FEATURE–BASED REPRESENTATION FOR ASSEMBLY MODELLING

by

WAN ABDUL RAHMAN JAUHARI BIN WAN HARUN

A Doctoral Thesis

submitted in partial fulfilment of the requirements

for the award of

Degree of Doctor of Philosophy

of the Loughborough University

September 1996

Department of Manufacturing Engineering

Loughborough University

ACKNOWLEDGEMENTS

I would like to express my sincere appreciation to my supervisor, Professor Keith Case for his expert guidance, support and encouragement throughout my research work. His ideas, suggestions and comments form a valuable part of this thesis.

I am also thankful to Dave Walters and Robb Doyle for their excellent services in the CAE laboratory and prompt responses to any technical problems.

Acknowledgements are due to the Govenrment of Malaysia for providing financial support for my study and to the Director General of the Standards and Industrial Research Institute of Malaysia (SIRIM) for granting me the study leave and provide encouragement for me to pursue the research work.

Last but not least, very special thanks are due to my family and friends for their constant support. The patience, understanding and encouragement of my wife and children are greatly appreciated.

To my parents,

Sharifah Ahmad and Wan Harun Wan Hussin,

who passed away on 15th March 1996 and 28th March 1996

respectively, at the time when I really need their love and encouragement

to finish this thesis.

May Allah bless their souls

SYNOPSIS

The need for a product model which can support the modelling requirements of a broad range of applications leads to the application of a feature–based model. An important requirement in feature–based design and manufacture is that a single feature representation should be capable of supporting a number of different applications. The capability of representing products composed of assemblies is seen to be necessary to serve the information needs of those applications. To achieve this aim it is an essential prerequisite to develop a formal structure for the representation of assembly information in a feature–based design system. This research addresses two basic questions related to the lack of a unified definition for features and the problem of representing assemblies in a feature–based representation. The intention is to extend the concept of designing with features by incorporating assembly information in addition to the geometrical and topological details of component parts. This allows models to be assembled using the assembly information within the feature definitions.

Features in this research are defined as machined volumes which are represented in a hierarchical taxonomy. The taxonomy includes several types and profiles of features which cover a general range of machined parts. A hierarchical assembly structure is also defined in which features form basic entities in the assembly. Each feature includes information needed to establish assembly relationships among features in the form of mating relationships. An analysis of typical assemblies shows that assembly interfaces occur at the face level of the mating features and between features themselves. Three mating relationships between pairs of features have been defined (against, fits and align) and are represented in the form of expressions that can be used for evaluations. Various sub–types of these major mating relationships can be identified (e.g. tight fit, clearance fit, etc.) and represented through the use of qualifying attributes. Component Relation Graphs, Feature Relation Graphs and Face Mating Graphs have been developed to represent each level of interaction in an assembly, and assembly relationships are combined with knowledge on process planning into a Component Connectivity Graph.

These graphs are used as the basis for deriving an integrated data structure which is used for defining classes for each level in the assembly hierarchy.

The implementation of a prototype system has been facilitated by use of an object–oriented programming technique which provides a natural method of adding functionality to the geometric reasoning process of features and the complex relationships between the parts that make up the assembly. The feature–based model is embedded in an object–oriented solid modeller kernel, ACIS®.

The research demonstrates the possibilities for a single feature representation to support multiple activities within a computer integrated manufacturing environment. Such a representation can form the basis of design improvement techniques and manufacturing planning as well as be a model to support the life cycle of the product.

TABLE OF CONTENTS

CHAPTER 3 – OBJECT ORIENTED TECHNIQUES

CHAPTER 4 – FEATURE REPRESENTATION

CHAPTER ONE
INTRODUCTION

1.1 THE NEED FOR A PRODUCT MODEL

The need for higher productivity in manufacturing industry has grown rapidly during the last few years. The requirement for shorter product life cycles, increased pressure for shorter time to market and demand for high quality products makes it imperative for industry to focus on new product development strategies in design and manufacturing processes. In recent years, issues such as *Simultaneous Engineering* (or *Concurrent Engineering*) and *Design for Manufacturability and Assembly* (DFMA) have received an increasing amount of attention by manufacturing industries. Simultaneous Engineering means a way of work where the various engineering activities in the product and production development process, as well as the management and control of production, are integrated and performed as much as possible in parallel rather than in sequence (Sohlenius 1992). DFMA is one of the tools used to achieve the aims of Simultaneous Engineering, and is defined as a technique by which a product is designed for ease and economy of manufacturing and assembly (Boothroyd, et. al. 1994). These concepts attempt to address the issue of product development productivity by helping the designer to make early decisions that minimise costs over the life of the product, thus shortening the lead time both for the development of new products and for individual orders. A critical part of implementing these concepts is the integration of design and manufacturing processes which involves an efficient communication of large amounts of data. This is achieved through the use of computers and computerised models.

Computer–Aided Design (CAD) and *Computer–Aided Manufacturing* (CAM) systems have been key components of the automation of design and manufacturing processes. Since its beginning in the 60's, CAD has passed through a number of distinct phases (Gero 1989). It commenced with a concern for graphical representation of the objects being designed. In the 1970's there was an emphasis on object modelling to support graphical representation of geometry and topology (connectivity). CAD has been used to

create geometric entities, which is often called *geometric modelling*. However, there was a recognition that aspects other than geometric were also needed, so many systems allowed the inclusion of non–geometric attributes by attaching them to geometric entities.

By the end of 1970's and early 1980's geometric modelling had reached sophisticated levels and at the same time engineering analysis tools were finding their way into CAD systems. The most prominent amongst these was the finite element analysis method. However, with some exceptions, CAD systems were not concerned with providing direct assistance to designers in their design decision making processes. Recently the need for the designer to consider the methods of manufacturing and assembly during the design process has been emphasised, and has led to the idea of a *product model.*

1.2 PRODUCT MODELLING AND FEATURES

Product modelling refers to the activities related to representing and utilising information related to products, their design and manufacturing processes and their production management (Mantyla 1989). The ultimate goal of product modelling is to be able to represent all this information in a way that makes it possible to capture and access the relevant information through the whole design–planning–manufacturing sequence with no loss of information at any stage. Although the definition of a product model varies according to the application, it should contain data, algorithms and a defined data structure suitable for the representation of the product. The ideal model should automatically generate the design, functions, service life, manufacturing methods and all data needed for the processing of customer orders (Rembold et. al. 1993). Due to the fast development of computer and information technologies and the increasing demand for productivity, the scope and approaches to product modelling have evolved rapidly in recent years (Krause et. al. 1993). Various modelling approaches have been proposed and implemented, but as much of the information needed in the design and manufacturing process deals with the geometric shape of the product, the geometric model forms the most important component in the representation of the product model. Mortenson (1985) identifies three purposes of geometric modelling in design and manufacturing –1) part

representation, which mandates a complete geometric definition of the part for manufacturing and other applications; 2) design, which allows the user to input a geometric specification and manipulate it and 3) rendering, which uses the geometry to paint a realistic picture of the object on the computer graphics output device.

In order to manipulate data in the geometric model for design and manufacturing activities, various *geometric reasoning* techniques have been developed. Geometric reasoning involves the application of computer techniques to spatial problems so that deductions can be made from geometry (Bonney et. al. 1989). To facilitate geometric reasoning, part geometry must be represented by higher level entities that relate directly to certain design functionalities or manufacturing characteristics. This necessitates the use of a system that is capable of reasoning about the geometry and topology of a design.

Conventional CAD systems allow users to draw lines, arcs and circles as geometric entities and store a part's geometry and topology that is used for display and geometric computation. There are three predominant types of geometric representations in CAD – wireframe, surface and solid models (Bedworth et. al. 1991). Hybrid schemes of representation such as combined surface and solid modelling have also been developed and are now becoming commercially available (e.g. Unigraphics). While conventional wireframe and surface models represent only edges and envelopes of a geometry, solid models also precisely define the material inside a part. Most solid modellers represent part geometry in terms of low level geometric and topological entities. The structure and contents of a solid modeller database represent the most robust part description available, and eliminates any ambiguity in interpreting the model and provides a more complete database for performing a range of functions. For these reasons, solid modelling has become a popular choice for CAD representation and is envisaged as becoming the de facto 3D modeller of the 1990s (Sharp 1993).

There are two predominant methods of representing solid objects – *Constructive Solid Geometry* (CSG) and *Boundary Representation* (B–Rep) (Requicha 1980, Zeid 1991). CSG is characterised by an internal data structure that defines solids in terms of Boolean

operations on solid primitives such as blocks, cylinders, cones and wedges. One difficulty with the usual CSG approach is that the primitives do not always have a direct relationship to the functional features of the part, and their sizing, position and orientation are usually added in a mathematical, rather than functional way (Faux 1988). In a B–Rep technique, solids are defined in terms of the faces, edges and vertices that form the boundaries of a solid object. The topology showing the relationships among these geometric elements provides the shape and structure of the solid. The advantages and limitations of each representation in design and manufacturing have been the subject of discussion by many authors such as Joshi et. al. (1986) and are very briefly described below.

The type of representation scheme supported by a CAD system is an important consideration, because this has an effect on how a model can be visualised and more importantly, it determines which information can possibly be derived from the CAD database. In many manufacturing applications, B–Rep is preferred to CSG as each part has a unique and explicit representation and thus a B–Rep data structure directly contains the required information whereas a CSG model has to derive this information when required. The geometric domain of CSG is practically limited to the quadric surfaces such as planes, cylinders, cones, tori and spheres whereas, in theory, a B–Rep model has no such limitations. To take advantage of each representation, a hybrid CSG/B–Rep representation has been proposed by some authors such as Falcidieno and Giannini (1991).

It has been recognised that many existing CAD systems do not provide the representation necessary for geometric reasoning and lack sufficient information to support downstream manufacturing applications. Even solid modellers do not provide higher level abstractions of the part that relate directly to certain design functionalities or manufacturing characteristics. A significant problem in the use of current CAD systems is the total effort required to capture the geometry of a product, which tends to limit the desire to make significant changes in product structure once this has been fully carried out (Boothroyd et. al. 1994). Consequently, the concept of *features* has been proposed to

serve the geometric reasoning needs of the CAD system. Instead of using a model consisting of graphics primitives such as points, lines and circles as the basis of geometry definition, the designer uses a set of features such as holes, pockets and slots. Features not only describe the product but also contain implicit and explicit information (Clark and South 1987). In the academic and industrial environments, feature technology is viewed as a key technology to the next generation of computer–aided design and manufacturing (van Houten 1992). The growing use of features in the CAD/CAM area is due to the fact that features offer many advantages over conventional CAD systems. Some of the advantages of features are summarised below (Clark and South 1987, Shah et. al. 1988, Mantyla 1989, van Emmerik and Jansen 1989, Chen et. al. 1991, Gui and Mantyla, 1994):

i. Features provide a more natural vocabulary for expressing the designed object than geometric primitives. Hence they capture more of the designer's intent in the design object representation than plain geometric models.

ii. Features facilitate the capture and management of parameter relationships and dependencies in a model and thus provide a more convenient path to fully parameterised design.

iii. Features effectively divide the geometry into two levels – feature types and geometric attributes of features. This allows the designer to leave geometric details unspecified until such time as they have to be determined.

iv. Features offer a good basis for modelling various kinds of manufacturing planning information, which require non–geometric data as well as geometric data.

v. It is easy to make design changes because of the associativities between geometric entities maintained in the data structure of feature modellers.

vi. In manufacturing, the use of features has the additional benefits of cost reduction in the long term due to the development of standards which will reduce tool

inventories, reduce process control and material management problems, provide effective dimensioning and reduce errors.

1.3 FEATURES IN APPLICATIONS

Many manufacturing applications require non–geometric as well as geometric data. The information carried by features can be embedded in a product model to serve as information carriers that will feed downstream manufacturing processes in the manufacturing environment. They provide an alternative component representation that forms a suitable basis for a wide ranging set of activities throughout the product's life cycle and this facilitates the bridging of the gap between design and manufacturing. Because of this potential, features have been used in many CAD/CAM applications (Pratt 1993). In design, features have been used as a fulfilment of functional requirements, for building of a geometric model and as preparation for design analysis activities (Case and Gao 1993). In manufacturing, most of the applications of features can be found in the process planning area, where the feature data provides a convenient way to model parts (Krause et. al. 1991, Gindy et. al. 1993). Applications such as casting (Corbett and Woodward 1991), injection moulding (Al–Ashaab and Young 1995), design for assembly (DeFazio et. al. 1990), assembly planning (Wang and Li 1991), inspection planning (ElMaraghy and ElMaraghy 1994) and manufacturing cost analysis (Nieminen and Tuomi 1991) have used feature representations.

A significant aspect of the development of the above applications is that a system is typically only capable of supporting a specific application domain. For example a feature–based system for process planning is intended as an input representation for use in process planning only and cannot be used to support other applications. This limitation is mainly due to the way features are defined and data is represented, as discussed in Chapter 2. The ability of a feature–based system to be applied to more than one application is important in a Simultaneous Engineering environment and to fulfil the requirement of a product model that can support the product development process.

1.4 THE ROLE OF ASSEMBLY MODELLING

Most of the existing CAD/CAM packages can be classified as geometric modellers. Their data structures are designed to store and manipulate geometric data of individual parts. However, in most engineering design, the product of interest is a composition of parts, formed into an assembly. As products become more complex, the demand to pay more attention to the assembly process during the design phase is becoming increasingly high. With current CAD systems considerable time and effort is still required to enter and design all parts and subassemblies of a product (Boothroyd et. al. 1994). There is thus a need for a system that allows a designer to create individual parts, assemble them and then perform the necessary analysis of the assembly. Modelling and representing assemblies, generating assembly sequences and analysing assembly are all relevant issues for geometric modelling and CAD/CAM technology.

Assembly modelling deals with the inter–relations among assembled parts rather than detailed shapes of each part. Functional understanding of assembly modelling is a key step towards a real CAD environment that can support early design (Gui and Mantyla 1994). The capability to represent products composed of assemblies is needed to support further integration of manufacturing systems at a more general level as well as to serve the information needs of the applications at the level of the part (Usher 1993). An assembly model provides data for generating assembly sequences and for assembly analysis, as discussed in Chapter 7. The role of assembly modelling in a CAD/CAM environment is shown in Figure 1.1 and it forms the main focus of this research.

A mechanical assembly can be represented by the description of its individual parts and their relationships in the assembly. Most of the interaction between parts occurs at mating surfaces. The modelling representation of these relationships and mating conditions are the distinguishing characteristics between modelling single parts and assemblies. Thus an assembly modeller can be considered as an extension of a geometric modeller where the data structure is extended to allow representation and manipulation of part relationships and mating conditions.

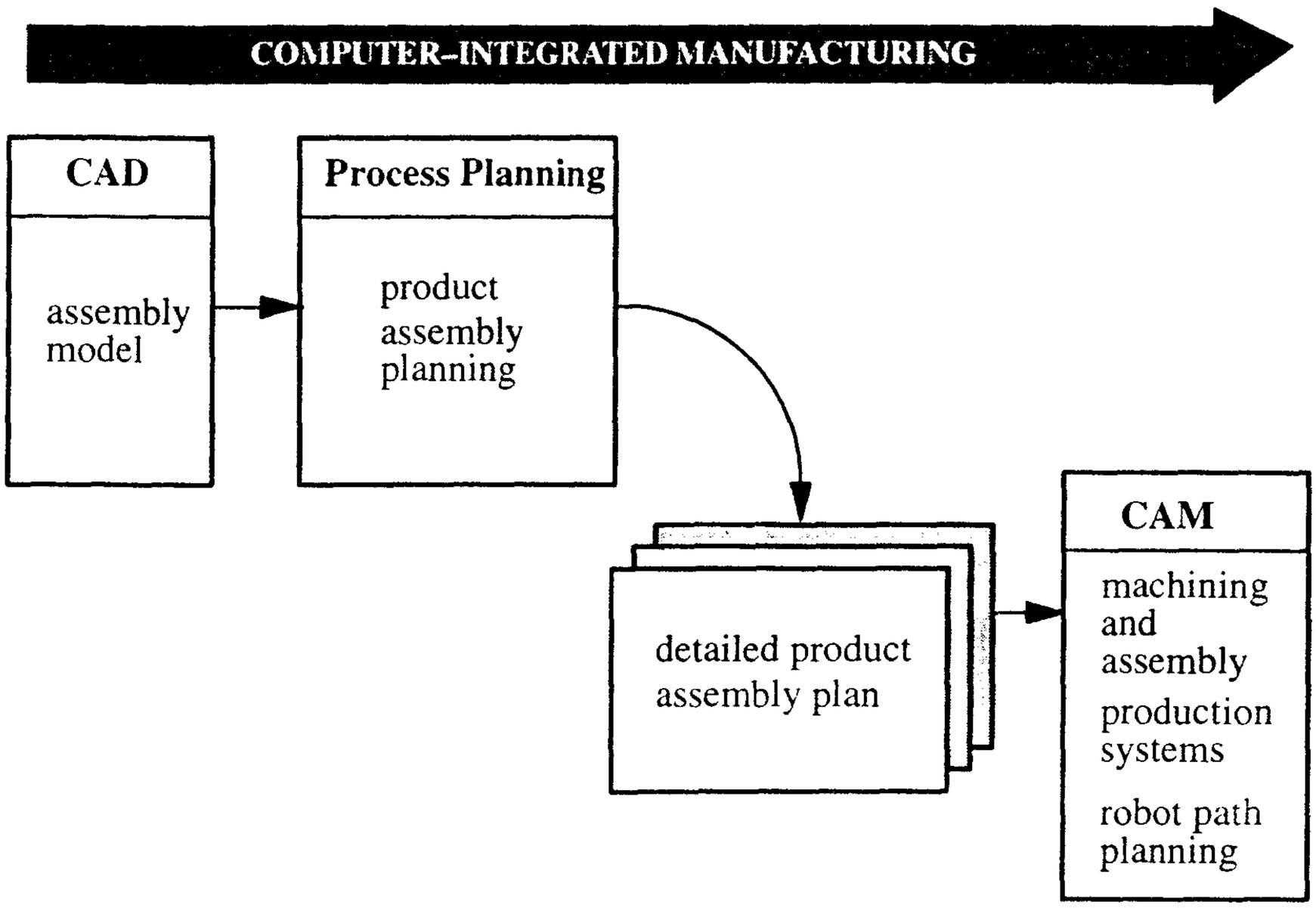


Figure 1.1: Role of Assembly Model in CAD/CAM (Lin and Chang 1993)

Individual parts are first created with the shape information (geometry and topology) and are then analysed and assembled. An ideal system allows the link to be established between the geometric and assembly model such that designers need only to modify individual parts for design modification by using the geometric modeller and the assembly model is updated automatically (Zeid 1991).

Due to the importance of assembly modelling, the activity has been the subject of much research work related to geometric modelling (Libardi et. al. 1988). Various assembly representation schemes and their related data structures are reviewed in Chapter 2. One of the significant developments in assembly modelling research in recent years is the use of features instead of piece parts as the lowest denomination of a product. This is because feature–based design has been found to facilitate assembly modelling applications by providing natural semantics for describing part interactions in a CAD system.

1.5 TOWARDS AN OBJECT–ORIENTED APPROACH

With the continuing demand to increase competitiveness, manufacturing software is becoming more sophisticated and complex (Nof 1994). Computational functions are extended and at the same time additional information types are included. The increasing level of complexity is needed to provide better computational support of necessary manufacturing functions. Since the seventies, structured programming has been the preferred method for building software systems. However, in the last few years, the concept of *Object–Oriented* (OO) programming has gained popularity in many computing areas. The OO technology has been recognised as a very promising software engineering tool that will help develop application software faster, cheaper and better through the reuse of existing program codes (Korah 1994). Many manufacturing applications have been developed to take advantage of this technology and have been shown to offer very high potential. The OO approach has already influenced the development of models for manufacturing decisions such as planning. design, control and simulation (Nof 1994).

