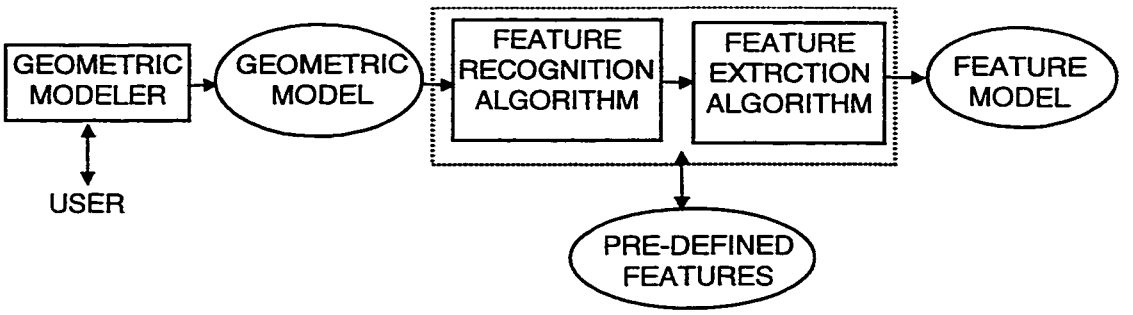


Figure 2.5: Automatic feature recognition approach [Shah95]

In machining region recognition technique, the main objective is to generate NC tool paths directly from CAD database by identifying the volumes to be removed by milling machining operations. In this approach, no comparison between the selected geometric object and the pre-defined features is required. Machine region recognition techniques are of the following types:

- Sectioning [Corney93]
- Convex hull decomposition [Kim94]
- Cell decomposition [Armstrong82]

In the pre-defined feature recognition method, the descriptions of the geometric model are compared with the pre-defined features. The tasks in pre-defined feature recognition method include: searching the CAD database to match topological/geometric parameters, extracting recognized features, determining feature parameters, completing the feature geometric model, and combining simple features to obtain higher level features [Shah95].

2.2.2 Feature-Based Design

Extraction of geometric features is a non-trivial task, if it is not impossible. To solve this problem, another feature modeling approach, namely feature-based design has been introduced and demonstrated by building prototype systems [Cutkosky88, Turner88]. In the feature-based design approach,

geometric features are first defined in a library. These geometric features are used as building blocks for modeling design candidates. The feature-based design approach is shown in Figure 2.6. Since geometric features are stored in the generated geometric model already, no feature extraction is required anymore.

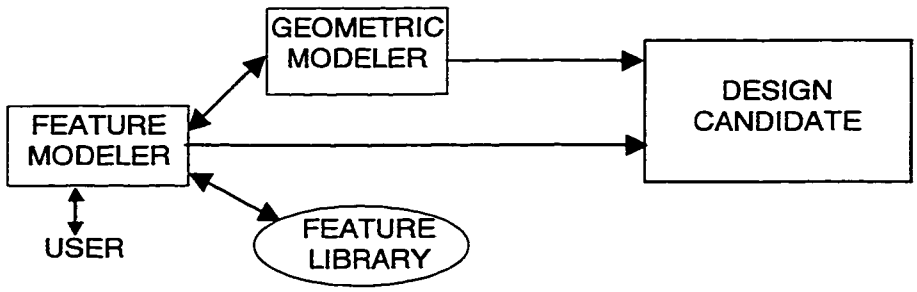


Figure 2.6: Feature-based design approach [Shah95]

Two methods commonly used in this approach are *destruction by machine features* [Turner88] and *synthesis by design features* [Miner85]. The destruction by machine features approach is also known as *destructive solid geometry* or *deforming solid geometry*. In this approach, the final geometric model is defined by removing features from the raw stock using machining operations. In synthesis by design feature approach, a geometric model is built by both adding and cutting geometric features.

The effectiveness of this approach primarily depends upon the definitions of features in the feature library. If the feature definitions mainly consist of microscopic geometric descriptions, the design is limited to geometric modeling. However, if the standard library comprises design features that provide macroscopic information such as functions, costs, etc., the product modeling effectiveness is considerably improved.

2.3 Knowledge-Based Design

The conventional CAD systems are primarily used for modeling design geometry. However modeling of product geometry is only part of the activities during the whole design process. The early stage of design, such as conceptual design, is not supported by the conventional CAD systems. To support the activities of the whole design process, new approaches have to be introduced. In this research, a knowledge-based system approach has been employed for modeling the conceptual design process. Techniques that are being investigated by other researchers are: top-down design [Mantyla90, Suzuki90], function-based design [Umeda92] and bond graphs-based design [Gui94].

The intent of using knowledge-based systems in engineering design is to replace human's intelligent activities by software systems [Rich91]. During the design process, designers use relevant knowledge to generate design candidates from design functional requirements, evaluating design candidates, and documenting design results.

2.3.1 Knowledge Engineering

Knowledge engineering was initiated due to the advances in computer science and artificial intelligence (AI) [Rich91] for solving the real world problems. The early effort in knowledge engineering focused on developing the expert systems. Expert systems are computer programs to model the knowledge of experts for solving domain dependent problems. Some successful expert systems are MYCIN [Buchanan84] for blood disease diagnosis, R1 [McDermott82] for computer system configuration, and DENDRAL for spectrographic analysis [Buchanan78]. Engineering applications of expert systems have been developed for design candidate generation, production process planning, and so on. Example design expert systems include a V-belt

design system [Dixon84], an air cylinder design system [Soni86] and a gear drive design system [Zarefar86].

Many knowledge representation schemes have been developed and used in implementing intelligent systems. These schemes include predicate logic, rule-based production inference system, semantic nets, frames, conceptual dependency and scripts [Rich91]. Each expert system uses one of these knowledge representation schemes and reasoning approaches. For example, the rule-based approach has been used in MYCIN [Buchanan84]. The advances in fuzzy logic and neural networks provide new approaches in developing intelligent systems. In this research, the rule-based approach has been employed.

2.3.2 Components of Rule-Based Knowledge System

A rule-based knowledge system consists of three components: a knowledge base, a database, and an inference engine, as shown in Figure 2.7.

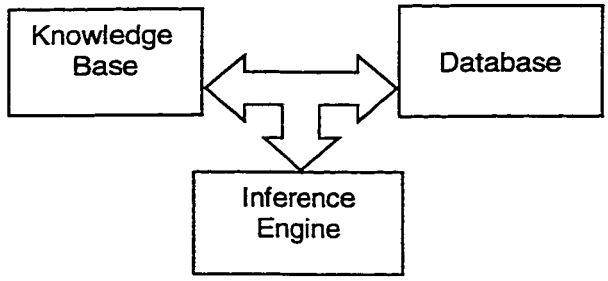


Figure 2.7: Components of a rule-based knowledge system

- *Knowledge base:*

  The knowledge base is the place to store knowledge. Knowledge is represented in form of rules. Each rule describes a piece of cause-result knowledge using the structure of

  $IF\ c_1\ \&\ c_2\ \&\ \ldots\ \&\ c_n\ THEN\ r_1\ \&\ r_2\ \&\ \ldots\ \&\ r_m$

where the *IF* part describes the conditions to be satisfied and the *THEN* part describes the resulting actions to be conducted. Both the condition part and result part are represented by a number of patterns linked with logic-and (&).

- *Database:*

  The database is the place to preserve the data. Since the data should be compared with the patterns of the rules in inference, representation schemes of the data and the patterns should be the same. In most rule-based reasoning systems, a piece of data is described by a predicate with the form of $(p_1, p_2, ..., p_n)$. During the reasoning process, data are added and removed by executing the result parts of rules.

- *Inference engine:*

  The inference engine is used to compare the condition parts of rules with the database, and execute the result parts of rules if their condition parts are matched with the database. When all the patterns in the condition part of a rule are matched with the database, then this rule can be *fired.* If a number of rules are being considered to fire, only the "best" rule is selected and its result part executed (fired). The method to select the best rule is called the conflict resolution strategy [Rich91]. The major conflict resolution strategies include: (1) the rule with the most number of conditional patterns should be fired first, (2) the rule with the most recently matched pattern should be fired first, etc.

2.4 Object Oriented Programming

In the real world, everything can be seen as an object having its own characteristics and behaviors according to which it interacts with other objects in the environment. These objects could be classified and arranged in a certain structure. This is nature's way of organizing things in a vast disordered world. The same concept is used in object-oriented programming (OOP) approach, which is now the dominant programming approach.

The key concepts in OOP are class, instance, variables and messages (also called functions or methods). Classes are generic abstractions of those physical objects with similar characteristics, attributes and behaviors. These classes have variables representing their attributes, and functions (or methods) representing the operations and behaviors through which the objects interact with other objects in OOP paradigm. Classes are used as templates to create instances, which are used for modeling the real world data. For example, "car" is a class representing the abstract concept. Specific cars, such as Tom's car or Jane's car, are instances of the concept car. A car concept provides *number-of-doors* as its member attribute, and *travelling* as its member function. Fundamental concepts in OOP are discussed below:

- *Abstraction:*

  Abstraction has been defined as the essential characteristic of an object that distinguishes itself from other kinds of objects, thus providing crisply defined boundaries. Abstraction is implemented by introducing classes and instances in OOP. For example, concept of gear can be represented by a class, while specific gears can be described by instances.

- *Encapsulation:*

  Encapsulation is the property that the inside (implementation) details of an object are hidden from the outside world. This is a very useful property in developing large systems due to its high modularity characteristic. The user of an OOP system only needs to know the functions to which objects respond. If object A wants to access object B, object A has to send a message to B and ask object B to do so. This mechanism is called message passing.

- *Inheritance:*

  In the OOP paradigm, classes are organized in a hierarchical data structure. The sub-class is able to inherit all the characteristics of its super-class

automatically. For instance, class *Car* is a super-class for classes *Toyota* and *Taurus*, therefore all the characteristics of *Car* can be inherited by *Toyota* and *Taurus*.

- *Polymorphism:*

  Polymorphism is a mechanism to respond to messages with the same name using different objects (method bodies). For instance, the function *display* can be defined in each of the classes of *Circle*, *Rectangle*, and *Triangle*. If the *display* method is sent to an object, a circle, a rectangle or a triangle is displayed depending on the class type of the object. Thus a circle is displayed on the monitor if the class *Circle* calls the *display* message, as shown in Figure 2.8. The polymorphism property of OOP is used to make the system more organized and small in size.

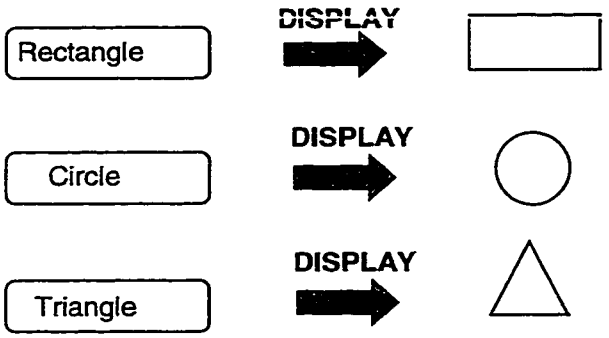


Figure 2.8: Polymorphism in object oriented paradigm

2.5 Smalltalk Language

The major portion of the feature-based intelligent design system has been implemented using Smalltalk - an object oriented programming language [Goldberg83]. Smalltalk was developed by Xerox Palo Alto Research Center (PARC). Its first commercial version, Smalltalk-80, is one of the pioneer OOP languages. Smalltalk-80 was also the first language to introduce windows programming, an interactive interface approach. Many concepts of this language

have been adopted by subsequent OOP languages such as C++ and Java. Its latest version, called VisualWorks 3.0, has been used for implementing the feature-based intelligent design system.

VisualWorks provides an OOP paradigm, a user-friendly interface environment, and a large library of classes. The weaknesses of this language are its slow computation speed, large memory requirement, and high price of software package. From the past experience [Xue92], it was found that it takes much less time to develop a system using Smalltalk than using other OOP languages such as C++ or Java. Therefore Smalltalk is an excellent tool for developing a research oriented software prototype system.

In Smalltalk, everything, including the compiler, debugger, window manager, etc., is defined as a class and is available for modifying. In VisualWorks 3.0, several hundreds of classes have been provided for modeling the components of this system. New classes can be defined in the same environment by declaring them as subclasses of the existing classes. Users can also modify the classes defined by the system.

In Smalltalk, every class has a super-class except the class called *Object,* which is the top class in the class hierarchy. A class may have two kinds of variables: *class variables* and *instance variables.* The class variables are shared by all instances of the class, i.e., the value of a class variable is the same for all the instances of that class. The values of instance variables are specific to a particular instance. The functions of classes are defined as class methods and instance methods. Classes respond to class methods (messages), while instances respond to instance methods. Unlike C++, in Smalltalk all the variables and methods defined in a class are inherited by its sub-classes. There are no private or protected variables or methods, as in C++.

2.6 Xue's Research on Integrated and Intelligent Design

The research presented in this thesis is a continuation of the research on integrated and intelligent design, which was initiated by Xue [Xue92, Xue93, Xue94a, Xue94b, Xue96, Xue97]. This section gives an overview of Xue's previous research.

The research on integrated and intelligent design aims at developing a computer-based design environment that supports the activities in all the product development life-cycle phases, thereby providing a theoretical and implementational framework for the next generation CAD/CAM systems.

In Xue's research, a feature-based product life-cycle modeling system was first introduced [Xue93]. In this system, different product life-cycle aspects were modeled using aspect primitives called features. Three types of features were developed: design features, geometry features, and manufacturing features. Design features are mechanical components and mechanisms, such as gears and shafts, for modeling design candidates based upon the functional requirements. Geometry features are geometric primitives, such as blocks and cylinders, to construct the product geometry. Manufacturing features are partial product geometry, such as holes and slots, for planning manufacturing process. Representation of features followed the scheme of a product modeling language – Integrated Data Description Language (IDDL) [Tomiyama87, Xue92]. IDDL was originally proposed at the Center for Mathematics and Computer Science in Amsterdam by Tomiyama [Tomiyama87]. This language was implemented by Xue at University of Tokyo [Xue92]. In IDDL a product is described by objects, their attributes, qualitative relations among objects (called facts), and quantitative relations among attributes (called functions) [Xue92]. Objects were employed for modeling aspect features in the feature-based product modeling system.

A qualitative intelligent system that combines knowledge-based reasoning and optimization was introduced to automatically generate the aspect models and to identify the optimal design using optimization [Xue94a]. To improve the efficiency of feature identification from the large feature library, a design-function based design feature coding system and a manufacturing-function based manufacturing feature coding system were developed for organizing the feature library and identifying appropriate features to automatically generate design candidates and plan production processes [Xue94b].

The objective to develop the integrated and intelligent design system was to achieve the design with best overall product life-cycle performance. Since production cost is a key measure for evaluating the design from manufacturing perspective, a number of cost models considering different production processes and tolerance requirements were introduced [Dong94]. An optimization model was developed for achieving the design with the best tradeoff between functional performance and production cost [Xue96]. A number of global optimization models were introduced to identify the optimal design considering variation of design alternatives and parameters [Xue97]. Mathematical models for concurrent design were also introduced [Yadav98a]. A multi-level heuristic search algorithm was introduced for generating the production process of the created design for further evaluation [Yadav98b].

This research aims at further improving Xue's integrated and intelligent design system. The improvement focuses on the following issues:

1. In the Xue's system, features are modeled by IDDL objects. Each object is composed of attributes. No special feature representation scheme was developed for modeling features. Due to the poor data representation scheme, many characteristics of features cannot be described using IDDL. Therefore in this research, a feature representation scheme and its modeling

environment have been developed. The different aspects of features, such as qualitative data, quantitative data, qualitative relations, quantitative relations, constraints, etc., are modeled separately. If a new aspect is required to model the feature, this new aspect can be easily added to the feature definition using the developed interface environment. In addition, a special data dependent relation maintenance system has also been developed to keep the consistency of the database.

2. In the previous system, geometric information is represented by geometry features, which are also described by IDDL objects. This symbolic geometric model is associated with a B-rep solid model by representing the geometric elements, including vertices, edges, faces, etc., using IDDL objects. The B-rep solid model was specially designed for IDDL and implemented using C. In the present research, a geometry representation scheme, based upon feature representation scheme, has been developed. The symbolic geometric model of the product is first translated into a neutral geometry representation scheme. The neutral geometric descriptions are further translated into the geometric modeling schemes by particular CAD systems. In this research, a 3D geometric modeling system called 3D Studio MAX [Michael96] was used to represent the product geometric information.

3. In the IDDL, a rule-based inference system was designed for accessing the database represented in the IDDL scheme. Therefore, for the present system, a new rule-based reasoning system was required to access the feature-based product descriptions. In this research, many built-in predicates have been developed to add, delete, and modify the feature descriptions. In addition, a mechanism to select only part of the knowledge base (rules) and database (features) has also been developed to improve the inference efficiency.

Chapter 3

A Feature-Based Intelligent Design System

In this chapter, a feature-based intelligent design system is proposed to solve the problems listed in Chapter 1 and Chapter 2. First, the functional requirements of the feature-based intelligent design system are introduced. Then the architecture of this system is discussed.

3.1 Functional Requirements for the System

The functional requirements of the feature-based intelligent design system were identified through extensive study of the feature-based design approach, geometric representation methods, intelligent systems, and object oriented programming.

3.1.1 An Efficient Product Modeling Method

Consider an automobile company that manufactures many different types of automobiles such as cars, vans, trucks, etc. A number of components such as pistons, brakes shoes, distributors, axles, clutch plates, and so on are common to all types of vehicles. These components that build up the products are standard components and used in all automobiles with minor differences. In a CAD system for designing automobiles, these standard components should be provided as basic elements that can be defined and stored in a standard library. Components from the standard library can be used to generate instances for modeling actual products.

This idea is illustrated in Figure 3.1. Standard objects are described as building blocks and these building blocks are used for modeling the actual

products. This product modeling approach is more efficient than to start the design from scratch using a geometric modeling tool.

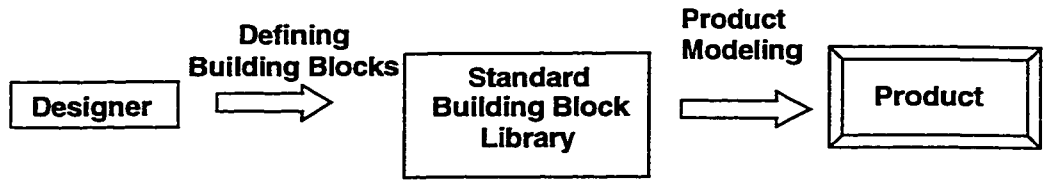


Figure 3.1: Use of standard building blocks for product modeling

The issues in the development of the system that uses a library of standard building blocks to model products are summarized as follows:

1. The data described in a building block should be well organized. This requires that a standard data scheme should be developed and used for representing the data of all the building blocks in the system.

2. The building block entities in the system should be editable, so that modifications to these standard building blocks can be carried out to model the actual products. For instance, the size and strength of the axle of a car are different from the parameters of a truck. Therefore editing is required after an axle is generated using a standard building block.

3. The definitions of the standard building blocks should not be affected after editing operations have been made to the instances during the modeling of a particular product. The standard generic entities should be only used for creating instances that can be then modified according to the specific requirements.

3.1.2 Building Blocks for Design

In a building block, data should be described in a standard format. The data in each building block should be classified into a number of groups, according to their types. These types include qualitative data, qualitative relations, quantitative data, quantitative relations, etc. A building block may also consist of a number of constituent building blocks. For instance, a gear pair mechanism consists of two gears and two shafts. The concept of building blocks is illustrated in Figure 3.2.

Data described inside the building block should include the following aspects:

1. *Geometric and non-geometric parameters:*

   Objects have geometric as well as non-geometric parameters. Both types of parameters are important from the design point of view. For example, a gear has geometric parameters, such as diameter and tooth height, and non-geometric parameters, such as rotational speed and direction of rotation. While the geometric parameters are used for building up the geometric model, the rotational speed is used for evaluating design functional performance. In most conventional CAD systems, non-geometric parameters cannot be defined. Such systems are basically used for geometric modeling. However, a design system should provide the environment in which both geometric and non-geometric parameters should be defined.

2. *Relations among the parameters:*

   Among all the parameters, some are calculated using other parameters. Therefore, parameters are linked through quantitative relations. For instance, the module of a gear is a function of the diameter and the number of teeth of the gear. Therefore, the design system should provide the capability to model quantitative relations among the parameters.

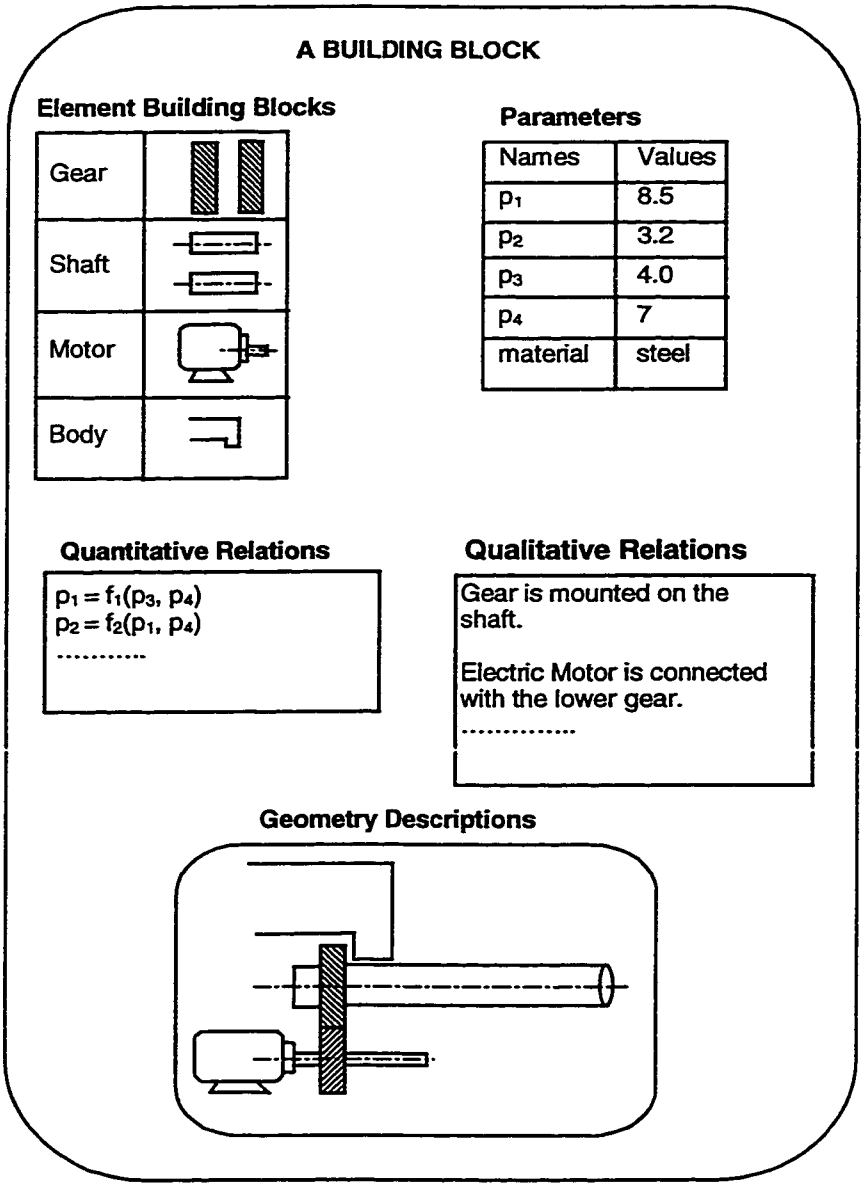


Figure 3.2: An example of building blocks

3. *Constituent building blocks:*

   A building block may be made up by other building blocks. For instance a gear-pair mechanism is a building block that consists of other building blocks including gears, shafts, etc. Using this approach, instead of defining all the

details of a building block at the same place, we can first define the primitive building blocks and then use them for modeling more complex building blocks. The building blocks form a tree data structure in which the sub-nodes represent the constituent building blocks of the building block at a higher level, as shown in Figure 3.3. The rela tionship between a building block and its constituent building blocks is a *"made up of"* relationship that is different from superclass-subclass relationship. The characteristics of a building block will not be inherited by its constituent building blocks.

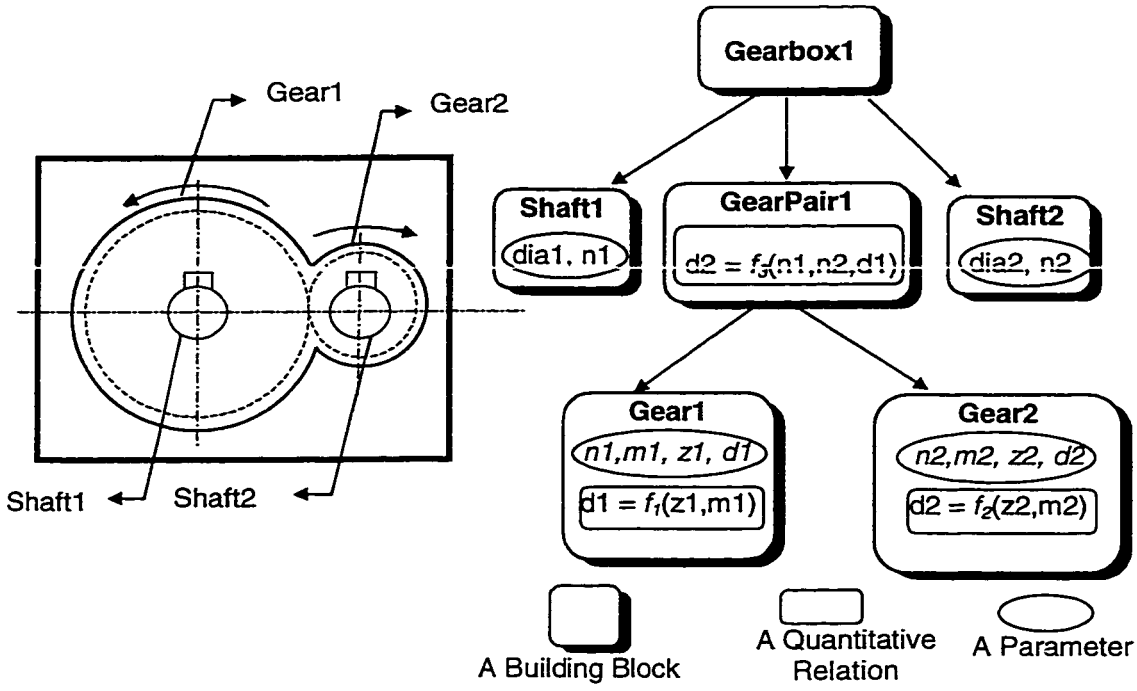


Figure 3.3: A gear pair mechanism represented by building blocks with a tree data structure

4. Qualitative data:

Product data could be quantitative (e.g., speed ratio of two gears is 1:2) or qualitative (e.g., a gear and a shaft are connected). Conventional CAD systems do not provide the capability for defining qualitative data. In order to develop the intelligent design system that supports the conceptual design

activities, qualitative information should be described in building block definitions. This type of description can be used for knowledge based reasoning.

5. *Constraints on the parameters:*

   Design parameters are bound by a number of constraints. For instance, the module parameters of the two gears should be the same in a gear-pair mechanism. In the building block definition, these constraints should be defined.

6. *Geometric descriptions:*

   Since geometric information plays a key role in product modeling, representation of the product geometry should also be provided in the building blocks. The geometric descriptions include 2D and 3D geometric descriptions. The geometric descriptions should be associated with symbolic descriptions of the product which are generated during the conceptual design stage.

3.1.3 A Mechanism to Maintain the Product Data Relations

Design is conducted through an iterative process that requires a number of redesigns before the final design is reached. If part of the design model is modified, the change may lead to inconsistency of the product data due to the pre-defined relations. Therefore, propagation of the data change using these pre-defined relations is required. In order to avoid inconsistency and to conduct the design in an efficient manner, a mechanism to maintain the relations of product data is required.

A dependent relation is defined by

$$d_j \leftarrow d_1, d_2, d_3, \ldots, d_n$$

where parameter d_j is calculated using parameters $d_1, d_2, d_3, \ldots, d_n$.

3.1.4 A Data Representation Scheme

In order to develop the intelligent design system, a data representation scheme that can be used for modeling the different aspects of the products is required. Product data should be represented in a simple and concise manner. All the elements listed in Section 3.1.2, including constituent building blocks, qualitative data and relations, quantitative data and relations, constraints, and geometric descriptions, should be represented using this data representation scheme.

3.1.5 Object Oriented Modeling Approach

The object oriented modeling characteristics, including abstraction, inheritance, encapsulation, etc., should be used in developing the feature-based intelligent design system. The requirements this imposes are explained as follows:

1. *Abstraction*:

   The general characteristics of objects should be defined in library building blocks. The actual building blocks used in modeling products should be generated using library building blocks as templates.

2. *Inheritance:*

   The library building blocks should be organized in a hierarchical data structure. A building block at a lower level should inherit all the definitions of its upper level building blocks. In this way the efficiency of modeling the standard library can be improved considerably.

3. *Encapsulation:*

   The data described in a building block should not be accessible directly by other building blocks. Communication among building blocks should be conducted by message passing. Using encapsulation can improve the modularity of the system, thus providing better software reusability.

3.1.6 Automated Product Modeling

The product design process should be automated to further improve product modeling efficiency. An intelligent system can serve for this purpose. An intelligent system consists of a knowledge base and an inference engine.

The knowledge base is described in terms of rules. Rules are pieces of knowledge described in the IF-THEN structure. The format to represent the rules should allow the use of parameters, constituent building blocks, etc. The rules in the knowledge base should be organized in groups to improve the modularity of knowledge base modeling.

An inference engine should be developed for knowledge based reasoning. This will enable the system to make decisions during the design process. A method should be developed to improve the reasoning efficiency by selecting only relevant knowledge bases and databases.

3.2 System Architecture

To satisfy the objectives and requirements specified in previous sections, an intelligent feature-based design system is proposed in this research. The architecture of this system is illustrated in Figure 3.4. The system consists of three modules:

- Feature-based Design System,
- Geometry Representation System, and
- Intelligent System.