In an OO programming environment, the basic unit of information is the object. which is defined by a name, a set of attributes that describe the object and methods to manipulate the object. A major advantage of using OO programming is that knowledge about the part is easy to maintain. The information is not scattered around the program structure but can be stored in objects that can be inherited many times. OO programming can improve the process of software development and programmer productivity. It can also result in a software product that is effective and flexible to subsequent modifications. These and other benefits, discussed in Chapter 3, have been utilised in the development of complex manufacturing software. To support the OO approach for computer programming many OO programming languages have been developed such as *Smalltalk, C++* and *Object Pascal.*

Pressures for software to interface with other systems has forced many companies to consider using OO technology as a basis for their next generation CAD systems. The OO technique has been shown to offer substantial help in simplifying the design and

implementation of CAD systems (Warman 1990, Wolf 1991). The use of OO for CAD modelling represents a means of expressing real world models and results in a design that is easier to maintain and extend to other applications. The Computer Aided Design Report (1991) suggested that the trend in the application of CAD in the next century is towards the use of *kernel modellers* using an OO approach which will be able to improve programmer productivity. Currently, one such modeller which is increasingly used is ACIS®. This software provides a collection of reusable codes to be used in the creation of solid models and the development of CAD/CAM systems, and is discussed in Chapter 3.

The OO approach was also found to provide an effective way to conceptualise and manipulate features for geometric reasoning, and has been used by several researchers to support various manufacturing applications (e.g. Unger and Ray 1984, Latif and Hannam 1993, Marefat et. al. 1993, Chen et. al. 1994).

1.6 PROBLEM STATEMENT

The overview in the previous sections shows that the integration of manufacturing processes through the support of CAD systems requires an efficient means of communicating design data to the various applications within a manufacturing enterprise. This requires an identification of the best means of representing design data in the form of a product model which can support the modelling requirements for a broad range of applications. The model should contain information from which all applications can either derive their data or access it directly. In order to fulfil these requirements, the trend points to the wider application of feature–based solid modelling with emphasis on the functionality of the product. Much previous work on feature–based design systems is concerned with using the method for the planning of machined and formed parts, with systems being dedicated to a particular application. However, an important concept in feature–based design and manufacture is that a single feature representation should be capable of supporting a number of different applications. There is a clear opportunity to extend the feature–based approach to other activities to verify the generic nature of the representation.

The development of a formal structure for the representation of assembly information in a feature–based design system is considered to be an essential prerequisite component to the generation of CAD/CAM systems that are capable of achieving the aims of optimising product design and manufacture. Such a representation can form the basis of design improvement techniques and manufacturing planning and help to support the life cycle of the product. There is a need to establish feature representations which can be an integrating agent across a number of manufacturing applications. The OO approach can provide a natural method of handling the complex relationships between the parts and sub–assemblies in the product. This research thus will address two basic issues:

1) the lack of a unified definition for features and
2) the problem of representing assembly in a feature–based representation.

The next section introduces the research objectives in consideration of the above stated problems.

1.7 OBJECTIVES OF THE RESEARCH WORK

The principal objective of this research work is:

to extend the knowledge of feature–based product representations as an aid to the automation of various aspects of design and manufacturing and to explore their use as supporting tools for assembly modelling.

To achieve the principal objective, the sub–objectives are:

i. To devise a feature representation that is capable of defining the assembly of mechanical parts

ii. To analyse typical mechanical assemblies and the interactions of features that constitute the assemblies

iii. To define and establish a taxonomy of assembly relationships

iv. To specify an enhanced version of a feature–based design system which incorporates assembly knowledge

v. To implement a prototype system using OO techniques

vi. To test the functioning of the model on typical assemblies

1.8 RESEARCH SCOPE

The research focuses on the static assembly of discrete mechanical components. Assembly parts are limited to feature types defined in this thesis. The assembly directions are limited to three primary axes of x, y and z. This is justified as seventy five percent of all products are assembled along three perpendicular directions (Delchambre 1992). The types of mating relationship are defined to suit the common surfaces available from the range of features. Other limitations are described in the relevant chapters. A simple proof of concept prototype feature–based assembly modeller is developed to validate the proposed model through testing of feature representations and profiles. The OO approach is employed in this research work through the use of the *C++ programming language* in a *UNIX* environment.

1.9 ORGANISATION OF THE THESIS

The thesis covers eight chapters. In the next chapter, a survey of the relevant literature is presented to highlight the current trends and problems of features technology and assembly modelling. Chapter 3 gives a general overview of the OO concept and the main tools, the C++ programming language and the solid modeller kernel ACIS, used in developing ideas in this research. Chapter 4 details the feature representation used, including feature definition, feature taxonomy and the application of the OO approach in representing features. In Chapter 5, the problems of assembly will be discussed through an analysis of assembly interactions involving typical assemblies. The definition of mating relationships are established and analysed with the relevant process planning knowledge. This results in a data structure which encompasses both types of knowledge.

The structure of a prototype feature–based design system is described in Chapter 6, and the prototype is tested on simple assemblies. A review of the approach and methodology used is provided in Chapter 7. Chapter 8 concludes the thesis by summarising the research findings, highlighting the main contributions and suggesting areas for future research.

CHAPTER TWO
REVIEW OF FEATURES AND ASSEMBLY MODELLING

2.1 INTRODUCTION

Feature technology has been recognised as a key technology for the next generation of computer–aided design and manufacturing systems. Research in this area is aimed at providing alternative component representations which are applicable for a wide ranging set of activities throughout the life cycle of a product. Features are used in this research as the basis for the development of an assembly model. In this chapter, relevant issues related to the research work are outlined, covering two important areas, namely features and assembly modelling. Section 2.2 presents some research issues in features technology. The research in assembly modelling and the application of features in this area are discussed in Section 2.3.

2.2 ISSUES IN FEATURES RESEARCH

Feature modelling can be considered as a relatively new development in the CAD/CAM area and much research is being undertaken to resolve the problems arising from this technique. Among major issues discussed are the definition of features, feature taxonomies, modelling approaches for feature data, representation of feature knowledge, feature mapping, standardisation of feature data, application areas and feature–based design systems. These are considered as major issues which affect this research work and thus are highlighted in the following sections. Comprehensive reviews on research in features are given by Shah et. al. (1988), Shah (1991), Salomons et. al. (1993), Case and Gao (1993), Bronsvoort and Jansen (1993) and Allada and Anand (1995).

2.2.1 FEATURE DEFINITIONS

Since features are used in the reasoning processes in various activities such as design, analysis and manufacturing, they are frequently associated with particular application

domains. Each application domain has its own definition of features, which differs from one to another. This results in a lack of a formal definition for features which is universally acceptable. The same geometry may have different interpretations according to its application. For example, a hole feature shown in Figure 2.1 may be viewed from three different perspectives – as a design feature for holding a shaft, as a manufacturing feature to be created by a machining process or as a geometric feature created by a Boolean operation (Xue and Dong 1993). The discussion on the various definitions of the concept of a feature as outlined by Unger and Ray (1988), Case and Gao (1993), Salomons et. al. (1993) and Lenau and Mu (1993) reflect the different technological or application viewpoints considered.

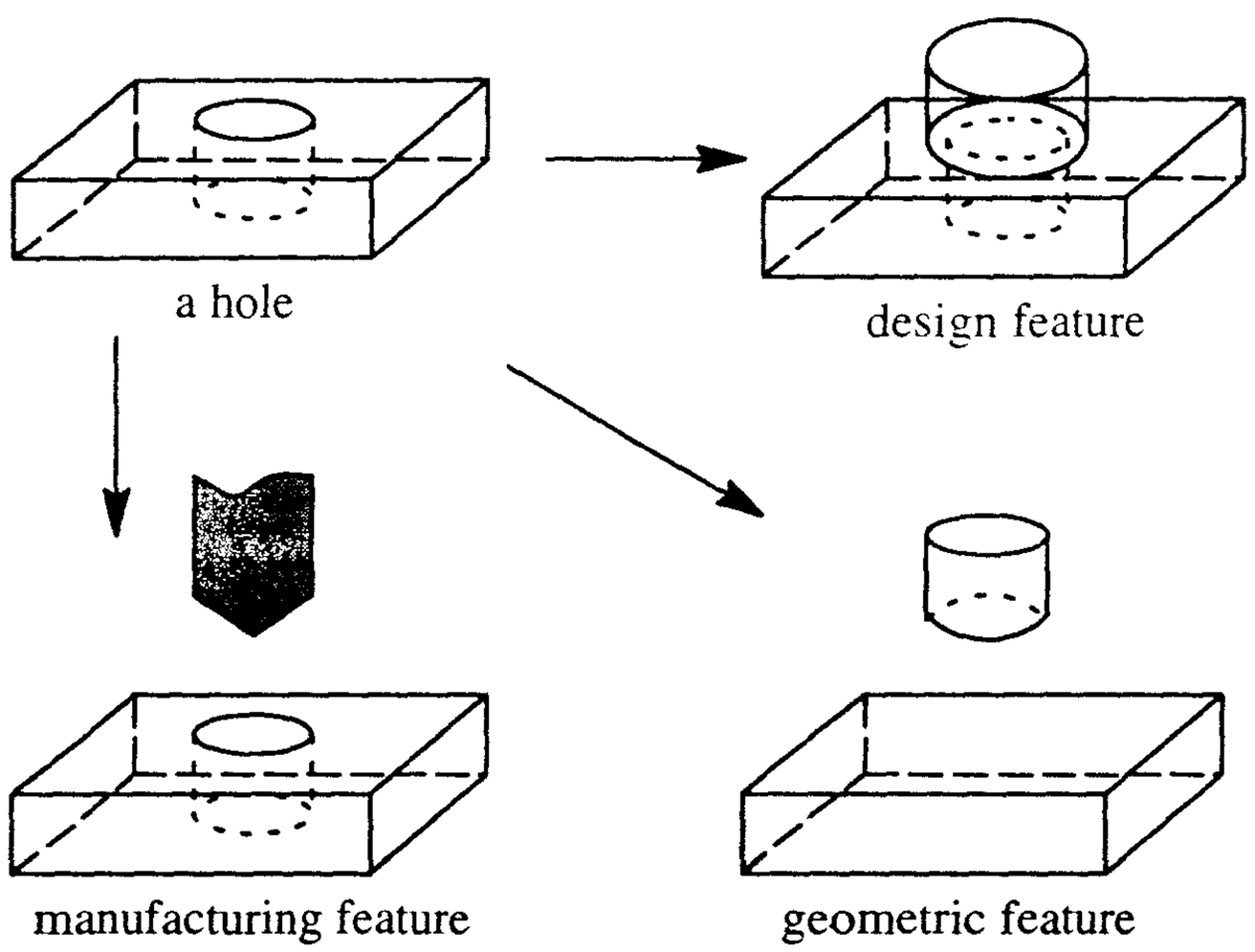


Figure 2.1: Feature definition from different perspectives (Xue and Dong 1993)

Shah et. al. (1988) analysed various definitions of features and proposed that the definitions converged into five major disciplines – design, process planning, geometric modelling, expert systems and databases. In general, the classification can be converged into two distinct applications – design and manufacturing (Van Emmerik 1991). A design feature defines generic shapes or specific geometries associated with well known technical functionality such as chamfers and keyways. It also describes a feature as it

should appear in the product model. Shah (1991) differentiates between design features and modelling features. Modelling features are groupings of geometric and topological entities that need to be referenced together while design features are elements used in generating, analysing or evaluating designs. As the motivation for feature research came from a desire to devise easier ways to define the geometry needed for process planning and NC programming, much of the earlier work defined features from manufacturing perspectives (Shah et. al. 1988). A manufacturing feature represents shape and technological attributes associated with manufacturing operations and tools, and as such they are defined according to the product type, application reasoning process and level of abstraction (Shah 1988). In process planning, features, as identified by process planners, are based upon machine tool processes and can usually be directly linked to a specific set of machine tools (van't Erve and Kals 1987). Many features are defined specifically for the process. For example, features needed to define parts for casting (Luby et. al. 1986) are significantly different from those needed for process planning of machined parts (Juri et. al. 1990). An example of features defined specifically for a product is given by Jones et. al. (1993), who defined a set of features for the design and machining of golf clubs.

As the need to consider the integration of design and manufacturing has become apparent, the application of features has been extended to cover many areas and the definitions tend to be stated in a broader and more general sense. An early attempt to define features in general terms was made by Pratt and Wilson (1985). They defined features as "an area of interest on the surface of a part". Luby et. al. (1986) defined a feature as "a geometric form or entity, whose presence or dimensions are required to perform at least one CIM function (e.g. graphics, analysis, process planning), and whose availability as a primitive permits the design process to occur". The definition given by Shah (1988) is more general – "information sets that refer to aspects of form or other attributes of a part, such that these sets can be used in reasoning about the design, performance or manufacture of the part or assemblies they constitute".

In much of the literature features are frequently referred to as form features. Form features are simply defined as shape elements with some function or meaning

(Bronsvoort and Jansen 1993) or elaborately as generic shapes with which engineers associate certain properties or attributes and knowledge useful in reasoning about the product (Sreevalsan and Shah 1992). As features do not necessarily relate to form, some definitions of features include the concept of shape. For instance, Sakurai and Gossard (1990) defined a feature as a single face or a set of contiguous faces called a face set possessing certain characteristic *facts* in topology and geometry. Masuki et. al. (1989) defined a feature as a set of faces with a distinctive pattern.

Other definitions emphasise the functions of features. Lenau and Mu (1993) classified features into functional features. Functional features represent surfaces that describe the different functions of the part and how they are positioned within the part, such as bearing and sealing surfaces. Assembly features are defined by Sodhi and Turner (1991) as form features that contain tolerance information and assembly functionality, and are used to model and create assemblies. Shah and Rogers (1993) defined an assembly feature as an association between two form features which are on different parts. Giacometti and Chang (1990) defined features used for assembly modelling as "a semantic grouping used to describe a part and its assembly. It groups functional, design and manufacturing information in a relevant manner". By grouping features into other features, design information is made available for mechanical, manufacturing and assembly analysis. In a more abstract form, the idea of fuzzy features was proposed by Clark and South (1987). Fuzzy features would be used in conceptual or exploratory design and would be less precise. For example, a designer would specify the existence of a connection, but not the type of connection. As the ideas firmed up, the connection would become more precise. Another use of fuzzy features would be to define various levels of detail depending upon the usage.

In an object–oriented environment, features are modelled as objects encapsulating various properties coupled with dedicated procedures (Wierda 1991, Wang 1991). Any set of information (geometric and non–geometric) that can be formulated in terms of generic parameters and properties and referred to as a set in the reasoning process of some application is considered as a feature (Shah et. al. 1988).

All the definitions above share the idea of a geometric entity and imply that features provide a higher level model of the object than the conventional CAD geometric model. Shah et. al. (1988) defined the least requirements a feature should fulfil – a physical constituent of a part, be mappable to a generic shape, have engineering significance and have predictable properties. In order for the feature definition to be useful, one must provide a database that has a complete definition of the part, not just geometry and topology (Shah et. al. 1988). The essence of the feature concept is that a product description not only says what the product is, but also contains implicit and explicit information on how it may be transformed to or from some other state (Case and Gindy 1991).

The discussion highlights the different definitions of features used for various applications. Although the problem of a lack of a formal definition of features has been recognised and some attempts have been made to unify the definitions, some authors believe that the need to define features to suit a particular application is inevitable and that the use of features is application specific (Chang 1990). However, in order for the feature definition to be fully useful, it should be defined to include a complete definition of the part, not just geometry and topology, and should be applicable to a wide variety of applications and functions. A model should be flexible to be adapted to the different applications found in a concurrent engineering environment.

2.2.2 FEATURE TAXONOMIES

The term feature taxonomy refers to the classification of features into classes which are often maintained in a hierarchical structure. The primary purpose of developing a feature taxonomy is to structure information in a way that relates to subsequent processing for application to problems. The success of feature modelling is largely determined by whether a useful taxonomy of feature types can be identified and organised in a modelling system and whether application–oriented data and knowledge bases can conveniently be organised on the basis of the taxonomy (Mantyla 1990). Feature taxonomies are also useful in developing product data exchange standards (Shah 1991).

The way of classifying features is highly dependent on the feature representation methodologies and the strategies for the eventual use of the feature data (Case and Gao 1993). Early attempts to classify manufacturing parts were addressed towards a geometric classification in which some typical design features were described by a numeric or alphanumeric code (Catania 1991). Some features are defined in terms of shapes, generic parameters and attributes (boss, holes), others in a variety of shapes such as ribs and webs. Examples of some feature taxonomies are described here.

The CAM–I form feature hierarchy is one of the most comprehensive classifications available (Butterfield et. al. 1986). It is organised in three groups: sheet, non–rotational and rotational features. Within these three groups, the model identifies 45 feature classes, 161 individual features and a number of attributes, notes and miscellaneous terminology. The scheme also contains classification for materials on the basis of material composition, stock form, heat treatment and surface condition. This scheme is intended for use in applications such as process planning and NC programming. Pratt and Wilson (1985) used a taxonomy of features based on the overall shape of features and the assumption that features will be incorporated in solid modelling systems. They distinguished between explicit and implicit features and produced a general classification of features. Implicit features are features that are unambiguously defined, for example by a generic description and a number of parameters for the specific occurrence, but are not evaluated into an explicit geometrical description. Explicit feature are features whose shape is explicitly described by a geometric model.

In Gindy's taxonomy (Gindy 1989), shown in Figure 2.2, form features are treated as volumes enveloped by entry/exit and depth boundaries. Feature classification is based on the External Access Directions (EAD) from which the feature volume could be machined by cutting tools. Form features are divided into three categories – protrusion, depression and surface. Feature geometry is described by defining the EADs, the boundary type (open, closed) and the exit boundary status (through/not through). The result of this grouping is a list of form feature classes that correspond to some common geometric shapes such as boss, pocket, hole, step, notch, through slot and non–through slot. The

scheme has been successfully used in process planning (Gindy et. al. 1993) and process capability modelling (Case 1994).

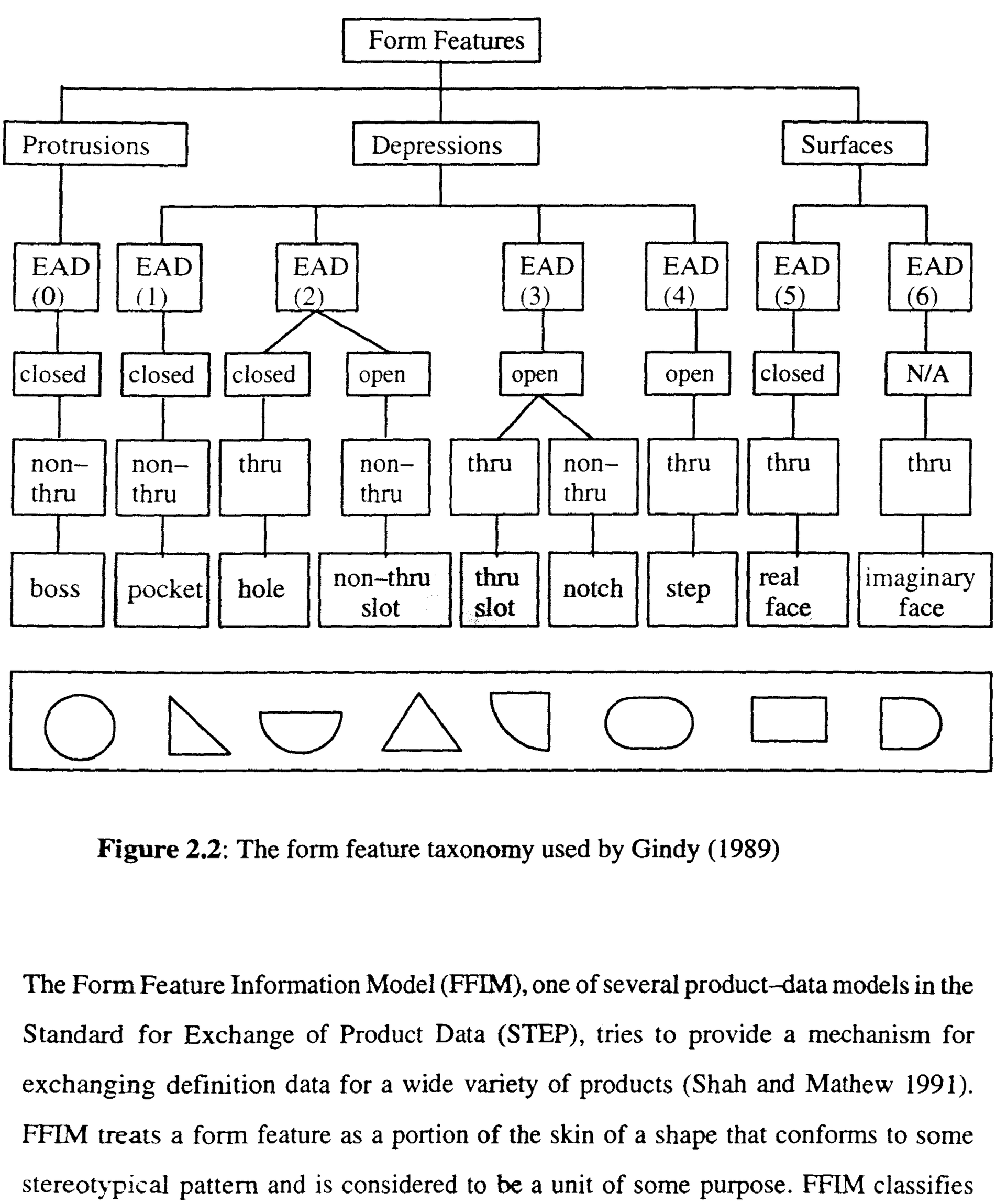


Figure 2.2: The form feature taxonomy used by Gindy (1989)

The Form Feature Information Model (FFIM), one of several product–data models in the Standard for Exchange of Product Data (STEP), tries to provide a mechanism for exchanging definition data for a wide variety of products (Shah and Mathew 1991). FFIM treats a form feature as a portion of the skin of a shape that conforms to some stereotypical pattern and is considered to be a unit of some purpose. FFIM classifies features along similar lines to the work of Pratt and Wilson (1985) into two main types:

explicit and implicit, as shown in Figure 2.3. Implicit features are further divided into six classes of depression, protrusions, passages, deformation, transition and area. Although FFIM is intended to be general purpose, the representations of some common profiles are rather complex. Criticisms of this model are highlighted by Shah and Mathew (1991). Some of these are the lack of positioning information, poor representation of certain popular profiles and the non–unique mapping of features between FFIM and the system in test. Recent information (Mill et. al. 1996) indicates that these difficulties have resulted in the indefinite postponement of adoption of the feature aspects as a part of the STEP standard.

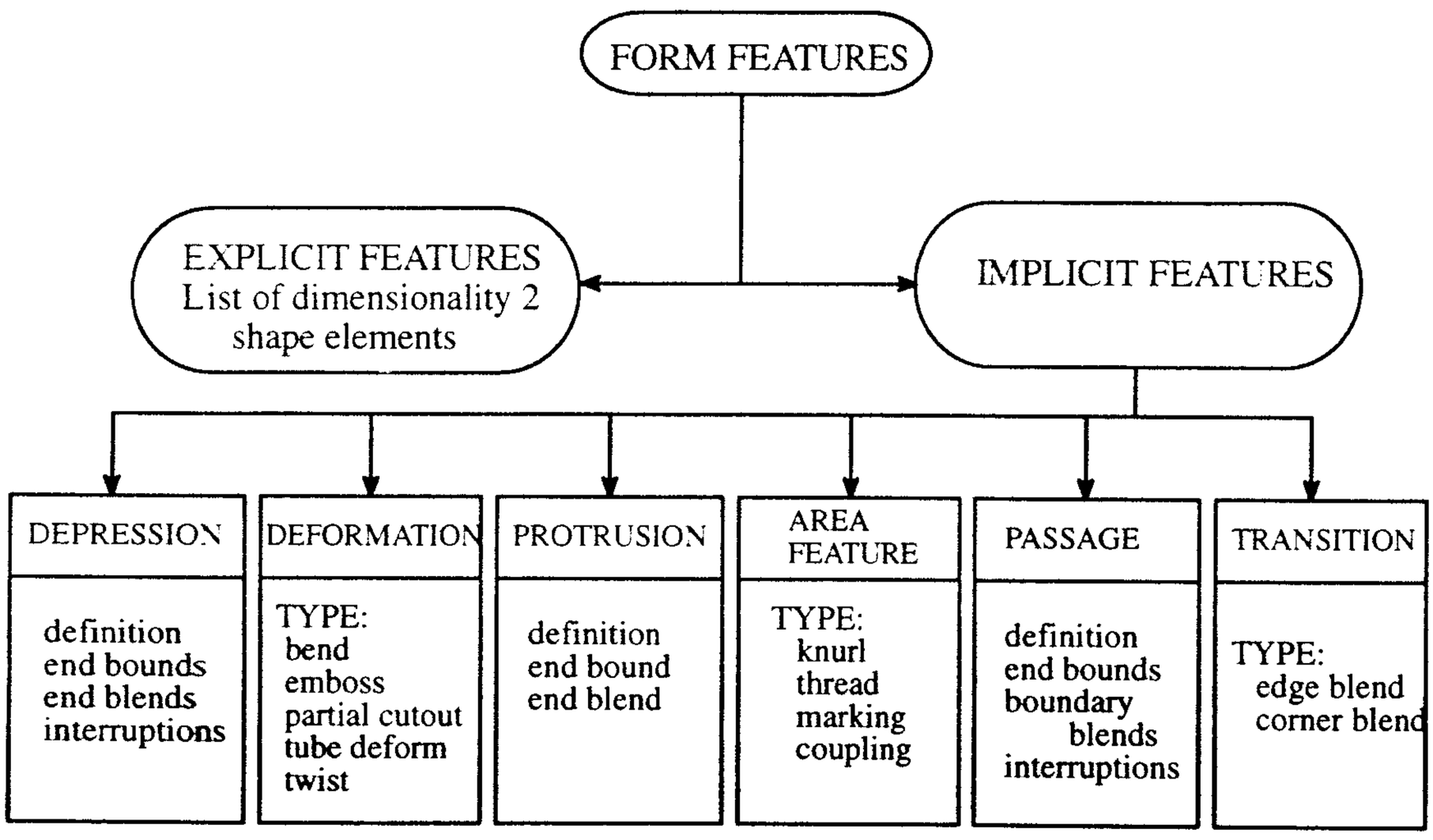


Figure 2.3: Form feature classification in the FFIM (Shah and Mathew 1991)

A feature taxonomy designed by Mantyla (1990) for an assembly modeller represents all geometric objects by means of a tree structure in which the nodes correspond to various kinds of volume features. The feature set includes both subtractive features that correspond to material removed from the parent feature (e.g. slots) and additive features

that represent material added to the parent (e.g. bosses). Arcs of the tree represent various kinds of geometric relationships between the features.

Marefat et. al. (1993) classify features into depressions and protrusions. Depression features can either be prismatic or rotational. Further classification of the features is shown in Figure 2.4. The taxonomy is used in an object–oriented environment for an integrated design, process planning and inspection system.

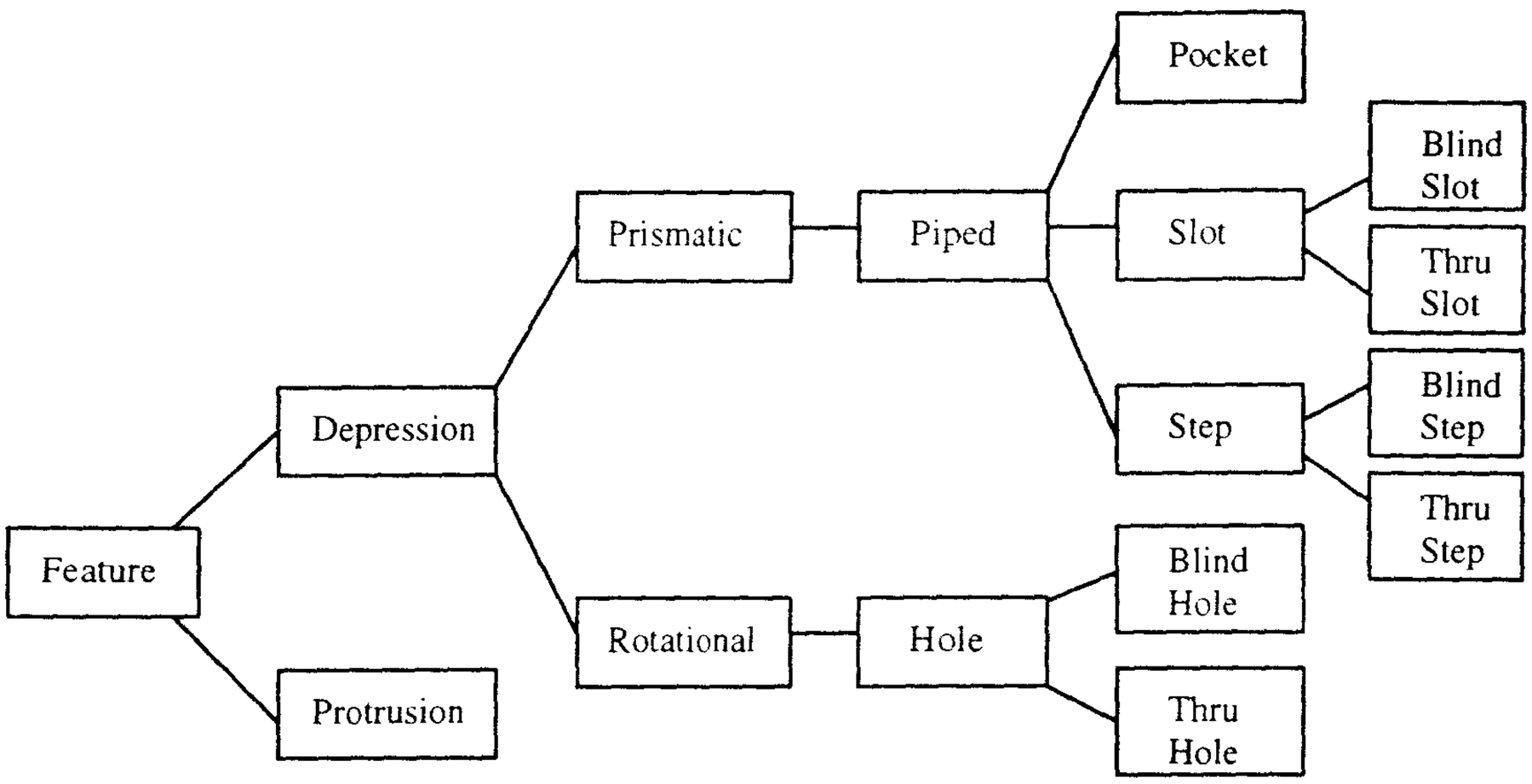


Figure 2.4: Feature hierarchy (Marefat et. al. 1993)

Although it has been argued that a general classification of features is difficult, if not impossible to develop (Bronsvoort and Jansen 1993), a well–defined feature taxonomy is an essential requirement in the object–oriented development environment, especially for manipulation purposes. Many taxonomies have been proposed to suit a particular feature representation and its eventual application. In assembly modelling, a hierarchical taxonomy is well–suited to the assembly structure as it allows the attributes of parts higher in the hierarchy to be inherited by those lower down. In this research, the taxonomy developed by Gindy (1989) was found to serve this purpose as it covers a good cross section of features involved in mechanical assembly.

2.2.3 FEATURE MODELLING APPROACHES

Previous research has established two predominant methods for creating a feature database to represent a part or a product model – *feature recognition* (or extraction) and *design by features*. Feature recognition allows the design of parts using conventional CAD systems such as 2D drafting, wireframe and solid modelling and then features are extracted from the geometric model using a recogniser and are stored in a separate database which forms the feature model (Case and Gao 1993). The process of feature recognition comprises three major tasks: feature definition, in which the rules for recognition are specified, feature classification in which potential features are classified and feature extraction, in which features are extracted from a solid model and stored for further analysis (Prabhakar and Henderson 1992). Feature recognition can be broadly classified into two approaches – human assisted and automatic. The latter method, shown diagrammatically in Figure 2.5, has been widely used in place of the former. Various approaches have been developed to achieve the goal of feature recognition, depending on the type of geometric model used i.e. whether it is based on B–Rep or CSG model. A graph–based method to recognise features is the most popular technique and a typical example of this is described by Joshi and Chang (1988). Other techniques are based on syntactic pattern recognition (Choi et. al. 1984), rule–based methods (Henderson and Anderson 1984), a decomposition approach (Nitschke et. al. 1991, Kim 1991) and the application of neural networks (Prabhakar and Henderson 1992). Feature recognition of machining features from 2D models has been demonstrated by Meeran and Pratt (1993).

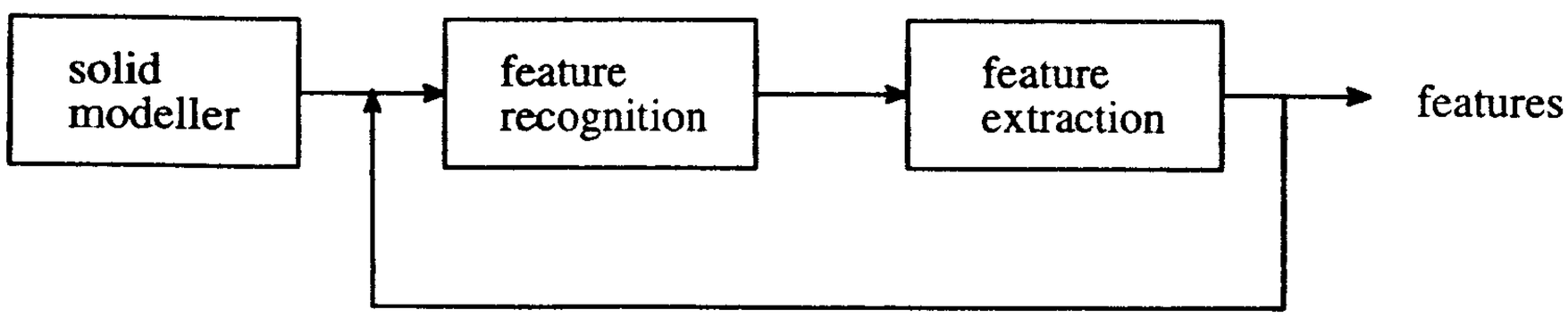


Figure 2.5: Automatic feature recognition

Feature recognition offers several potential advantages of consistency, applicability to different processes and a saving in manpower (Woodwark 1988). The approach is also seen as the most versatile for the transformation of product models between application domains (van Houten 1992) or it can be made application–specific allowing each application program to have its own recognition program (Shah et. al. 1988). These advantages have been utilised in a variety of applications such as part modelling (Nitschke et. al. 1991), process planning (Lee et. al. 1993), determination of tool approach directions (Karra and Phelps 1990), as input to Design for Assembly analysis systems (Rosario and Knight 1989), set–up planning and fixture design (Sakurai and Gossard 1991) and automatic dimensioning of 3D solid models (Oh and Lee 1990). Although much of the early work in features involved feature recognition, not much emphasis has been given to more development of this approach in recent years. This is due to the many drawbacks of the approach. It cannot retrieve information that is not in the CAD database such as tolerances, surface conditions and geometrical information (Sreevalsan and Shah 1992). Most of the systems have a restricted domain of recognisable features (Bronsvoort and Jansen 1993). There are also errors caused by multiple translation from product model to the CAD model and then to feature recognition model (Chamberlain et. al. 1993). Objects such as sculptured surfaces and interacting features make the feature recognition task more difficult (Case and Gindy 1991). In general, the algorithms and techniques involved in feature recognition are complex and require intensive programming. Above all, a technique which involves detecting features which are already there is considered to be redundant effort (Shah et. al. 1988).

In the design by features approach shown in Figure 2.6, the designer is provided with a feature library. In most of the systems, the form of a feature is created within a geometric model by a procedure based on a given set of feature parameters. Once features have been created and are available in the feature model, they can be used and accessed by a variety of downstream applications. The approach can eliminate the need for feature recognition and gives a unique, pre–defined feature list with which designers may construct their parts and thus improve the design environment provided by CAD systems (Case and Gao

1993). The pre–packaged solutions to commonly occurring functional requirements in a product which is represented by features will simplify and standardise the processes of design and manufacturing (Faux 1986). The approach allows a greater depth of understanding of features and feature interactions to be generated, which can ultimately help in the identification of a combined feature pre–definition and feature analysis approach to manufacturing planning (Young and Bell 1993). It also offers the possibility of considering manufacturing and assembly concerns early in the design process (Salomons et. al. 1993). This will lead to lower design costs and lead times, more reliable cost estimating and more predictable control of manufacturing costs, times and quality (Faux 1986). However, there are some limitations to this approach. It assumes that the designer has ample manufacturing knowledge with which he/she can transform the design into manufacturing details. The method imposes limitations on designers due to the finite nature of the features library and thus not all operations are possible (Sreevalsan and Shah 1992). This problem can be overcome by extending the range of features for the application of interest or by incorporating higher level information in the features.

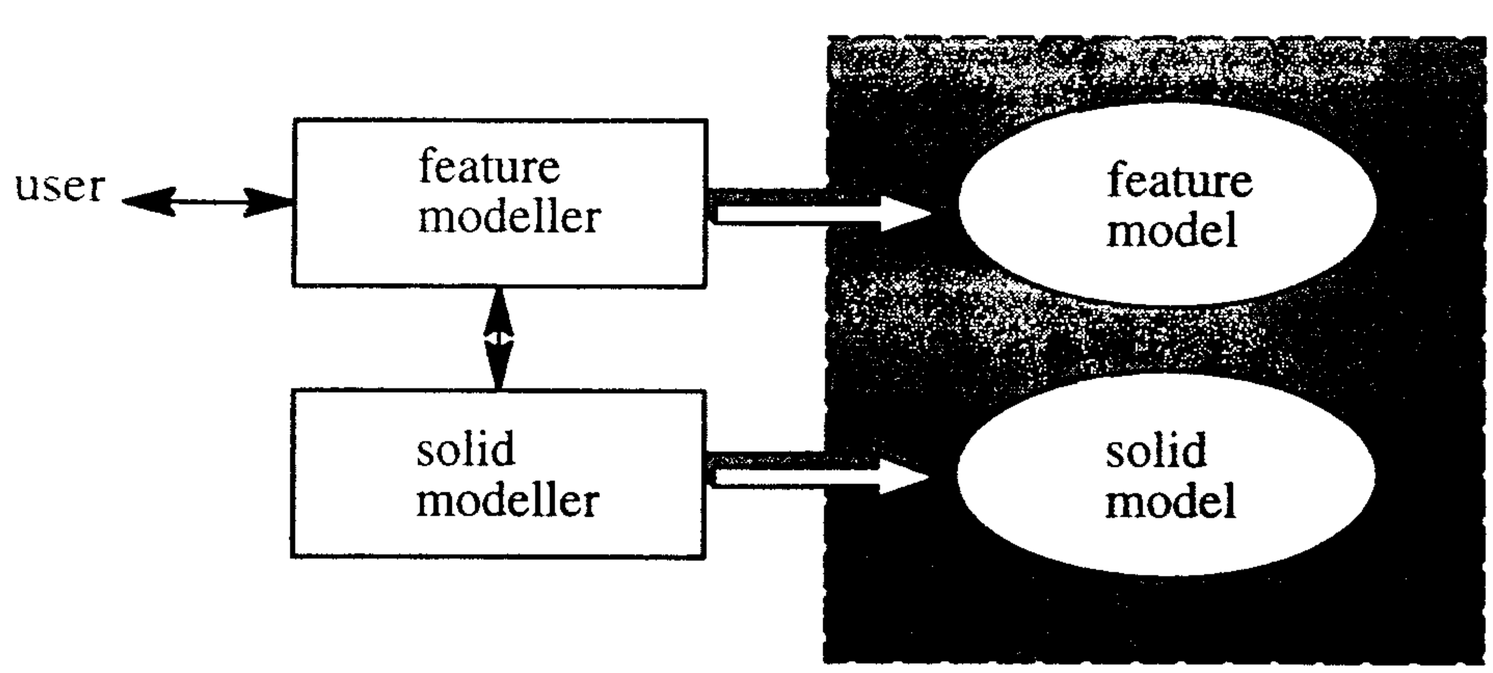


Figure 2.6: Design by features

Many researchers believe that feature recognition or design by feature approaches on their own are not enough to fulfil the requirements of a flexible feature–based design system (Falcidieno and Giannini 1991, Sreevalsan and Shah 1992, Case and Gao 1993). Both should be integrated to gain the benefits of each other. An attempt to incorporate a

feature extraction system in the design by feature system with the purpose of extracting protrusion features was reported by Chamberlain et. al. (1993). Laakko and Mantyla (1993) designed a feature–based modelling system which implements a hybrid of design by feature and feature recognition techniques in a single framework. Fu et. al. (1993) have also combined both approaches for the transformation of feature representations.