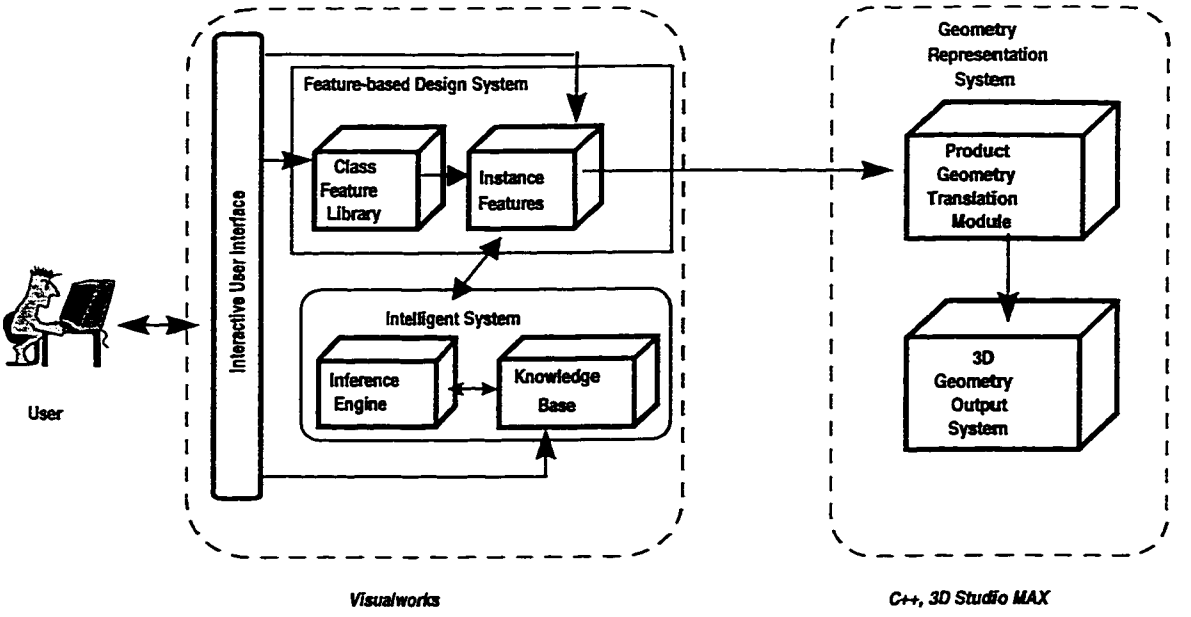


Figure 3.4: System architecture showing the three modules

3.2.1 Feature-based Design System

In this system, features are used as building blocks for product modeling as described in Section 3.1. Features are defined at two levels: class level and instance level. Class features are used to capture the generic characteristics of objects. Instance features are the specific product data generated using class features as templates.

The class features are stored in a standard library. The object oriented programming approach is used for modeling the class feature library. Class features are organized in a hierarchical data structure. A feature at a lower level in the hierarchy data structure inherits descriptions defined in its super-class features.

Descriptions of a class feature are organized in a number of groups, called "aspects" in this system. These aspects include attributes (parameters),

attribute relations, element features (constituent building blocks), feature relations, constraints, geometry descriptions, etc.

All the descriptions in the class features can be derived in the instance features automatically when they are generated using the class features as templates. The descriptions in the instance features can be modified during product modeling process. Since the data in instance features are associated by their relations and any change of partial product data descriptions can be propagated to other parts automatically using these relations.

3.2.2 Geometry Representation System

In this system, geometric descriptions are defined as part of the feature definitions. The geometric descriptions include (1) 2D geometric descriptions and (2) 3D geometric descriptions.

The 2D geometric descriptions are used for displaying the 2D product geometry using a specially developed browser. The 2D geometric elements, including lines, circles, arcs, etc., are used for modeling the product geometry. The 2D geometry representation function was implemented using VisualWorks directly.

The 3D geometric descriptions of the features are defined by (1) 3D primitives (e.g., box, cylinder, etc.), (2) moving and rotating operations to these primitives, and (3) Boolean operations (e.g., union and subtraction) to these primitives. To utilize the functions of conventional geometric modeling systems for displaying the geometric results, a neutral feature geometry representation scheme is developed in this research. When generation of the product 3D geometry is required, all the 3D geometric descriptions in the instance features are extracted and changed into this neutral geometric representation scheme. The geometric modeling system then translates the neutral geometric

descriptions into its geometric representation format to display the 3D geometry. In this work, a 3D geometric modeling system, namely 3D Studio MAX [Michael96], has been employed for this purpose.

3.2.3 Intelligent System

An intelligent system has been developed to achieve the objective of product modeling automation. The intelligent system is composed of a knowledge base and an inference engine. Product modeling is automated by knowledge-based inference.

Knowledge is encoded in form of rules. A rule has IF-THEN data structure. The IF part of a rule describes the condition that has to be satisfy before the THEN part of rule can be executed. The database is represented by instance features. To improve the reasoning efficiency, only the relevant part of the knowledge base and database are considered.

Details regarding the feature-based design system, geometry representation system, and intelligent system will be discussed in Chapters 4, 5, and 6.

The feature-based intelligent design system provides a user-friendly interface environment. Using the class features defined in the class library, modeling of products can be achieved by generating instance features either manually or using the knowledge-based system.

The object oriented programming language - Smalltalk has been primarily used for implementing the feature-based intelligent design system. It was used due to its large class library and good programming/debugging environment.

The geometry representation system was implemented using C++ and 3D Studio MAX. For translating the neutral geometric descriptions into the specific

descriptions for 3D Studio MAX, a translation module has been implemented using Microsoft Visual C++ 5.0. 3D Studio MAX is used for geometric representation due to its excellent graphics functions. A snapshot of the implemented system is shown in Figure 3.5.

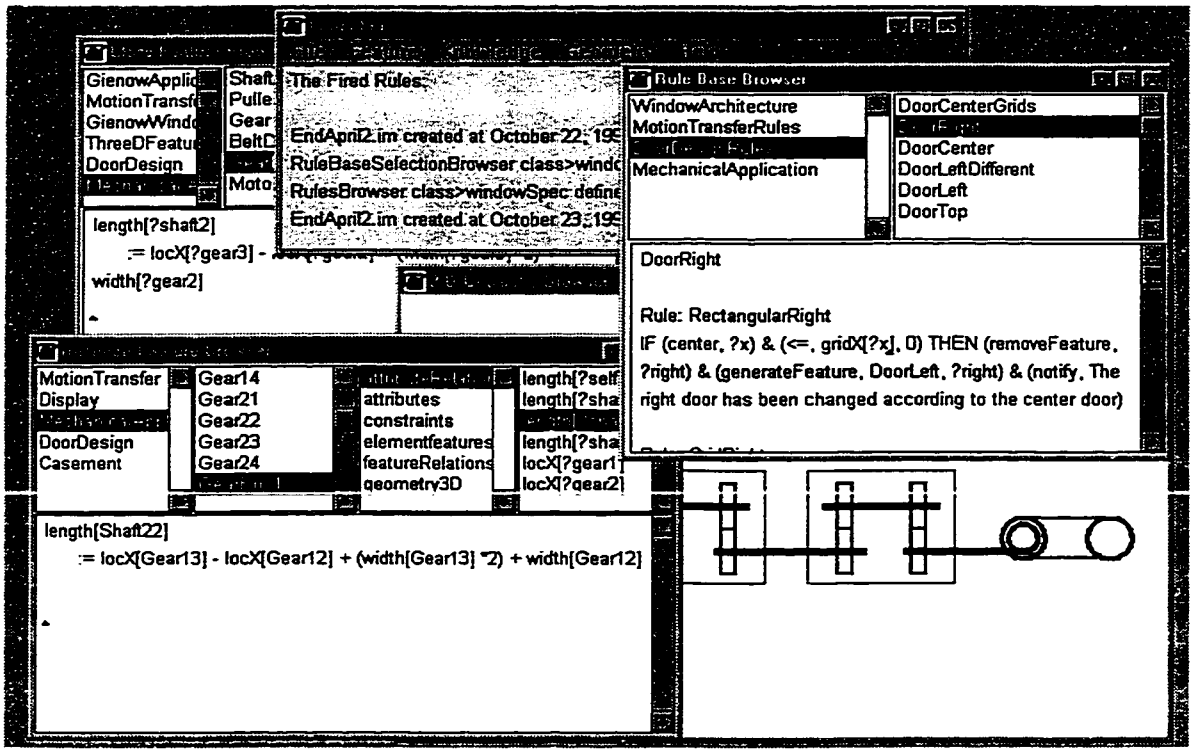


Figure 3.5: Snapshot of the implemented system

Chapter 4

Feature-based Design System

This chapter introduces the feature-based design system. The methods of modeling the standard class feature library, generating instance features for building up products, and maintaining product data relations are discussed in this chapter. An example is given to demonstrate the functions of the developed system.

4.1 Introduction

In this research, features are used as building blocks for product modeling. First, generic characteristics of objects are captured in class features. Class features are stored in the class feature library of the design system. Using class features as templates, instance features can be generated for modeling product data. All the class feature descriptions are inherited by the corresponding instance features automatically.

In a class feature definition, all the descriptions are organized in a number of groups called *aspects*. These groups include quantitative parameters, relations among quantitative parameters, qualitative descriptions, relations among qualitative descriptions, constraints, geometric descriptions, and so on. This structure provides good software re-use and management capabilities. For example, a new aspect can be added to the feature definitions, when such a requirement is raised.

The object oriented programming approach has been employed for modeling class features. The class features are organized in a hierarchical data structure. *S*ub-class features inherit all the characteristics of their super-class

features. In this way, modeling of class features can be accomplished in an efficient manner.

Instance features are the specific product data generated using class features as templates. An instance feature can be modified without affecting its class feature definition. A product model generally comprises a collection of instance features that are associated with qualitative relations among themselves and quantitative relations among the parameters.

4.2 Creation of Class Features

All the class features are stored in different *categories* due to the large number of class features. The categories play no role in the product library modeling process except for helping to search and find the required class features easily. To create a new class feature, either an existing category should be selected or a new category should be added. A new class feature is defined as a sub-class feature of an existing class feature. For instance, the class feature *SpurGear* can be defined as a sub-class feature of the class feature *Gear*. The descriptions in the *Gear* class feature can be inherited by the *SpurGear* class feature automatically.

4.2.1 Class Feature Browser

The class feature browser is the interface environment to define class features. A snapshot of the class feature browser is shown in Figure 4.1. The browser consists of four list views and one text view as illustrated in Figure 4.2.

In the class feature browser, the categories are listed in the category list view. By selecting one of the categories, the features in that category are shown in the feature list view. By selecting a feature in the feature list view, a list of aspects is shown in the aspect list view. The aspects include: attributes,

element-features, attribute-relations, feature-relations, geometry3D, constraints, class-methods, instance-methods, etc. The aspect list is a built-in list that is common to all features. Element name list view shows the names of the feature

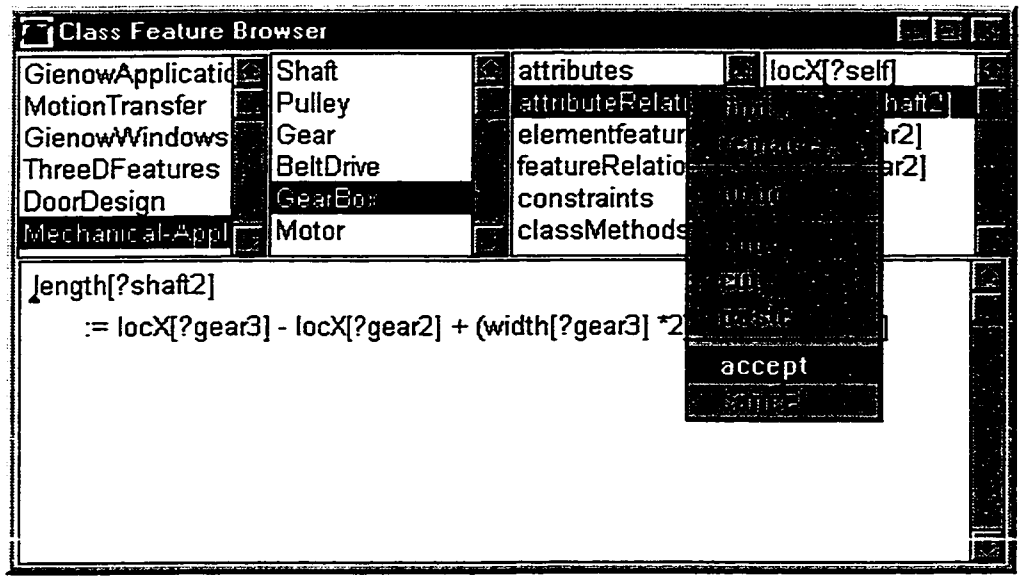


Figure 4.1: Snapshot of class feature browser

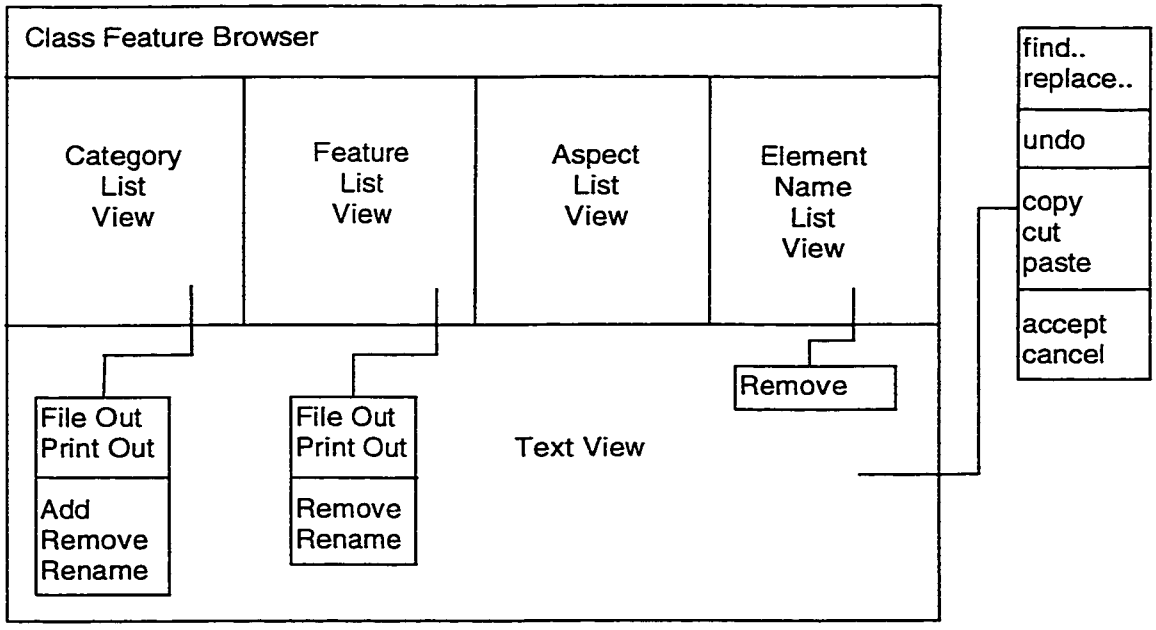


Figure 4.2: Views and menus in the class feature browser

elements in corresponding aspect. These element names are represented with different formats. Text view is used to edit the feature definitions.

4.2.2 Defining Class Features

Class features are defined in the text view and saved in the class feature library. After the category, in which the class feature is to be defined, has been selected, the following template is shown in the text view of the class feature browser:

```
<Super-Class Feature> subclass: <Sub-Class Feature>
instanceVariableNames: <instance variables>
classVariableName: <class variables>
category: <category name>
```

To define a new class feature, the user is required to replace the text within brackets with the actual feature definitions. A class feature name starts with an uppercase letter. A super-class feature should be a class feature already defined in the system. Instance variables and class variables are the same as those in Smalltalk. The category name is the one selected for generating the new class feature. An example of feature definition is given as follows:

```
<Feature> subclass: <Gear>
instanceVariableNames: <addendum pitch>
classVariableName: <Pi>
category: <Mechanical-Applications>
```

After filling in this template, the class feature can be saved by executing the *accept* menu item in the text view menu. Common editing functions, such as *find, replace, cut, copy, paste*, etc., are provided in the text view menu. An existing class feature can be removed or renamed by selecting the corresponding menu items of the feature list view.

The next step is to define the feature elements using the text view. Details regarding these feature definitions are introduced in Section 4.3.

4.3 Representation of Class Features

Class features are abstractions of physical objects. The characteristics of different perspectives of objects, such as parameters and behaviors, are captured as aspect descriptions of a class feature. In this system, the aspects are: attributes, element-features, attribute-relations, feature-relations, constraints, geometry3D, instance-methods and class-methods. An example of class feature definition is given in Table 4.1.

Table 4.1: An example of a class feature description

Items	Examples
class definition	<Feature> subclass: <GearPair> instanceVariableNames: <> classVariableNames: <> category: <Mechanical-Applications>
attributes	width 100 speedRatio 4
element-features	?gear1 Gear ?gear2 Gear ?shaft1 Shaft ?shaft2 Shaft

Table 4.1: An example of a class feature description (continued)

Items	Examples
attribute-relations	rpm[?gear1] = rpm[?shaft1] rpm[?gear2] = rpm[?shaft2] speedRatio[?self] = rpm[?gear1] / rpm[?gear2]
feature-relations	connection (connect, ?gear1, ?shaft1) (connect, ?gear2, ?shaft2) (pair, ?gear1, ?gear2)
constraints	widthConstraint width[?self] > d[?gear1] + d[?gear2]
geometry3D	Geometry3D (box, plate1, plateWidth[?self], plateThick[?self], plateLength[?self]) (rotate, plate1, 0, 90, 0) (move, plate1, 0, 50.0, 18.0) .....
instance-methods	drawingOn: gc from: anInstanceName | originPoint aRectangle locX locY | locX := locX[?self]. locY := locY[?self]. originPoint := Point x: locX y: locY. .....

In the remainder of this section, the aspects defined in the class features are introduced with details.

(1) Element-Features:

A mechanical product is usually made up of a number of components. For instance, a car consists of components including body, engine, wheels, and so on. Similarly features should also be able to define the composing components. These composing components are referred as element-features in the feature-based design system. When an instance feature is generated, its element-features are also generated automatically. Element-features are listed under the *element-features* item in the class feature browser. An element feature is defined by a variable name and a feature type in a class feature. A variable name is represented by a string of characters starting with a question mark "?". The type of the element-feature must be a class feature already defined in the system. Examples of element-feature descriptions are:

Example I

```
?X
      Shaft
```

Example II

```
?gear1
      Gear
```

?self is a special variable to refer to the feature itself.

(2) Attributes:

The parameters of features are described as attributes. Attributes include numerical attributes (e.g., width and length) and descriptive attributes (e.g., color and type). In this feature-based design system, an attribute is defined by an attribute name and an attribute value. An attribute name is described by a string

of characters. An attribute value is described by either a number or a string of characters. Examples of attribute definitions are shown below:

Example I

```
m
        2
```

This example shows the definition of an attribute name *m* and its default numerical attribute value *2.*

Example II

```
color
        red
```

This example shows the definition of an attribute name *color* and its default descriptive attribute value *red.*

Attributes are used in the format of *attribute[feature]* in other parts of the feature definitions. An attribute *m* of the element feature *?gear1* is represented as *m[?gear1].* Other examples of attributes are: *color[?self]* and *diameter[?pulley1].*

(3) Attribute Relations:

Relations among numerical attributes are described by functions. Each function is defined by a number of input attributes and one output attribute. A general representation of a relation takes the form:

```
<attribute name>
        := <an expression>
```

The syntax of <an expression> follows the syntax of Smalltalk. Element-features and attributes are allowed in the expression. When a complicated relation is to be described, generally a Smalltalk method is implemented first,

and this method is subsequently called in the expression. Examples of attribute relations are:

Example I

```
d[?self]
      := m[?self] * z[?self]
```

Example II

```
d[?gear1]
      := Gear diameterWithM: m[?gear2] Z: 30
```

(4) Feature Relations:

Relations among features are defined by predicates. Each predicate takes the form of $(x_1,x_2,.....x_n)$, where, $x_1,x_2,.....x_n$ are called terms. The first term in a predicate is used to specify the relation among the rest of the terms. Feature relations are qualitative in nature. A term in a feature relation is described by a symbol constant, a numerical constant, a variable, or an attribute. Feature relations are organized in groups. A group of feature relations defined in class feature *GearPair* is described as

```
connection
      (connect, ?gear1, ?shaft1)
      (connect, ?gear2, ?shaft2)
      (pair, ?gear1, ?gear2)
```

(5) Constraints:

Constraints specify the requirements that a feature must satisfy. A constraint is defined by an expression with a Boolean return value. Each constrain is associated with a name, which is defined by a string of characters. Attributes and element feature variables are allowed in constraint expressions. An example of constraint is shown below:

Example

```
moduleConstraint
     ^#(1.2 1.6 2.0 2.5) includes: m[?gear1]
```

In this example a Smalltalk method is used to evaluate the constraint.

(6) Class Methods

Definitions of class methods are the same as those in Smalltalk. In addition, element-feature variables and attributes are allowed in the class method definitions. Class methods are executed by class features directly.

(7) Instance Methods

Definitions of instance methods are the same as those in Smalltalk. In addition, element-feature variables and attributes are allowed in the instance method definitions. Instance methods are executed by instance features. The 2D product geometry is also defined as an instance method. An example instance method to achieve the height of a standard casement window is defined as:

Example:

```
getHeight
| material height |
height := 0.
material := frameMaterial[?self].
material == #primedWood
     ifTrue: [height := heightForPrimedWood]
     ifFalse: [height := heightForSevilleMetalClad].
^height
```

In this instance method, two instance variables, `heightForPrimedWood` and `heightForSevilleMetalClad`, are used to obtain the value of the `height` attribute.

(8) Geometry3D

The 3D product geometry descriptions are described in this aspect. The 3D geometry of a feature is defined by generating primitives (e.g., box and cylinder), transformation operations (moving and rotating) and Boolean operations (e.g., union and subtraction) to these primitives. Details regarding product geometry representation will be discussed in Chapter 5.

4.4 Generating Instance Features

Instance features are used for modeling actual product data. Class features are used as templates for generating instance features. Instance features can be generated manually by the user or automatically by the intelligent system. The *instance feature browser* of the feature-based design system is the user interface for generating and modifying instance features. A snapshot of the instance feature browser is shown in Figure 4.3. This browser consists of four list views and one text view as illustrated in Figure 4.4.

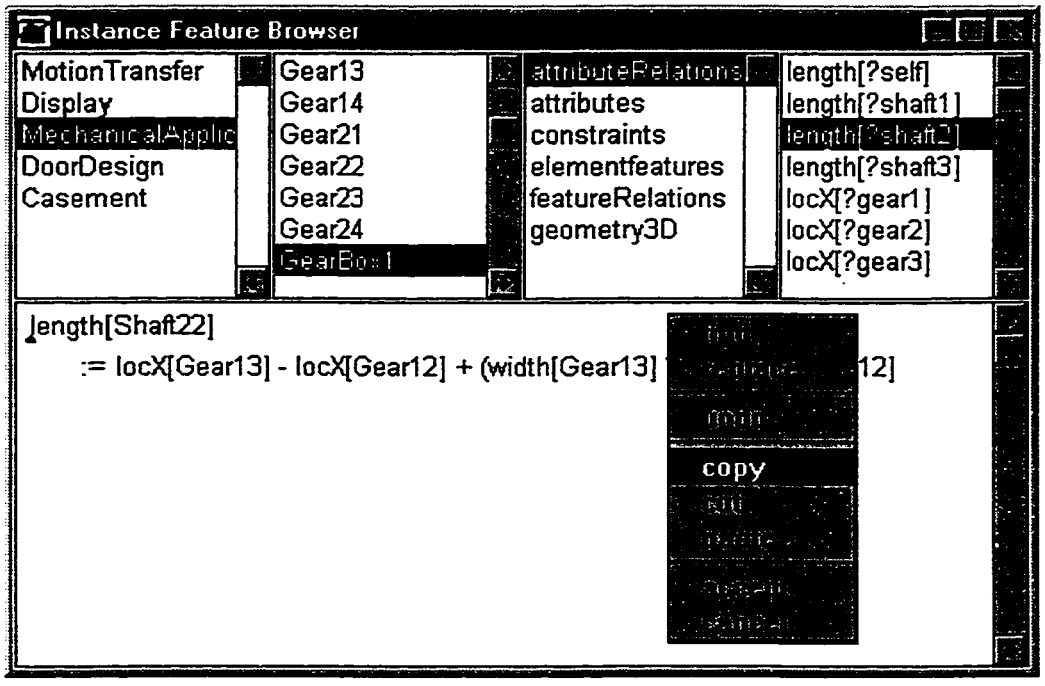


Figure 4.3: Snapshot of instance feature browser

Different products are generated in separate categories. Instance features can be added to or removed from a category using the feature list view. A product description can also be saved in an external file using the *File Out* function of the category list view menu.

Instance features can be modified manually using this browser or automatically using the intelligent system. Manual modification to instance feature definitions is carried out using the text view of the instance feature browser.

Representation of instance features is similar to the representation of class features, except that the element-feature variables are replaced by actual instance feature names. An instance feature, generated from the class feature shown in Figure 4.5, is illustrated in Figure 4.6.

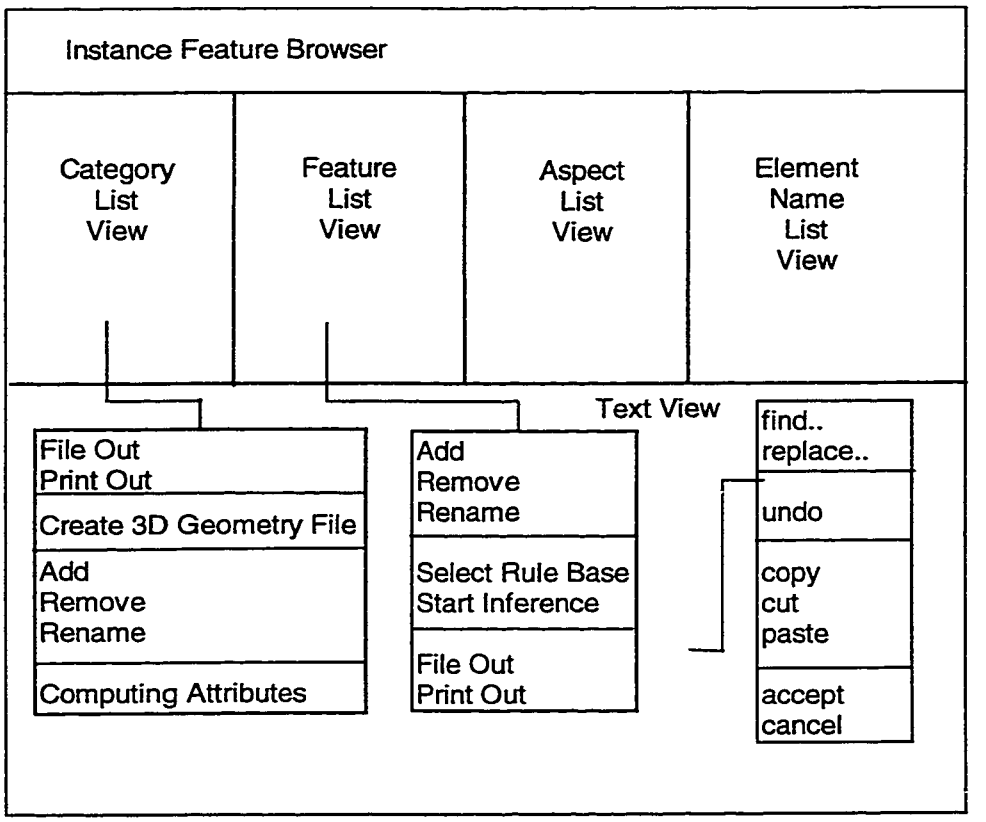


Figure 4.4: Instance feature browser - the interface for product modeling

The following steps are used to generate an instance feature manually:

- Add a new category or select an existing category from the instance feature browser,
- Select the menu item "add" from the feature list view of the instance feature browser,
- Select the class feature, which is used to generate the instance feature, by specifying its category name and class feature name using dialog boxes,
- Specify the name of the instance feature to be generated,

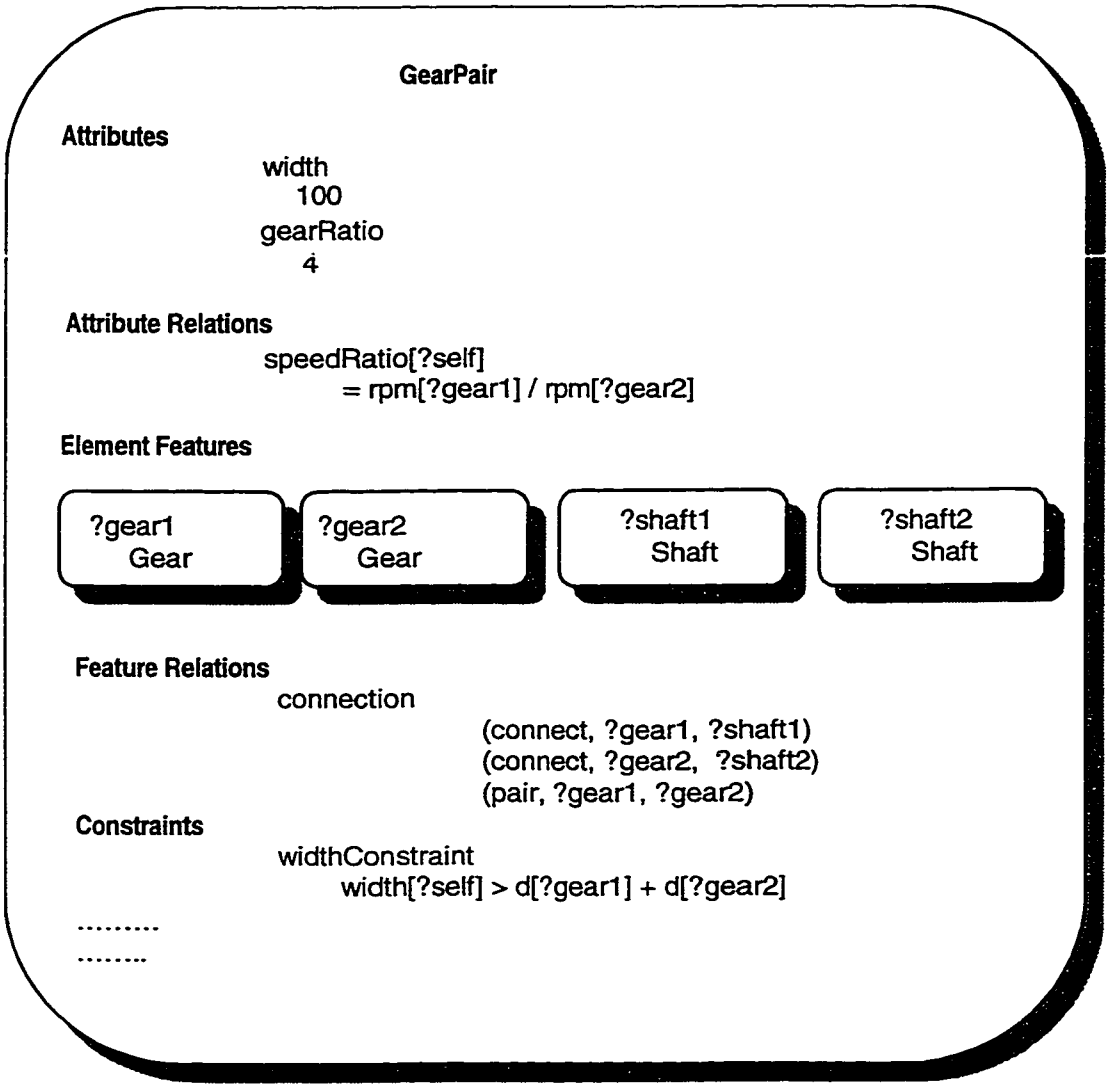


Figure 4.5: Representation of *GearPair* class feature

- If the class feature consists of element features, these element features should also be generated. The user is then asked to specify the names for these element features.