In this thesis, the design by feature approach is chosen as it provides the advantage of storing relevant information for applications during the design process, as well as offering the possibility for considering assembly concerns early in the design process. This is not possible using the feature recognition approach.

2.2.4 REPRESENTATION OF FEATURE KNOWLEDGE

Since features arise from the reasoning processes and languages used by humans, computable representations of features and feature languages have been developed (Rosen 1993). Features have been represented using codes. particularly based on the Group Technology (GT) approach in many early process planning systems. Since this is inefficient and more suited to a manual approach, they are no longer used.

The application of *Artificial Intelligence* (AI), particularly the *Expert System* or *Knowledge–Based Systems* (KBS) technique in CAD systems has been commonplace. Feature–based design systems have been used to provide representations which serve KBS that reason about the geometry and topology of designed parts. Databases built from features extracted from solid models can be submitted to a KBS for further analysis. Feature knowledge is represented using various methods such as special descriptive languages, frames (Joshi et. al. 1988) or rules (Henderson and Chang 1988). Most of the systems which integrate feature–based design and KBS are used for process planning where extensive data on process capabilities and material properties require an appropriate handling mechanism. Examples of such systems have been developed by Bond and Chang (1988), Unger and Ray (1988), Henderson and Chang (1988), Chung et. al. (1988) and Catania (1991).

In recent years, object–oriented (OO) techniques have been used widely in many computer applications. OO software seems to be able to support the feature concept as well as the feature taxonomy idea (Salomons et. al. 1993). Using an OO structure provides a general way to think about and manipulate features for geometric reasoning (Chung et. al. 1988). From an OO point of view, features are perceived as objects that have a name for identification, a number of attributes to describe their characteristics and methods to manipulate them. This information is declared and stored in an entity called a *class*, which acts as a template description for objects of a specific type. Different classes can be organised in a hierarchy or taxonomy which is readily extensible to include additional data and relationships as appropriate. Each feature is modelled as an object encapsulating various properties coupled with dedicated procedures (Wang 1991). An example of an OO representation of an instance of a feature class (which is a face) is shown in Figure 2.7 (Zhang et. al. 1992). Other examples of OO feature–based systems can be found in Luby et. al. (1986), Unger and Ray (1988), Kuttner (1988), Masuki et. al. (1989), Catania (1991), Chen et. al. (1991) and Chen et. al. (1994).

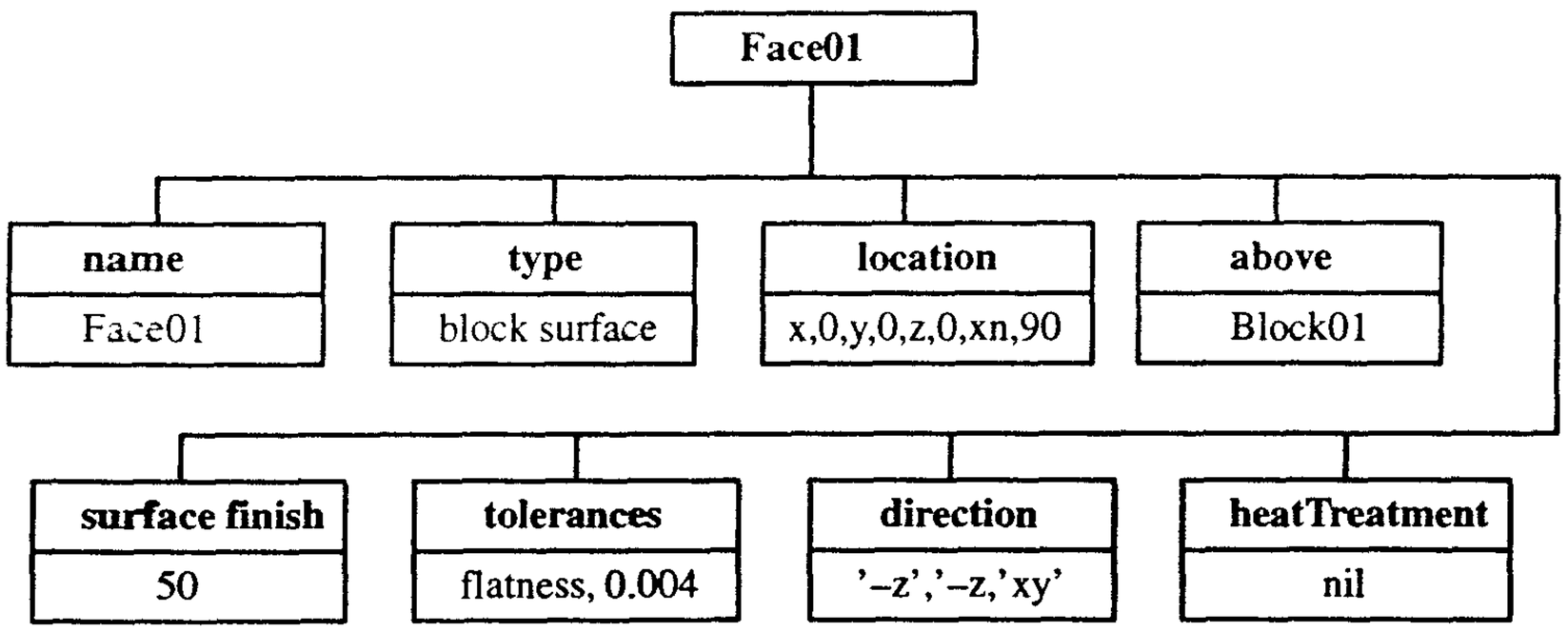


Figure 2.7: An instance of feature class (Zhang et. al. 1992)

There is an increasing trend in the use of the OO approach as the rich data types of the representation make it possible for a feature model to capture more information on the applications. There are many other benefits offered by the OO technique and this is discussed in Chapter 3. Because of these benefits, the OO approach is utilised in the

representation of features and assembly in this research work as discussed in Chapters 4 and 5.

2.2.5 MANUFACTURING APPLICATION AREAS

In the following sections, two predominant manufacturing applications that have employed features – process planning and assembly planning – are discussed. Descriptions of other manufacturing applications can be found in Shah et. al. (1994).

2.2.5.1 PROCESS PLANNING

Process planning is the activity to determine appropriate procedures to transform raw material into a finished product, as specified by the design specification. The need to automate this activity leads to the use of computers in systems that are generally called *Computer Aided Process Planning* (CAPP) systems. There are two approaches used in CAPP – variant and generative (Chang and Wysk 1985). The variant approach uses a data retrieval system to retrieve an existing process plan using a GT approach for identification. The plan is then modified (made a variant of the original) and possibly added to the database. A generative CAPP system synthesises process information in order to create a process plan for a new component automatically. For process planning, part dimensions and geometric tolerances need to be available, and to achieve this, the CAD interface must be able to convert the part description into an explanation of the part's features and characteristics.

Most of the generative CAPP systems are based on the feature description of parts (Wierda 1991). Features provide a high–level description of the part, which is a fundamental requirement for reasoning to determine processes, operation sequences, machine and tool selections and other decisions related to the process planning activity. A feature can be associated with some machining operation and this makes the operation selection in process planning relatively easy. For example, a hole can be bored with a particular type of boring machine or a slot can be milled with a particular type of milling machine using a specific tool (Bronsvoort and Jansen 1993). The feature data also provides the most convenient way to model the machined surfaces in process planning.

Various representations and syntaxes of the feature information required for process planning and the sources and means to obtain it have been reviewed by Shah et. al. (1988). Chang (1990) summarises various work on feature recognition used in process planning and NC programming applications. Feature recognition has been the most common approach to extract manufacturing features from CAD for process planning application (van't Erve and Kals 1987, Bond and Chang 1988, Unger and Ray 1988, Henderson and Chang 1988, Krause et. al. 1991, Nitschke 1991, Young and Bell 1993). A number of process planning systems use a design by feature approach (e.g. Gindy et. al. 1993). Shah et. al. (1991) uses feature databases where the user inputs feature information in a text format.

The use of features in process planning has been extended to the design and planning of fixtures (Dong et. al. 1991, Nee et. al. 1992) and as an input to a knowledge–based cost analysis system (Nieminen and Tuomi 1991).

Although features provide a natural form of representing parts for process planning, they are usually limited to single parts. Most of the feature–based process planning systems have a restricted domain of recognisable features that limits the application domain. As there is an increasing need to extend the manufacturing applications beyond the process planning domain, the way features are defined becomes important.

2.2.5.2 DESIGN FOR ASSEMBLY AND ASSEMBLY PLANNING

Computer support for the design and analysis of assemblies is essential since individual component optimisation will not necessarily mean an optimum assembly (Rosen 1993). The use of computers in assembly has been evident for some time (e.g. Swift 1987). More recently, features have been seen as a means of modelling products in a way that is suitable for *Design For Assembly* (DFA) analysis or as an input to assembly planning systems.

One of the objectives of DFA is to achieve assembly through simplification and redesign and reduction of parts by integrating the functions of the parts. DFA analysis procedures require certain geometric properties for each component part and sub–assembly (Rosario

and Knight 1989). The main types of information needed in DFA are component positions and orientations, mating features, mating operations and component/feature geometry (Molloy et. al. 1993). A Feature–Based Design system has been integrated with several assembly analysis and synthesis algorithms to be used in the DFA systems (DeFazio et. al. 1990, Molloy et. al. 1993). In the work by DeFazio, a feature–based design system captures design intent in the form of assembly topology, product function and manufacturing or field use. The work involved identifying the information important to DFA tasks and how that information could be captured using feature–based design. The feature–based design was then integrated with assembly analysis and synthesis algorithms. Li and Huang (1992) developed an automatic assembly coding from a feature–based model and this is used for an automated DFA system.

Assembly planning is concerned with creating steps of assembly operations based on connectivity relationships between component parts, from which a product is assembled (Wang and Li 1991). It involves the application of algorithms and heuristic rules to produce alternative feasible assembly plans. Five types of information are required to generate an assembly plan (Delchambre 1992) – component geometry, component attributes, final assembly information (assembly directions), topology and technological aspects (additional constraints). The quality of the plan generated by the assembly planning system depends on the representation of the parts and their relationships. Thus the description of an assembly to the computer in terms of geometric relationships and physical constraints is a critical problem and crucial for automatic assembly planning. The geometric input to the system can be provided by features which can identify connections between parts that make up the assembly.

Many knowledge–based systems have been developed for assembly planning as they are suited to handling a large amount of data and the existence of insufficient or ambiguous information. A review of research work in computer–based environments for supporting the concurrent design of products and assembly is given by Lim et. al. (1995). The review includes detailed discussion on the roles of features and mechanical assembly modelling in providing an effective environment for the design of components and assemblies. The

different approaches for representing assembly models of parts are discussed in Section 2.3.

This brief overview of the application of features in DFA and assembly planning highlights the need for a CAD modelling system which is based on the application of features that can model assembly efficiently. A well defined model provides a means of examining complex geometric interactions before anything is built and thus will be useful for further analysis and planning activities. Assembly modelling is reviewed in Section 2.3.

2.2.6 FEATURE MAPPING

Most of the work in feature–based applications as described in preceding sections concentrates on one application and product type. In an integrated manufacturing environment, it is beneficial to have features that can be transformed from one application to another. There is a need for a system that supports multiple applications driven by a common or stored database (Shah et. al. 1988). Each application can have its own view of an object or definition of the object, with features relevant for that application.

In order to integrate the various applications, features identified in a particular domain have to be partially or fully transformed to other domains. The desirable situation is for the design feature to comprise manufacturing aspects and manufacturing features to include information on the design intent. A feature mapping system is necessary to transform information in shared or neutral databases to application specific features most suited to a given reasoning process. Shah and Rogers (1988) define feature mapping as the selective extraction of relevant data by applications and transformation of this data to conform to the application view for use in its reasoning process. This may involve selective feature extraction, decomposition into lower level entities, reconstruction by geometric reasoning and in some cases, augmentation with the addition of new entities.

Feature mapping is seen as a critical area for the success of feature–based design systems (Shah and Rogers 1988, Shah 1991). As the discussion in Section 2.1.4 suggests, many

feature–based systems are confined to one application area, thus avoiding the problem of feature mapping. However, some attempts have been made to support multiple applications. A generic mapping shell has been developed as part of the feature–based design system based on a general theory of feature transformation between application specific feature spaces (Shah 1988). The shell supports three related applications – Group Technology classification, process selection and manufacturability evaluation. Feature spaces represent collections of features relevant to a specific application domain. Dong et. al. (1991) applied feature mapping from design features to manufacturing features for fixture design. Falcidieno and Giannini (1991) proposed a method which allowed the user to extract features from a B–Rep model and represent them in the context of multiple functional viewpoints like manufacturing, handling and assembly. This is done by mapping features into a new model called a Shape Feature Object Graph which is considered a neutral format description, independent of the application model.

Most of the multiple view problems above have been solved at the single component level. It is useful to also take assembly relations into account when solving the problem. There is a need for a system that supports multiple applications driven by a common or shared database.

2.2.7 STANDARDISATION OF FEATURES

The realisation of an integrated manufacturing environment is not possible without powerful, widely–accepted and standardised interfaces which will contribute to harmonising data structures. Unless data can move freely between the various computer–aided systems throughout the life cycle of the product, full integration will not occur. In order for feature–based design to be useful in application, the feature data should be able to be transferred efficiently without any loss of information. An independent platform is required to fulfil this requirement. The need for standardisation of a means of defining features has been highlighted by several authors such as Pratt (1993).

One of the earliest efforts to improve the data exchange and sharing process between functions found in a manufacturing enterprises was through the *Initial Graphics*

Exchange Specification (IGES). IGES is an engineering data exchange specification supported by major CAD/CAM systems. However IGES suffers from several drawbacks such as its limitation to the geometric data only, lack of interfaces to CAD systems and its lack of ability to be used with application programs (Shah 1988). As a spin–off of the IGES activity, the *Product Data Exchange Specification* (PDES) was developed with the aim of creating an international standard for the exchange of product model data (SME 1989). In the international community, a co–ordinated effort with similar objectives is called STEP – *Standard for the Exchange of Product Model Data.*

STEP is a CAD/CAM product data exchange standard designed to support data sharing through the exchange of physical files as well as common application programming interfaces and database implementations. It uses the EXPRESS data definition language as a tool for providing object–oriented, integrated views of product data (ISO 1991). The objective of this standard is to provide a mechanism capable of describing product data throughout the life cycle of a product. STEP is seen as a promising platform which can provide a common language for data exchange and the project represents the most concentrated international effort so far to meet this need. Its aim is to develop and standardise specifications for exchange and sharing of product life cycle data between heterogeneous computer systems in a Computer Integrated Manufacturing environment. Some parts of STEP such as the geometric modelling aspects have been adopted as international standards, whereas other aspects are still at the proposal stage.

As noted before, considerable difficulty has been experienced in standardising features and the assembly applications. Research involving the application and examination of STEP has been highlighted by few researchers. An experiment to determine whether there was a mismatch between the Form Feature Information Model (FFIM), a product data model in STEP shown in Figure 2.3, and a feature–based design system was conducted by Shah (1991). The study involved mapping of the features of a feature–based system into sets of FFIM entities, inverse mapping and transferring data to and from the system to the FFIM format by creating models in the system. A limited neutral exchange structure has been developed to enable the transfer of feature–data

between a feature–based design system, LUT–FBDS, and the form feature representation schema outlined in Part 48 of the STEP standard (Smith 1993). In design for assembly, the STEP/EXPRESS standard has been proposed to define a product model (Molloy et. al. 1993).

It is clear that standardisation in the areas of interest to this research has either not been established or has not reached a stage of adequate maturity. However the methods adopted, particularly the object–oriented programming, are believed to be useful in any future attempts at compatibility with standards.

2.2.8 FEATURE–BASED DESIGN SYSTEMS

The development of feature–based design systems is necessary to support the various applications discussed earlier. In order for the systems to be useful, they have to fulfil a number of requirements as outlined by Broonsvort and Jansen (1993), Shah and Rogers (1988) and Duan et. al. (1993) Among them are:

i. The system must have an integrated data representation

ii. Mechanisms for mapping features into application systems should be provided

iii. The system should be interactive and graphical

iv. There must be a mechanism to define generic descriptions of features as well as application–oriented features and store these in a feature library

v. There must be a mechanism to create instances of a feature by specifying the required parameters

vi. There should be the ability to carry out consistency verification of geometry and attributes. Constraints must be available to guarantee the validity of the features

Many feature–based design systems have been developed in conjunction with the research work described in earlier sections. Examples of the systems are Casper (Luby et. al. 1986), DLink (Patel and McLeod 1988), CADETS (Lawlor–Wright and Hannam 1989), ASU Features Testbed Modeller (Shah and Mathew 1991), LUT–FBDS (Case et. al. 1993), FSMT (Duan et. al. 1993) and DEFP (Lenau and Mu 1993).

A few commercial CAD systems have also incorporated a feature–based approach. Most recent versions of the major CAD/CAM systems (e.g. Unigraphics, Catia) have some claim to have the capability of design by features. Some systems, such as Pro–Engineer (Parametric 1993) and MicroStation® Modeler (Bentley 1994) are fully committed to a features approach. Pro–Engineer is a parametric, feature–based mechanical design system. Using Pro–Engineer, feature–based design can be enhanced through pro/FEATURE, a module which allows users to create 'user–defined' features and complex design features such as shells, 3D swept features, features created by blending non–parallel cross sections and others.

In the main, features in the commercial systems are seen as a convenient mechanism for defining the parametrics of geometric primitives and simple boolean operations and can only be considered as design features. In general it is not possible to associate attributes such as surface finish, and nor is it possible to meaningfully export the feature descriptions to activities such as process planning. In some cases features are further restricted in their use to initial geometry creation and the effects of modifying the feature's (geometric) parameters are poorly defined and may lead to model corruption.

The following sections describe three feature–based design systems developed mainly for academic and research purposes, and serve to illustrate the various approaches and capabilities.

2.2.8.1 ASU Features Testbed Modeller

The ASU (Arizona State University) Features Testbed is a proof–of–concept system that primarily uses the design by features approach (Shah and Mathew 1991). The system is a collection of modules for the design, documentation and evaluation of mechanical parts. It is organised into two shells, one for design (modelling shell) and the other for mapping and applications. The shells can be customised by various organisations to fit their needs. The system allows users to define their own generic features without making any changes to the code. The structure of the system is shown in Figure 2.8. ASU has been used for

manufacturability evaluation (Shah and Hsiao 1991) and for assembly modelling (Shah and Tadepalli 1992).