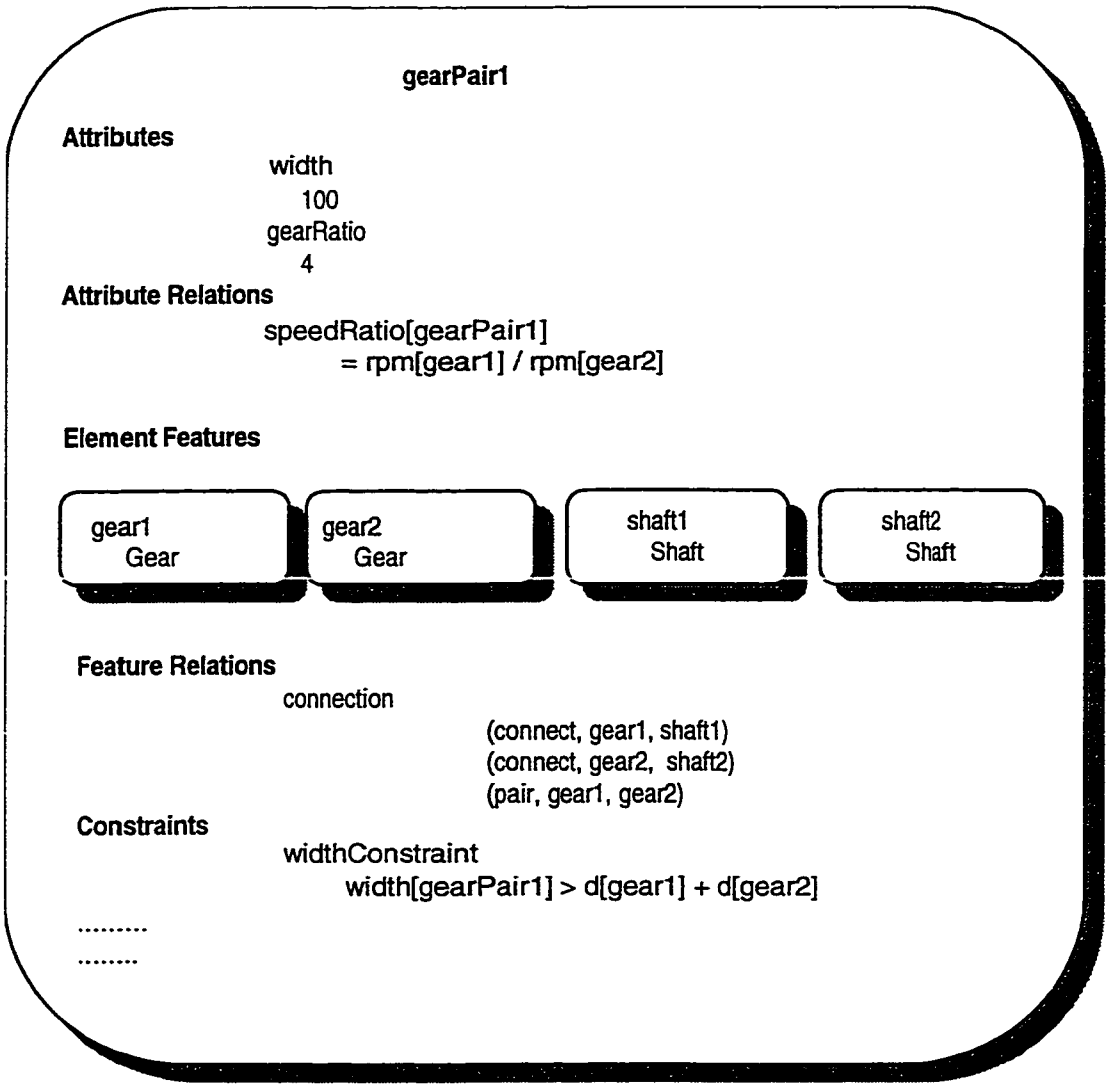


Figure 4.6: Representation of *gearPair1* instance feature

When an instance feature is created using a class feature as its template, the instance feature inherits the descriptions in that class feature and its super-class features. The element feature variables in the class feature descriptions are replaced by the actual element feature names. The process of instance feature generation and inheritance among features are shown in Figure 4.7.

During the creation of instance features, the default attribute values defined in class features are inherited to their instance features automatically. These attribute values can be modified during modeling the product. Element features, by default, are of the types that are specified in the class feature definitions.

4.5 Maintaining Product Data Relations

Modifications and redesigns are often required before a product model is finalized. Since the product data are associated through their relations, modifications in one part of the data should be propagated to other parts using these relations. This approach is similar to the traditional parametric design, in which numerical relations among parameters are defined and used to keep the consistency of parameter values. In addition, this system also provides the

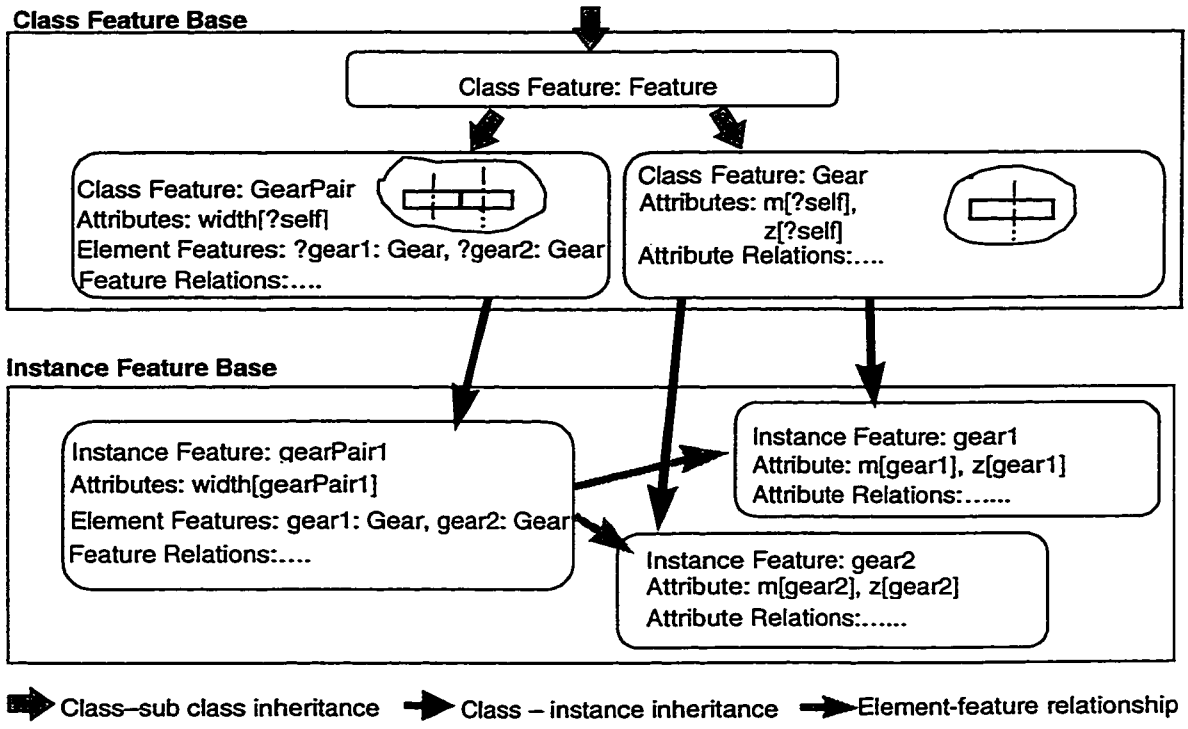


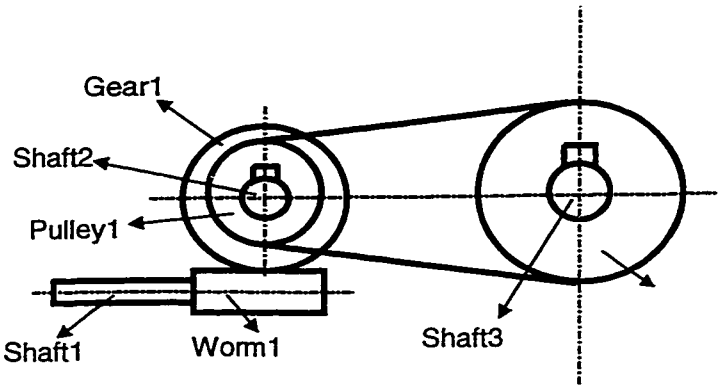
Figure 4.7: Generation of instance features using class features as templates

following functions: (1) modeling of the relations among non-geometric attributes, (2) modeling of the relations among non-numerical attributes, and (3) modeling of the qualitative relations among features. Figure 4.8 shows a mechanism that is used to transfer rotational motion from one shaft to another. Some of the relations among the attributes are listed in this figure. The feature-based design system uses a data dependent relation management mechanism (introduced below) to keep the consistency of the product data descriptions.

The data dependency relation network has been developed based upon the relations for propagating product data changes. The propagation of changes is carried out in the following manner:

- For each product defined in a category, the system maintains a list representing the attribute dependency relations. Each element on the list specifies the input and output relation defined by a function. If an attribute is changed, the system then checks whether this attribute is used as an input attribute in a function. If it is so, the output attribute value should be updated using the function. This process is continued until no propagation is required.
- When an instance feature is deleted, all its element features should be deleted.
- When an attribute or an element feature is removed, all the relevant qualitative and quantitative relations should be deleted.
- The constraints are checked when new features and attributes are created or modified. A violation of constraint can be resolved by changing the relevant attribute values.

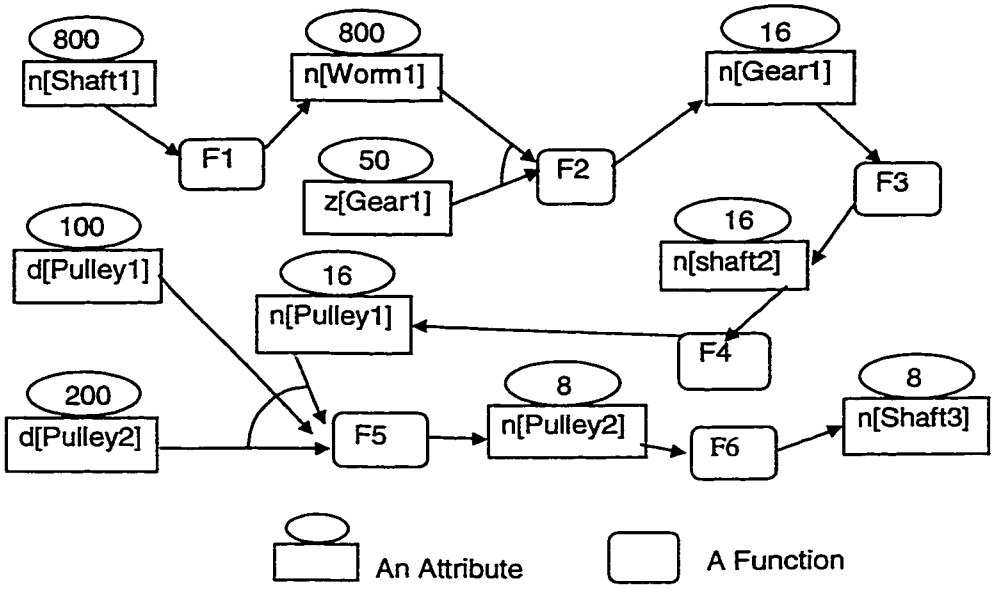
In the developed data dependent relation management system, an output attribute value is updated when the input attribute values are changed using an introduced relation. This approach is different from the conventional constraint modeling method in which updating the attribute values in both directions can be carried out using the constraint [Kremer97, Zweben94]. In the conventional



(a) A mechanism

F1: n[Worm1] := n[Shaft1]
F2: n[Gear1] := n[Worm1]/z[Gear1]
F3: n[Shaft2] := n[Gear1]
F4: n[Pulley1] := n[Shaft2]
F5: n[Pulley2] := n[Pulley1] *d[Pulley1]/d[Pulley2]
F6: n[Shaft3] := n[Pulley2]

(b) Attribute relations



(c) A relation network

Figure 4.8: A relation network among the attributes of a mechanism that transfers rotational motion from one shaft to another

constraint-based problem solving systems, first a set of constraints are defined. Constraint propagation is usually used to deduct more constraints from the available constraints to reduce the problem solving space [Zweben94]. In the system developed in this research, the data dependent relations are not the constraints used in the conventional constraint-based problem solving systems. If we want to define mutual dependent relations between two attributes, two quantitative relations should be defined.

4.6 An Example

A mechanical design example is given in this section to show the effectiveness of the introduced feature-based design approach. In this example, a mechanism was designed to transfer motion from a shaft of a motor to another shaft using a number of mechanisms including gear-pair mechanisms, worm-gear mechanism, pulley-belt-drive mechanism, etc.

The first step in the product design using the feature-based design system is to define the class features. All the class features in this example were defined as sub-classes of a class feature called *FeatureGeometry*. The class feature, *FeatureGeometry,* provides attributes representing 2D location coordinates, namely *locX* and *locY*. The class features used in this mechanism are listed in Table 4.2. Functions are used to define the quantitative relations among attributes. For instance, the relation among module, *m*, number of teeth, *z*, and diameter of a gear, *d*, is defined as a function:

```
d[?gear1] := m[?gear1] * z[?gear1]
```

When an instance gear feature is generated, the data dependent relation network is then used to keep the consistency of the attribute values. For instance, if the module of a gear is changed, the diameter can be updated automatically using the data dependent relation network.

Table 4.2: List of class features in this example

Classes	Attributes	Element Features
Gear	m (module), z (number of teeth), d (diameter), n (rotational speed), width, etc.	None
Shaft	d (diameter), n (rotational speed), width, etc.	None
Pulley	d (diameter), n (rotational speed), etc.	None
PulleyBeltDrive	outN, inN (rotational speeds), width, etc.	?pulley1, ?pulley2
GearBox	inN, outN (rotational speeds), etc.	?gear1, ?gear2, ?gear3, ?gear4, ?shaft1, ?shaft2, ?shaft3
Motor	width, length, outN (rotational speed), etc.	None
WormGear	d (diameter), inN, outN (rotational speeds), etc.	None
Mechanism	width, length, etc.	?motor, ?pulleyBeltDrive, ?gearbox1, ?gearbox2, ?wormGear

Product modeling is conducted by generating instance features using class features as templates. When the instance feature *mechanism1* is generated using class feature *Mechanism*, its element features are also generated. The class feature *Mechanism* consists of 2 element-features of the

class type *GearBox.* Each *GearBox* has 7 element-features, which are of the types *Gear* and *Shaft.* Therefore, 2 instance features of the GearBox and 2 sets of the 7 instance features for these 2 gear boxes should be generated. The whole mechanism consists of 22 instance features, as illustrated in Figure 4.9.

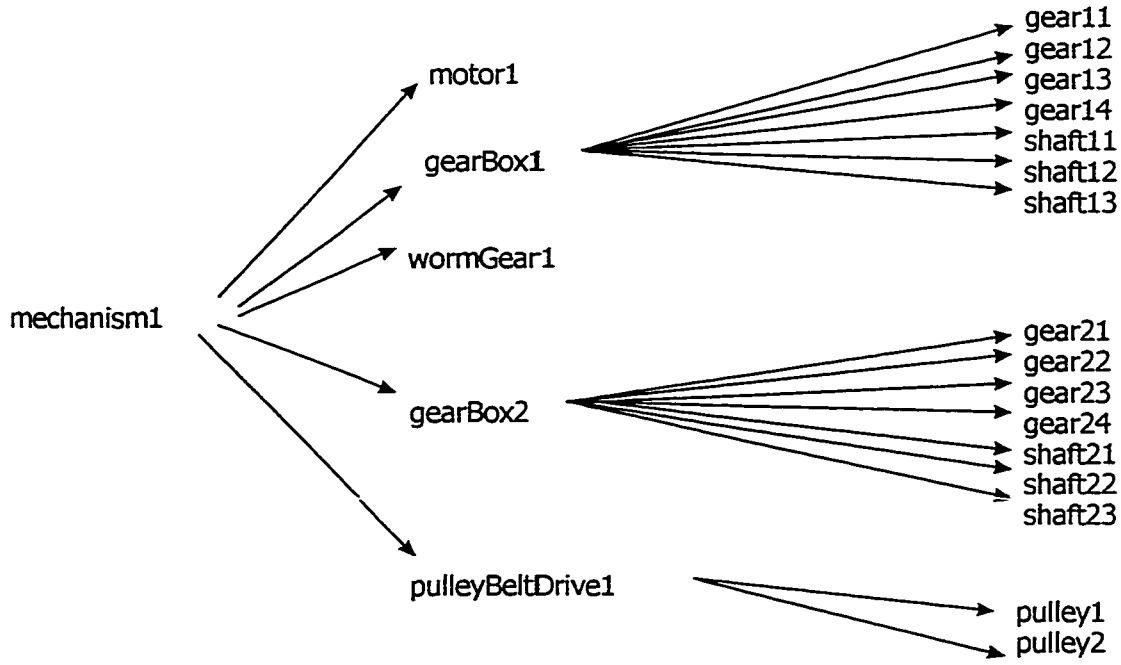


Figure 4.9: Diagram showing instance features that build up the product

The 2D representation of the product is displayed in the *2-D Geometry Browser* of the feature-based design system. A snapshot of the geometry browser displaying the created mechanism is shown in Figure 4.10. The attributes can be changed according to the requirements of the design. The data dependent relation network is used to keep the consistency of these attributes. For instance, if the value of the attribute representing the diameter of the first gear in the first gear-box is increased, the values of related attributes, such as the rotational speeds of the shafts in the pulley-belt-drive mechanism, should also be updated. When the diameter of the gear is changed, the dimensions of

the gearbox are also updated accordingly. The changed product representation is shown in Figure 4.11.

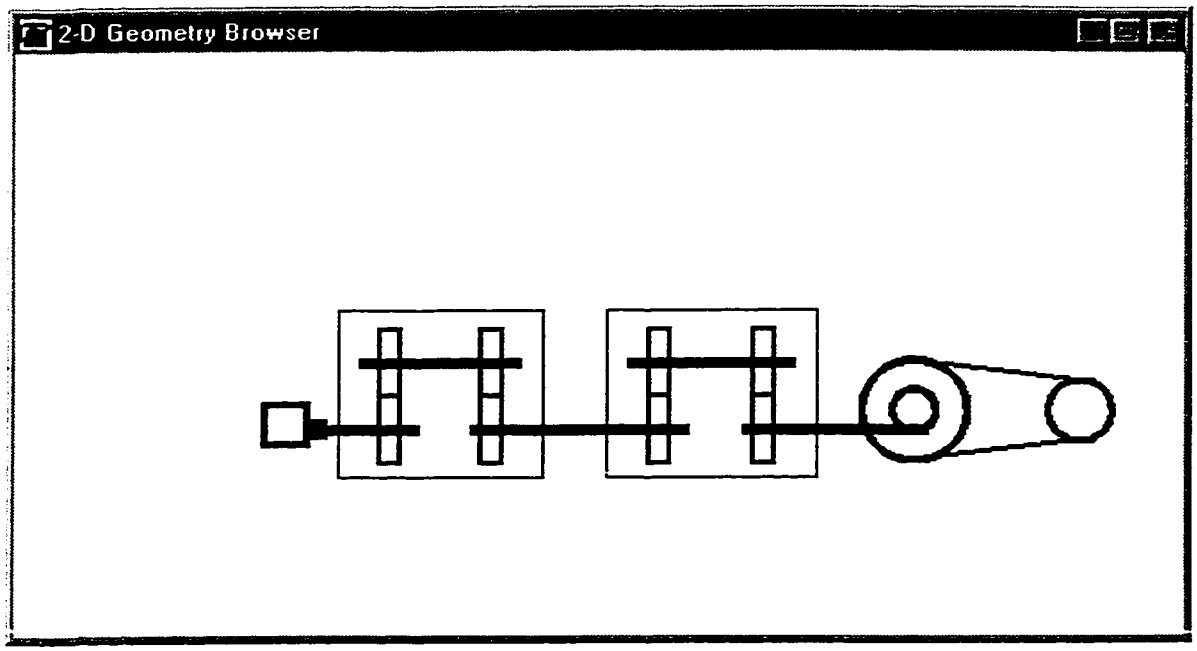


Figure 4.10: Representation of 2-D geometry of *mechanism1*

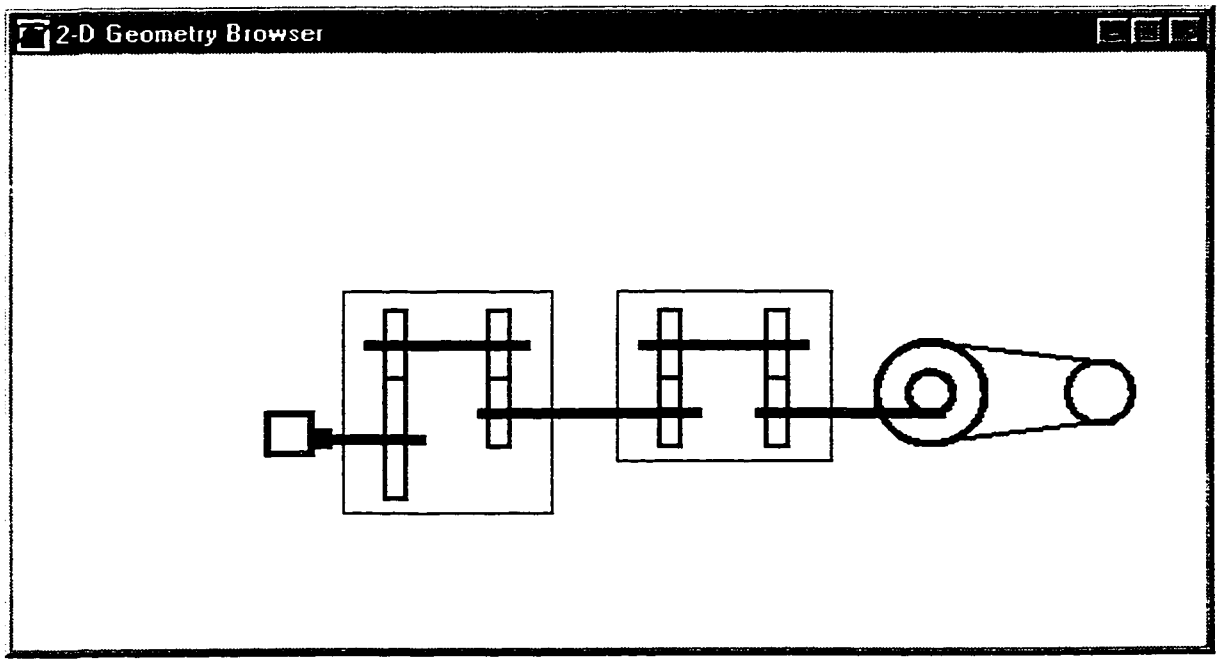


Figure 4.11: Representation of 2-D geometry of *mechanism1* after changing attributes

Chapter 5

Geometry Representation System

This chapter introduces the geometry representation module of the feature-based intelligent design system. First, geometry representation scheme in the feature definitions is introduced. Then the method to associate feature-based symbolic descriptions with 3D solid model is presented. At last, an illustrative example based upon the introduced method is given.

5.1 Geometric Representation in Feature-based Design

Geometric descriptions are part of the feature definitions in the feature-based design system. Geometric descriptions include 2D and 3D descriptions. A feature definition is composed of a number of aspects. There are two different aspects for describing product geometry in this system: (1) the 2D geometric aspect and (2) the 3D geometric aspect.

5.1.1 2D Product Geometry Representation

The 2D geometric descriptions of a class feature are preserved in an instance method. Instance methods in class features are similar to the instance methods in Smalltalk. In addition, in these methods use of attributes and element features is allowed. In a feature definition, the instance method to create 2D geometric output is called *2DDrawing*. This method uses the attributes preserved in feature definitions and Smalltalk 2D drawing functions to display the 2D design result.

The instance features with 2D geometric descriptions are used for displaying the 2D product geometric information through a specially developed interface called *2-D Geometry Browser*, as shown in Figure 5.1.

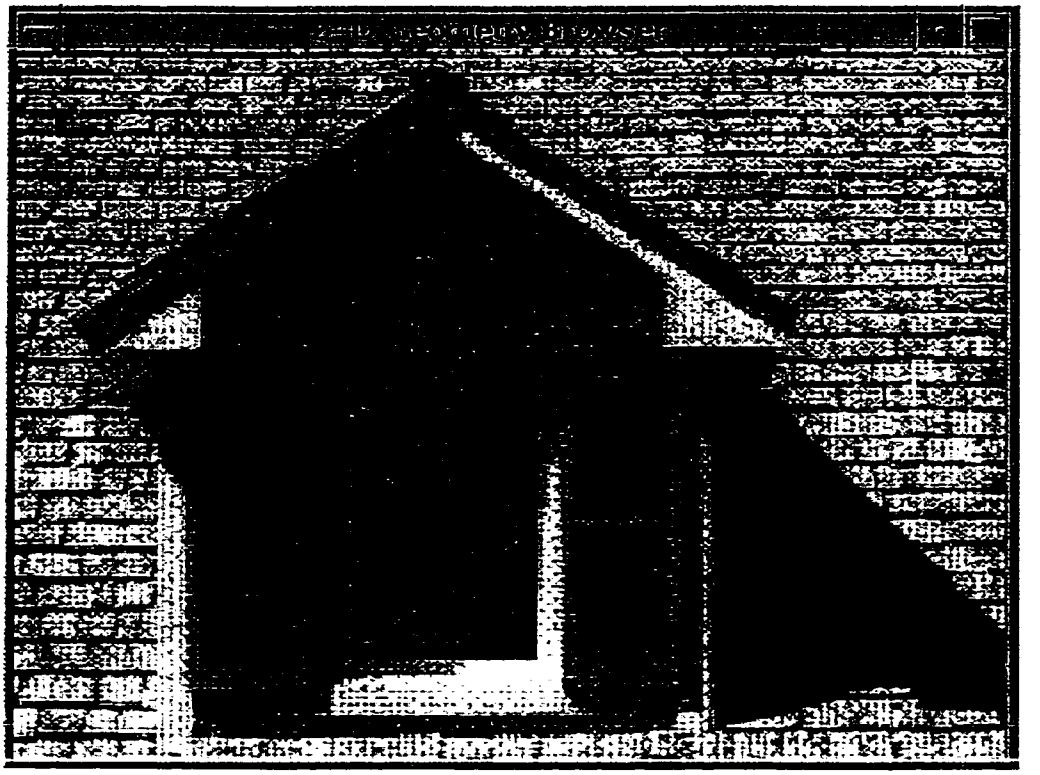


Figure 5.1: The 2-D product geometric output window

Many functions have been developed for this 2D geometric information display window. An object can be displayed against a background to see if the object is matched with the background. An image of the background can be selected from the screen of the computer monitor and stored in the system. The background is then displayed in the 2D product geometry output window, with the 2D output of the selected instance features on the top of the background, as shown in Figure 5.1. Background can be moved and scaled in both *X* and *Y* directions to match itself with the 2D output of the product.

5.1.2 3D Product Geometry Representation

The 3D product geometry is described in a feature aspect called *geometry3D*. A representation scheme has been developed in this research to describe the 3D geometry. In this scheme, generation of objects using 2D and

3D geometric primitives, transformations of the objects including moving and rotating, Boolean operations to these objects including union and subtraction, assignment of material and color properties to the generated objects, etc. are defined using feature descriptions. Details of the representation scheme are given in Section 5.2.

The symbolic 3D geometric descriptions in the instance features are extracted and translated into a neutral format. The neutral geometric descriptions are further translated into 3D solid models by the 3D geometry translation module provided in the geometric modeling systems. This approach is illustrated in Figure 5.2. A 3D geometric modeling system, called 3D Studio MAX [Michael96], has been employed for developing the geometry representation system.

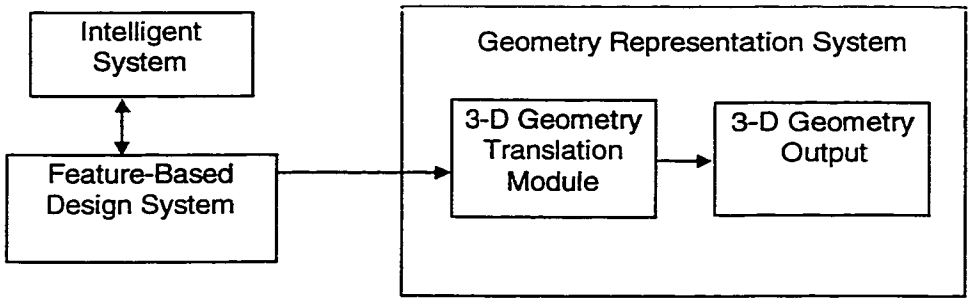


Figure 5.2: Geometry representation system receives the input from the feature-based design system

The geometry representation system provides the following functions:

- Modeling of primitive geometric objects: to create 2D and 3D primitives such as rectangles, polygons, circles, arcs, boxes, cones, cylinders, spheres, etc.
- Creation of 3D objects from 2D objects: to generate 3D solids by *extrude* and *lathe* operations on the 2D primitives.
- Transformation of objects: to change locations and orientations of objects using transformation operators including moving, rotating, etc.

- Assignment of materials and colors: to assign material properties to objects and to set up appearance parameters such as color, shininess, color diffusion, etc.
- Organization of objects: to associate objects in groups.
- Boolean operations to objects: to generate complex objects using Boolean operations including union and subtraction.

5.2 A Geometry Representation Scheme

The 3D geometry of objects is described in terms of primitive geometric objects and a series of operations to these objects. The basic steps involved in representing the geometry are (1) generating objects using primitives, (2) positioning objects, (3) Boolean operations to objects, and (4) assigning material and color properties to the objects. These steps are described by predicates. Each predicate takes the form of (t_1, t_2, ..., t_n), where t_1, t_2, ..., t_n are the fields of this predicate. Fields are described by symbols, strings, integers, floats, variables, attributes and element-features. Therefore the 3D geometry of a product is defined by a collection of predicates.

5.2.1 Generation of Geometric Primitives

A complex object is usually modeled by combining a number of primitives. Geometric primitives are classified as 2D primitives and 3D primitives. During the development of this scheme, issues for ease of understanding and implementation were considered. Each primitive is associated with a name and an ID number. The name is provided in the geometric description, while the ID number is automatically provided by the system when the object is created. The 2D geometric primitives are shown in Figure 5.3. The 3D geometric primitives are illustrated in Figure 5.4. Generation of these primitives is carried out by the following built-in predicates.

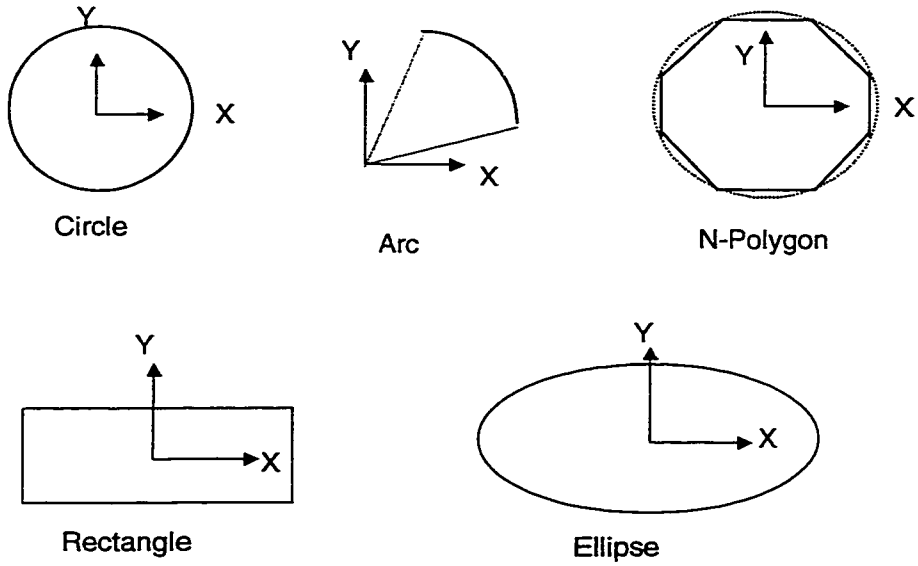


Figure 5.3: 2-D geometric primitives

1. **Circle**

   Format: (circle, <name>, <radius>)

   <name>: name of the new object.