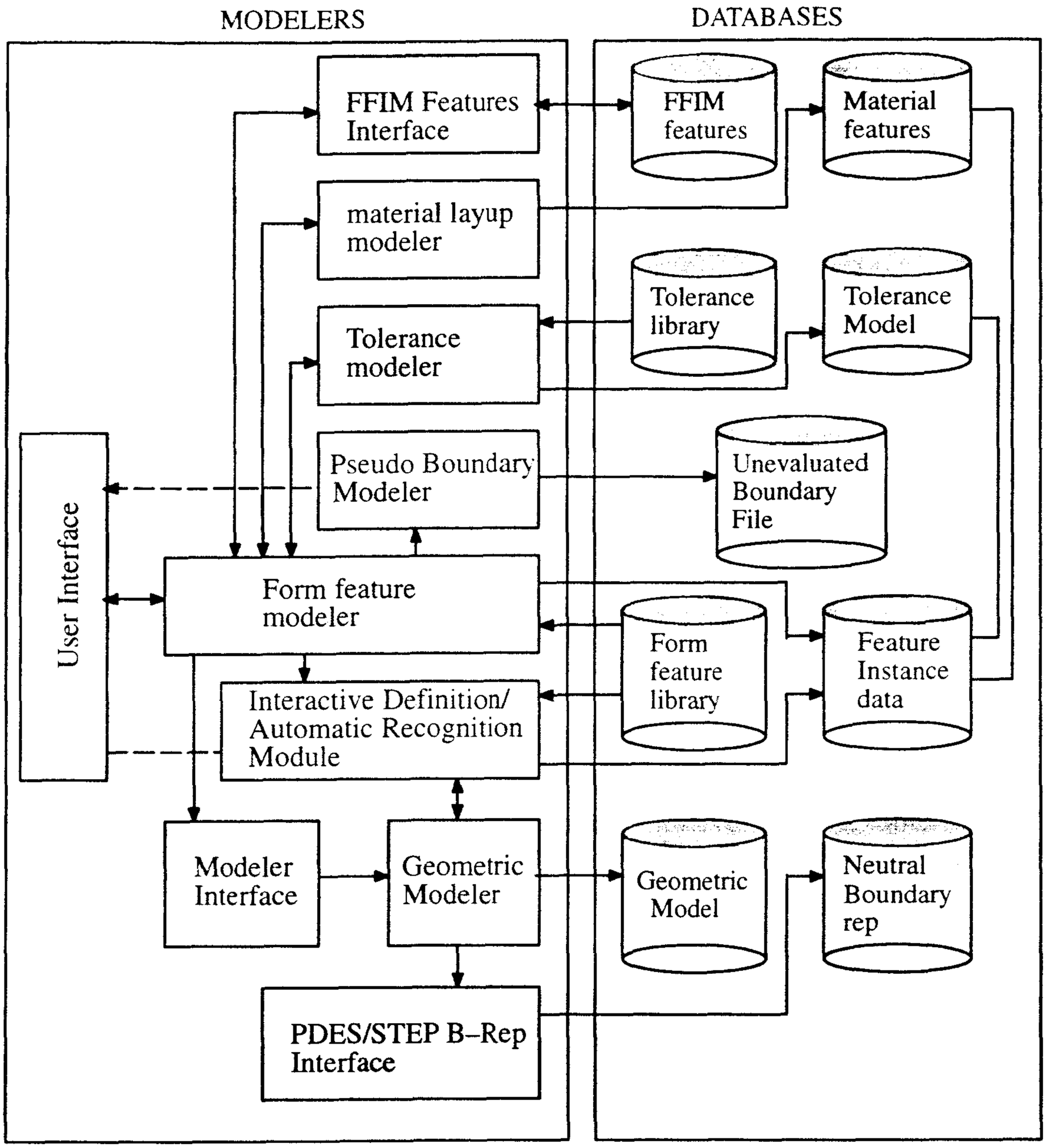


Figure 2.8: Schematic diagram of ASU Features Testbed (Shah and Mathew 1991)

2.2.8.2 FSMT

FSMT is an acronym for Feature Solid Modelling Tool, developed by Duan et. al. (1993). It consists of seven components, as shown in Figure 2.9, a feature definition and management system (FDMS), a Boolean operation processor (BOP), a geometry and

attributes consistency checker (CVS), a knowledge base (KB), a database (DB) and a user interface management system (UIMS). FSMT claims to use a generalised sweeping method that is capable of defining all kinds of parametric features. A designer can build up his/her own feature library dedicated to a particular application. The system has the ability to solve the problem of mapping from feature definitions into Finite Element Mesh generation, process planning and NC programming, and has also been used for tolerance analysis and synthesis (Huang et. al. 1993).

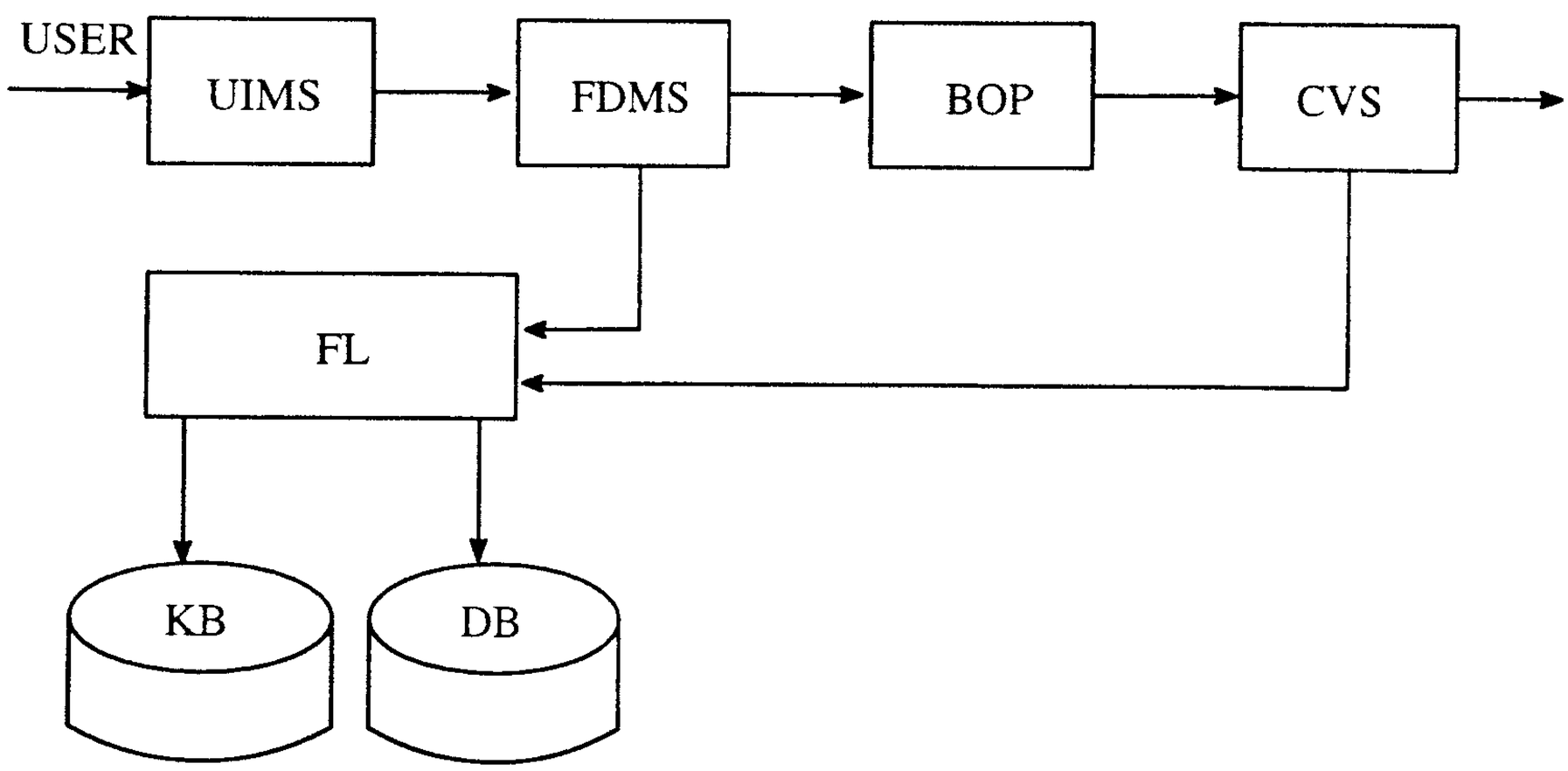


Figure 2.9: Architecture of FSMT (Duan et. al. 1993)

2.2.8.3 LUT–FBDS

LUT–FBDS stands for Loughborough University of Technology–Feature–Based Design System. LUT–FBDS is a prototype feature–based design system which was developed in relation to research on process capability modelling for design and selection of processing equipment (Case et. al. 1993, Case 1994). The structure of the system is shown in Figure 2.10. The system consists of a design by feature user interface to a solid modeller (PAFEC Imaginer), a feature processor and a geometric reasoner. The design by features interface allows designers to create features by evaluating sets of parameters for feature primitives; to perform feature edit operations such as move, rotate, copy and

delete; and to define feature relationships such as parent–child relationships and tolerances between features.

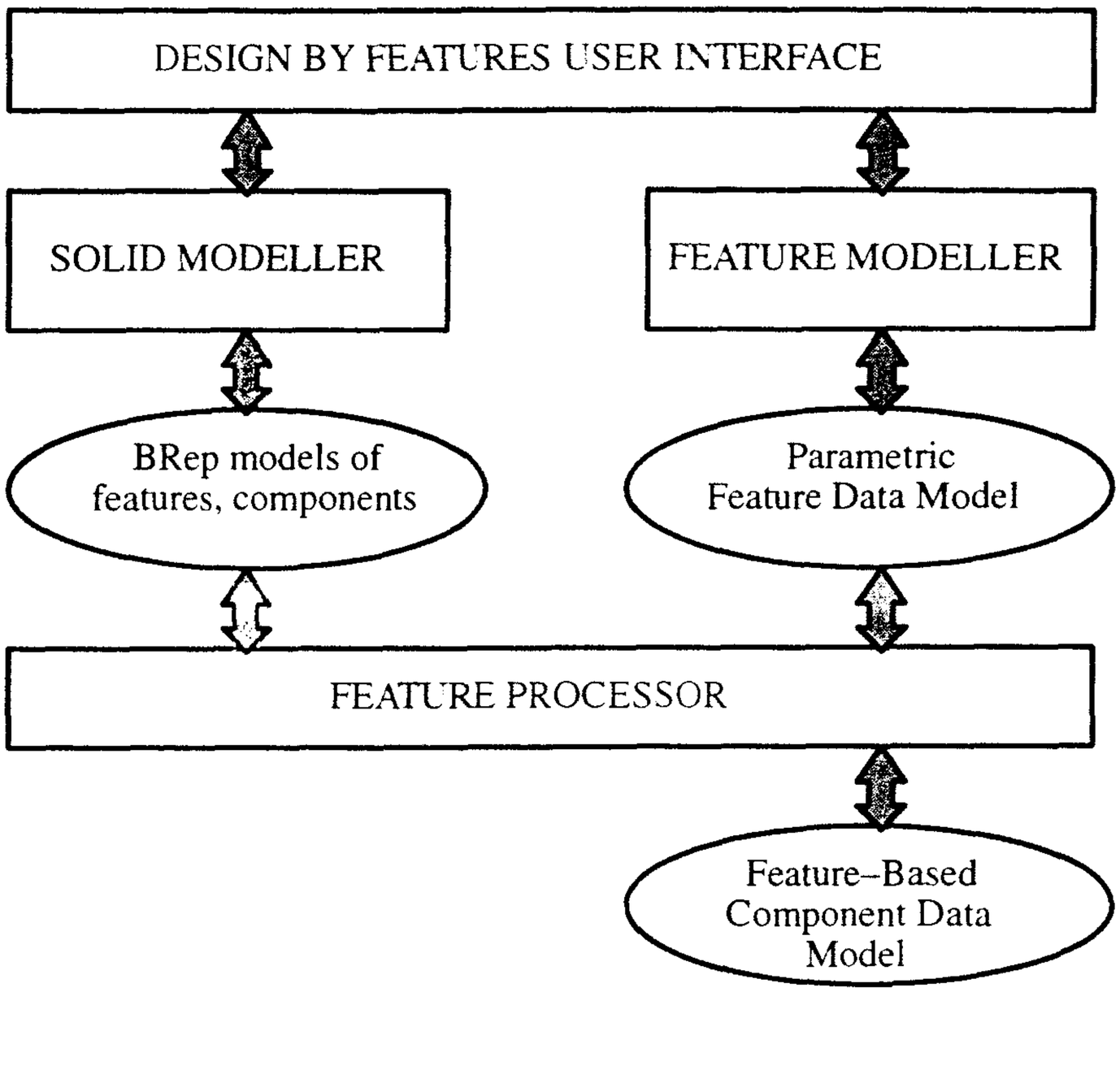


Figure 2.10: Structure of LUT–FBDS (Case et. al. 1993)

Once a feature is created through the design by features interface, a B–Rep model is generated for the feature and stored in the Imaginer database. At the same time, information about the feature such as its dimensional and positional parameters, tolerances, surface finish and relationships with other features are stored in the Parametric Feature Data Model, which is then processed by the feature processor/geometric reasoner to generate a detailed and well structured data model, known as the Feature–Based Component Data Model (FBCDM). The feature processor contains the functions to create the data structure and to calculate implicit data such as access directions, imaginary face information and parent–child relationships at the face level. The feature processor also contains functions for communication between the

FBCDM and the design interface. LUT–FBDS is provided with a feature validation mechanism which detects exceptional situations and the consequent changes in the class, dimensions and relationships of all the affected features.

The review of various feature–based design systems indicates that there is still no general purpose system existing which is flexible and can be adapted to many types of application. Most of the systems also suffer from the problem of a limited number of features and simple shapes of features for part construction (Chen et. al. 1991). Efforts are required to produce a generic geometric representation method that will satisfy the diverse requirements of different applications. One solution suggested by Chen et. al. is to allow designers to create their own "user defined features" for the construction of complex parts. Attempts have been made to provide a system which allows user defined features in a feature–based modelling system (Dong and Wozny 1991). However, such features may bring undesirable consequences such as the inability of the system to support manipulation or validation of features. User–defined features may be unintelligible to downstream applications and could destroy or alter pre–existing features and the system will be unable to detect such changes or to react suitably.

2.3 ASSEMBLY MODELLING

Assembly modelling has been the subject of research in many areas such as kinematics, AI, robotics and geometric modelling. Assembly is defined as the process of creating a connection between components or sub–assemblies to form complex end products (Wang and Li 1991). To model assembly properly, it is important to represent the nature and structure of dependencies between parts in an assembly. As mentioned in Chapter 1, the modelling representation of relationships and mating conditions are the distinguishing characteristics between modelling individual parts and assemblies and consequently between geometric modellers and assembly modellers. Assembly modellers can be thought of as more advanced geometric modellers where the data structure is extended to allow representation and manipulation of hierarchical relationships and mating conditions. Figure 2.11 depicts the role of a geometric modeller as a preprocessor to the assembly modeller in the creation of an assembly model. A link is

established between geometric and assembly modellers such that designers need only to modify individual parts for design modifications using the geometric modeller, and the assembly model is updated automatically (Zeid 1991).

There are three requirements for assembly modelling: the modelling of individual parts, specifying the hierarchical relationships between parts in the assembly and specifying the mating conditions between parts or specifying the locations and orientations of the parts in their assembled positions (Zeid 1991). These requirements are discussed in the following sections with the emphasis on the research work done in those areas.

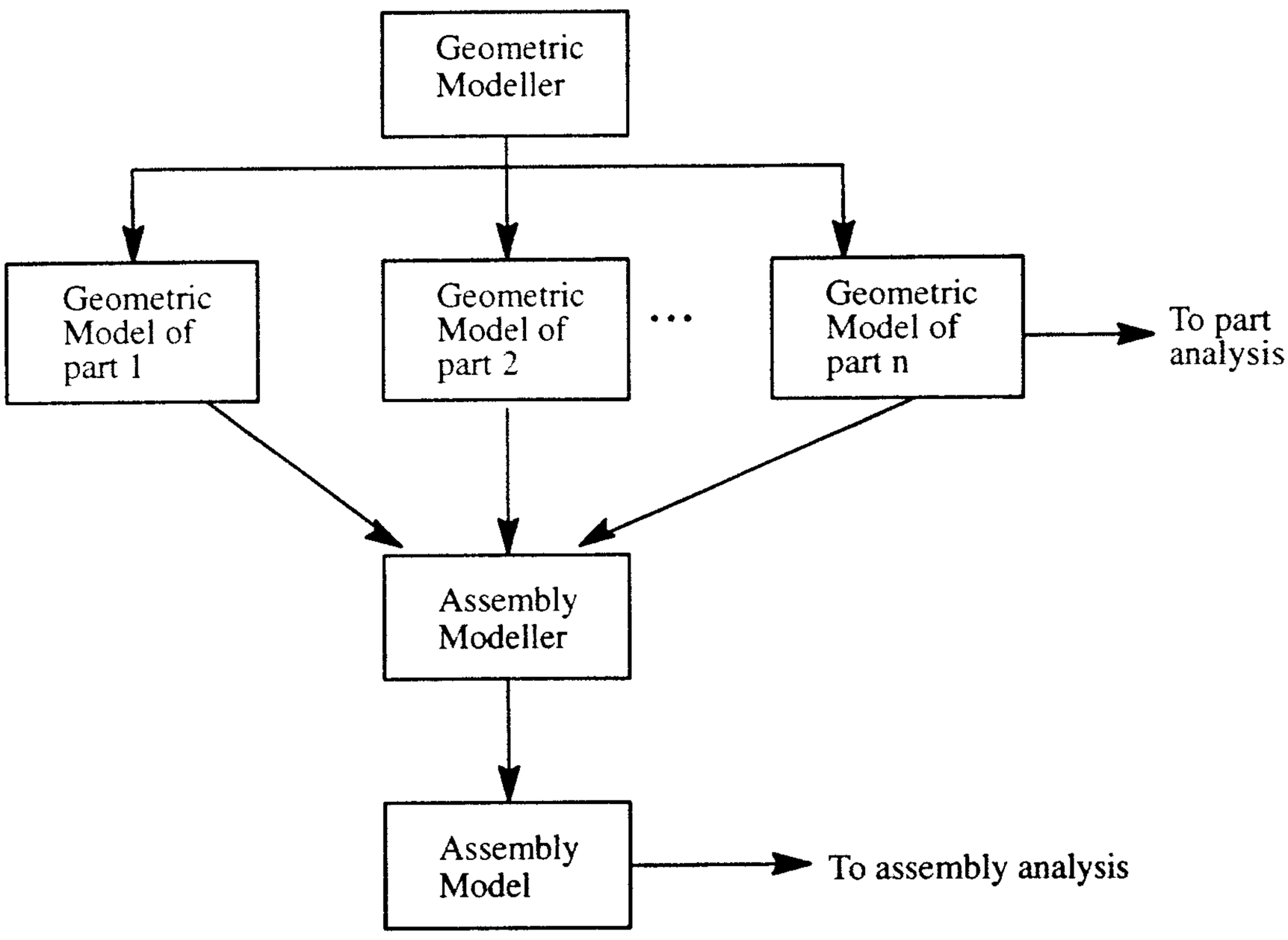


Figure 2.11: Generation of an assembly model (Zeid 1991)

2.3.1 MODELLING OF PARTS

This is the first step in creating an assembly model. Individual parts can be created using a geometric modeller with a proper representation scheme. In most assembly operations, specific features of objects dictate how these objects may be assembled together. Solid modelling, especially B–Rep schemes, have been used for this purpose because the

mating conditions are related to the faces, edges and vertices of the assembled parts (Zeid 1991). However, features are seen as a more natural method of representing the assembly of parts by capturing assembly mating information and enhancing the assembly design environment (Shah and Tadepalli 1992). Features contain information relating to the position and dimensions required to define the geometry and information pertaining to how features of a single component or assembly are positioned with respect to each other. The feature may be defined based on either shape or connectivity. The latter is used for representing a mating position in an assembly operation. Due to these factors, features have been used in much recent assembly modelling research work (Wang and Ozsoy 1990, Giacometti and Chang 1990, Shah and Rogers 1993, Molloy et. al. 1993).

2.3.2 ASSEMBLY STRUCTURE AND MATING RELATIONSHIPS

An assembly database stores the geometric models of individual parts, the spatial positions and orientations of the parts in the assembly and the assembly or attachment relationships between parts (Zeid 1991). Some representation schemes have been developed, but the inherent problem that all these structures are attempting to solve is how to assign assembly data interactively to build or develop the assembly. The main difference among these schemes stems from the way the user provides the assembly data, that is the locations and orientations of the various parts and their hierarchical relationships. Some of the representations of assembly and mating relationships are discussed below.