   <radius>: radius of the circle.

   Reference point: center of the circle.

2. **Arc**

   Format: (arc, <name>, <radius>, <from>, <to>)

   <name>: name of the new object.

   <radius>: radius of the arc.

   <from>: starting angle.

   <to>: ending angle.

   Reference point: center of the arc.

3. **N-Polygon**

   Format: (n-polygon, <name>, <radius>, <side>)

   <name>: name of the new object.

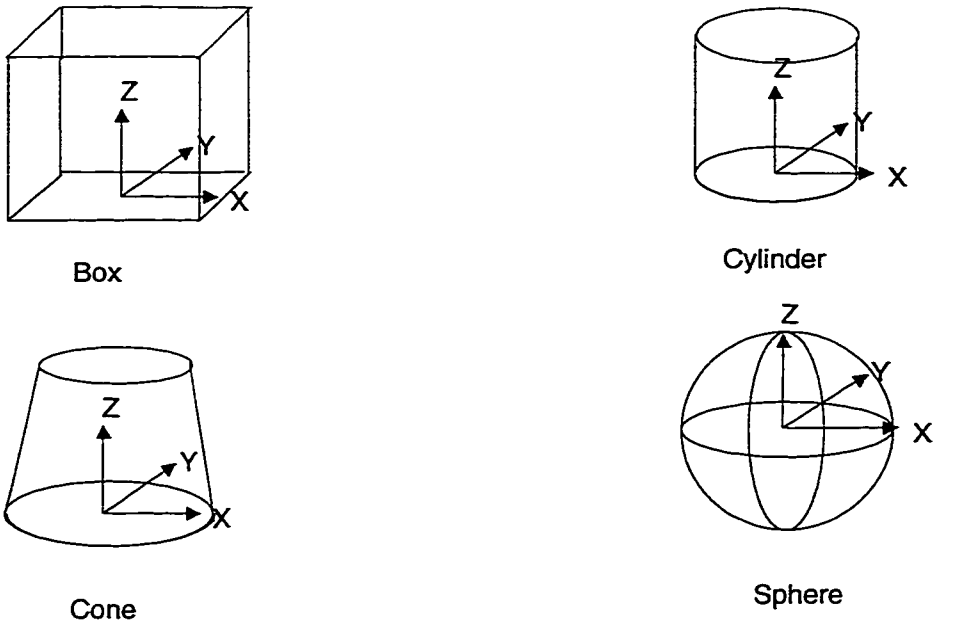


Figure 5.4: 3-D geometric primitives

<radius>: radius of the regular polygon.

<side>: number of sides of the regular polygon.

Reference point: center of the regular polygon.

4. Rectangle

Format: (rectangle, <name>, <length>, <width>)

<name>: name of the new object.

<length>: dimension in x direction.

<width>: dimension in y direction.

Reference point: center of the rectangle

5. Ellipse

Format: (ellipse, <name>, <length>, <width>)

<name>: name of the new object.

<length>: dimension in x direction.

<width>: dimension in y direction.

Reference point: center of the ellipse.

6. Box

Format: (box, <name>, <length>, <width>, <height>)

<name>: name of the new object.

<length>: dimension in x direction.

<width>: dimension in y direction.

<height>: dimension in z direction.

Reference point: center point of the bottom plane.

7. Cylinder

Format: (cylinder, <name>, <radius>, <height>)

<name>: name of the new object.

<radius>: radius.

<height>: height.

Reference point: center point of the bottom plane.

8. Cone

Format: (cone, <name>, <radius1>, <radius2>, <height>)

<name>: name of the new object.

<radius1>: radius of the bottom circle.

<radius2>: radius of the top circle.

<height>: height.

Reference point: center point of the bottom circle.

9. Sphere

Format: (sphere, <name>, <radius>)

<name>: name of the new object.

<radius>: radius.

Reference point: center point of the sphere.

5.2.2 Operations for Creating 3D Primitives from 2D Primitives

Extrusion and Lathe are the operations used for generating 3D solids from 2D sections. Extrusion is an operation to generate a 3D object with uniform cross-section, by sweeping a 2D section in the direction of its normal. Lathe is an operation to revolve a 2D section about an axis by a certain angle to generate the 3D object. Extrusion and lathe operations are illustrated in Figure 5.5.

Generation of a 3D solid using either of these two operations is carried out by a two-step process. The first step is the creation of the 2D section on which the operation will be applied. In the case of *extrude* operation, the 2D section is a horizontal cross-section of the solid to be generated. In the case of lathe operation, the 2D section is half of the longitudinal cross-section of the solid to be generated. The second step is the application of *extrude* or *lathe* operation to the 2D section created in the first step to achieve the 3D solid. The two operations are illustrated in Figure 5.5. The built-in predicates for these two

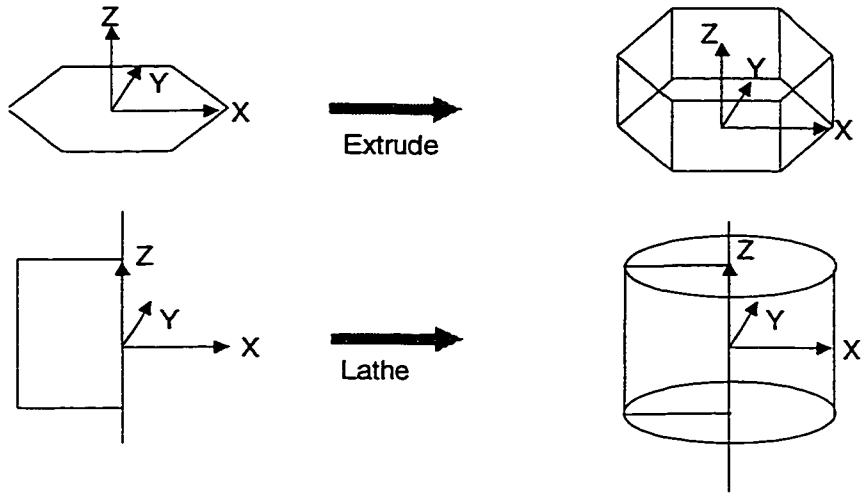


Figure 5.5: Operations to the 2-D primitives

operations are defined as:

1. Extrude

Format: (extrude, <name1>, <name2>, <height>)

<name1>: name of the existing 2D object.

<name2>: name of the new 3D object.

<height>: dimension in z direction.

Reference point: reference point of the 2D section.

2. Lathe

Format: (lathe, <name1>, <name2>, <degree>)

<name1>: name of the existing 2D object.

<name2>: name of the new 3D object.

<degree>: angle to be rotated.

Reference point: geometrical center of the resulting object.

5.2.3 Transformation Operations

Transformation operations are used to change the locations and orientations of objects to place them at desired positions. Two types of transformation operations are provided in this system: *move* and *rotate*. These operations are shown in Figure 5.6.

1. Move

Format: (move, <name>, <x>, <y>, <z>)

<name>: name of the existing object.

<x>, <y>, <z>: translation in x, y, and z directions.

2. Rotate

Format: (rotate, < name >, < θ_x >, < θ_y >, < θ_z >)

< name >: name of the existing object.

< θ_x >, < θ_y >, < θ_z >: rotation about x, y, and z axes.

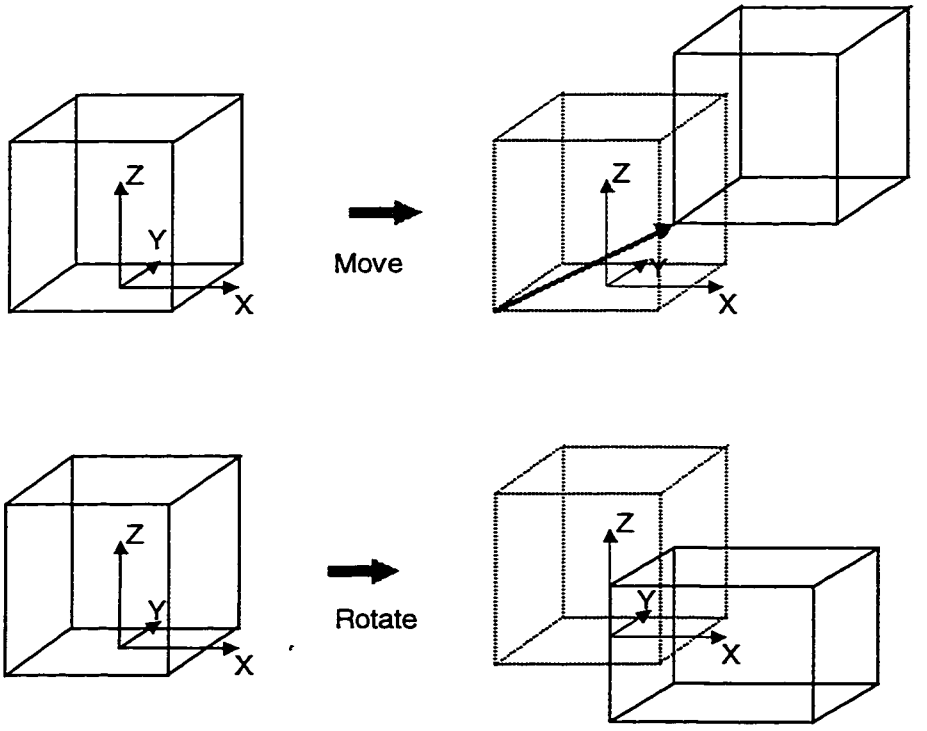


Figure 5.6: Transformation operations

5.2.4 Boolean Operations

Boolean operations are used to create complex objects. In this system, two types of Boolean operations are provided: union and subtraction, as shown in Figure 5.7. In the current system, Boolean operations have not been fully implemented in the 3D solid modeling module, due to the limitation of the 3D Studio MAX system. The following syntax is used for representing the Boolean operations:

1. Union

Format: (union, <name1>, <name2>, <name>)

<name1>, <name2>: names of the two existing objects.

<name>: name of the new object.

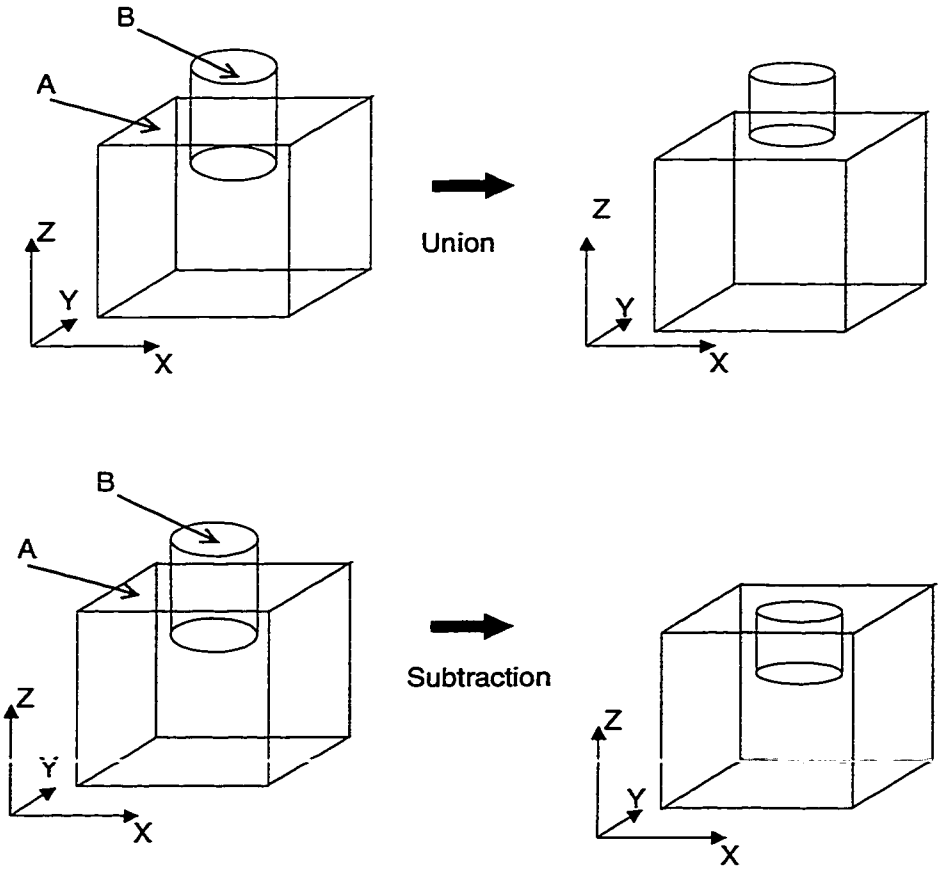


Figure 5.7: Boolean operations

2. Subtraction

Format: (subtraction, <name1>, <name2>, <name>)

<name1>: name of the base object.

<name2>: name of the object to be subtracted.

<name>: name of the new object.

5.2.5 Assignment of Material and Color Properties

The geometric model can be made to look more realistic and attractive by assigning the properties of material and color. The following built-in predicates are defined for this purpose.

1. **Material**

   Format: (material, <name>, <type>)

   <name>: name of the existing object.

   <type>: type of the material to be assigned to the object. The materials currently defined in the system are glass, wood, copper, steel, ferrous, etc.

2. **Diffuse-Color**

   Format: (diffuse, <name>, <red>, <green>, <blue>)

   <name>: name of the existing object.

   <red>, <green>, <blue>: the three color components represented by values between 0 and 1.

3. **Specular-Color**

   Format: (specular, <name>, <red>, <green>, <blue>)

   <name>: name of the existing object.

   <red>, <green>, <blue>: the three color components represented by values between 0 and 1.

4. **Ambient-Color**

   Format: (ambient, <name>, <red>, <green>, <blue>)

   <name>: name of the existing object.

   <red>, <green>, <blue>: the three color components represented by values between 0 and 1.

Definitions of diffuse color, specular color, and ambient color can be found in Michael [Michael96].

5.2.6 Organizing Objects in Groups

Objects can be organized in groups. Each group is composed of a number of objects and sub-groups. Therefore, all the objects of a product are organized in a hierarchical data structure. The group operation is defined by:

Group

Format: (group, <name1>, <name2>, <name>)

<name>: name of the new group.

<name1>, <name2>: objects and sub-groups.

5.3 3D Studio MAX

3D Studio MAX by Kinetix is an system for modeling 3D objects and displaying the 3D objects using its graphics environment. Since nearly all the functions in 3D Studio MAX are plug-in components programmed using C++, modification and expansion of the 3D Studio MAX system can be carried out easily. A snapshot of the 3D Studio MAX user interface is shown in Figure 5.8.

3D Studio MAX is one of the best 3D computer-graphics environment so far. The 2D and 3D geometric objects can be created, transformed, modified, rendered, grouped, etc. using the menu items provided in the system. Compound objects, such as Boolean objects, can be created using primitives. In addition, material properties, texture maps, and colors can be assigned to the created objects. Lights, camera, viewing direction, etc. can also be defined and selected. In this research, the following functions are primarily used:

- Generation of 2D and 3D primitives,
- Transformation of objects to change locations and orientations by translating and rotating these objects,
- Assignment of material properties to the objects to enhance the display visualization,

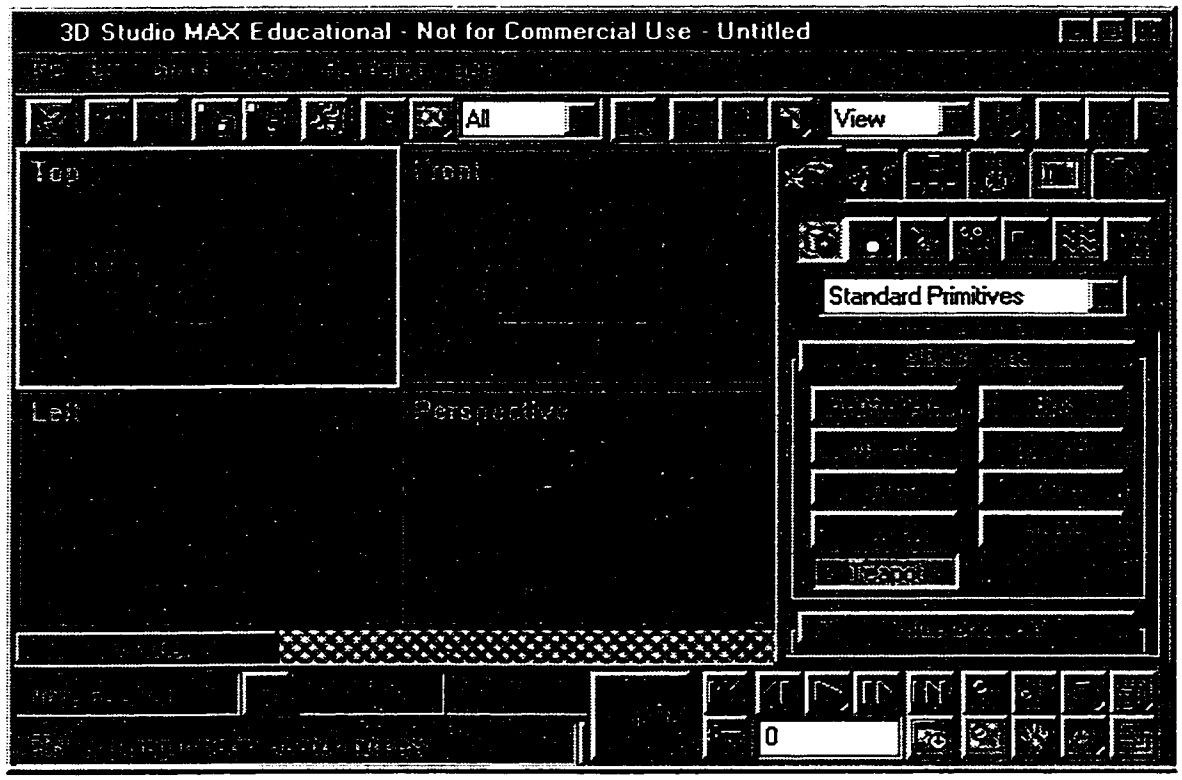


Figure 5.8: User interface of 3D Studio MAX

- Association of objects in groups,
- Boolean operations to the objects such as union, subtraction, etc. In the current system, the Boolean operations have not been fully implemented.

5.4 Generation of 3-D Solid Model

The 3D solid model of a product is created through 4 steps:

1. *Defining the 3D geometric descriptions in class features:*

   The 3D geometric descriptions of a product in a feature definition follow the scheme introduced in Section 5.2. Attributes and element features can be used in these descriptions. An example of 3D geometric descriptions for the feature *Gear* is illustrated in Table 5.1. The 3D geometry of a gear is

created by extruding a 2D circle, and rotated and moved to the desired position. In addition, the material and color properties are also defined in the 3D geometric descriptions. The final geometry is described by a variable *?self.*

2. *Generating 3D geometric descriptions in instance features using the class features as templates:*

   Geometric descriptions in the instance features are generated using the class feature descriptions as templates. The variables that have been used in the class feature descriptions for representing element features are replaced by the instance feature names. The instance feature, *gear1*, is generated using class feature, *Gear,* as template. In this example, the variable *?self* is replaced by the name of the instance feature *gear1*.

3. *Extracting the 3D geometric descriptions from instance features and translating these descriptions into a neutral form:*

   The symbolic geometric descriptions in the instance features are then *filed out (extracted)* to a neutral format from the feature-based design system. First all the instance features in a category, representing a complete product, are selected. Then the 3D geometric descriptions in the instance features are extracted and translated into the neutral form. During the extraction of geometric descriptions from the instance features, the following steps are taken:

   (1) As described in Section 5.2, every geometric object in a feature is associated with a name. To ensure that each object in the translated form is unique, the object name in the neutral form is defined by combining the instance feature name and the object name described in the instance feature.

(2) Attributes in the instance feature descriptions are replaced by their values.

(3) If the product consists of more than one instance feature, the geometric descriptions of these instance features are combined together. The 3D geometric descriptions of an instance feature are extracted, only after the geometric descriptions of its element features have already been extracted.

The extracted 3D geometric descriptions of instance feature *gear1* are shown in Table 5.1.

Table 5.1: Geometric descriptions of a gear in different places

Places	Geometric Descriptions
Class Feature: *Gear*	(circle, cir1, radius[?self]) (extrude, cir1, ?self, thickness[?self]) (rotate, ?self, rotX[?self], rotY[?self], rotZ[?self]) (move, ?self, locX[?self], locY[?self], locZ[?self]) (material, ?self, copper) (ambient, ?self, 0.9, 0.4, 0.5)
Instance Feature: *gear1*	(circle, cir1, radius[gear1]) (extrude, cir1, gear1, thickness[gear1]) (rotate, gear1, rotX[gear1], rotY[gear1], rotZ[gear1]) (move, gear1, locX[gear1], locY[gear1], locZ[gear1]) (material, gear1, copper) (ambient, gear1, 0.9, 0.4, 0.5)

Table 5.1: Geometric descriptions of a gear in different places (continued)

Places	Geometric Descriptions
Extracted Neutral Data	circle gear1-cir1 22.0 extrude gear1-cir1 gear1 10 rotate gear1 0 90 0 move gear1 -73.3333 0 187.0 material gear1 copper ambient gear1 0.9 0.4 0.5

4. *Translating the neutral 3D descriptions into the 3D solid model:*

In this research, the extracted 3D geometric descriptions are first preserved in a file, and then translated into the 3D solid model. The 3D Studio MAX has been used as the platform for displaying the 3D geometry. The module to translate the 3D neutral geometric descriptions preserved in the file was developed as a plug-in application of the 3D Studio MAX. A plug-in application in 3D Studio MAX is a piece of program written using C++, and used to access the objects in 3D Studio MAX. The geometric description translation module, developed as a 3D Studio MAX plug-in application, reads the neutral 3D descriptions preserved in the file, and translates these descriptions into 3D Studio MAX command for generating the 3D product geometry. A generated gear mechanism in the 3D Studio MAX environment is shown in Figure 5.9.

5.5 An Example

In this section, an example is given to show how the 3D design geometry is described in the features using the introduced method. An assembly, as shown in Figure 5.10 (a), is composed of two components, a base and a shaft. The two components and the assembly are described by three class features: *Base, Shaft,* and *BaseShaft,* respectively as shown in Figure 5.10 (b). In the

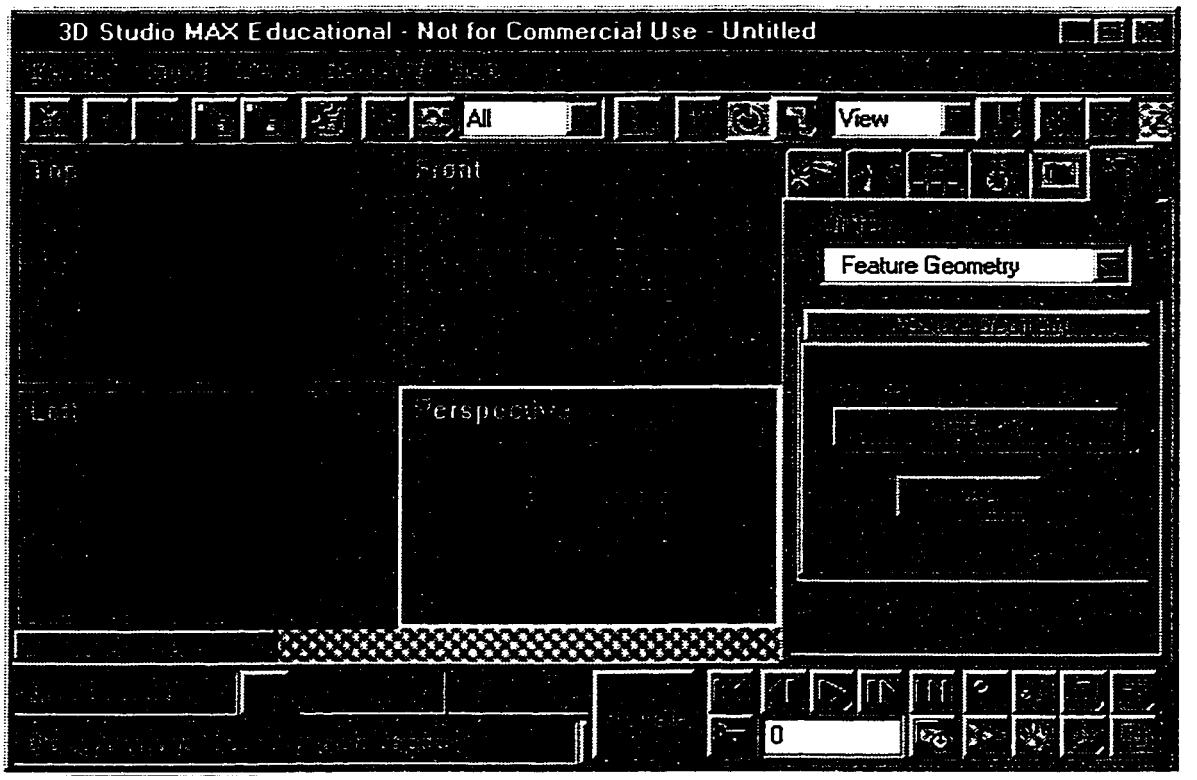
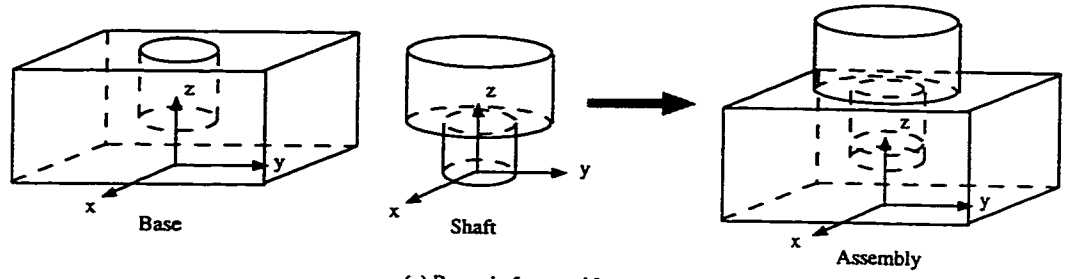


Figure 5.9: A snapshot of 3D Studio MAX with a gear mechanism

class feature *Base*, the geometry of base is defined by three steps: (1) A rectangle is extruded to obtain a block with required dimensions. (2) A cylinder is generated and positioned at the desired location. (3) The base is created by a subtraction Boolean operation to the block and cylinder. In the *Shaft* class feature, the geometry of shaft is defined as the result of a union Boolean operation to two cylinders.

The final geometric object name is defined using the variable *?self.* Three instance features, *base1, shaft1,* and *baseShaft1,* are generated from the three class features respectively. The instantiated 3D geometric descriptions in these three instance features are illustrated in Figure 5.10 (c). In the three instance



(a) Base-shaft assembly

Class Feature: Base

3D-Geometry:

(rectangle, rec1, length[?self], width[?self])
(extrude, rec1, box1, height[?self])
(cylinder, cyl1, holeRadius[?self], holeHeight[?self])
(move, box1, locX[?self], locY[?self], locZ[?self])
(move, cyl1, locX[?self], locY[?self], locZ[?self] + height[?self] - holeHeight[?self])
(subtraction, box1, cyl1, ?self)

Instance Feature: base1

3D-Geometry:

(rectangle, rec1, length[base1], width[base1])
(extrude, rec1, box1, height[base1])
(cylinder, cyl1, holeRadius[base1], holeHeight[base1])
(move, box1, locX[base1], locY[base1], locZ[base1])
(move, cyl1, locX[base1], locY[base1], locZ[base1] + height[base1] - holeHeight[base1])
(subtraction, box1, cyl1, base1)

Class Feature: Shaft

3D-Geometry:

(cylinder, cyl1, radius1[?self], height1[?self])
(cylinder, cyl2, radius2[?self], height2[?self])
(move, cyl1, locX[?self], locY[?self], locZ[?self])
(move, cyl2, locX[?self], locY[?self], locZ[?self] + height1[?self])
(union, cyl1, cyl2, ?self)

Instance Feature: shaft1

3D-Geometry:

(cylinder, cyl1, radius1[shaft1], height1[shaft1])
(cylinder, cyl2, radius2[shaft1], height2[shaft1])
(move, cyl1, locX[shaft1], locY[shaft1], locZ[shaft1])
(move, cyl2, locX[shaft1], locY[shaft1], locZ[shaft1] + height1[shaft1])
(union, cyl1, cyl2, shaft1)

Class Feature: BaseShaft

Element-Features:

?BaseFeature Base
?ShaftFeature Shaft

3D-Geometry:

(group, group1, ?BaseFeature, ?ShaftFeature)

Instance Feature: baseShaft1

Element-Features:

base1 Base
shaft1 Shaft

3D-Geometry:

(group, group1, base1, shaft1)

(b) Class feature definitions

(c) Instance feature definitions

```
rectangleb base1-rec1 14.0 20.0
extrude base1-rec1 base1-box1 12.0
cylinder base1-cyl1 3.0 7.0
move base1-box1 0.0 0.0 0.0
move base1-cyl1 0.0 0.0 5.0
subtraction base1-box1 base1-cyl1 base1
cylinder shaft1-cyl1 3.0 5.0
cylinder shaft1-cyl2 6.0 7.0
move shaft1-cyl1 0.0 0.0 7.0
move shaft1-cyl2 0.0 0.0 12.0
union shaft1-cyl1 shaft1-cyl2 shaft1
group baseShaft1-group1 base1 shaft1
```

(d) 3-D geometric descriptions for 3D Studio MAX

Figure 5.10: 3D geometric descriptions of a base-shaft assembly

features, the positions of the two components are defined by 6 coordinate attributes. The attribute values and relations are:

```
LocX[base1] := 0.0
LocY[base1] := 0.0
LocZ[base1] := 0.0
LocX[shaft1] := LocX[base1]
LocY[shaft1] := LocX[base1]
LocX[shaft1] := height[base1] - height1[shaft1]
```

When the 3D geometric descriptions in the instance features are used to generate the input data for the 3D Studio MAX, all the attribute names are replaced by attribute values. The temporary geometric object variables defined in these instance features are associated with the unique instance feature names for generating intermediate geometric objects in 3D Studio MAX environment. The generated 3D geometric descriptions for 3D Studio MAX are shown in Figure 5.10 (d).

Chapter 6

Intelligent System Based Product Modeling

This chapter introduces the intelligent system that is used to automate the product modeling process. The methods for modeling the knowledge base and the database are presented first. The knowledge-based reasoning mechanism is then introduced. Finally, an example is given to illustrate the introduced approach for automated product modeling.

6.1 Introduction

An intelligent system has been developed in this research for conducting decision-making activities during the process product modeling through knowledge-based reasoning. The intelligent system consists of the following two components:

1. a knowledge base
2. an inference engine

The knowledge base is the place to store knowledge that is described as rules using the IF-THEN data structure, while the database is the place to preserve product data that are described by instance features. During the product modeling process, the condition parts of the rules stored in the knowledge base are compared with the database. If the condition part of a rule is matched with the database, the result part of the rule should be executed. This process is called knowledge-based reasoning or inference. Matching and executing of rules are conducted by an inference engine. The architecture of an intelligent system is shown in Figure 6.1.

In this research, an intelligent system has been developed for conducting automated product modeling. As described in the previous chapters, product

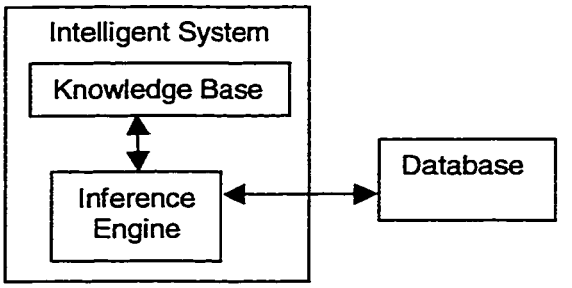


Figure 6.1: Intelligent system architecture

modeling is carried out by generating instance features using class features as templates. During the product modeling process, the instance feature descriptions such as attributes, element features, etc. can also be changed.

6.2 Modeling of the Knowledge Base and Database

6.2.1 Modeling of the Knowledge Base

In a knowledge base system, knowledge is described using a knowledge representation method. Typical knowledge representation methods include rules, frames, semantic networks, predicates, etc [Rich91]. In this research, knowledge is represented in form of rules. To improve the inference efficiency, only relevant rules and data are selected in automated product modeling.

A method of selecting only the relevant part of the knowledge base and the database has been developed. Since the database representing a product is described by instance features, the partial database of a product is selected by choosing only the relevant instance features. The knowledge, represented in form of rules, is organized in groups. In the knowledge-based reasoning, only the relevant groups of the knowledge are selected. This idea is illustrated in Figure 6.2.

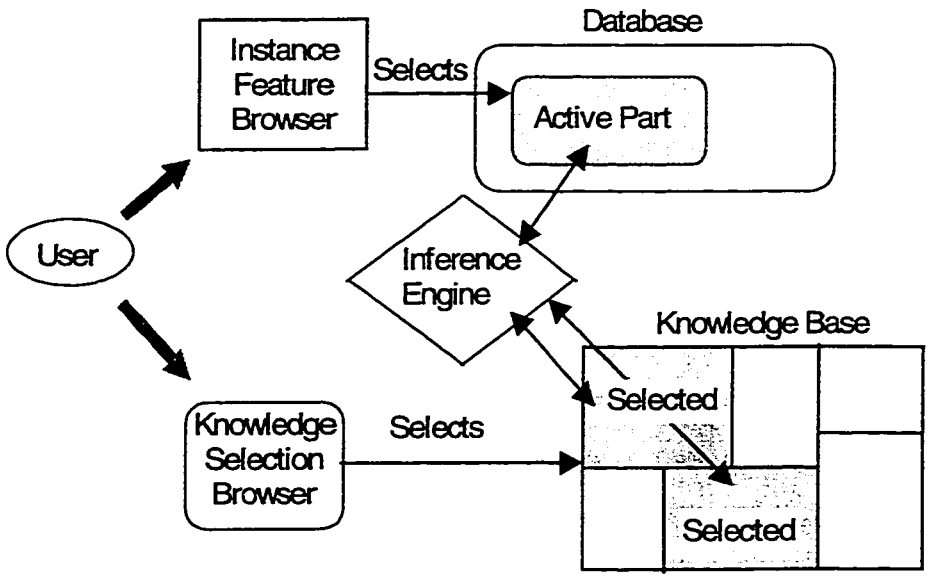


Figure 6.2: Only the relevant parts of knowledge base and database are involved in the inference process

The rules representing knowledge are grouped into a number of rule-bases. For instance, the rules related to gear design are bundled into a rule-base. When a gear is being designed, this rule-base should be selected for inference. Rule-bases are organized by *categories*. The categories do not affect the inference process, except for identifying required rule-bases in an efficient manner due to the large size of rule-bases.

The purpose of not describing all the rules at the same place but in different rule-bases is to improve the inference efficiency and knowledge modeling modularity. Every rule-base is associated with a rule-base name. A rule-base is the smallest unit that can be selected for inference. The method of selecting a rule-base is described in Section 6.3.1. The user is free to select any rule-bases from any categories for the inference.

A rule represents a piece of knowledge in the form of an IF-THEN data structure. The IF part of the rule describes the conditions that have to be satisfied before the THEN part of the rule can be executed. In this system, each

rule has a rule name. Hence a rule is identified by its: (1) category name, (2) rule-base name, and (3) rule name.

In engineering practice, a number of conditions have to be satisfied simultaneously before a sequence of actions can be performed. To represent this type of knowledge, both the IF part and THEN part of the rule are described by a collection of *patterns*, each of them representing a separate condition or action. These patterns are linked together by operator of logical-and (&). A rule is matched with the database, only when all the patterns of the rule are satisfied. The patterns in THEN part of the rule represent actions such as to add certain data in the database. A pattern is described in the form of $(x_1,x_2,.....x_n)$, where $x_1,x_2,.....x_n$ are called the terms. A pattern is also known as a *predicate*.

A term in a pattern can be a symbol constant, a numerical constant, a variable, or an attribute. During the inference, a variable can be matched with a number of data. In this system, a variable is described by a string starting with the letter '?'. The methods of representing the attributes and variables are the same as those introduced in Chapter 4.

A rule-base for building product design is shown in Figure 6.3. In this example, the rule-base consists of three rules. The rule-base name is described at very beginning. The keyword *"Rule:"* precedes the rule name. Both IF and THEN part of a rule consist of one or several patterns.

Built-in predicates are the patterns (predicates) to perform special actions. These special actions include generating instance features, assigning attribute values, removing attributes, and so on. The built-in predicates are described in both the condition parts and the result parts of the rules. Part of the built-in predicates in the condition parts and results parts of rules are described in the Table 6.1 and Table 6.2, respectively.

```
Rule-base: DoorRightSectionDesign

Rule: RectangularRight
IF (center, ?x) & (<=, gridX[?x], 0)
THEN (removeFeature, ?right) & (generateFeature, DoorRightA, ?right)
& (notify, The right door has been changed according to the center
door)

Rule: GridRight
IF (center, ?x) & (>=, gridX[?x], 1)
THEN (removeFeature, ?right) & (generateFeature, DoorRightB, ?right)
& (notify, The right door has been changed according to the center
door)

Rule: GridNumbers
IF (center, ?x) & (right, ?y)  & (>=, gridX[?x], 1) & (>=,
gridY[?x], 1)
THEN (assignAttribute, gridX[?y], gridX[?x]) & (assignAttribute,
gridY[?y], gridY[?x])
```

Figure 6.3: A rule base consisting of three rules

Table 6.1: The built-in predicates in the condition parts

Names	Functions	Examples
=	To check if two terms are equal. Terms could be attributes or numerical constants.	(=, m[?x], 2) (=, n[?x], n[?y])
<	To check if the value of the second term is less than the value of the third term. Usage is same as = built-in predicate.	(<, d[?shaft1], 10.0) (<, n[?gear1], n[?gear2])
<=	To check if the value of the second term is less than or equal to the value of the third term. Usage is same as = built-in predicate.	(<=, locX[?gear1], 100.0) (<=, radius[gear1], radius[?x])
>	To check if the value of the second term is greater than the value of the third term. Usage is same as = built-in predicate.	(>, d[?shaft1], 10.0) (>, n[?gear1], n[?gear2])

Table 6.1: The built-in predicates in the condition parts (continued)

>=	To check if the value of the second term is greater than or equal to the value of the third term. Usage is same as = built-in predicate.	(>=, locX[?gear1], 100.0) (>=, radius[gear1], radius[?x])
question	To ask the user to answer yes or no to a question.	(question, Do you want to change the color of the window panel)

Table 6.2: The built-in predicates in the result parts

Names	Functions	Examples
generateFeature	To generate an instance feature of the selected class feature and to preserve it in the current active instance feature as an element-feature.	(generateFeature, Gear, ?x)
removeFeature	To remove an element-feature of the current active instance feature.	(removeFeature, ?x)
assignAttribute	To assign a value to an attribute.	(assignAttribute, m[?gear1], 2.0)
removeAttribute	To remove an attribute.	(removeAttribute, m[gear2])
notify	To inform the designer with a message.	(notify, The window now has a semicircular top)

This system provides a user interface for viewing and editing the rule-bases. This interface is called the *Rule Base Browser*, as shown in Figure 6.4. In this browser, the rule-bases are organized in different categories. To define a new rule-base, one has to select an existing category or add a new category. The bottom view of the Rule Base Browser is used to define a new rule-base. The menu in the bottom view of the Rule Base Browser is used for editing and compiling the rules. The user should provide a name at the top of a rule-base. Each rule has a rule name, which is preceded by the keyword *Rule*. A rule-base is compiled by executing the *accept* item of the bottom view menu. In the knowledge-based design, only relevant rule-bases are selected for rule-based inference to improve the reasoning efficiency.

6.2.2 Modeling of the Database

The database for representing the product is modeled by instance features. Instance features are organized in a hierarchical data structure. Details regarding database modeling can be found in Chapter 4. The partial database for reasoning is selected by choosing an instance feature as the active instance feature. When the active instance feature is selected, all its elements, including element features, attributes, qualitative relations among features and quantitative relations among attributes, are considered active in the rule-based inference.

The database is dynamically updated during the inference process. When the rules are fired, new product description data are added to the database. The instance features generated during inference are considered as element features of the active instance feature.

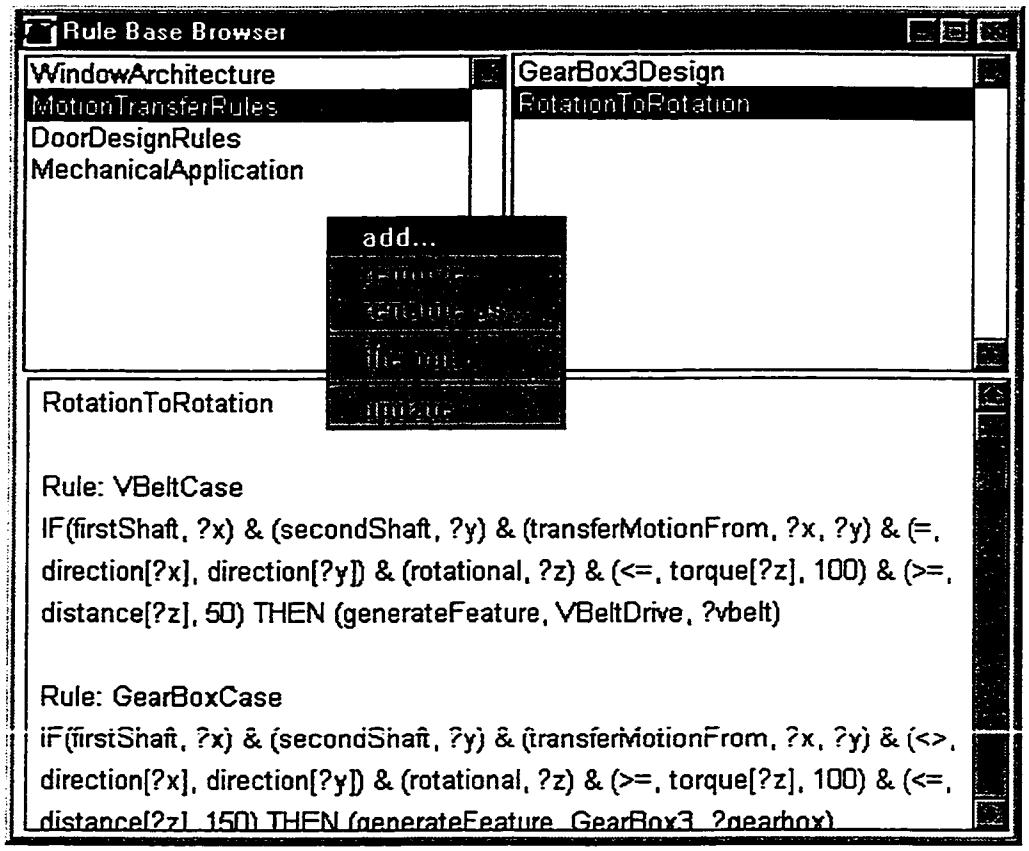


Figure 6.4: A snapshot of the rule base browser, used for defining and editing knowledge

6.3 Knowledge-based Inference

In this research, knowledge-based design starts with the selection of relevant knowledge base and database. The knowledge-based inference is conducted to model a design database.

6.3.1 Selection of Partial Knowledge Base and Database

During the knowledge-based design, the user should first select the relevant rule-bases. The Rule Base Selection Browser is used for selecting the relevant rule-bases for the inference. A snapshot of the Rule Base Selection Browser is shown in Figure 6.5.

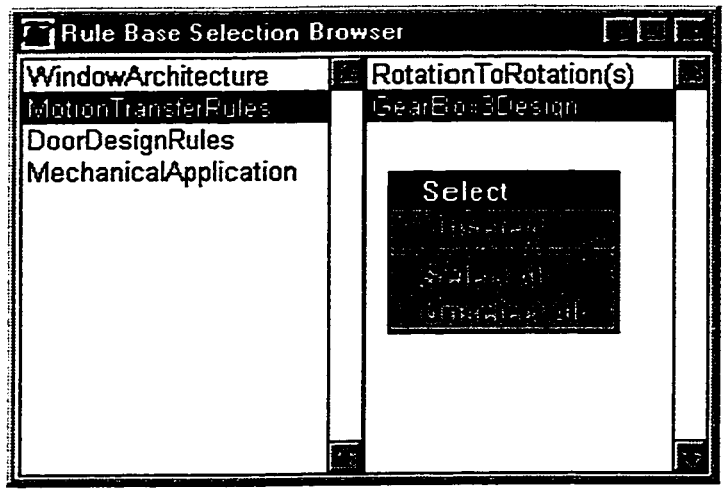


Figure 6.5: A snapshot of the rule-base selection browser

The menu items for the right-side view are used to select/unselect the rule-bases. By comparing the Rule Base Selection Browser with the Rule Base Browser, it can be seen that the list of categories and the lists of rule-base names in the two browsers are identical. This is expected since only the rule-bases already defined in the Rule Base Browser can be selected in the Rule Base Selection Browser. The selected rule-bases are marked with "(s)" in the browser. The selected rule-bases can also be unselected.

The partial database used for inference is selected by choosing an instance feature as the active instance feature, which consists of element features, attributes, qualitative relations among features, and quantitative relations among attributes. Selection of the active instance feature is conducted using the Instance Feature Browser.

The advantages of selecting only the relevant knowledge base and database are:

1. The computation combinatory explosion problem due to the large size of knowledge base and database can be solved.

2. Since the results of inference depend upon the selected rule-bases, different design candidates can be generated by selecting different rule-bases.

6.3.2 Sequence of Matching and Executing Rules

Once the relevant rule-bases and active instance feature have been selected, the inference can be started by selecting the menu item *Start Inference* from the Instance Feature Browser.

Inference is carried out by matching the condition parts of the rules (IF parts) with the data in the active instance feature, and executing the THEN parts of the rules if the condition parts are satisfied. All the selected rules are considered to be at the same level during inference, irrespective of the categories and the rule-bases they belong to. First all the rules in the selected rule-bases are stored in the same place. Matching and execution of these rules is conducted from the first one to the last one. The inference does not stop after the last rule has been accessed. Instead, inference goes back to the matching of the first rule again. This inference process is continued until no rule can be matched. Therefore, if there is no contradiction in knowledge base, the sequence of rules is not important to the reasoning results, as illustrated by the following example:

Example

```
Rules:
          Rule1: IF (p, a, b) THEN (r, a)
          Rule2: IF (r, a) THEN (p, a)
          Rule3: IF (s, a) & (t, b) THEN (p, a, b)
Facts:
          (s, a), (t, b)
```

The reasoning is carried out as follows:

1. Reasoning begins with the Rule1. At this stage, this rule cannot be matched.

2. The reasoning engine then starts to access the Rule2. At this stage Rule2 cannot be fired either.
3. When Rule3 is accessed, since the condition part of this rule is satisfied, the result (p, a, b) is added to the database.
4. Reasoning returns back to Rule1. Now the condition part of this rule is matched, and (r, a) is then added to the database.
5. The Rule2 is then matched, and (p, a) is added to the database.
6. Since Rule3 has already been fired with the same condition, this rule is not fired again. In the same way, Rule1 cannot be fired again. Since Rule2 is the last rule fired so far, and no other rule can be fired, the reasoning is terminated.

In the conventional programming with IF-THEN expressions, the sequence of these IF-THEN expressions is important. For instance, if the rules in the above example are these IF-THEN expressions in the conventional program, only Rule3 can be fired and (p, a, b) added to the database. If Rule3 is moved before Rule1, all these three rules should be fired and (p, a, b), (r, a), and (p, a) added to the database. However, in the rule-based knowledge representation, reasoning results using rules with different sequences should be the same.

6.3.3 Matching of a Rule

A rule is composed of a number of patterns. Matching of a rule is carried out by comparing all its patterns in the IF part of the rule with the data in the active instance feature. During the matching process, a pattern in the IF part of the rule is picked up and its terms are compared with the database. If all the terms are matched, the pattern is considered matched. The next pattern is picked up for matching only when the previous pattern has been matched. If a pattern could not be matched, matching of this rule is considered as a failure.

When all the terms in a pattern are constants, then these terms are simply compared with the data preserved in the active instance feature. If this pattern has been defined in the database, matching of this pattern is a success. When variables are used in the patterns, these variables should be matched with the constants in the database. A variable could be matched with a number of constants. For example the pattern *(P, ?X)* can be matched with both of the facts *(P, A)* and *(P, B)*. If the pattern is matched with *(P, A)*, the variable *?X* is instantiated to *A*. A variable defined in different patterns of the same rule should be instantiated with the same value.

During an inference process, variables in a rule are instantiated to a certain set of values only once. For example, if a rule with variables *?X* and *?Y* has already been fired for *?X = A* and *?Y = B*, the rule will not be fired again for the same set of variable values. A rule without any variables can be fired only once in the inference process. On the other hand, a rule with variables can be fired many times with different variable instantiations. After matching, the variables in the THEN part of a rule are replaced by their instantiations for execution.

6.3.4 Matching of Built-in Predicates

The matching of the built-in predicates is different from the matching of the ordinary predicates. As an example, consider the matching of a built-in predicate, (=, m[?x], 2). The first term '=' is defined as a built-in keyword in this system. The variable ?x has been instantiated with an instance feature name. The attribute *m* belongs to the feature that has been instantiated to variable ?x. If the value of the attribute *m* is equal to 2, the pattern is considered to be matched. Similarly the matching procedure is the same for other built-in predicates such as <, >, =<, =>, etc.

6.3.5 Execution of a Rule

A rule is fired when its condition part is matched. As a result of firing, new data are added to the database and action(s) conducted according to the rule. Before adding a pattern described in the THEN part of a rule to the database, the variables in this pattern are replaced by their instantiations obtained in the process of rule matching. Built-in predicates in the THEN part are not added to the database. Instead these predicates are used for conducting pre-defined actions such as generating new instance features changing attribute values, etc.

6.4 An Example

The example is to design a mechanism to transfer rotational motion from one shaft to another. The first step is to define the design requirements. The design requirements are described by:

(1) two shafts,
(2) the motion to transfer from one shaft to another,
(3) the magnitude of the torque to be transferred,
(4) the distance between the two shafts,
(5) the rotational speed of the driver shaft,
(6) the desired rotational speed of the driven shaft,
(7) rotation direction of the driver shaft and the desired rotation direction of the driven shaft.

The standard design requirements are defined in a class feature called *Rotational*. Two features with the type *Shaft* are defined as element features of the class feature *Rotational*. The magnitude of the torque to be transferred and the distance between the two shafts are described as attributes. Rotational speeds of the shafts and their rotation directions are also described as attributes. Facts (qualitative descriptions/relations) are used to define that there

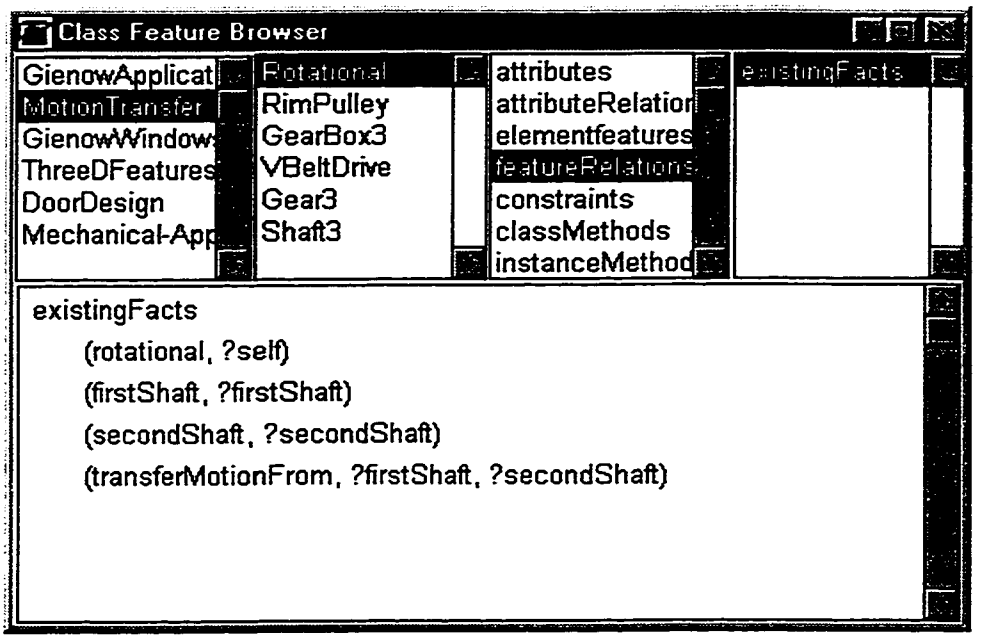


Figure 6.6: Snapshot of class feature browser showing the existing facts

are two shafts and motion is to be transmitted from the driver shaft to the driven shaft. The variables *?firstShaft* and *?secondShaft* refer to the driver and the driven shafts respectively. The *Shaft* feature has its own attributes such as rotational speed, direction of rotation, and diameter. Relations among the various attributes are also defined. A snapshot of the *Class Feature Browser* showing the *Rotational* class feature is given in Figure 6.6.

To generate the product using the class feature, an instance feature called *Rotational1* with its element features *DriverShaft1* and *DrivenShaft1* are generated. To conduct product modeling automatically, relevant rule-bases are selected for reasoning. This is carried out using the instance feature browser by selecting the menu item *Select Rule Base,* as shown in Figure 6.7. The design knowledge used for inference in this example is described in different rule-bases. One of the rule-bases used in this example is shown in Figure 6.8. In this rule-base, the rule *VbeltCase* represents the knowledge:

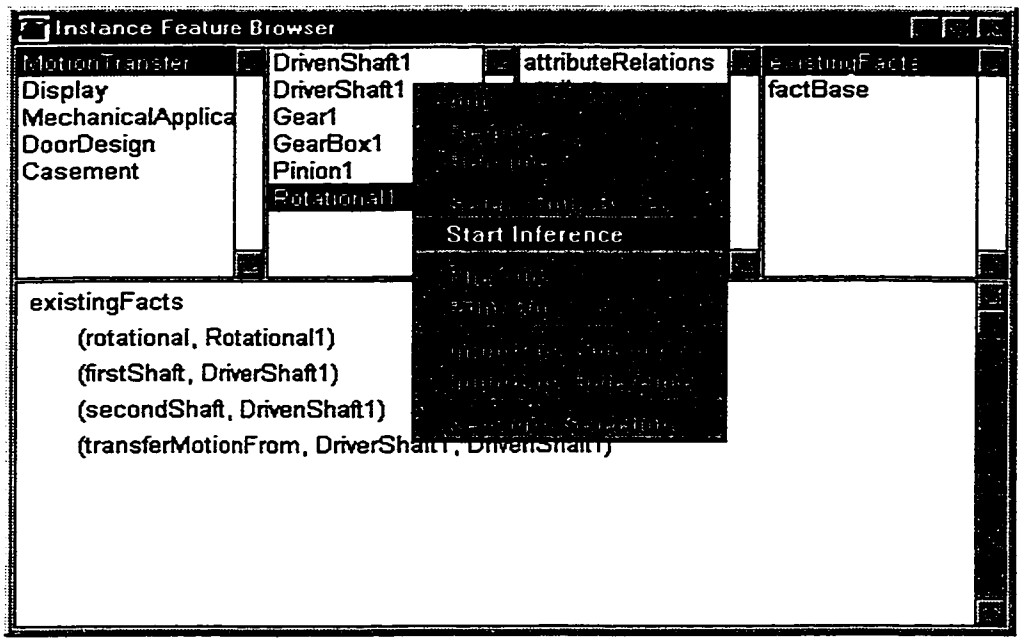


Figure 6.7: Snapshot of the instance feature browser showing the selection of rule-bases

```
IF
      (there is a shaft called ?x) and (there is a shaft
      called ?y)  and (motion has to be transferred from the
      ?x shaft to the ?y shaft) and (directions of both
      shafts are the same) and (there is a rotational
      mechanism called ?z) and (the magnitude of the torque
      to be transferred is less than 100) and (the distance
      between the shafts is more than 50)
THEN
      (generate an instance feature of V-Belt drive)
```

Similarly the rule *GearBoxCase* describes the case to generate a gear mechanism. A gear mechanism is used if the torque to be transferred is large and the distance between the two shafts is small. A V-belt drive mechanism is used if the torque is small but the distance is large. One of the two design alternatives is generated depending upon the design requirements. The conditions of motion transfer in this example are summarized in Table 6.3. The existing facts are shown in Figure 6.7. Under these conditions the *GearBoxCase* rule is fired. Firing of the rule is summarized in Table 6.4.

Table 6.3: The values of the attributes

Attribute Names	Attribute values
torque[Rotational1]	250
distance[Rotational1]	100

RotationToRotation

Rule: VBeltCase
IF(firstShaft, ?x) & (secondShaft, ?y) & (transferMotionFrom, ?x, ?y) & (=, direction[?x], direction[?y]) & (rotational, ?z) & (<=, torque[?z], 100) & (>=, distance[?z], 50) THEN (generateFeature, VBeltDrive, ?vbelt)

Rule: GearBoxCase
IF(firstShaft, ?x) & (secondShaft, ?y) & (transferMotionFrom, ?x, ?y) & (<>, direction[?x], direction[?y]) & (rotational, ?z) & (>=, torque[?z], 100) & (<=, distance[?z], 150) THEN (generateFeature, GearBox3, ?gearbox)

Rule: GearBoxInput
IF (firstShaft, ?x) & (GearBox3, ?y) THEN (assignAttribute, inputRPM[?y], rpm[?x]) & (assignAttribute, inputDirection[?y], direction[?x])

Rule: GearBoxOutput
IF (secondShaft, ?x) & (GearBox3, ?y) THEN (assignAttribute, outputRPM[?y], rpm[?x]) & (assignAttribute, outputDirection[?y], direction[?x])

Figure 6.8: Descriptions of rules used in this example

As a result of firing the rule *GearBoxCase,* an instance feature *Box1* and its two element-features *Pinion1* and *Gear1* are generated. *Pinion1* serves as the input gear connected to the driver shaft, whereas *Gear1* is the output gear. The instance feature *Box1* is of the type *GearBox.* It has a number of attributes such

as inputRPM, outputRPM, inputDirection, outputDirection, and so on. Their values are generated by the execution of rules. At the next step, the design of *Box1* is accomplished by selecting relevant rule-bases and conducting inference. These rules are shown in Figure 6.9. Computation is carried out for automatically updating the attribute values using their dependency relations.

Table 6.4: Firing of the rule *GearBoxCase*

Patterns	Matching
(firstShaft, ?x)	?x is matched with DriverShaft1.
(secondShaft, ?y)	?y is matched with DrivenShaft1.
(transferMotionFrom, ?x, ?y)	?x and ?y are matched as above.
(<>, direction[?x], direction[?y])	The values of the direction attribute in DriverShaft1 and DrivenShaft1 are compared. The condition is true since they are not equal.
(rotational, ?z)	?z is matched with Rotational1.
(>=, torque[?z], 100)	Verify that torque to be transferred is more than 100.
(<=, distance[?z], 150)	Verify that the distance between the two shafts is more than 150.

GearBox3Design

Rule: Input

IF (GearBox3, ?x) & (inputGear, ?y) THEN (assignAttribute, rpm[?y], inputRPM[?x]) & (assignAttribute, direction[?y], inputDirection[?x])

Rule: Output

IF (GearBox3, ?x) & (outputGear, ?y) THEN (assignAttribute, rpm[?y], outputRPM[?x]) & (assignAttribute, direction[?y], OutputDirection[?x])

Figure 6.9: A rule-base for design of a gear box

Chapter 7

Implementation of the Feature-based Intelligent Design System

This chapter presents the architecture and implementation of the feature-based intelligent design system. The system was implemented using VisualWorks, C++, and 3D Studio MAX.

7.1 System Architecture

The feature-based intelligent design system is composed of three sub-systems: the feature-based design system, the geometry representation system, and the intelligent system, as shown in Figure 7.1. The various interface windows of the feature-based intelligent system and their interactions are shown in Figure 7.2.

The feature-based design system and the intelligent system have been implemented using Visualworks 3.0 – a window-based environment for programming in the Smalltalk language [Goldberg83]. The geometry representation system has been implemented using Microsoft Visual C++ 5.0 [Gregory97] and 3D Studio MAX [Michael96].

The feature-based intelligent design system is managed by an interface environment called *Launcher.* All the interface windows of the system, including the *Class Feature Browser, Instance Feature Browser, Rule Base Browser, 2-D Geometry Browser,* etc., can be opened using this launcher. These interface windows are used for defining class features, generating instance features, defining rule-bases, and displaying 2D product geometry results, respectively. A snapshot of the Launcher is shown in Figure 7.3.