Most of the assembly systems are represented by a hierarchical structure. Wesley et. al. (1980) created a comprehensive engineering database to allow representation of objects and their inter–relationships. A graph–based structure was used to model assemblies where components and assemblies (parts, sub–parts and assembly) are represented by nodes interconnected through corresponding edges that represent relations among components. Four types of relationships are defined – "part–of", "attachment", "constraint" and "assembly". The nodes also store positional relationships between objects and material properties. The above relationships between parts and subassemblies in the data structure are modelled using a world model. A program called

AUTOPASS was developed using a world model which represents the above relationships between parts and subassemblies in the data structure. The model does not provide an interactive user interface and requires the transformation matrix of each component as an input to constrain the location and orientation of each component in an assembly.

Sekiguchi et. al. (1983) divides the relationships between parts in an assembly into two main groups – "fit", which implies a pair of external and internal cylindrical surfaces and "contact" between two planar surfaces. These are classified into the relative degree of difficulty of assembly, which is determined by the combination of the degrees of freedom of motion and the required force to change the relative position of parts in assembly and/or disassembly, as shown in Figure 2.12. For instance, push fit is ranked lower than pressure fit, as the former is less difficult than the latter to disassemble. A connective matrix is built for each assembly direction (x, y, z) and for each type of relationship. From this, the rules which govern the assembly sequence is determined.

Connective relations		Code
Fit	Pressure fit	Pr
	Push fit	Pu
	Screw fit	Sc
	Taper fit	Ta
	Spline fit	Sp
	Position fit	Po
	Movable fit	Mo
	Gear coupling	Ge
	Ring fit	Ri
	Key fit	Ke
Contact	Clamp contact	Cl
	Taper contact	Ta
	Plane Contact	Pl
	Gear meshing	Ge
	Gap plane	Ga

Figure 2.12: Connective relations (Sekiguchi et. al. 1983)

To avoid the problem of using a transformation matrix, Lee and Gossard (1985) proposed a hierarchical tree structure as shown in Figure 2.13. This contains basic information such as mating features between the components, plus the concept of virtual links that are introduced to connect pairs of mating components or sub–assemblies.

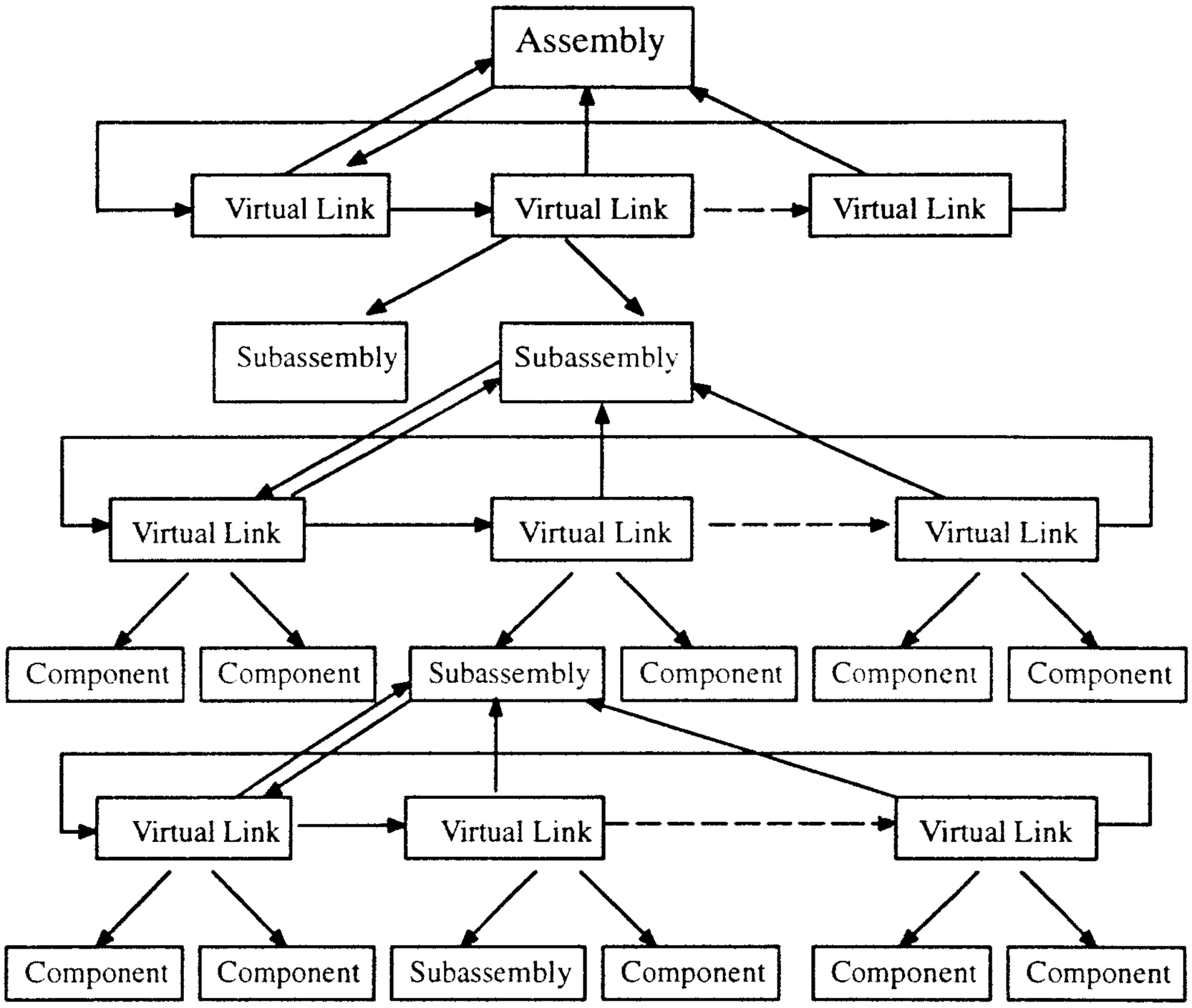


Figure 2.13: Assembly tree structure (Lee and Gossard 1985)

A virtual link is defined as the complete set of information required to describe the type of attachment and the mating condition between a mating pair. Mating features are used to describe the mating information in detail. Two mating conditions of "against" and "fits" are used to describe the mating relationships between mating features. The "against" condition holds between planar faces of a pair of components. The "fits" condition holds between centrelines of a solid cylinder and a hole. Any mating pair of two subassemblies, two parts or one subassembly and one part is connected by one virtual

link. The transformation matrices are derived automatically from the associations contained in the virtual links (Lee and Andrews 1985).

The mating conditions defined by Lee and Gossard have been used successfully by Rocheleau and Lee (1987) to establish the relationships between components and compute the location and orientation of the component. Although the two mating conditions can accommodate a wide range of possible assemblies, they proposed other mating conditions to enable thread and gear conditions and other special cases to be represented. Kim and Lee (1989) extended the use of the model for dynamic and kinematic analysis of assembly components.

The virtual linked assembly structure cannot explicitly describe the natural structure of an assembly and cannot provide enough mating information to support subassembly instances. Ko and Lee (1987) further developed the ideas of Lee and Gossard (1985) by representing an assembly in a hierarchical tree. An assembly is divided into several subassemblies and each subassembly is divided into several groupings, which are further composed of several components. Any two components are in different subassemblies if the components have relative motion with respect to each other and any two components in a subassembly are in different groupings if the component do not mate directly. Two additional mating conditions are proposed – "tight–fit" and "contact". The "tight–fit" condition is a "fit" condition whereby the rotational movement is constrained. A "contact" condition is introduced to prevent any movement in the "against" condition. The approach is used to generate an assembly plan.

The idea of representing assembly as an assembly graph was further consolidated by Wang and Ozsoy (1990). In this graph, shown in Figure 2.14, the assembly, its sub–assemblies and components are hierarchically structured as the topmost, intermediate and terminal nodes respectively. The concept of instance is introduced to accommodate more than one occurrence of a component or a subassembly at different locations with different orientations in an assembly. The connectivity information between the elements of an assembly is made available through the instances instead of through the components or subassemblies. The mating condition of "against" and "fit are

used with an additional mating condition of "parallel". "Parallel" constrains two planar faces to have a specified separation distance and to have their surface normals pointing in the same direction. The mating information between a pair of mating entities is stored as a set of mating links.

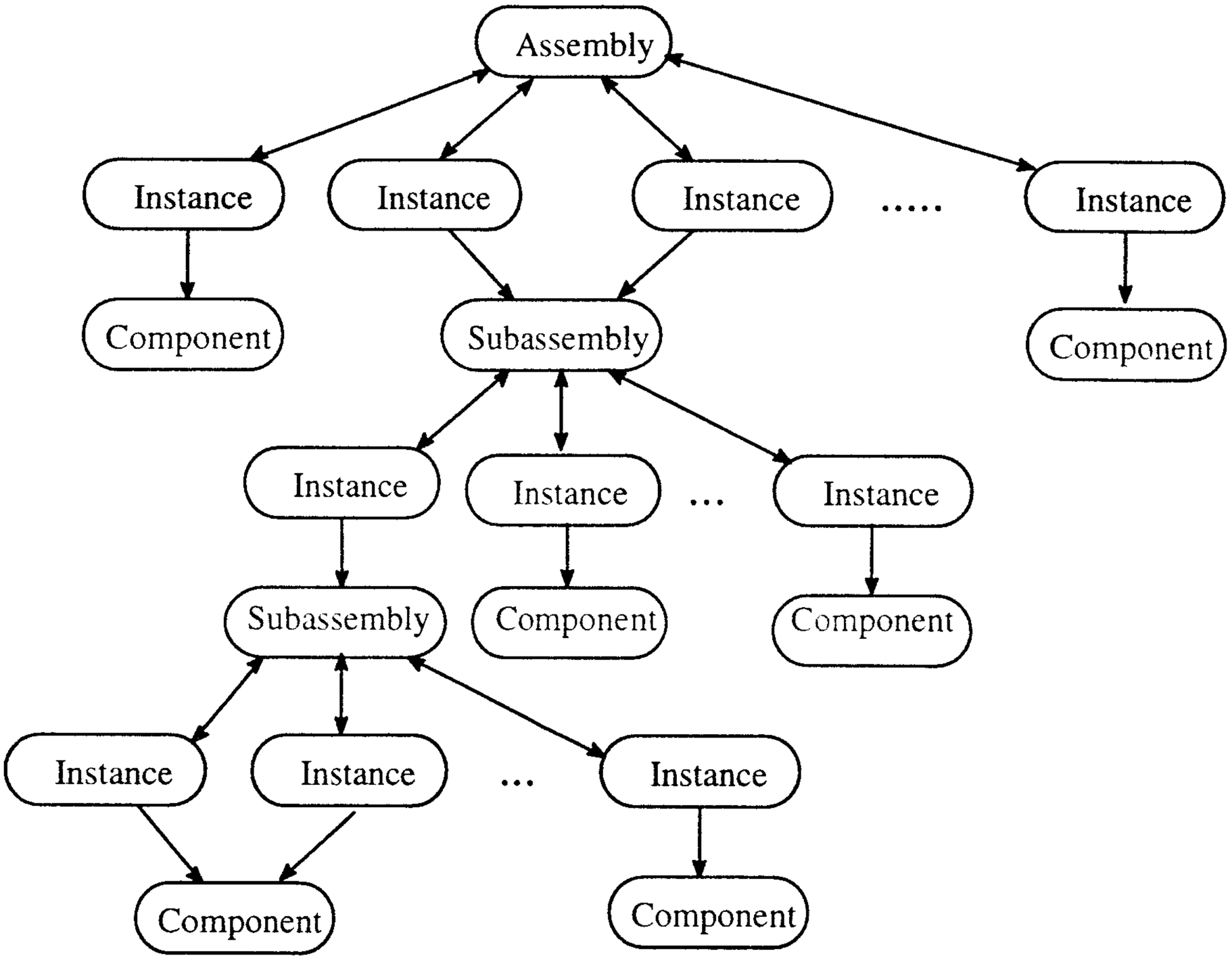


Figure 2.14: Assembly Graph (Wang and Ozsoy 1990)

Mating links are created and linked together according to the user specified mating conditions. The position and orientation of an instance of the assembly is derived from the mating conditions carried by the mating links of that instance. The detailed mating information about where and how the mating happens is provided by mating conditions and mating features. Mating features contain the specific geometrical information referred to by mating conditions. For instance, if the mating condition is against, the two

mating features will be two planar faces and if the mating condition is fit, the two mating features will be two cylindrical faces.

Huang et. al. (1993) also used the mating conditions of "against" and "fits" and developed a technique that allows a designer to interactively create an assembly of components by specifying the mating conditions and/or the relative location and orientation among the individual components in a feature–based system. A dimension/tolerance chain is then created automatically. Baxter et. al. (1992) proposed an extension of the mating conditions proposed by Lee and Gossard (1985), by writing rigourous definitions for a set of mating conditions, including the degrees of freedom that they constrain.

Shah and Rogers (1993) distinguished between the representation of assembly and the derivation of assembly relationships. They claim that the hierarchical structures used in most assembly representation research can only model "part–of" relationships. To fully model assembly, many other types of relationship need to be included. To achieve this aim, five types of relationships between subassemblies are defined: "part–of", "structuring relations" (SR), "degree of freedoms" (dof), "motion limits" and "size constraints" applied to dimensions. The assembly structure consists of low–level geometric entities (axes, faces) to high–level subassemblies, as shown in Figure 2.15. Sub–assemblies consist of parts, and parts can be thought of as an assembly of form features. Form features are composed of simple volumes combined together by Boolean operations and feature volumes are defined by boundary entities. An assembly may consist of several sub–assemblies, which themselves may consist of several units, either a part or a sub–assembly.

The work done at the University of Leeds on the development of a product data framework considers assembly as lists of parts without reference to physical or functional connectivity (Henson et. al. 1993). The framework considers product, assemblies, components and features each of which have their own set of entity attributes. A product description may either be a component or an assembly. An assembly description may be implemented as a list of parts where a part is either a component or an assembly. This

representation is only suitable for a limited application and is not sufficient to support applications which require information about the relationships between components, such as tolerance analysis and design for assembly.

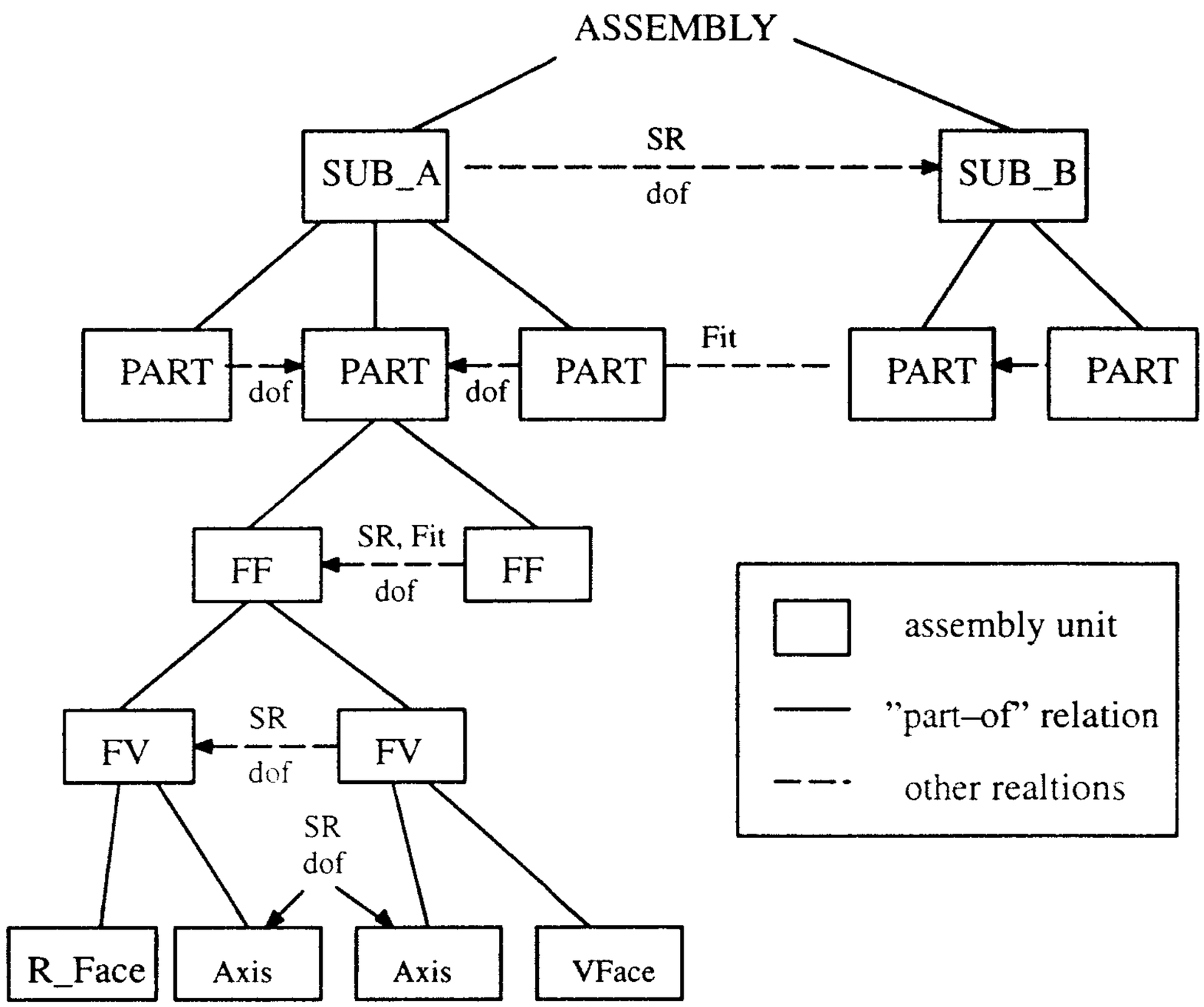


Figure 2.15: Assembly entities and structure (Shah and Rogers 1993)

Other forms of graphs have also been used to represent assembly and mating conditions. Roy and Liu (1988) proposed a semantic association model (SAM)–based assembly database. The components of the assembly are represented by a feature–based structural face adjacency graph. The required mating conditions between features of different components in an assembly are defined by a functional relationship graph. The assembly database could support further functional analysis such as assembly evaluation and tolerance analysis. However this is only a conceptual model. Sturges and Kilani (1992) use a component graph to describe the mating conditions between the features in a subassembly. Nodes in the graph represent either individual components, subassemblies

or void regions while links represent the mating conditions between the mating nodes. Mating consists of the faces that are shared by the two nodes concerned. Porchet and Zhang (1993) model an assembly by specifying parts, connection types between parts and the number of restricted degrees of freedom by a connection in a graph. The graph is used to determine tolerances of functional parts in a product at the assembly level.

Mantyla (1990) proposed hierarchical part–of–graphs that support relationships between components in multiple views. An object–oriented (multi–tree) data structure was designed and certain consistency rules for the views were imposed. An assembly design system was developed that supports top–down design, multiple levels of detail, feature–based design of components and limited constraint–based geometric relationship maintenance. The system is intended to support applications such as process and assembly planning. However the system can only represent 2D geometry.

The importance of Dimensioning and Tolerancing (D&T) information to support the assembly application has been recognised and many researchers have proposed data structures which include this information (Roy and Liu 1988, Sodhi and Turner 1991). An important aspect of tolerance design is to establish the functional relationship between parts. However, many tolerance design techniques attached to features are conceptual in nature and are not practical for application (Porchet and Zhang 1993). The inclusion of D&T knowledge in features is an extensive area of research and this is not considered in this research.

The review on assembly modelling highlights two important aspects – the structure of the assembly and how relationships among parts in an assembly are defined. A hierarchical assembly structure has been widely adopted as it is the most natural way of representing assemblies. The structure represents the way in which an assembly is actually modelled by the designer. It also suits well with the way features are structured. This structure is adopted in this thesis, as described in Chapter 6. The above discussion also points out some common approaches in defining relationships among assembled parts. Two common types of mating relationship which involve contact between two planar surfaces and contact between a hole and a shaft have been identified by most of the researchers.

The differences are only in the naming of the relationships and the level of detail in describing the relationships among the parts. The established relationships are adopted in the feature knowledge in this research.

2.4 SUMMARY

In this chapter, several issues pertaining to the application of feature–based technology in manufacturing and assembly modelling of mechanical parts have been discussed. Although there are criticisms on the use of features in design and manufacturing (Gui and Mantyla 1994), features are seen to have the most potential in representing manufacturing knowledge efficiently. The discussion on features emphasises the need for a definition and taxonomy which can be used in multiple applications. To achieve this, features should be defined to fully incorporate the knowledge of the application domains and be supported by a well defined taxonomy. A design by features approach is preferable to the feature recognition method due to the possibility of considering manufacturing and assembly concerns early in the design process and the advantage of storing relevant information for the application, which is not possible in the latter approach.

A single feature representation would be useful across many applications and eliminate the need for feature mapping. Since process planning and assembly modelling are frequently feature–based, it is most appropriate to use a feature–based model as the internal data representation for both of these applications. Assembly modelling is seen as a very important activity in the design process as the output from assembly modelling can be used in various applications such as Design For Assembly (DFA) and assembly planning. Several approaches in representing assembly models have been discussed.

Research in the application of features for assembly modelling is still lacking compared to process planning. From this review, it is evident that little work has been done on integrating feature definitions to cover the two major activities of process planning and assembly. This research attempts to look into this problem and propose a means of representing features that are applicable for both applications. To achieve this aim, a feature–based design approach is seen as most appropriate to represent a part while an

object–oriented approach is deemed most suitable to represent the feature knowledge. An object–oriented technique is reviewed and discussed in the next chapter while Chapter 4 describes how features are defined using this approach.

CHAPTER THREE
OBJECT–ORIENTED TECHNIQUES

3.1 INTRODUCTION

"Object–oriented (OO) models are recognised as being useful for understanding problems, communicating with application experts, modelling enterprises, preparing documentation and designing programs and databases" (Rumbaugh et. al. 1991). This statement highlights the capability and the importance of the OO technique which is now widely used in the development of manufacturing application software. The aim of this chapter is to present an overview of the technique with a particular emphasis on OO programming. OO concepts are explained as a basis for understanding the development of the feature and assembly models and the prototype feature–based assembly modelling system described in the following chapters. The chapter also gives an overview of the C++ programming language and a solid modeller kernel, ACIS®. These tools represent the current state of the art in developing application systems and thus a significant amount of time has been spent in studying and applying them to the problem of creating a feature–based assembly modelling system. Section 3.2 gives an overview of the OO programming concepts and Section 3.3 outlines the benefits of using this approach. Section 3.4 describes the approach to OO design used in this research work including the notation used in representing objects and their relationships. The C++ programming language is described in Section 3.5 and Section 3.6 gives a general description of the ACIS® solid modeller kernel and how it is used in this research. The description of the main features of these tools will help in the understanding of the research work presented in the following chapters.

3.2 OBJECT–ORIENTED PROGRAMMING CONCEPTS

Approaches to programming have changed dramatically since the invention of computers to accommodate the increasing complexity of programs and the development of hardware. The language development process has passed through various phases,

moving from binary machine–code instructions through low–level assembly language to high–level languages such as Pascal, Fortran and C. However these programming languages do possess sufficiently powerful enough abstractions required by large and complex software systems (Zhou et. al. 1994). The structured programming approach was developed in the 60's in an attempt to solve this problem. Structured programming is an approach which divides systems into functional modules, so that each module is highly cohesive. Communication between modules is strictly controlled, thus allowing the program to be debugged more easily (Zhou et. al. 1994). The approach eases the organisation and control of the software development task, but it remains difficult to control a project once it reaches a certain size. There is a problem of maintenance, extension and integration of the system developed, and OO programming was introduced to address these problems.

OO development emphasises a number of essential concepts and principles which provide guidance for the construction of programs based on the ideas of objects, classes and class relationships. Some of the important principles involved in OO programming are outlined in the following paragraphs (Rumbaugh et. al. 1991, Korah 1994). Sections 3.5 and 3.6 provide examples of these principles applicable to the C++ language and the ACIS modeller.

3.2.1 ENCAPSULATION

The term object–oriented means that software is organised as a collection of discrete objects that incorporate both data structure and behaviour (Rumbaugh et. al. 1991). The concept of an *object* is the central feature of OO programming. An object is a self–contained software entity that consists of both *data* and program code (*procedures*) to fulfil the required functions which manipulate the data. Data is information or space in a program where information can be stored, such as a name or a dimension. Procedures or *methods* are parts of a program that cause the computer to actually do something, such as display the output, perform calculations or store information on a disk. In traditional programming, code (sequences of computer instructions) and data have been kept apart. In OO methods, code and data are merged into single indivisible entity. Within an object,

some data and/or methods may be private to the object and inaccessible to anything outside the object. In this way, an object provides a significant level of protection against other unrelated parts of the program. This relationship between data and function in an object is referred to as *encapsulation* or *information hiding*. Encapsulation supports the separation of the specification of the component from its implementation. It offers two kinds of protection – it protects an object's internal state from being corrupted by the program that uses it (the *client* program), which in turns protects the client's code from changes in the object's implementation. An object does not tell the outside world how it does an operation. This prevents a program from becoming too interdependent and eases the problem of maintenance (Korah 1994).

Mitchell (1993) outlines the situations where the use of objects is necessary. Objects should be used in the following situations:

i. To represent real world concepts such as animals, cars, features.

ii. To represent well–known data structures or algorithms. For example, a feature is a linked list of faces and a component is a linked list of features. Thus features and components are objects.

iii. To encapsulate design decisions which are difficult to make or involve machine dependencies. For example, a mouse, a keyboard and a screen of a computer are machine dependent and thus can be represented as objects.

iv. To hide complexity to the end user, for example to handle certain types of curves such as Bezier curves.

v. To create a more convenient OO interface to existing libraries, such as a window object which provides an interface to the text handling and graphics library supplied with the compiler.

Each object is defined by a *class* declaration. The class is a collection of objects sharing the same set of characteristics (data format) and functionality. Classes and objects are closely related concepts. Every object is a specific instance of a class and the class

definition ensures that all objects of that class will have the same structure and behaviour. For example, in the development of a feature-based assembly modelling system, a component class is developed to represent any component that makes up the assembly. An object called a block can be created to represent an instance of the class Component. Figure 3.1 illustrates the concept of class, objects and how they interact in an OO program.

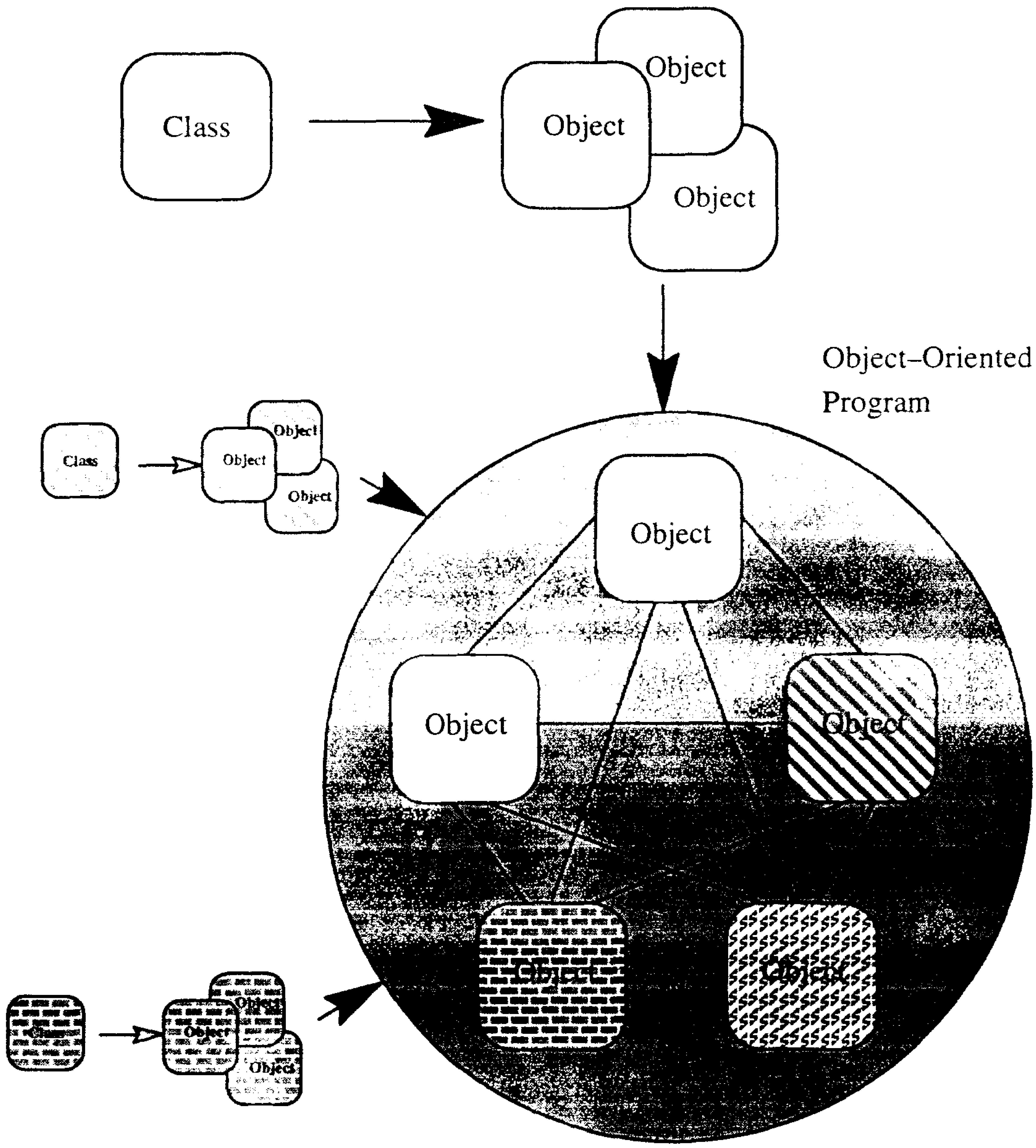


Figure 3.1: Class, Object and OO Program

3.2.2 POLYMORPHISM

Polymorphism in the context of OO is the ability to hide many different implementations behind a single interface or the ability to issue the same command to different objects. This means that the same message can be understood differently by objects of different classes and therefore produce a different, but appropriate result. It allows a function name to be shared up and down a class hierarchy. The client code can invoke an object's operation without knowing its type and if the implementation of the object's operation changes, the client code is not affected. For example, the message *DrawFeature* sent to a boss feature will result in a boss being drawn. When the same message is sent to a hole or slot, it would result in a hole or slot being drawn. This is referred to as *dynamic binding*, as it is the establishment at run time of an association between a method call and the code executed.

Polymorphism also allows *function overloading* and *operator overloading*. By function overloading, it is possible to define different functions with the same name, each processing different data. For example, two functions of the same name could be written, one to move the feature using cartesian coordinates and another to execute the same function using polar coordinates. The arguments and return type of the function determine which function is used. Similarly, with operator overloading, mathematical operators, such as +, – and /, can be defined to operate on various data types including objects.

3.2.3 INHERITANCE

One useful property of OO methods is that a class produced for one program may be usable in a new program by a slight modification and this can be achieved by defining a new class which inherits the properties of the existing class.

Inheritance is the property that allows the building of objects from other objects or creation of new classes by extending and adapting old classes, based on hierarchical relationships. The class from which one inherits is called a *base class* (parent class) and the class which does the inheriting is called a *derived class* (child class). A derived class

may inherit all of the data formats and methods from its parent class but it has the opportunity to change anything it inherits by adding new data and/or methods or redefining inherited methods. In the last case, a method declared in a base class may have several definitions since it may be redefined in multiple derived classes. When the method is called to perform an operation on an object, the definition actually used is determined at execution time based on the class of the object, as explained in Section 3.2.2. A base class may have multiple derived classes and a derived class may in turn serve as the base class for other derived classes, producing a tree–structured organisation of classes as shown in Figure 3.2. In the example, point and shape are derived classes of the drawing object class. The shape class in return is a base class for polygon and circle classes. The triangle and quadrilateral classes are derived classes of a polygon class and thus can inherit all data and methods from the classes higher up in the hierarchy.

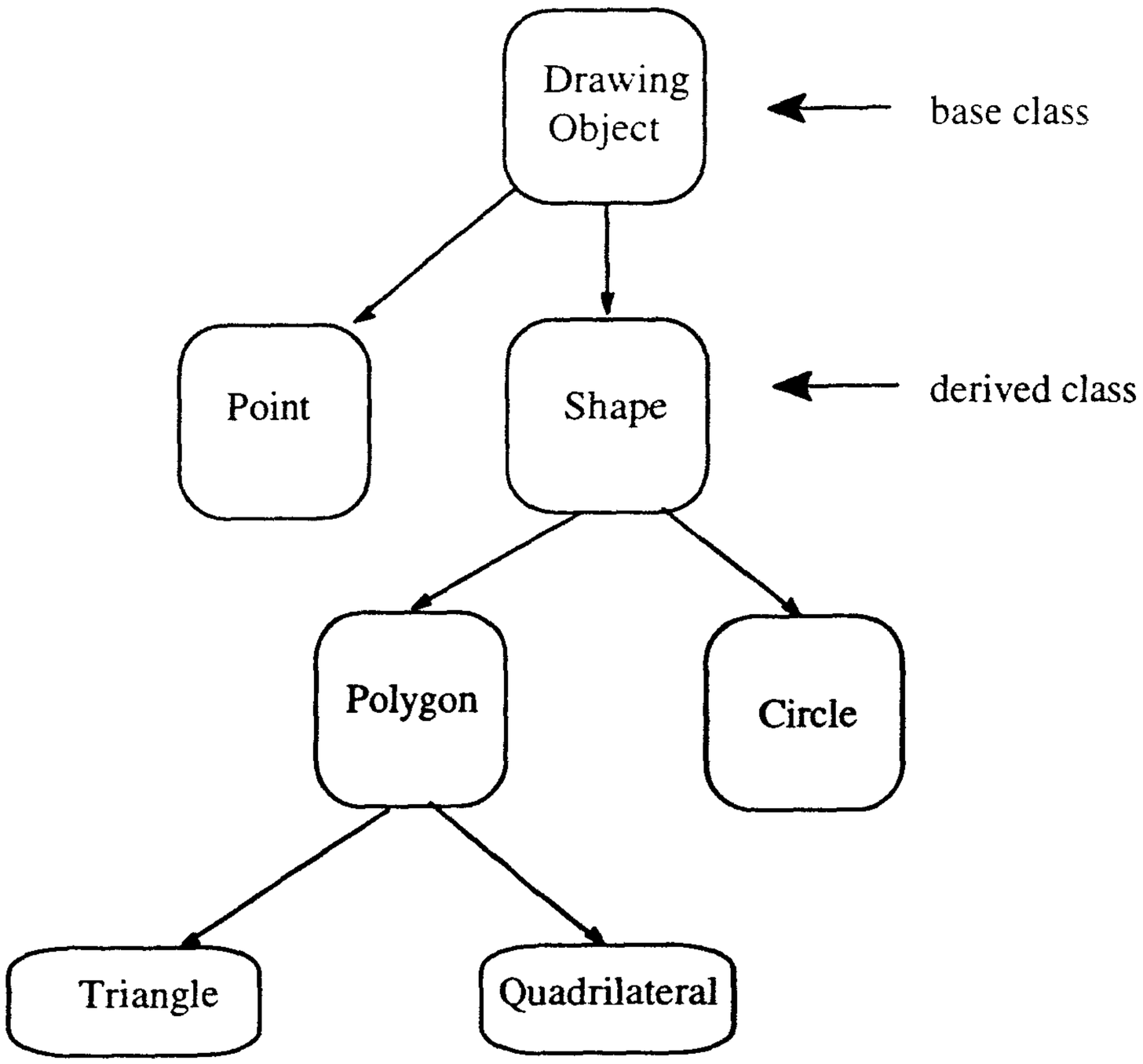


Figure 3.2: Class hierarchy for shape (Gorlen 1987)

In an OO design process, inheritance is used in the following situations (Mitchell 1993):

i. If one class is just an extension of another, then it should inherit the parent class, for example a rectangle inherits a line and an arc inherits an ellipse.

ii. If various classes have many member data or methods in common, but with some differences, it is worth creating a base class containing these common members and let other classes inherit them. For example, a line and a circle share the x, y coordinates.

iii. If a group of objects share a complex algorithm, then a base object may be appropriate.

iv. If a class is a specific example of a general case, create a base class for the general case and let the specific class inherit it. For example, a feature class described in Chapter 4 is a general class for all feature types and profiles.

3.3 BENEFITS OF OBJECT–ORIENTED PROGRAMMING

Many development benefits can be achieved using the OO model. Most advantages come from the reusability of code, and the fact that analyses and designs are easier to achieve than with the traditional development model. For example, OO languages have built–in support for reusability through classification, inheritance, information hiding and encapsulation. The concept of class provides the benefit of reusability by providing a template which programmers can use over and over again to create many objects.

Inheritance and dynamic binding make programs easier to extend by defining classes which inherit the properties of other classes. Each time a new sub–class is added to the inheritance hierarchy, it can automatically reuse the attributes and operations defined by its base class, as explained in the previous section. Any inherited method can be redefined to perform a task more suited to the new class. This allows the customisation of existing parts and only the codes for the new features need to be written. Furthermore, a base class can be defined and only partially implemented so that it can become a generic (or abstract) class. The rest of the class is left to the specialised users to define and implement.

This idea leads to the extensive use of existing libraries of proven facilities which the programmer can use in the development of an application system. Classes and objects could become the equivalent of interchangeable, standard components, similar to selecting parts from a catalogue and snapping them together. The building of software using the concept of standard "components" or building blocks that have already been tested in many systems can improve the quality of the software and makes it easier to model complex systems. When less code is written, there is less chance of making an error and the task takes less time. This is the approach used in developing an application system using a kernel modeller such as ACIS. The OO solution also allows the user to prototype portions of a problem and add to the prototype later. The result is a fast response to changing user requirements.

By exploiting the concept of encapsulation, a programmer can change the implementation of an object and not affect any of the other objects in the system. The interface between the user program and the object is well–defined and localised in the object's class definition. A well designed implementation also hides the complexity of its operation from the user program, making objects easier to use. This leads to another benefit of modularity. Modularity makes debugging a system much easier. When the system is modular, it is easier to isolate the problem and identify it within a specific module. Programmers can change any of the object's internal algorithms without disturbing the system, but they may not alter the object's interfaces and services. This is also important during the test phases of a project because the programmer does not have to retest modules that have not been changed or reused without modification.

All the benefits of OO programming described above are relevant issues in the development of manufacturing applications, especially in the present competitive environment. For these reasons, the OO approach has been chosen in this research for describing features, the feature taxonomy, representing an assembly model and developing a prototype feature–based system.

3.4 OBJECT-ORIENTED DESIGN APPROACH

To help in the OO design process, various guidelines have been introduced and these are often referred to as OO Analysis and Design. The purpose of OO analysis (OOA) is to determine which objects need to be programmed and how these objects will interrelate. The OO design (OOD) process provides more details about the objects to be programmed, including any associated user interface objects and database architecture. A certain amount of OOA and OOD must be done before detailed coding can begin and this can be achieved by numerous different methodologies which address the analysis, design and implementation phases of the development. Among the methodologies are the *Booch method* (Booch 1991) and the *Object Modelling Technique* (OMT) (Rumbaugh et. al. 1991). Detailed discussion of the OO methodology is beyond the scope of this thesis, but an important result of OOA and OOD is one or more object models, which are presented in diagram form and graphically show the objects to be programmed and the interactions among them. The object models also show the data and methods inside each object.

The following approaches in OO design, adapted from Mitchell (1993), are adopted in this research:

i. Selection of objects. This requires determination of an appropriate class, as discussed in Section 3.2.1. Classes for features and assembly are defined in Chapters 4 and 5 respectively.

ii. Determination of the interaction between classes, that is which classes use which other classes.

iii. Determination of the relationships among classes to help in their organisation. Two classes may be related by inheritance, one class may be a client of another (use the other) or there may be no relationship. An inheritance relationship is developed if two classes meet the criteria outlined in Section 3.2.3. Otherwise, a class will be a client of another class.

iv. Design of the system which involves making a high level decision about the overall structure of the program and division of the program into separate modules.

v. Design of the contents of classes which involves determining the data and methods.

vi. Consideration of the interface between each module. This involves specifying the form of all interactions and the information flow among them.

vii. Test and develop the program.