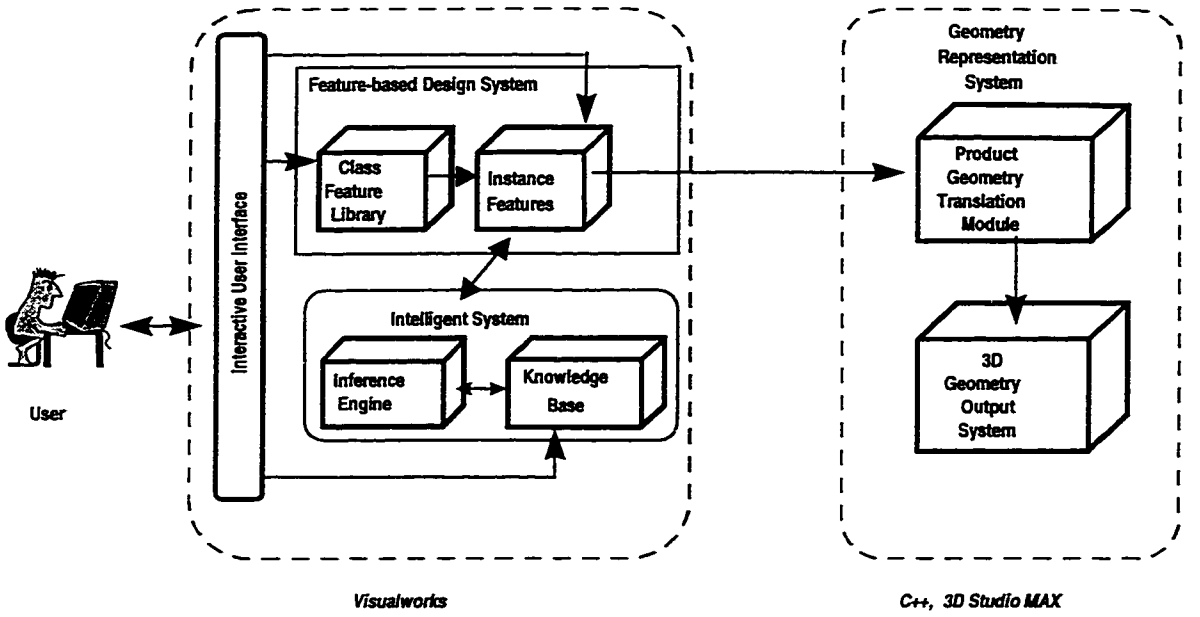


Figure 7.1: System architecture

Class features are used to define the general product descriptions. Each class feature consists of a several aspects, including element-features, attributes, feature relations, attribute relations, constraints, and so on. Class features are organized in a hierarchical data structure. Due to the large size of class library, these class features are organized in categories. Class features are defined using class feature browser, as shown in Figure 4.1.

Instance features are used to model the real product data. Instance features are generated using class features as the templates. The generated instance features can be modified by the user. Instance features are modeled using the instance feature browser, as shown in Figure 4.3. Since each product is described by a number of instance features, these instance features are also organized in categories. Each category represents one product. In addition, calculation of attribute values using the generated quantitative relations is also conducted using the instance feature browser.

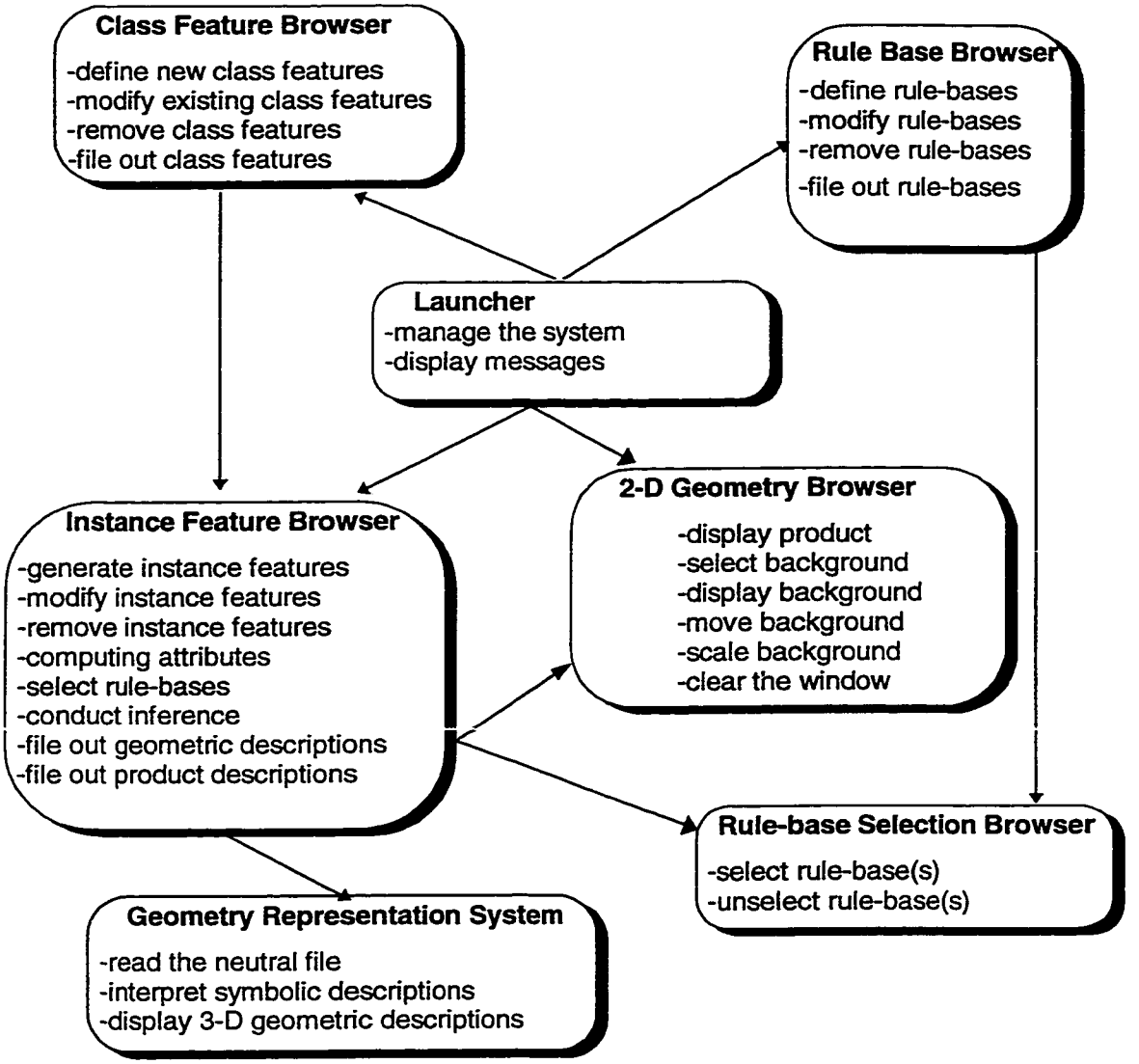


Figure 7.2: Representation of feature-based intelligent design system

An intelligent system has been developed to automate the product modeling process. Knowledge in the form of rules is defined in the Rule Base Browser. In this browser, rule-bases are defined in different categories. Each rule-base consists of a number of rules. Using the rules described in the knowledge base, product modeling is conducted automatically through rule-based inference.

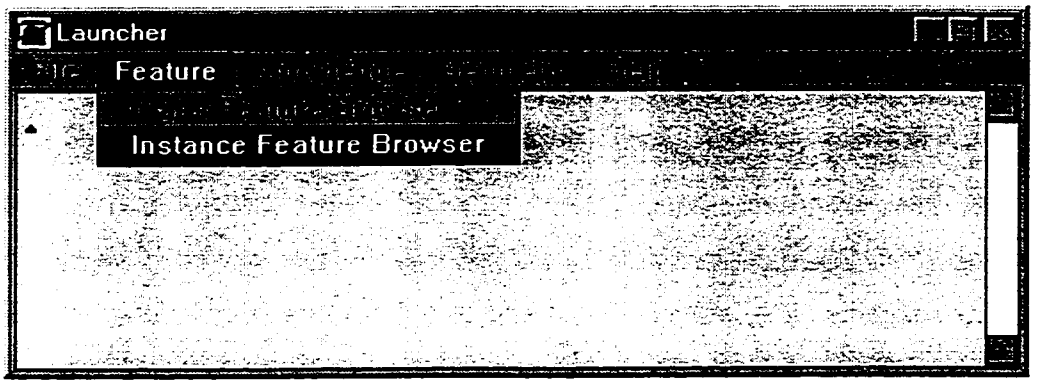


Figure 7.3: A snapshot of the feature-based intelligent design system launcher

Only the relevant knowledge base and database are involved in the inference process. First, an instance feature is selected as the database for the reasoning. A number of rule-bases are then selected for this active instance feature. The rule-base selection browser is used for selecting the relevant rule-bases. The active instance feature is selected using the instance feature browser.

The 2D product geometry is displayed using the 2D geometry browser. In this window, the background of the product can also be displayed. The background can be selected from the screen of computer monitor directly, or from a graphics file. The background can also be moved in location and scaled in size.

To represent the 3D geometry of the product, a 3D geometry representation system has been developed. The symbolic geometric descriptions in the instance features are extracted and translated into a neutral format. These symbolic geometric descriptions are further translated into 3D solid descriptions using the translation module for the 3D geometric modeling CAD system. The translation module has been implemented using the Microsoft Visual C++ 5.0 programming environment. The translation module is linked with the CAD package - 3D Studio MAX.

7.2 New Classes Used for System Implementation

VisualWorks provides a very large library of classes for system implementation. New classes can also be defined as sub-classes of the system classes. In the implementation of this system, commonly used system classes are: *Object, ApplicationModel, Dictionary, OrderedCollection, String,* etc. Explanations to these classes are given in Table 7.1. In Smalltalk all the variables and methods of the super-classes are inherited by the sub-classes automatically. The classes are organized in different categories. The newly created categories and the classes are given in Table 7.2.

Table 7.1: The system classes used for implementation

Class Names	Explanations
Object	Class at the top of the hierarchy of the system classes
ApplicationModel	Class for developing interface
Dictionary	Class for storing data in form of a *key-and-value* pair where key is described by a symbol and value can be any Smalltalk object
OrderedCollection	A collection of Smalltalk objects stored in a sequence and identified by sequence numbers
String	Class representing an array of characters

A number of global variables have been defined in the system to preserve the knowledge and data. For instance, the global variable *RulesAspectDic* stores the rule-bases in this system. Similarly the global variable *FeatureCategoryDic* is used to store all the class features. When a new class feature is defined (using the class feature browser), its feature description is stored in the global variable

FeatureCategoryDic. Some of the major global variables used in this system are listed in Table 7.3.

Table 7.2: Newly created categories and classes

Category Names	Class Names
Feature-Design	• ClassFeatureBrowser • FeatureClass • FeatureInitialize • MainBrowser
Feature-Design-Instance	• InstanceBrowser • FeatureInstance • InstanceDrawer • BackgroundEditor • BackgroundImage
IntelligentDesign	• RulesBrowser • RuleBase • Rule • Fact
InferenceSupport	• RuleBaseSelectionBrowser • SelectedRule • SelectedRuleCollection

Table 7.3: Major global variables and their descriptions

Variable Names	Stored Data
FeatureCategoryDic	Class features defined in the system
FeatureInstanceDic	Instance features generated in the system
FeatureDrawingCategory	Instance features to be displayed in the 2-D drawing browser

Table 7.3: Major global variables and their descriptions (continued)

Variable Names	Stored Data
FeatureAspectsList	List of aspects shown in the class feature browser
FeatureInstanceAspectsList	List of aspects shown in the instance feature browser
RulesAspectDic	The rule-bases defined in the system

A Smalltalk class called *FeatureInitialize* has been defined in the system for initializing all the global variables. When a new copy of Smalltalk system is used, the user should execute the instance method *initialize* of the class *FeatureInitialize.*

7.2.1 New Classes for Implementation of Class Features

All class features in this system are translated into Smalltalk classes. Since all class features are sub-classes of the system class feature called *Feature*, the common characteristics of class features are described in a Smalltalk class corresponding to the class feature *Feature*. In addition, a Smalltalk class, called *FeatureClass,* is also defined to store the class feature definitions using the instance variables. The methods defined in this class are mostly related to storing the data inside these instance variables. Sub-class features can inherit the properties of the super-class features. To achieve this goal, methods such as *findSuperClassLink, addFeatureInheritance,* etc. have been implemented.

ClassFeatureBrowser is the Smalltalk class for implementing the class feature browser. The methods in this class are used to define the interface environment and conduct the actions of menu items. When a class feature is being defined, the actions to be performed are: saving the feature descriptions, including attributes, element-features, attribute relations, feature relations,

constraints, etc., removing feature descriptions, and so on. The class feature definitions are stored in Smalltalk variables. For instance, the attributes should be stored in the variable called *attributeDic.* The data structure to store the class feature *GearBox* is shown in Figure 7.4. A snapshot to show the data structure of GearBox class feature is given in Figure 7.5. A mechanism to check the syntax of the class feature definition has also been developed.

7.2.2 New Classes for Implementing Instance Features

Instance features are described by the Smalltalk instances that are generated using the Smalltalk classes corresponding to the class features. The instance variables of these instances include: *instanceName, attributeDic, attributeRelationDic, instanceOfClass,* etc. Many methods have been implemented to access the data of these instance variables.

InstanceBrowser is the class for implementing instance feature browser and its menu actions. The methods in this class can be categorized as follows:

- *methods for implementing the browser interface:*
  The instance feature browser consists of five views. Four of them are list-views and elements on these lists are preserved in instance variables *worldList, featureList, aspectList* and *nameList.* The bottom view is a *textEditor.* Many methods have been implemented to carry out the actions of the menu items in these views.
- *methods for generating instance features and their aspects:*
  Instance features are generated using class features as their templates. The methods for generating instance features include: *featureAdd, addAttributes, addAttributeRelations, addFeatureRelations, addConstraints,* etc.
- *methods for modifying instance descriptions:*
  Instance features can be modified using the instance feature browser. New elements can be added to the instance feature definitions. The existing

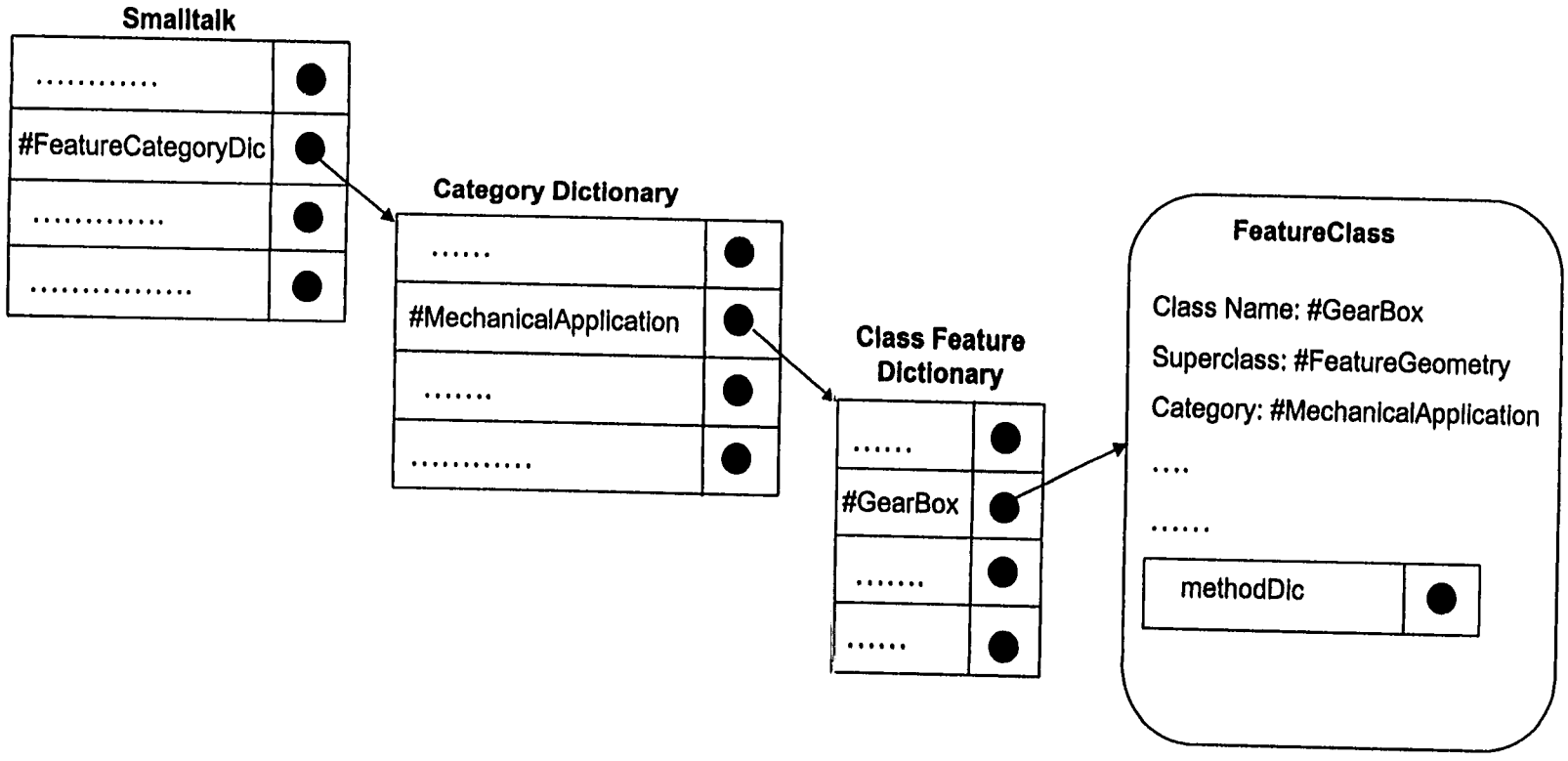


Figure 7.4: The data structure in which a class feature is stored in the system

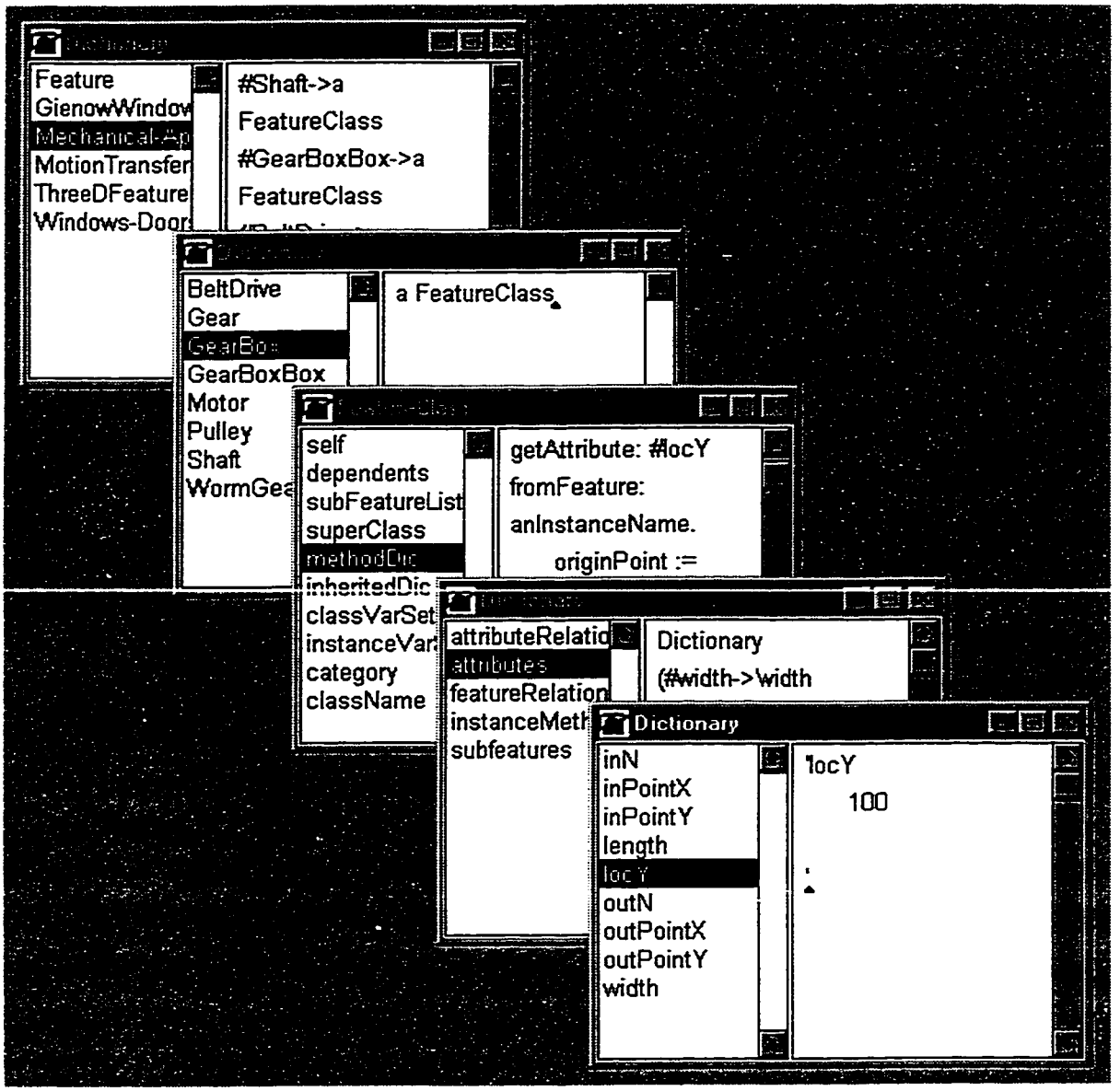


Figure 7.5: Snapshot showing the storage of class feature *GearBox* in the system

elements can also be removed. The methods to add new elements include: *acceptAttribute:, acceptAttributeRelation, acceptConstraint:*, etc.

- *methods for computing attribute values using the attribute relations:*
  The attribute relations are used for keeping the consistency of attribute values. The methods used for attribute calculation include: a*ttributeComputing*, etc.

7.2.3 New Classes for Implementing Knowledge Base and Inference

Four classes have been created for knowledge base modeling. These classes are:

- RulesBrowser
- RuleBase
- Rule
- Fact

The class *RulesBrowser* is used for implementing the interface for defining rule-bases. The methods defined in this class are used to add and remove rule-bases.

The other three classes are used for preserving the descriptions of rule-bases, rules, and facts (predicates), respectively. Each of these classes has a number of instance variables to describe the different types of descriptions. The instance variables of the class RuleBase include:

- text
- ruleNameCollection
- numberOfRules

The instance variables of the class Rule include:

- ifPart
- thenPart
- ruleName

The instance variables of the class *Fact* include:

- termCollection
- numberOfTerms

All the above classes are in a category called *IntelligentDesign* in the VisualWorks system browser. The accepted rules are stored in a global variable called *RulesAspectDic*.

Two classes have been created for selecting rule-bases for inference. These two classes are:

- RuleBaseSelectionBrowser
- SelectedRule

in a category called *InferenceSupport*. The class *RuleBaseSelectionBrowser* is a class for implementing the Rule-base Selection Browser. The *SelectedRule* class is used to preserve the descriptions of the selected rules, including the name of the rule, the name of the rule-base, the name of the category, the content of the rule, the variable instantiations that have been executed, the variable instantiations that are currently being matched, the variable instantiations to be executed, etc.

The rule-based inference was implemented by the methods including:

- *methods for matching of rules:*
  Relevant rule-bases are selected and stored in the active instance feature in the form of an instance variable, which is an instance of *OrderedCollection*. Matching of a rule is carried out by matching all the patterns at the IF part. The methods for matching rules include: *matchRule:at:, matchPattern:with:, matchTerm:*, etc.
- *methods for execution of rules:*
  During firing a rule, since the variables of the rule can be matched with a number of instantiations, verification is required to ensure that one

instantiation can be fired only once. All the instantiations for a rule are stored in instances of *OrderedCollection.* These collections are: *matched, executed* and *execute.* The collection *matched* contains all the possible instantiations, *executed* contains instantiations that have been fired already. The collection *execute* contains the matched instantiations that have never been executed. The collection *execute* is achieved by subtracting the *executed* from *matched.*

7.2.4 New Classes for 3D Product Geometry Representation

The 3D geometric descriptions in the instance features are extracted to a neutral file and then translated to the format that can be understood by 3D Studio MAX. Both Smalltalk and C++ are used for implementing 3D geometric representation system.

The geometric description extraction module was implemented using Smalltalk. The methods for this extraction include *worldGeometryFileOut*, etc.

The translation module has been developed using the Visual C++ 5.0 programming environment. The 3D Studio MAX package provides plug-in application development functions. Plug-in applications are external programs that can be integrated into existing packages such as 3D Studio MAX. The translation module is executed from the 3D Studio MAX system, and used to read the neutral 3D geometry description file, translate these descriptions into 3D Studio MAX data format, and display the 3D product geometry using 3D Studio MAX functions.

Each line in the neutral geometric description file describes a step to build up the 3D product geometry. The translation program reads, interprets and executes the file line by line. In 3D Studio MAX, all types of geometrical shapes are sub-classes of class the *Object.* These classes are identified by their class IDs and used for representing the geometry with different shapes. Every class defined in 3D Studio MAX has an unique ID. Before implementing a new class, a new class ID has to be generated using the *Class ID Generator* as shown in

Figure 7.6. The new classes created in this research include *Feature*, *FeatureClassDesc*, etc. The existing 3D Studio MAX classes used in the implementation include *Object*, *Inode*, *Modifier*, *GeomObject*, *Mtl*, *Interface*, *IUtil*, *IParamArray*, *IParamBlock*, etc.

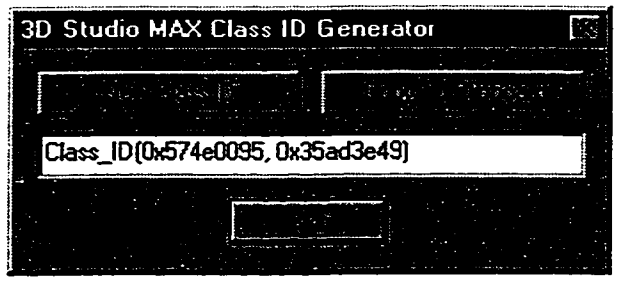


Figure 7.6: Every class in 3D Studio MAX has a unique ID which is generated using Class ID Generator

Global variables have been defined to store the objects being created, their names, transformation matrices, and so on. The global variables defined in this program are shown in Figure 7.7. The major global variables and their explanations are given in Table 7.4.

Table 7.4: Global variables and their descriptions

Global Variables	Descriptions
objPtr	An array of pointers to the class *Object* - one for every object in the view.
nodePtr	An array of pointers to the class *INode* - one for every object in the view.
mat	An array of objects of class *Matrix3*, used during transformations.
objName	A 2-dimensional character array for storing the names of objects generated.
objCount	A counter for registering the number of objects generated.

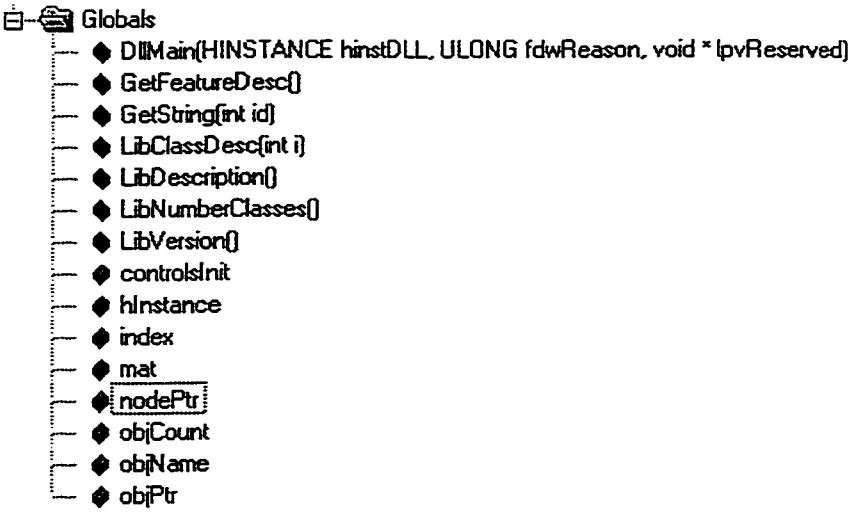


Figure 7.7: Global variables used in the translation module

In 3D Studio MAX, all the plug-in applications are listed in the *Utilities* menu. On selecting the plug-in called *FeatureGeometry,* the standard IBM PC's *File Open* dialog-box pops up. Using this dialog box, the 3D geometric description file can be selected for displaying.

Chapter 8

Development of an Industrial Application

This chapter introduces an industrial application developed using the feature-based intelligent design system for a local manufacturing company – Gienow Building Products Ltd.

8.1 Background of Gienow Building Products Ltd.

Gienow Building Products Ltd. is a manufacturer of quality building products including windows and doors. In 1947, a Calgary home builder, Bernard Gienow, started a window manufacturing company, Gienow Sash and Door. This company has since undergone many phases of growth and mergers with other companies such as Gienow Constructions Ltd., Humphrey Window Franchise and Almecto. Today its operations are spread in Alberta and British Columbia of Canada with sales and distribution centers in five cities. In its international operations, its products reach over a dozen countries.

The product range at Gienow is very large with various types of doors and windows. These are high quality windows and windowed doors with frames made of wood, aluminum, and vinyl for the residential building market. The vinyl-framed windows take an 80% share of the total manufactured windows. The windowed doors are classified into standard doors and patio doors.

In Gienow, production of windows and doors is based upon the orders from the customers. Due to the adoption of computer systems for organizing production activities, product development lead-time has been reduced considerably. In 1985, the company adopted the concept of *Just-in-Time (JIT)* manufacturing [Browne90] for further improving its manufacturing efficiency.

A computer software system for supporting the JIT philosophy has also been developed and used in the last 10 years. In this system, many modules have been implemented for partially assisting various activities in the product realization process, including product ordering, design modeling, resource (materials, machines, and workers) allocating, production process planning, quality control, production cost estimation, and so on.

The company has also been involved in a number of R&D projects. A project collaborated with Alberta Research Council (ARC) was aimed at increasing productivity, throughput and quality of the wood window facility. This project was completed in 1992 and resulted in 22.5% increase in efficiency and 67% gain in throughput.

8.2 Objective of Developing the Feature-based Intelligent Building Product Design System

The objective of developing the feature-based intelligent building product design system is to automate the process of building product design, thereby shortening product development time and efforts at Gienow. This design system provides an environment to efficiently model building products, including windows and doors. In this system, the building product library is modeled by class features. Actual products are modeled as instance features, which are generated using class features as the templates. Since attributes of instance features are associated by a relation network, any change of attribute values can be propagated to relevant attributes automatically using this attribute relation network. The intelligent system is used for further improving the design efficiency. The product geometry is represented by the 3D geometry representation system. The design system, the geometry representation system, and the intelligent system work in an integrated environment.

8.3 Implementation of the Building Product Design System

As described in the previous section, building products, including windows and doors, are created using the feature-based design approach. Features are described at two levels, class level and instance level, representing standard library and actual product data respectively. The class features are used as templates for generating instance features. Complete product descriptions, including attributes, attribute-relations, element-features, feature-relations, constraints, and geometry, are preserved in the feature definitions. The intelligent system is used for automated modeling of building products. The functions for enhanced 2D and 3D geometric representation of the building products have also been implemented in this system.

8.3.1 Feature-based Modeling of Building Products

A class feature library has been created to represent the building products for Gienow. Features are organized in a hierarchical data structure. A class feature *GienowWindow* is defined in the category *GienowApplications.* This class is the super-class feature of all the class features in the building product design system. *GienowWindow* is a sub-class of *FeatureGeometry.*

Casement One Wide Windows is a family of windows being produced at Gienow, as shown in Figure 8.1. A class feature, called *CasementOneWide,* representing this type of windows is defined in the system. *CasementOneWide* is a sub-class of *GienowWindow.* Different types of casement one wide windows are modeled as sub-classes of *CasementOneWide.* The hierarchical data structure of these class features is illustrated in Figure 8.2.

In the class feature *CasementOneWide*, a number of attributes have been defined. These attributes include: height, width, price, metalCladding, openingDirection, frameMaterial, etc. Since all the casement one wide windows

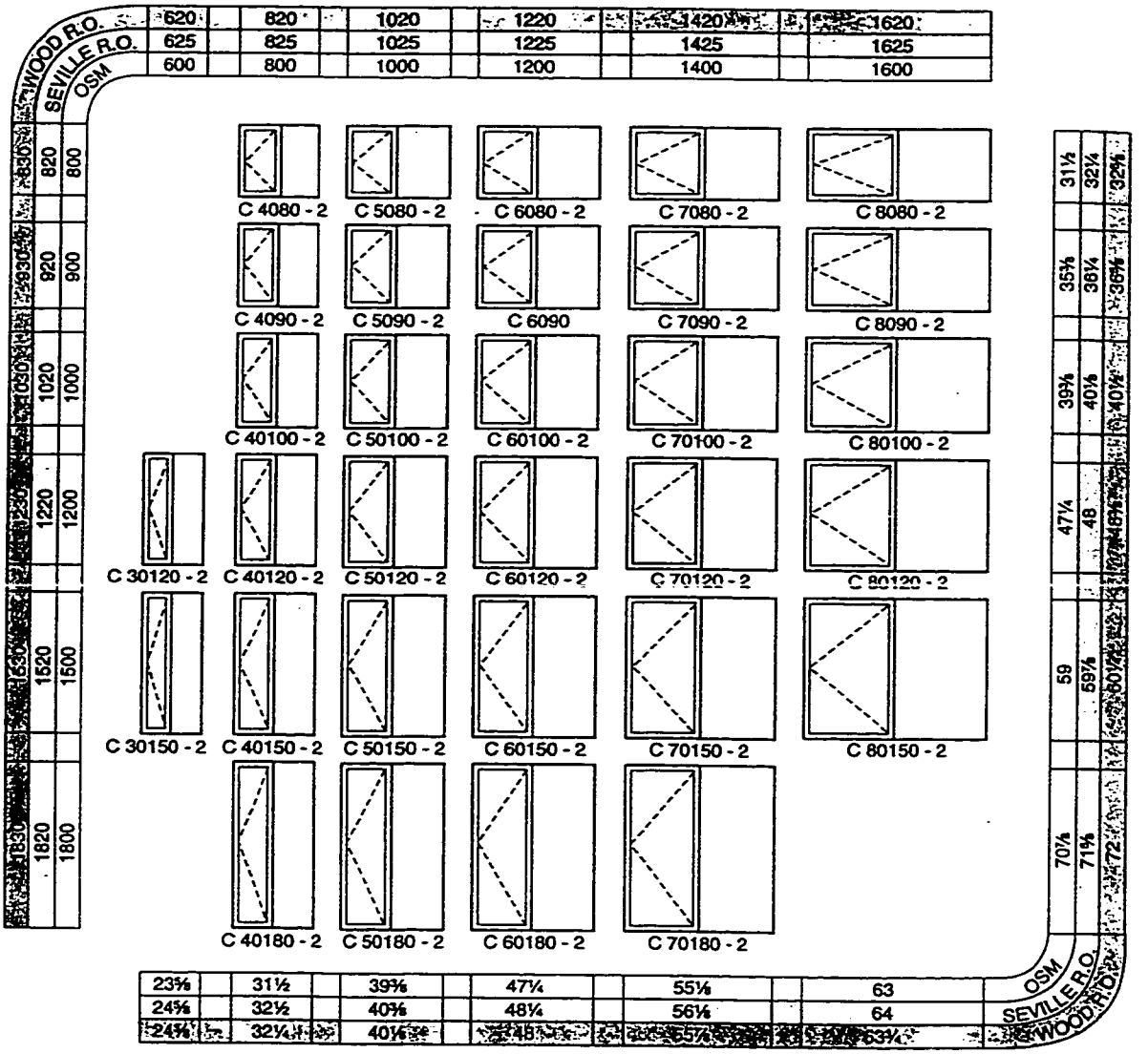


Figure 8.1: Family of casement one wide windows

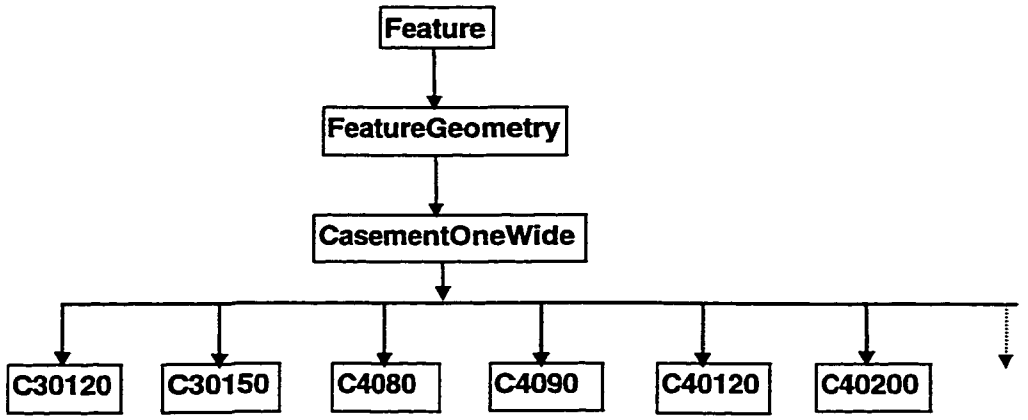


Figure 8.2: Hierarchy of class features in this application

have similar shape, the 2D and 3D geometric descriptions of these windows are defined in the class feature *CasementOneWide*. Methods to estimate the price and to calculate geometric attributes of the windows are also defined in this class feature. Major attributes of *CasementOneWide* are listed in Table 8.1.

Table 8.1: Attributes defined in the class feature *CasementOneWide*

Attribute Names	Default Values	Values After Computation
width	12 ¼ inch	Calculated using an attribute relation and a method named getWidth
height	48 ¼ inch	Calculated using an attribute relation and a method named getHeight
type	C	Set by the user, could be C, F or P
openingDirection	Left	Set by the user, could be Left or Right

Table 8.1: Attributes defined in the class feature *CasementOneWide* (cont.)

Attribute Names	Default Values	Values After Computation
price	$ 451.00	Calculated using an attribute relation and a method named getPrice
frameMaterial	primedWood	Set by user, could be primedWood or sevilleMetal-clad
metalGrill	yes	Set by user, could be yes or no
solR1	yes	Set by user, could be yes or no
BarThick	2 ½ inch	Calculated using an attribute relation
glassThickness (for 3D geometry representation)	¼ inch	Calculated using an attribute relation
color (for 2D geometry representation)	White	Set by user, could be any color

To simplify feature representation, the unnecessary details should be hidden from the users. For the casement windows, a number of constant parameters don't need to be accessed by the users. These parameters are defined as instance variables. A method to initialize the instance variables has also been defined in *CasementOneWide.* This method is executed automatically when an instance feature is generated.

Many types of the casement windows are available for production at Gienow, such as *C30120, C30150, C4080, C4090, C40100, C40120,* and so on. These types of windows are defined as sub-class features of the

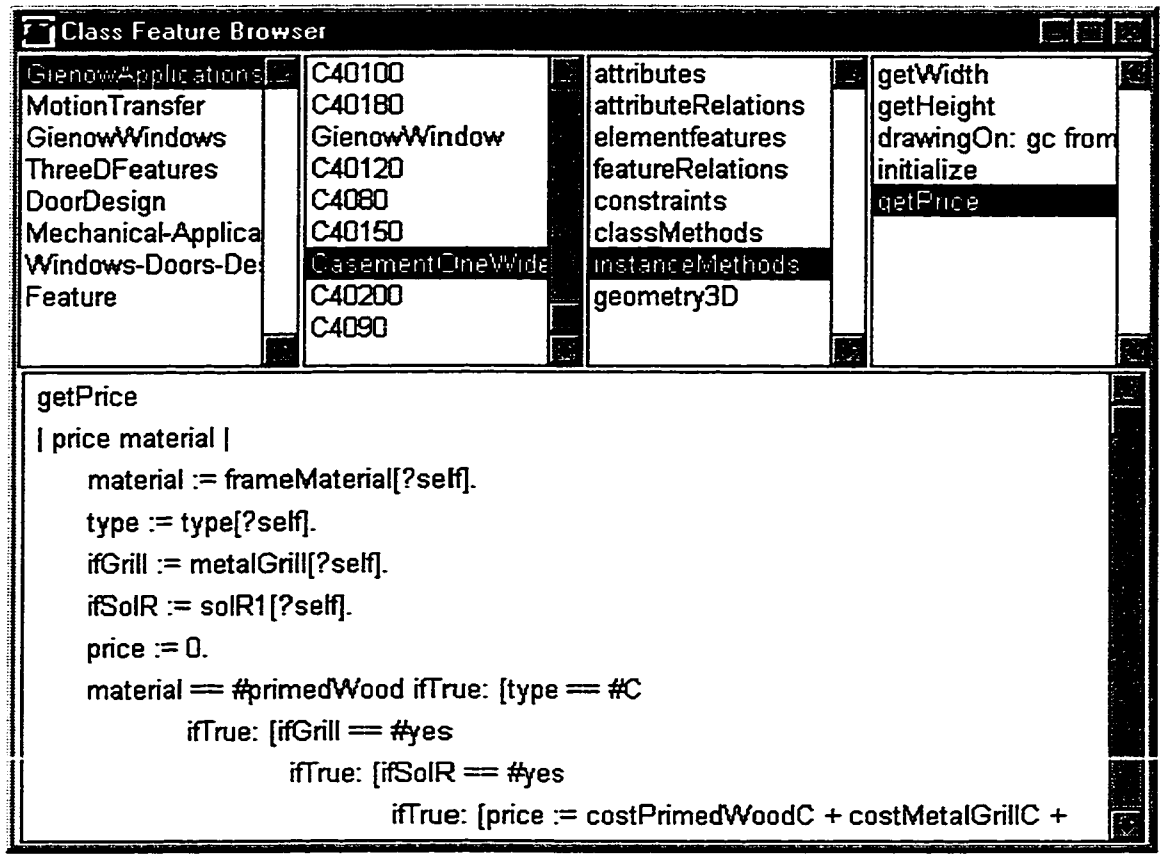


Figure 8.3: Class features defined in the Building Product Design System

CasementOneWide. The instance variables defined in *CasementOneWide* are inherited by its sub-class features automatically. These instance variables are initialized with different values. A snapshot of the class feature browser with the various window class features is shown in Figure 8.3.

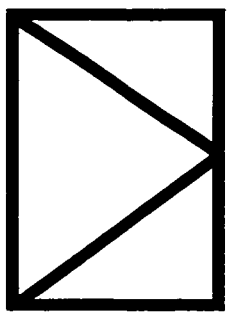
The price of a window depends upon the type of the window, material used to make the window, dimensions of the window, etc. Additional cost is incurred for metal cladding or glass fitting. It is apparent that price of a window can not be calculated using an attribute relation, since some non-quantitative attributes are involved in calculation. To calculate the price of a window, an instance method called *getPrice* was implemented.

8.3.2 Geometric Representation of Building Products

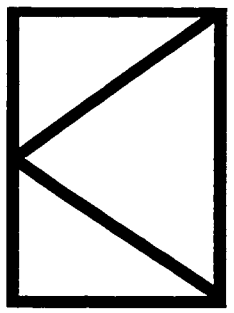
In the instance features, the 2D and 3D geometric descriptions are preserved in different aspects. The 2D geometric descriptions of a feature are described by an instance method, while the 3D geometric descriptions are represented in an aspect called *geometry3D*.

The 2D product geometry is displayed using the 2-D geometry browser. Figure 8.4 shows two 2D windows with left-side opening direction and right-side opening direction. Windows and doors can be displayed against background images. This function is provided in the 2-D geometry browser. This function is useful for both designers and customers. For instance, matching of a new window with an existing house can be checked by displaying the window with the image of the house as background, as shown in Figure 8.5.

An image of the background can be selected from a rectangular area of the computer monitor's screen and stored in the system by executing the menu item *Background* → *Select* of the 2-D geometry browser. This background image can be displayed in the 2-D geometry browser by executing the menu item



Left opening window



Right opening window

Figure 8.4: Representation of 2D windows

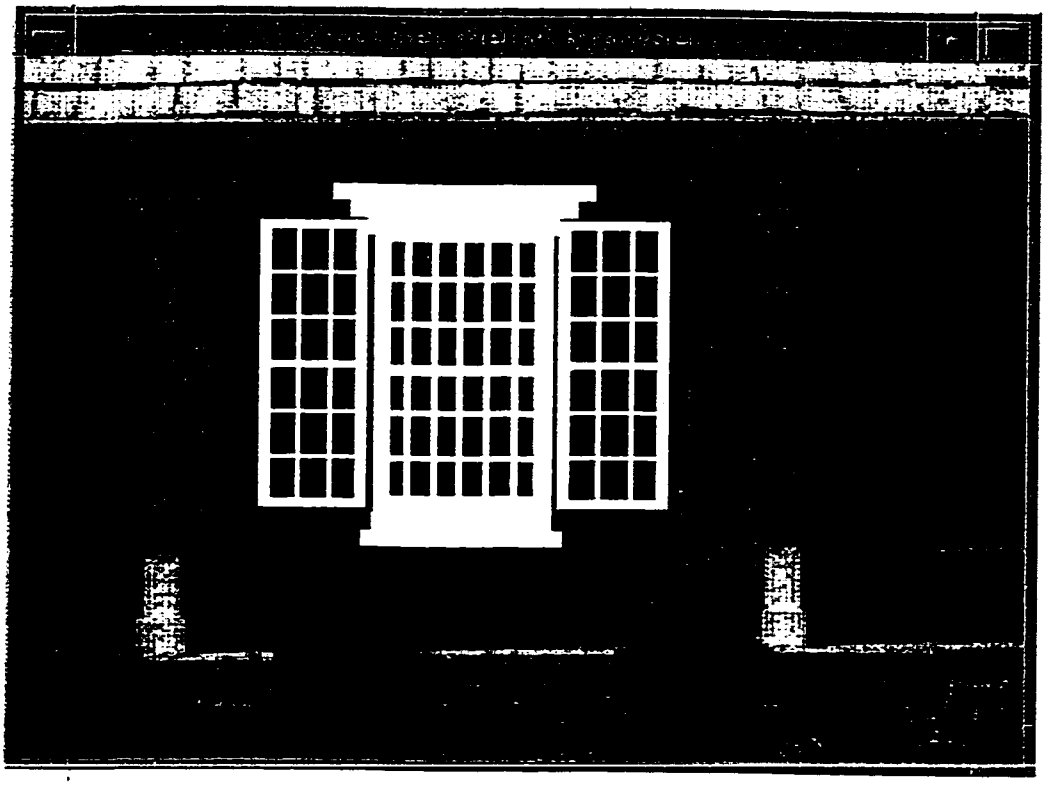


Figure 8.5: A window shown against a background in 2D geometry browser

Background → *Display*. The instance feature is displayed on top of the background by executing the menu item *Object* → *Draw*.

In order to place the building products in desired positions, the location attributes of the instance features should be changed using the instance feature browser. However since it is tedious to change the locations of all the instance features, functions to move and scale the background image have been developed. The background image can be moved and scaled using *Background* → *Move* and *Background* → *Scale* menu items respectively. On selecting the scale operation, a dialog box is poped up for entering the x-scale and y-scale values respectively. If a scale is greater than 1, the image is then enlarged, otherwise reduced. Similarly for moving operation, both the new x and y

coordinates should be provided. The menu functions of the 2D geometry browser are summarized in Table 8.2.

Table 8.2: Functions for selecting, displaying and placing a background

Menu Items	Functions
Background → Select	Select a desired background by specifying a rectangular area on the computer monitor screen
Background → Display	Display the background in the 2-D geometry browser
Background → Scale → X-Scale	Scale the background in x direction
Background → Scale → Y-Scale	Scale the background in y direction
Background → Move → X-Coordinate	Move the background in x direction
Background → Move → Y-Coordinate	Move the background in y direction

The 3D geometric descriptions of an instance feature are preserved in a feature aspect called *geometry3D*. These geometric descriptions are defined using built-in predicates. A 3D product geometry is created by generating geometric primitives and applying operations to these primitives. The material properties can also be assigned to the geometric entities. For instance, the transparency property of the glass of a window can be defined.

The 3D geometric descriptions in instance features can be extracted into a file. The symbolic descriptions are then translated into 3D Studio MAX format and displayed using 3D Studio MAX. The 3D window/door model can be

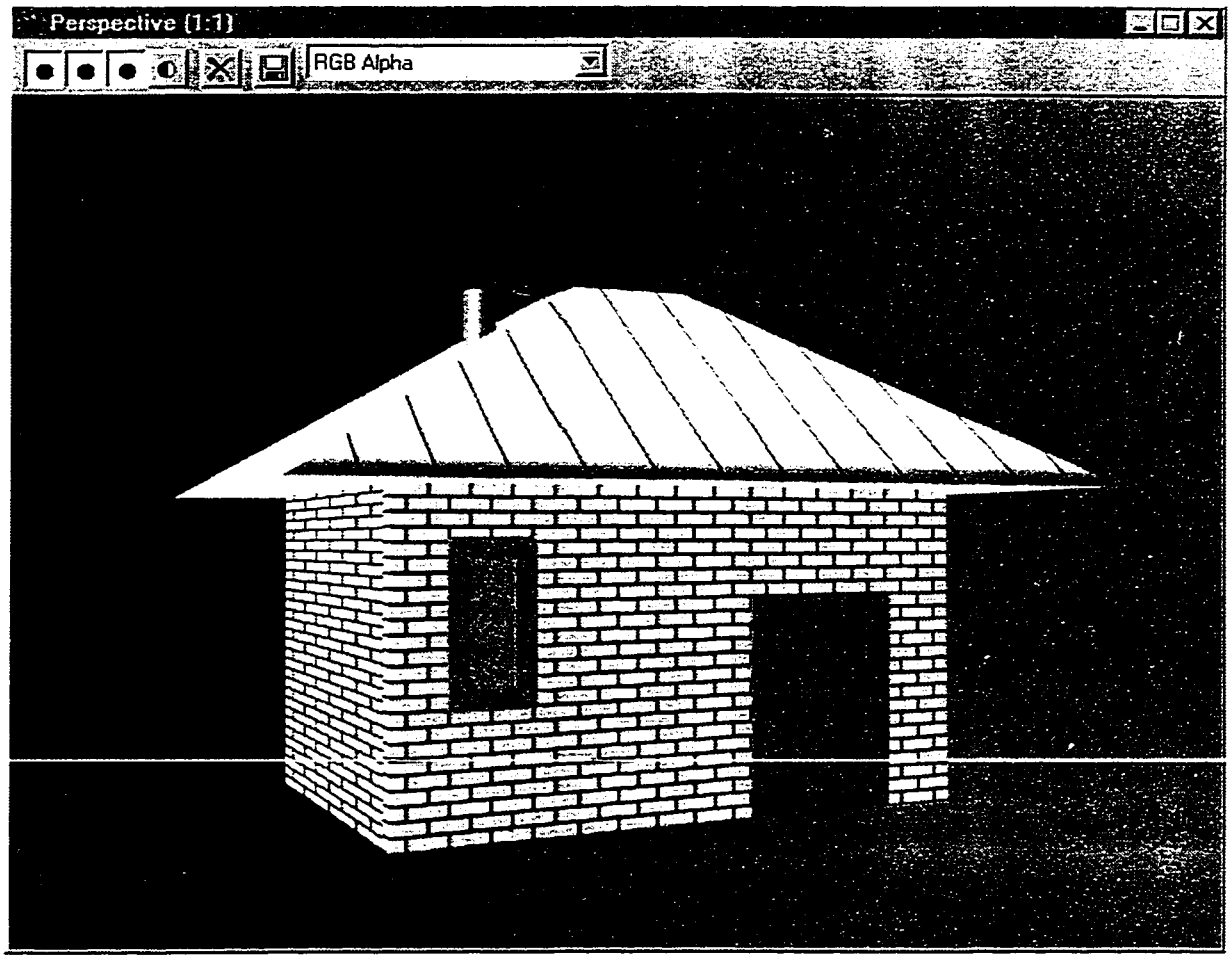


Figure 8.6: 3D Geometric model of a window placed against a model of a house

displayed against a model of a house to check the matching of this window/door with the house. A snapshot of C4080 window placed against a house in 3D Studio MAX is shown in Figure 8.6.

8.3.3 Data Relation Maintenance for Building Products

The effectiveness of data relation network for keeping the consistency of the product data was verified by developing a door design system. In this system, a standard door is defined as a class feature *Door*. A door has four

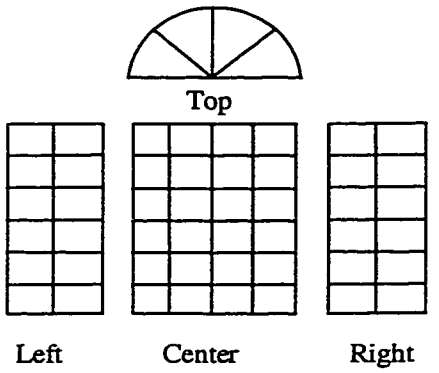


Figure 8.7: A door and its four parts

parts, defined as four element features of this class feature: *top, center, left* and *right,* as shown in Figure 8.7.

A number of quantitative relations were defined among the attributes of the element features of the door. The purposes of defining attribute relations are (1) to calculate the positions of the element-features, and (2) to calculate the dimensional parameters of the element features. For instance, relations are defined among the location coordinates of the element features *left* and *center.* If the location of the left component is modified, the relation is used to calculate the new location of the center component. Some of the important attribute relations defined in this example are listed in Table 8.3.

Table 8.3: Major relations among the attributes

Attribute Relations	**Explanations**
height[?right] := height[?left]	Set the height of the right component equal to that of the left component
locY[?center] := locY[?top] + (width[?top]/2) + dt[?self]	Calculate the y coordinate of the center component. In this relation, dt is the distance between the center component and the top component

Table 8.3: Major relations among the attributes (continued)

Attribute Relations	Explanations
height[?center] := height[?left]	Set the height of the center component equal to that of the left component
locX[?center] := locX[?left] + width[?left] + dl[?self]	Calculate the x coordinate of the center component. In this relation, dl is the distance between the center component and the top component
width[?left] := width[?right]	Width of the left component is set equal to width of the right component
locX[?top] := locX[?center]	The x-coordinate of the top component is equal to the x-coordinate of the center component
locX[?right] := locX[?left] + width[?left] + dl[?self] + width[?center] +dr[?self]	The x-coordinate of the right component is the sum of the x-coordinate of the left component, widths of the left and center components, and the distance between the right and center components
locY[?right] := locY[?center]	The right and the center components have the same y-coordinates
locY[?left] := locY[?center]	The left and the center components have the same y-coordinates

A number of class features are defined for each element of the door. Different design candidates are modeled using different element feature types. For instance, the top component of the door can have semicircular shape represented by the class feature *DoorTop* or triangular shape represented by class feature *DoorTopC*.

8.3.4 Intelligent Design of Building Products

In manual design, a human designer uses his/her knowledge to select the class features to generate instance features and model the relations. In this research, a rule-based inference system has been developed to conduct the automated design. A snapshot of the *Rule Base Browser* with various rule-bases defined in the system for door design is shown in Figure 8.8. A rule-base used for the design of the center component of the door is shown in Figure 8.9. This rule-base is explained in Figure 8.10.

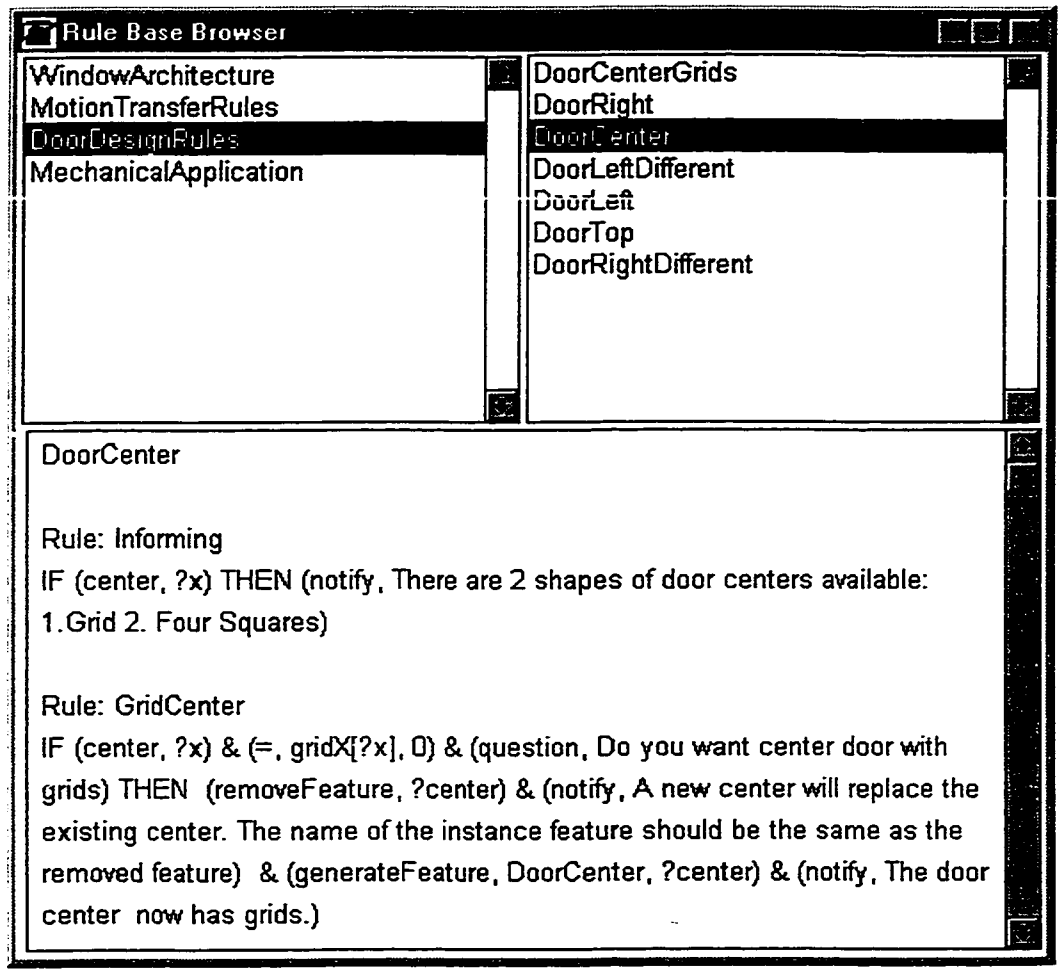


Figure 8.8: Rule-bases defined for automatic design of door components

DoorCenter

Rule: Informing
IF (center, ?x) THEN (notify, There are 2 shapes of door centers available: 1.Grid 2. Four Squares)

Rule: GridCenter
IF (center, ?x) & (=, gridX[?x], 0) & (question, Do you want center door with grids) THEN (removeFeature, ?center) & (notify, A new center will replace the existing center. The name of the instance feature should be the same as the removed feature) & (generateFeature, DoorCenter, ?center) & (notify, The door center now has grids.)

Rule: FourSquareCenter
IF (center, ?x) & (>=, gridX[?x], 0) & (question, Do you want a center door with four squares) THEN (removeFeature, ?center) & (notify, A new center will replace the existing center. The name of the instance feature should be the same as the removed feature) & (generateFeature, DoorCenterB, ?center) & (notify, The door center now has four squares.)

Rule: GridsInXDirection1
IF (center, ?x) & (>=, width[?x], 101) & (>=, gridX[?x], 1) THEN (assignAttribute, gridX[?x], 6)

Rule: GridsInXDirection2
IF (center, ?x) & (<=, width[?x], 100) & (>=, gridX[?x], 1) THEN (assignAttribute, gridX[?x], 4)

Rule: GridsInYDirection1
IF (center, ?x) & (>=, height[?x], 101) & (>=, gridX[?x], 1) THEN (assignAttribute, gridY[?x], 7)

Rule: GridsInYDirection2
IF (center, ?x) & (<=, height[?x], 100) & (>=, gridX[?x], 1) THEN (assignAttribute, gridY[?x], 5)

Figure 8.9: A rule-base for design of the center component

The rules are organized in rule-bases representing the knowledge for designing the door components. For instance, the rule-base *DoorCenter* contains the rules that are used for the design of the center component of a door. In this rule-base, rules are used for informing the user of the types of

DoorCenter

Rule: Informing
IF *there is a element-feature called center*, THEN *notify that there are 2 shapes of door centers available: 1.Grid 2. Four Squares.*

Rule: GridCenter
IF *there is a center and it is not of the grid type and if the designer wants a grid type center door*, THEN *remove the current center, notify the user that he should provide the same instance name, generate a grid type center and inform the designer about the change effected.*

Rule: FourSquareCenter
IF *there is a center and it is not of the four square type and if the designer wants center door of the four square type*, THEN *remove the current center, notify the user that he should provide the same instance name, generate a four square type center and inform the designer about the change effected.*

Rule: GridsInXDirection
IF *there is a center and it is of the grid type and its width is greater than 100*, THEN *set the number of grids in x direction on the center equal to 6.*

Rule: GridsInXDirection
IF *there is a center and it is of the grid type and its width is less than 100*, THEN *set the number of grids in x direction on the center equal to 4.*

Rule: GridsInYDirection
IF *there is a center and it is of the grid type and its height is greater than 100*, THEN *set the number of grids in y direction on the center equal to 7.*

Rule: GridsInYDirection
IF *there is a center and it is of the grid type and its height is less than 100*, THEN *set the number of grids in y direction on the center equal to 6.*

Figure 8.10: DoorCenter rule-base in plain English language

center components available, generating instance features, and calculating attribute values. Similarly in the rule-base *DoorLeft*, rules are used to decide the type of the left component based upon the type of the center component. When the left and the center components are not matched, the existing left component is replaced by the correct component through reasoning.

Using this system, a door is first generated with its four default components. If the user wishes to modify the design of the door, he/she then selects the relevant rule-bases and carries out inference. Figure 8.11 shows the result of the door when it is first generated using the class feature as the template.

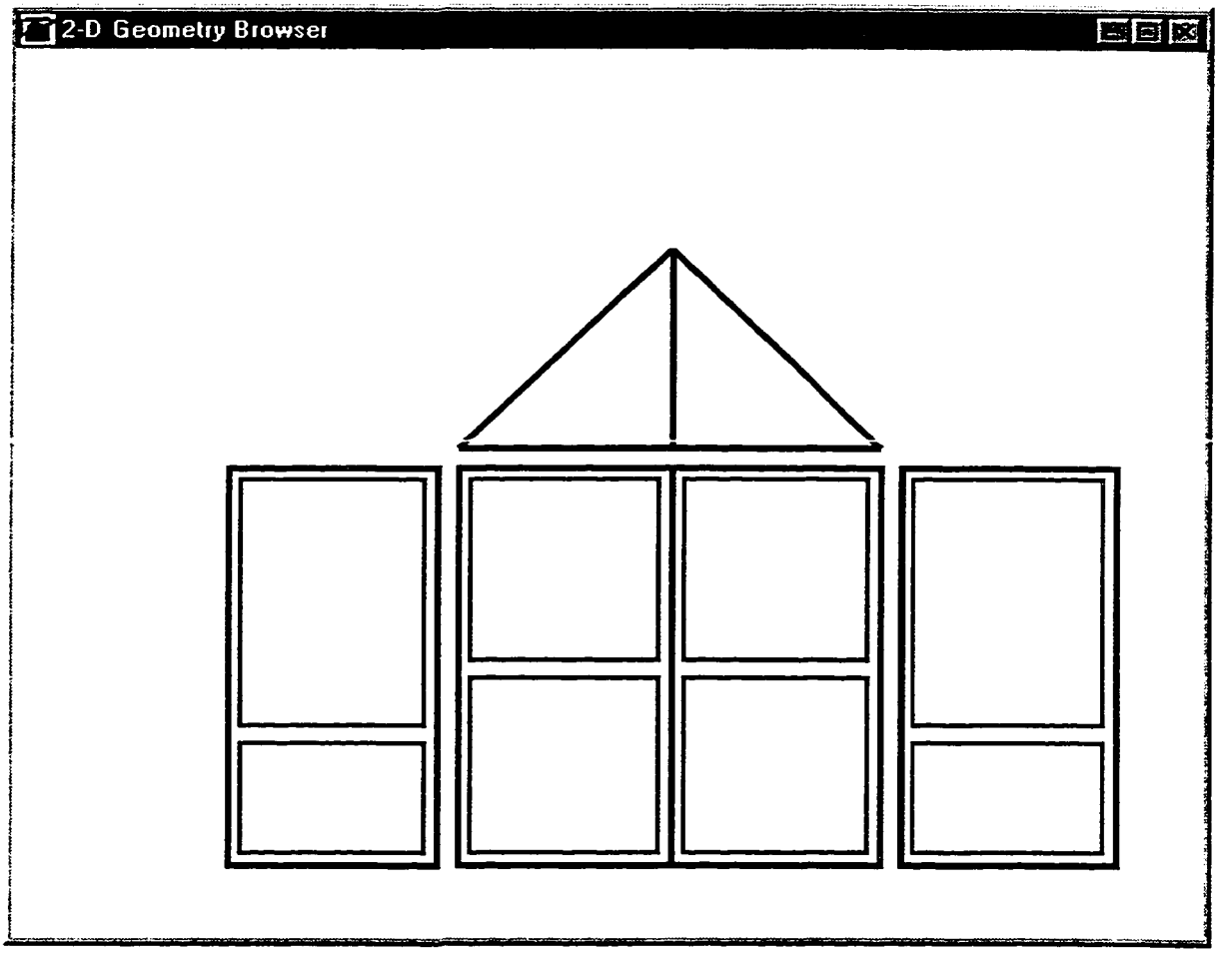


Figure 8.11: Appearance of the door before replacing the top

Now suppose the designer wishes to modify the top component of the door, and selects the rule-base DoorTop as the design knowledge. During the inference, the designer is (1) informed of the types of top components available, and (2) questioned if he/she wishes to replace the existing top component with a rectangular one or with a semicircular one. Suppose the semicircular top

component is chosen, the system then inquires about the details of the top component design, such as the number of spikes of the semicircular top component.

A snapshot of the 2-D geometry browser showing the door before the replacement of top component is given in Figure 8.11. Snapshot of the 2-D geometry browser showing the door after the replacement of the top component is given in Figure 8.12. The center component shown in Figure 8.12 has four squares. If the user wishes to modify the design of the center component, the rule-bases *DoorCenter* and *DoorCenterGrids* are then selected. Since the

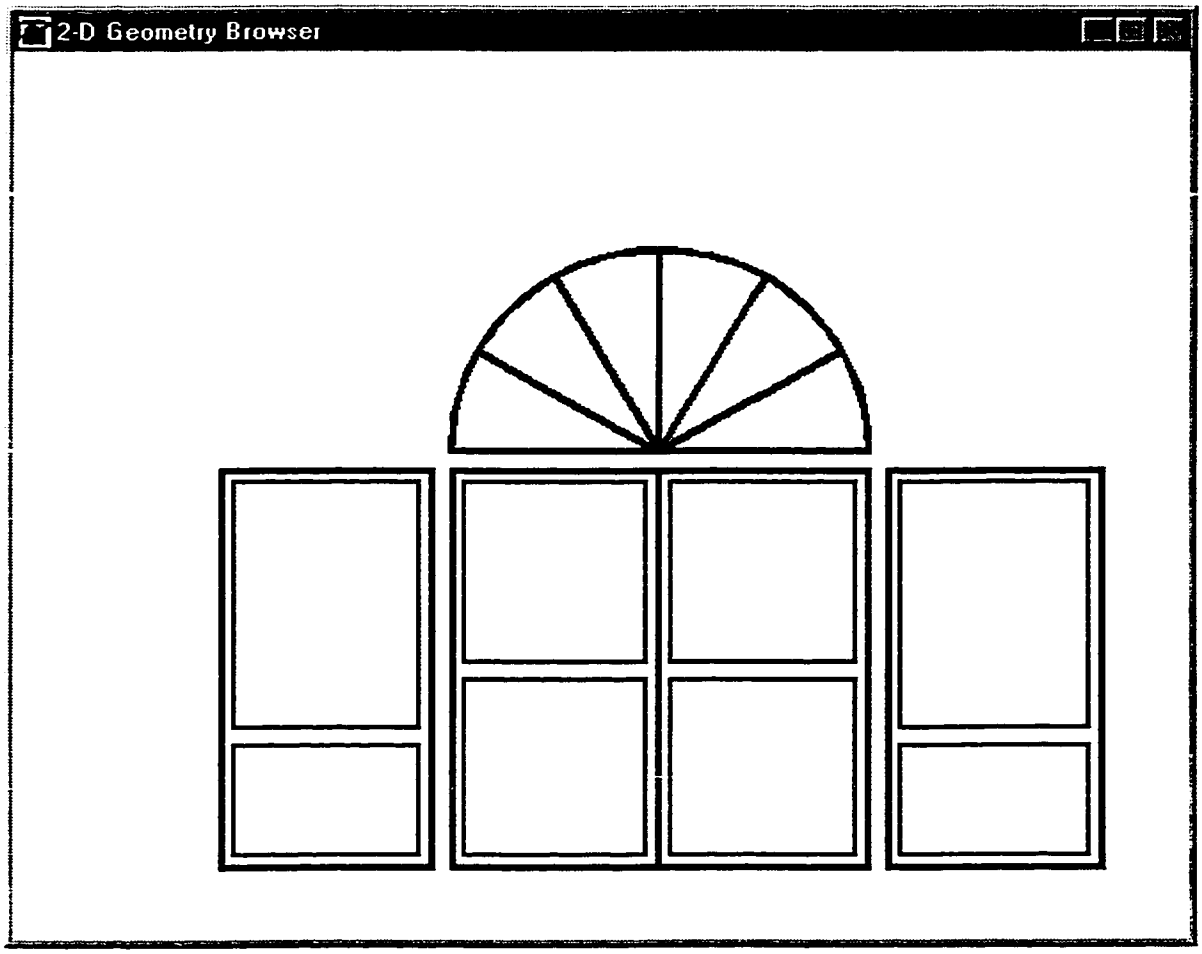


Figure 8.12: Appearance of the door after replacing the top

design of the top component has already been completed, the rule-base *DoorTop* selected previously can be *un-selected* at this stage. By reasoning, the type of the center component can be changed according to the user's preference. In this example, the center component is replaced by the one with rectangular grids. The number of grids in the horizontal and vertical directions are determined according to the rules *GridsInXDirection1*, *GridsInYDirection1*, *GridsInXDirection2*, *GridsInYDirection2*, respectively.

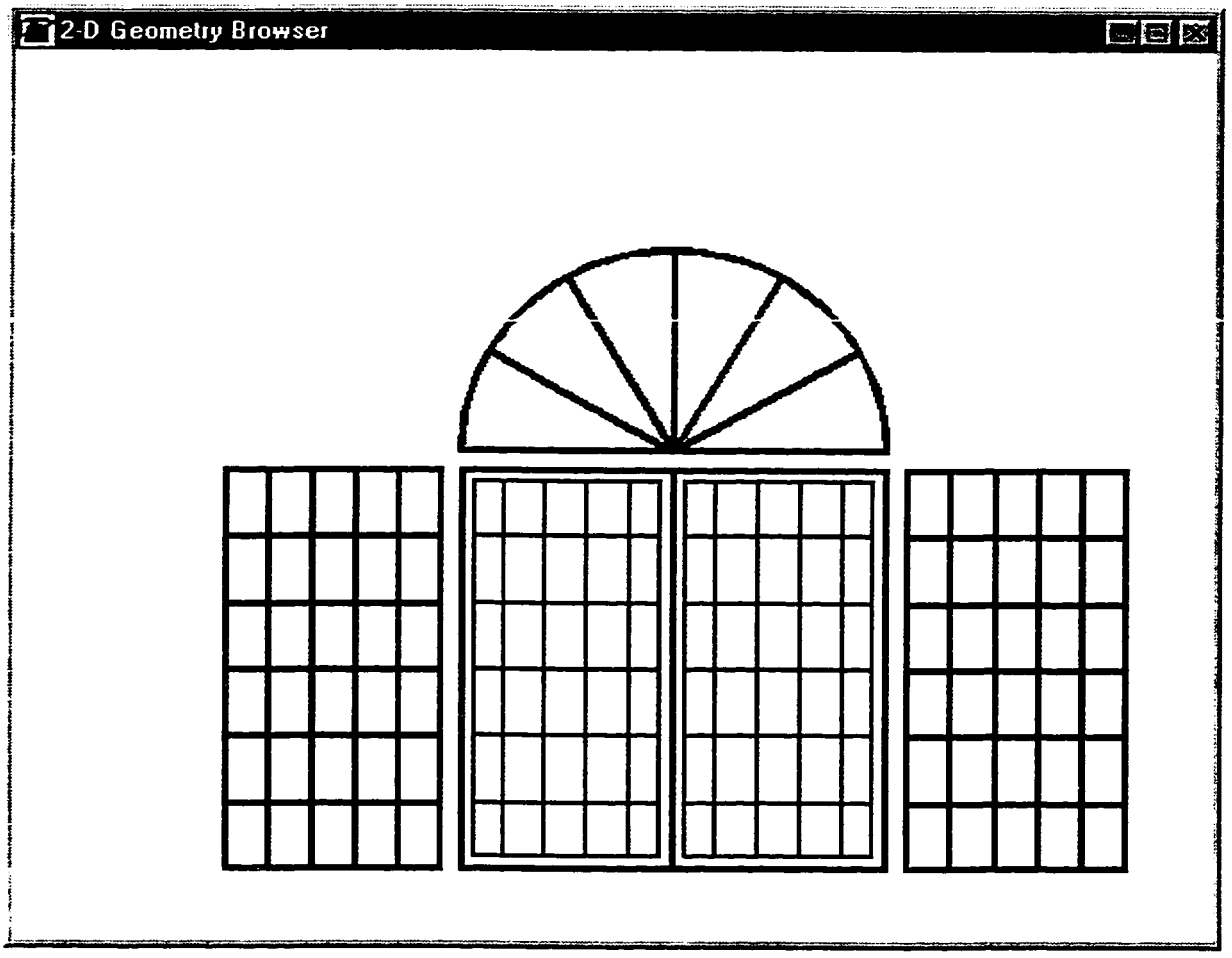


Figure 8.13: Appearance of the door after knowledge based reasoning

Next, the right and the left components of the door should be designed. To achieve this, the rule-bases related to right and left door component design

are selected. Rule-bases selected for the previous inference can be un-selected to streamline the inference process. In order to maintain the symmetry of the door, the left and the right components should be of the same type. In the rule-base *DoorLeft,* rules are defined to replace the left component if it is inconsistent with the center component. *DoorRight* is the rule-base for the right component design. The design candidate generated as a result of the knowledge-based inference is shown in Figure 8.13. Different design candidates can be generated by selecting different rule-bases. For instance, if the user wants the left and right components to be different from the center component, he/she should select rule-bases *DoorLeftDifferent* and *DoorRightDifferent,* instead of *DoorLeft* and *DoorRight.*

Chapter 9

Conclusions and Future Work

In this chapter, first conclusions of this research are summarized. Future work is then outlined.

9.1 Conclusions

The research was devoted to the development of an intelligent design system. A prototype system was developed in which functional design can be conducted efficiently. In this research, feature-based design approach was employed for modeling products. Class features were defined in a library. Product modeling was conducted by generating instance features using the class features as templates. Geometric descriptions were a part of the feature definitions. A geometry representation system was developed for associating the symbolic geometric descriptions with the 3D solid geometric descriptions. A rule-based reasoning approach was utilized for automated product modeling. An industrial application was developed using the developed intelligent design system for a local manufacturing company – Gienow Products Ltd.

The conclusions during the development of the feature-based intelligent design system are discussed in the following sub-sections.

9.1.1 Modeling of Products using Class Features and Instance Features

In the conventional CAD systems, only geometric data are modeled in the CAD database. In this research, features are used for modeling both geometric and non-geometric data. In addition, different types of information, including qualitative properties, quantitative attributes, qualitative relations among features, quantitative relations among attributes, constraints, etc., can also be described using the feature-based product modeling system.

Class features and instance features are modeled using the object oriented programming approach to improve the modularity of the system. The standard components of products are modeled as class features. Actual products are described by instance features, which are generated using class features as their templates. Class features are organized in a hierarchical data structure, so all the characteristics defined in a super-class are inherited by the sub-classes automatically. When an instance feature is generated from a class feature, all the definitions in the class feature and its super-class features are inherited by the instance feature. The descriptions of instance features can be modified based upon the design requirements.

The consistency of the product database is maintained using the data dependent relation network. When part of the design data is modified, the change is then propagated to other parts of the product descriptions using the relations among data. This function also serves as the basis for modeling the different product life-cycle aspects, such as design, manufacturing, maintenance, recycle, etc., and integrating these life-cycle aspects into the same environment by associating these life-cycle aspect descriptions using the relations.

9.1.2 Representation of Product Geometry

Geometric information is described as part of the feature definitions. Both 2D and 3D geometric descriptions are defined in class features first and then instantiated as instance features. Because the geometric descriptions in features are represented by attributes and facts, these descriptions form a symbolic geometric model of the product. A special geometry representation scheme was developed in this research for modeling the symbolic geometric model. In this scheme, the geometry of a product is described by primitives (rectangles, circles, cylinders, cubes, cones, etc.) and operations to them (extrude, lathe, move, rotate, union, subtract, and so on).

To display the 3D product geometry using a conventional CAD system, a module has been implemented to extract the symbolic geometric descriptions

from the feature-based design system and translate these descriptions into the 3D solid model format. In this research, 3D Studio MAX serves as the 3D solid modeling system due to its excellent computer graphics functions. The symbolic geometric descriptions in instance features are first extracted to a neutral file. A 3D Studio MAX plug-in system was developed using C++ to read the data in the neutral file and translate this data into 3D Studio MAX commands to create the 3D solid geometry of the product. When the instance features for representing a product are modified, the 3D geometry of the product can be updated accordingly.

9.1.3 Automation of the Product Modeling Process

The product modeling process is automated by developing the rule-based inference system in this research. Various product modeling activities, including generating new instance features and their descriptions, modifying existing instance feature descriptions, deleting instance features and their descriptions, automatically updating the product data change, etc., are carried out through rule-based reasoning.

Since the sizes of both the knowledge base in the form of rules, and the database in the form of instance features, are large, a special inference mechanism has been developed to improve the reasoning efficiency. In this approach, only part of the knowledge base and database are selected and considered during the inference. To achieve this goal, rules are organized in groups, called rule-bases. Only the relevant rule-bases are selected as the knowledge base. The partial database is selected by specifying an active instance feature.

9.2 Future Work

The effectiveness of product modeling using the developed feature-based intelligent design system has been proved by the theoretical analysis and

application implementation. To further improve this system, the research should focus on the following aspects:

1. *Integration of design with other product development life cycle aspects:*
   This research focuses on modeling the design aspect of the product life-cycle. However, development of a product involves the activities in many different product life-cycle aspects such as manufacturing, assembly, maintenance, and recycling. This design system should be integrated with other product life-cycle modeling systems in the future.

2. *Improvement in system interface environment:*
   The system interface environment should be improved. Currently the system interface environment consists of a number of browsers. Definitions of features and rules, and the result of rule-based inference are coded using text. To improve the interface environment, graphical editing functions for defining the knowledge base and database should be introduced.

3. *Improvement in 3D geometry representation:*
   In this research, only the fundamental built-in predicates are developed and implemented for modeling the 3D product geometry. More functions should be added to improve the capability of the feature-based intelligent design system. In the current system, only a small number of material types, such as copper, wood, steel, etc., have been defined. More material types should be added.

4. *Improvement in intelligent system:*
   In order to make the intelligent system more effective, more built-in predicates should be added. For instance, the built-in predicates to ask the user for entering data and to print out the design result using a certain format should also be developed in the future. In the current system, the relevant rule-bases for conducting the inference are selected manually by the users. If a user doesn't have good knowledge about the knowledge base, he/she may find it is hard to select appropriate rule-bases for modeling and developing the

products. Therefore meta-knowledge, the knowledge about the knowledge base, should also be modeled in this system to help the users to select appropriate rule-bases during the design process.

5. *Integration of design system with other product development systems using agent-based distributed knowledge base and database modeling approach:* With the advances in Internet technologies, agent-based distributed computing methods have been developed in many computer-based design and manufacturing systems [Maturana96]. The feature-based intelligent design system developed in this research can serve as a design module in the integrated agent-based product development systems.

References

[Armstrong82]
Armstrong, G. T., "A Study of Automatic Generation of Non-invasive NC Machine Paths from Geometric Models," PhD. Dissertation, Department of Mechanical Engineering, University of Leeds, April 1982.

[Bjarne94]
Bjarne, S., *Design and Evolution of C++,* Addison - Wesley Longman, 1994.

[Browne90]
Browne, J., Harhen, J., and Shivnan, J., *Production Management Systems,* Addison - Wesley, 1990.

[Buchanan78]
Buchanan, B. G. and Feigenbaum, E. A., "Dendral and Meta-Dendral," *Artificial Intelligence*, vol. 11, 1978.

[Buchanan84]
Buchanan, B. G. and Mitchell, T. M., *Rule Based Expert Systems: The Mycin Experiments,* Addison - Wesley, 1984.

[Carringer86]
Carringer, R. A., "Product Definition Data Interface," *Proceedings of CIMTECH Conference,* March 1986.

[Casale85]
Casale, M. S. and Stanton, E. L., "An Overview of Analytic Solid Modeling," *IEEE CG & A*, pp. 45-56, February 1985.

[Corney93]
Corney, J. and Clark, D. E. R., "Face-based Feature Recognition: Generalizing Special Cases," *Computer Integrated Manufacturing*, vol. 6, no. 1, pp. 39-50, 1993.

[Cutkosky88]
Cutkosky, M., Tenebaum, J. M., and Muller, D., "Features in Process Based Design," *ASME Computers in Engineering Conference*, ASME press pp. 557-562, July 1988.

[Dave95]
Dave, P. and Sakurai, H. "Maximal Volume Decomposition and its Application to Feature Recognition," *Proceedings of ASME Computers in Engineering Conference and the Engineering Database Symposium*, 1995.

[Dixon84]
Dixon, J. R. and Simmons, M. K., "Expert Systems for Engineering Design: Standard V-Belt Drive Design as an Example of the Design-Evaluate-Redesign Architecture," *Proceedings of ASME Computers in Engineering Conference,* August 1984.

[Dixon88]
Dixon, J. R., "Designing with Features: Building Manufacturing Knowledge into more Intelligent CAD Systems," *Manufacturing International '88 Symposium on Manufacturing Systems - Design Integration and Control,* vol. 3, pp. 51 - 57, April 1988.

[Dong94]
Dong, Z., Hu, W., and Xue, D., "New Production Cost -Tolerance Models for Tolerance Syntheis," *Journal of Engineering for Industry, Transactions of ASME*, vol. 166, pp. 199-206, 1994.

[Eastman79]
Eastman, C. and Weiler, K., "Geometric Modeling using Euler Operators," *Proceedings of First Annual Conference on Computer Graphics in CAD/CAM Systems*, pp. 248-259, April 1979.

[Forest68]
Forest, A. R, "Curves and Surfaces for Computer-Aided Design," PhD. Dissertation, University of Cambridge, 1968.

[Gindy89]
Gindy, N. N. Z., "A Hierarchical Structure for Form Features," *International Journal of Production Research*, vol. 27, no. 12, pp. 2089-2103, 1989.

[Goldberg83]
Goldberg, J. and Robson, D., *Smalltalk-80: The Language and its Implementation*, Addison Wesley, 1983.

[Grayer76]
Grayer, A. R., "A Computer Link between Design and Manufacture," PhD. Dissertation, University of Cambridge, UK, September 1976.

[Gregory97]
Gregory, K., *Using Visual C++ 5*, QUE Corporation, 1997.

[Gui94]
Gui, J. K. and Mantyla, M., "Functional Understanding of Assembly Modelling," *Computer-Aided Design*, vol. 26, no. 6, pp. 435-451, May 1994.

[Hagen85]
Hagen, H., "Geometric Spline Curves," *CAGD Journal*, vol. 2, pp. 223-227, 1985.

[Hovart96]
Hovarth, I., "A Workbench Architecture for Object Oriented Handling of Features," *Proceedings of 1996 ASME Design Engineering Technical Conferences and Computers in Engineering Conference*, August 1996.

[Ishii93]
Ishii, M., Tomiyama, T., and Yoshikawa, H., "A Synthetic Reasoning Method for Conceptual Design," In Wozny, M. J., and Olling, G., eds., *Proceedings of Towards World Class Manufacturing*, North-Holland, pp. 3-16, 1993.

[Johnson86]
Johnson, R. H., "Product Data Management With Solid Modeling," *Computer-Aided Engineering Journal*, pp. 129-132, August 1986.

[Johnson90]
Johnson, B.M. and Ruwe, M., *Professional Programming in COBOL,* Prentice-Hall, 1990.

[Kim94]
Kim, Y. S., "Volumetric Feature Recognition using Convex Decomposition," *Advances in Feature Based Manufacturing*, Elsevier Publications, pp. 39-63, 1994.

[Kremer97]
Kremer, R. C., "Constraint Graphs: A Concept Map Meta-Language," PhD. Dissertation, Department of Computer Science, University of Calgary, 1997.

[Krouse85]
Krouse, J. K., "Solid Modeling Catches On," *Machine Design*, pp. 50-55, February 1985.

[Lee85]
Lee, E. T. Y., "Some Remarks Concerning B-Splines," *CAGD Journal*, vol. 2, pp. 145-149, 1985.

[Mantyla90]
Mantyla, M., "A Modeling System for Top-Down Design of Assembled Products," *IBM Journal of Research and Development*, vol. 24, no. 5, pp. 639-659, 1990.

[Marisa88]
Marisa, R., "Proposed Extensions to PADL-2," Technical Note COMAPP, Cornell University, May 1988.

[Maturana96]
Maturana, F. and Norrie, D. H., "Multi-Agent Mediator Architecture for Distributed Manufacturing," *Journal of Intelligent Manufacturing*, vol. 7, pp. 257-270, 1996.

[McDermott82]
McDermott, J., "R1: A Rule Based Configurer of Computer Systems," *Artificial Intelligence*, vol. 19, 1982.

[Michael96]
Michael, T. P., *3D Studio MAX Fundamentals,* New Riders, 1996.

[Miner85]
Miner, R. H., "A Method for the Representation and Manipulation of Geometric Features in a Solid Model," M. Sc. Thesis, Mechanical Engineering Department, MIT, 1985.

[Mittal89]
Mittal, S. and Araya, A., "A Knowledge-Based Framework for Design," In Tong, C. and Sriram, D., eds., *Artificial Intelligence in Engineering Design vol. 1: Design Representation and Models of Routine Design,* Academic Press, pp. 273-293, 1992.

[Mortenson85]
Mortenson, M. E., *Geometric Modeling,* John Wiley, New York, 1985.

[Nau86]
Nau, D. and Gray, M., "SIPS: An Application of Hierarchical Knowledge Clustering to Process-Planning," *ASME Winter Annual Meeting Symposium Integrated and Intelligent Manufacturing*, 1986.

[O'Grady91]
O'Grady, P. and Young, R. E., "Issues in Concurrent Engineering Systems," *Journal of Design and Manufacturing*, vol. 1, pp. 27-34, 1991.

[Potts88]
Potts, C. and Bruns, G., "Recording the Reasons for Design Decisions," *Proceedings of 1988 IEEE International Conference on Software Engineering*, pp. 418-427, 1988.

[Putnam86]
Putnam, L. K. and Subrahmanyam, P. A., "Boolean Operations on *n*-Dimensional Objects," *IEEE CG & A*, pp. 43-51, June 1986.

[Rich91]
Rich, E. and Knight, K., *Artificial Intelligence,* McGraw - Hill Inc, 1991.

[Rosenblatt91]
Rosenblatt, A. and Watson, G. F., "Concurrent Engineering," *IEEE Spectrum*, vol. 28, no. 7, pp. 22-37, 1991.

[Shah86]
Shah, J.J. and Bhatnagar, A., "GT Coding Scheme for Sheet Metal Features," Technical Report, Department of Mechanical Engineering, Arizona State University, 1986.

[Shah88a]
Shah, J. and Rogers, M. "Functional Requirements and Conceptual Design of the Feature-Based Modeling System," *Computer-Aided Engineering Journal*, February 1988.

[Shah88b]
Shah, J. J., Sreevalsan, P., Rogers, M., Billo, R., Mathew, A., "Current Status of Features Technology, Report on Task 0," Technical Report R-88-GM-04.4, CAM-I Inc., 1988.

[Shah89]
Shah, J., "Feature Transformations Between Application Specific Feature Spaces," *Computer-Aided Engineering Journal*, vol. 6, no. 6, 1989.

[Shah95]
Shah, J. J. and Mantyla, M., *Parametric and Feature - Based CAD/CAM*, John Wiley & Sons Inc., 1995.

[Shenoy83]
Shenoy, R. S. and Patnaik, L. M., "Data Definition and Manipulation Languages for a CAD Database," *Computer Aided Design Journal*, vol. 15, no. 3, pp. 131-134, 1983.

[Soni86]
Soni, A., "An Intelligent Mechanism Selections Consultant," *ASME Computers in Engineering Conference*, ASME Press, 1986.

[Subra94]
Subrahmanyam, S. and Wozny, M., "An Overview of Automatic Feature Recognition Techniques for Computer-Aided Process Planning," *Computers In Industry*, August 1994.

[Sutherland63]
Sutherland, I. E., "Sketchpad: A Man-Machine Graphical Communication System," *Proceedings of Spring Joint Computer Conference*, Baltimore, 1963.

[Suzuki90]
Suzuki, H., Ando, H., and Kimura, F., "Geometrical Constraints and Reasoning for Geometrical CAD systems," *Computers and Graphics*, vol. 14, no. 2, 1990.

[Tan86]
Tan, S. T. and Yuen, M. M. F., "Integrating Solid Modeling with Finite Element Analysis," *Computer-Aided Engineering Journal*, pp. 133-137, August 1986.

[Tan87]
Tan, S. T., Yuen, M. M. F., and Hui, K. C., "Modeling Solids with Sweep Primitives," *Computers In Mechanical Engineering (CIME) Magazine*, pp. 60 -73, September 1987.

[Tomiyama87]
Tomiyama, T. and Ten Hagen, P. J. W., "The Concept of Intelligent Integrated Interactive CAD Systems," CWI Report No. CS-R8717, Center for Mathematics and Computer Science, Amsterdam, The Netherlands, 1987.

[Turner88]
Turner, G. and Andeson, D. C., "An Object Oriented Approach to Interactive Feature - based Design for Quick Turnaround Manufacturing," *ASME Computers in Engineering Conference*, pp.45-49, 1988.

[Umeda92]
Umeda, Y., Tomiyama, T., and Yoshikawa, H., "A Design Methodology for Self Maintenance Machine Based on Functional Redundancy, Design Theory and Methodology - DTM," *ASME Press*, pp. 317-324, 1992.

[Vickers88]
Vickers, D. L. and Swanson, K. A., "A Form Features-Centered Architecture for Product Definition Exchange," *AUTOFACT '88 Conference Proceedings*, pp. 25 - 37, 1988.

[Wilkinson87]
Wilkinson, D. and Hallam, R., "A Study of Product Data Transfer Using IGES," *Computer-Aided Engineering Journal*, vol. 4, no. 3, pp. 131-136, 1987.

[Wilson85]
Wilson, P. R., "Euler Formulas and Geometric Modeling," *IEEE CG & A*, pp. 24 - 36, August 1985.

[Woo82]
Woo, T. C., "Feature Extraction by Volume Decomposition," *Proceedings of Conference on CAD/CAM Technology in Mechanical Engineering*, pp. 76-94, 1982.

[Xue92]
Xue, D., Takeda, H., Kiriyama, T., Tomiyama, T., and Yoshikawa, H., "An Intelligent Integrated Interactive CAD," *In Intelligent Computer-Aided Design, Waldron, M. B., Brown, D., and Yoshikawa, H., eds.*, Amsterdam: North-Holland, pp. 163-192, 1992.

[Xue93]
Xue, D. and Dong, Z., "Feature Modeling Incorporating Tolerance and Production Process for Concurrent Design," *Concurrent Engineering: Research and Applications*, vol. 1, pp. 107-116, 1993.

[Xue94a]
Xue, D. and Dong, Z., "Developing a Qualitative Intelligent System for Implementing Concurrent Engineering Design," *Journal of Intelligent Manufacturing*, vol. 5, pp. 251-267, 1994.

[Xue94b]
Xue, D. and Dong, Z., "Coding and Clustering of Design and Manufacturing Features for Concurrent Design," *Proceedings of 1994 ASME Design Automation Conference: Advances in Design Automation 1994*, DE-vol. 69 -1, pp. 533-545, 1994.

[Xue96]
Xue, D., Rousseau, J. H., and Dong, Z., "Joint Optimization of Performance and Costs in Integrated Concurrent Design: Tolerance Synthesis Part," *Engineering Design and Automation*, vol. 2, no. 1, pp. 73-89, 1996.

[Xue97]
Xue, D., "A Multilevel Optimization Approach Considering Product Realization Process Alternatives and Parameters for Improving Manufacturability," *Journal of Manufacturing Systems*, vol. 16, no. 5, pp. 337-351, 1997.

[Yadav98a]
Yadav, S. and Xue, D., "Feature-Based Concurrent Design For Improving Manufacturability," *Proceedings of CSME Forum Design Integration and Optimization Conference*, vol. 3, pp. 7-14, 1998.

[Yadav98b]
Yadav, S. and Xue, D., "A Multi-level Production Scheduling Mechanism," *Proceedings of IASTED International Conference on Robotics and Manufacturing*, pp. 169-172, July 1998.

[Zarefar86]

Zarefar, Z., Lawley, T., and Eesami, F., "PAGES: A Parallel Axis Gear Drive Expert System," *ASME Computer in Engineering Conference*, 1986.

[Zeid91]

Zeid, I., *CAD/CAM Theory and Practice,* McGraw-Hill Inc., 1991.

[Zweben94]

Zweben, M. and Fox, M. S., *Intelligent Scheduling,* Morgan Kaufmann Publishers, 1994.