One of the significant aspects of the methodology presented in this thesis is the use of common graphical notations for defining problems and requirements related to the representation of objects and their relationships. This helps to define an object using OO programming concepts without dealing with the complexity of the programming language syntax. The use of graphical notation allows essential information to be attached to the analysis model. The main items of information of importance to this research are the data attributes of the class, the methods and the relationships among classes, as described in the following paragraph.

Using this approach, which is based on OMT methodology, the class is represented by a rectangle divided into regions. An example of the notation for a class called Circular Feature is shown in Figure 3.3. The name of the class is given in the top region in bold. A second region is used to list data or attributes of the class. For the Circular Feature class, there are four items of data – a pointer to the body of the feature, the radius, height and the position of the feature. The third region contains a list of public members of the class (explained in Section 3.5). In this case, there are four public methods – Get Dimension, Draw, ShowRadius and Save. The objects can have physical or conceptual connections between them, and these are referred to as links. Various graphical notations are used to represent these links and Figure 3.4 shows how inheritance is represented using this approach for a set of classes. A triangle connects a base class to its derived class. The base class is connected by a line to the apex of the triangle while the derived classes are connected by lines to a horizontal bar attached to the base of the triangle.

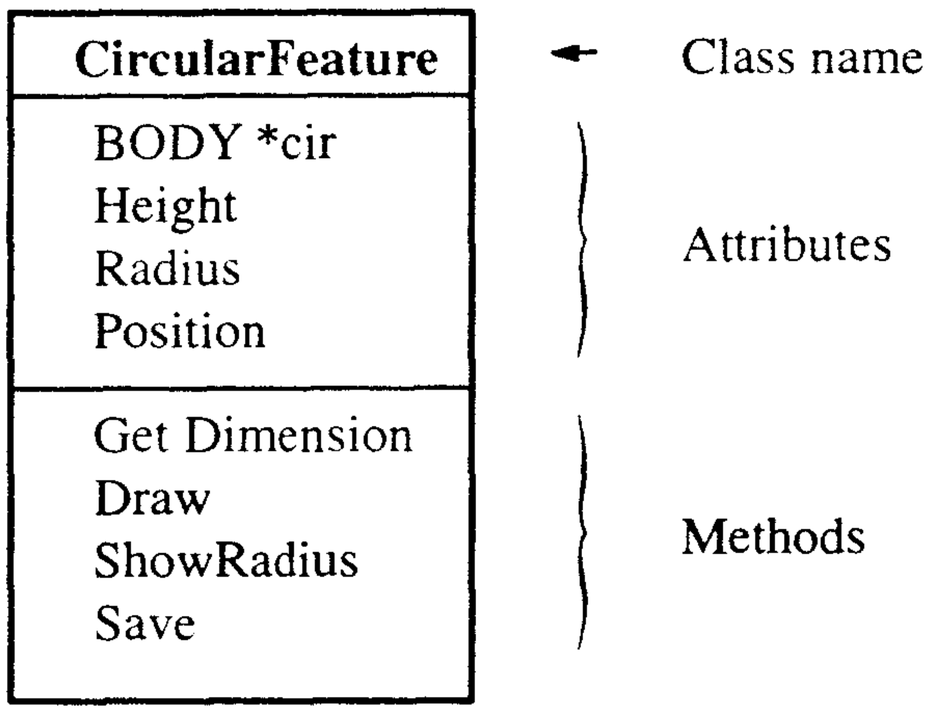


Figure 3.3: Representation of a class

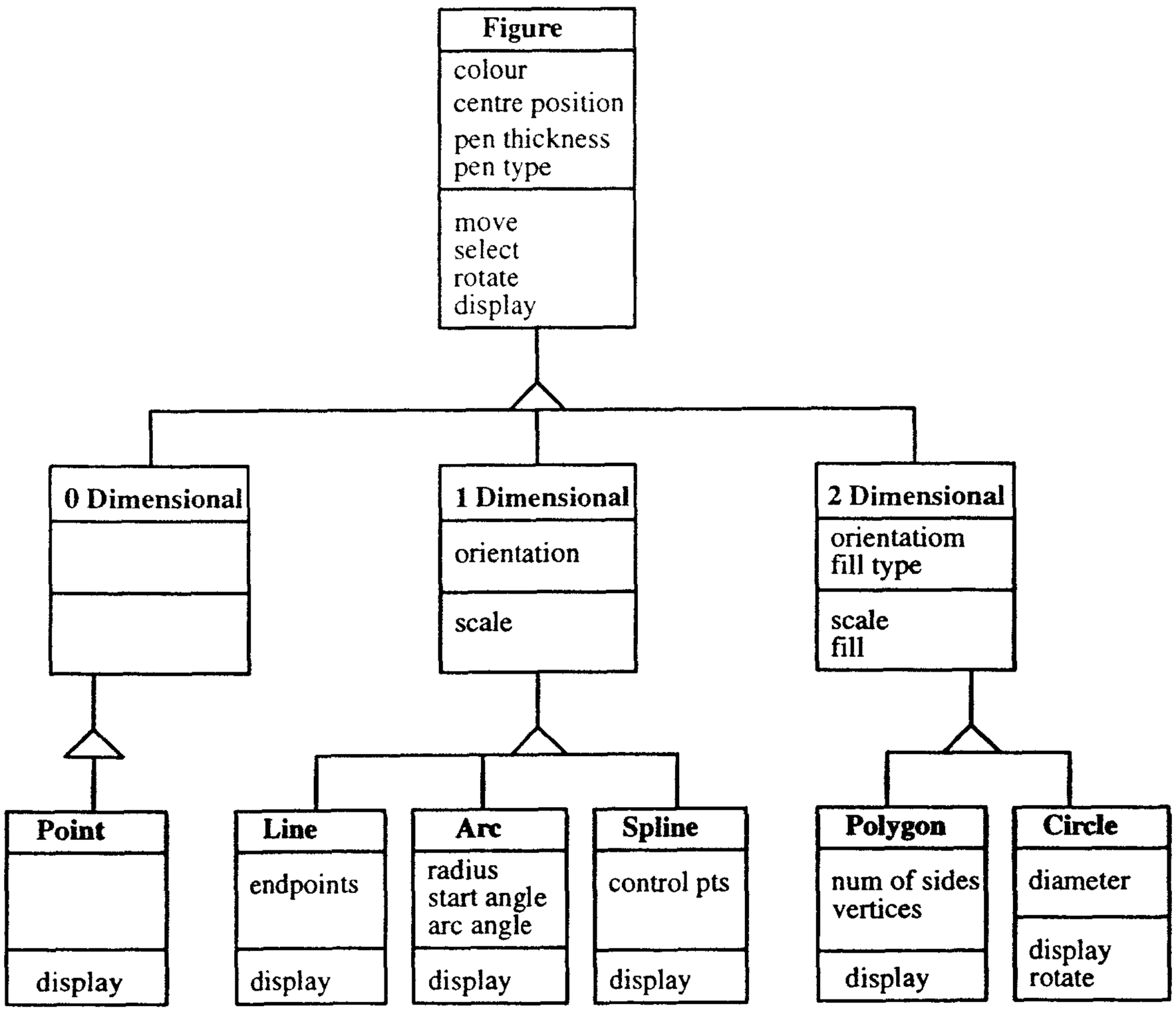


Figure 3.4: Notation for hierarchical relationships (Rumbaugh et. al. 1991)

In Figure 3.4, the dimensional classes are derived from the Figure class and the geometrical entities (point, line, arc, spline, polygon and circle) are derived from the appropriate dimensional classes.

3.5 THE C++ PROGRAMMING LANGUAGE

There are many OO programming languages currently in use, but two of the most popular are *C++* and *Smalltalk*. The C++ language is a relatively new programming language, derived and enhanced from the C language (Stroustrup 1991). It contains many improvements and features that make it a "better C" and adds support for data abstraction and OO programming techniques.

C++ supports the three key features of the OO concept, namely encapsulation, polymorphism and inheritance by means of the class declaration. A class is a data type defined by users to describe what sort of information it can represent and what sort of actions can be performed with that data. This example of the definition of a Circular Feature class illustrates these aspects:

```
class CircularFeature: public Feature {
private
        BODY *cir;
        double radius, height, pos_x, pos_y, pos_z;
public:
        CircularFeature();
        ~CircularFeature();
        void GetDimension();
        void Draw();
        void ShowRadius() {return rad;}
        void Save();
```

A data abstraction of the class is defined by the access functions. In C++ these can be *public* or *private*. A public member can be used by other functions that do not belong to the class, while a private member can be used only by other members of the class. This

class declares a pointer to the feature body, dimensions and the position with private member variables and the rest of the codes as public member functions. Since radius and height are private, they can be accessed only by member functions. A user program may read the values of radius by calling the member function ShowRadius, but may not write into these variables. This ability to combine data structure and functions in a single entity makes encapsulation cleaner and more powerful than conventional languages.

A member function with the same name as its class such as CircularFeature() is a special function called a *constructor*. A constructor function creates a new instance of its class and initialises it, and is implicitly called whenever an object of its class is declared or allocated via the C++ *new* operator. A function with the same name prefixed with the character ~ such as ~CircularFeature() is called a *destructor*. It is used to clean up memory when an object is deleted. In some classes, particularly the base class, the key word *virtual* specifies that dynamic binding is to be used for the function to which it is applied. The virtual function allows another class derived from the base class to provide alternative versions of the function.

The notation *class CircularFeature:public Feature* denotes that CircularFeature is a derived class of a Feature class. Private members of the base class are inherited, but cannot be accessed by the derived class, thus preserving the encapsulation. Public members of the base class are inherited as private members of the derived class by default, but usually they are caused to be inherited as public members by qualifying the name of the base class with the keyword public, as in the above example.

Some of the advantages of using the C++ programming language are good runtime performance, well developed supporting technology for program development in areas such as class libraries, domain–specific application skeletons and classes and OO design techniques and tools. Due to these advantages, it is claimed to become a de facto programming language for software development and is adopted as the programming language for developing the feature–based assembly model in this research.

3.6 ACIS® SOLID MODELLER

3.6.1 GENERAL DESCRIPTION

ACIS is an OO geometric modelling system designed for use as a "geometry engine" for applications that require 3–D modelling (Spatial 1993). It is a B–Rep solid modeller and provides an open architecture framework for wireframe, surface and solid modelling from a common, unified data structure. ACIS supports a wide range of geometry types and provides a set of geometric operators for the construction and manipulation of complex models. As an OO system written in C++, ACIS provides extensive facilities through a set of class libraries for application development.

Unlike other commercial solid modellers such as Unigraphics and Pro–Engineer, which are menu driven, the solid model can be built within ACIS using class libraries which can be accessed through an *Application Procedural Interface* (API) or direct–object interface to all internal objects, classes and methods, shown in the structure of ACIS in Figure 3.5. Application developers can add, derive and extend classes or access the system from any language such as C++, C, FORTRAN, PASCAL and LISP. This approach provides flexibility, especially in the development of a customised application program as developers are not tied to the proprietary rights of other software.

The functionality of ACIS can be enhanced through the development of application–specific facilities, called Kernel Extensions or Husks. Husks can be coupled to ACIS to provide additional application development support such as rendering, constraint management and feature modelling. It provides an infrastructure to allow system developers to manipulate and manage their applications development. Specific applications are then built on top of ACIS Husks and ACIS. Figure 3.6 illustrates this idea whereby a feature–based design system is developed as a husk and the assembly modelling is built on top of this husk as an application. Further applications such as assembly planning and Design For Assembly system can be developed on top of this and other husks.

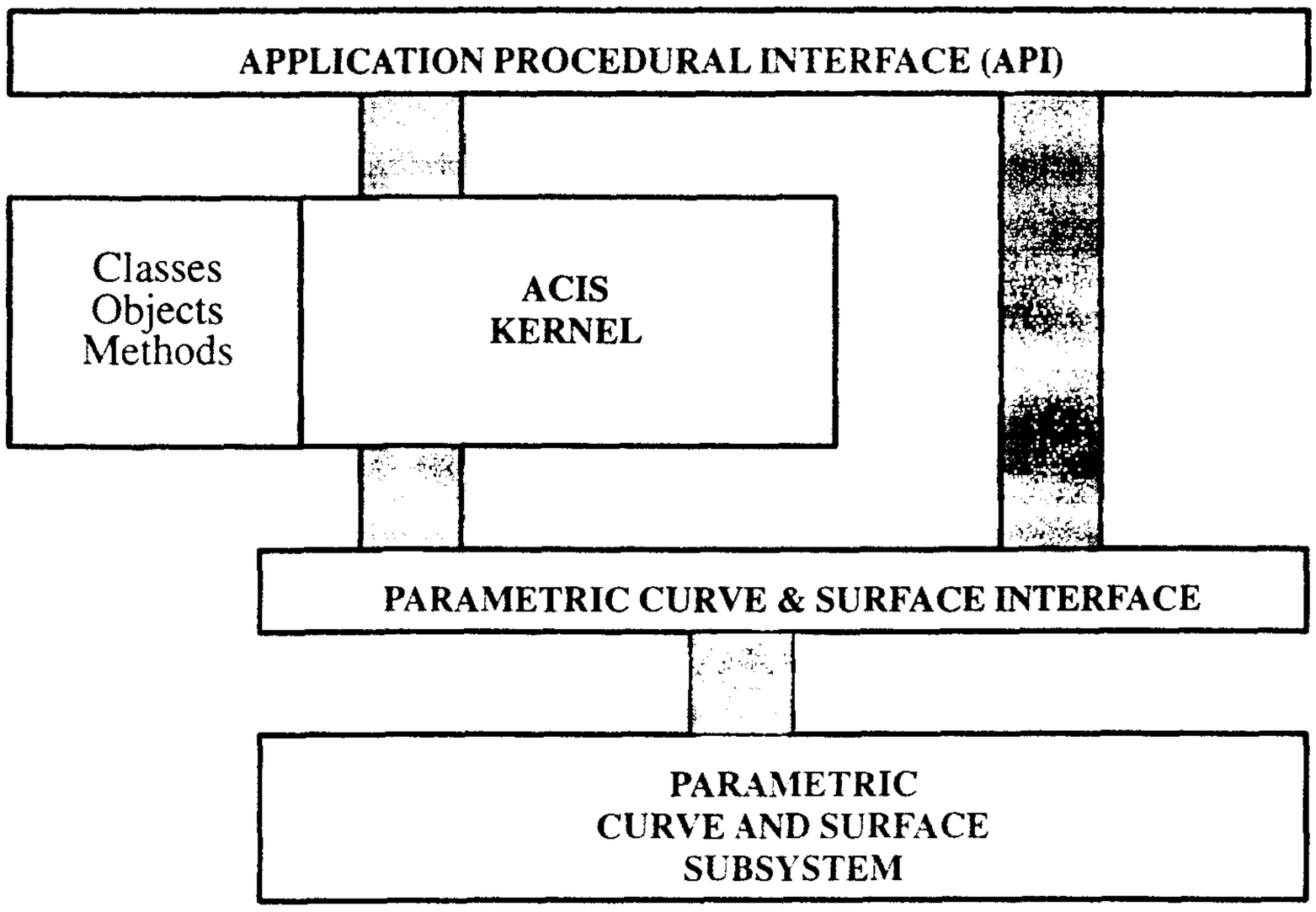


Figure 3.5: The structure of the ACIS solid modeller

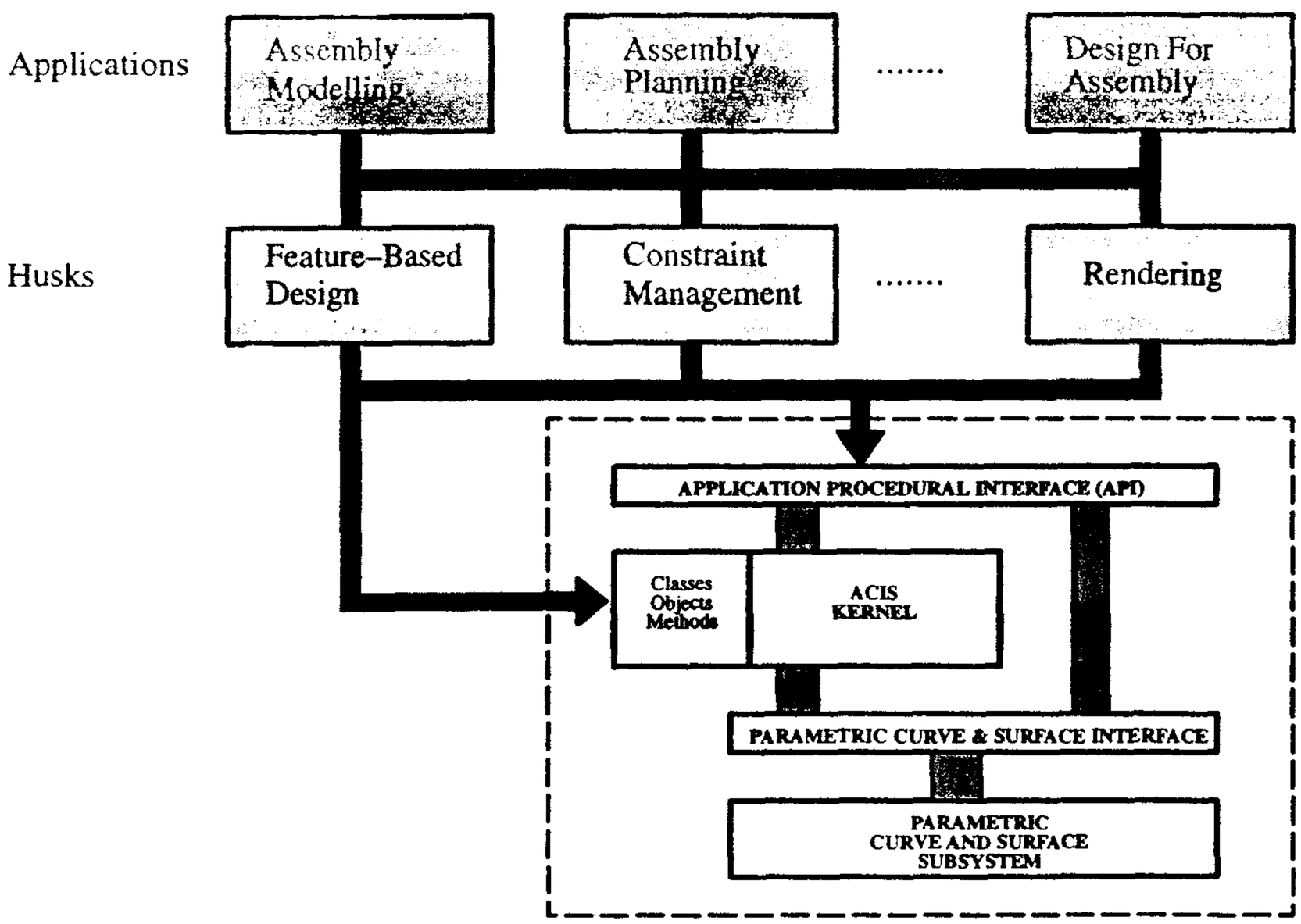


Figure 3.6: Husks and applications

There are few commercial feature husks available, such as the Feature Management Husk (Spatial Europe 1993). The husk offers some standard features and at the same time allows application developers to provide their own definitions of features. The user–defined features can be described in C++ or in an external command language, such as Scheme, a LISP–based language. It is claimed that by using this approach, many new types of features can be created at a faster speed. The husk can be interfaced to an external constraint modeller.

With the increasing application of OO techniques, the use of ACIS in the development of manufacturing applications has increased. In the feature–based domain, ACIS has been used as a geometric modeller for a process planning system (Krause et. al. 1991, Wang 1991); as a kernel for a feature recognition system for multiplying connected holes, pockets and islands in 2.5D objects (Corney 1991), and to extract feature model data from a 2.5D component (Murray 1993). ACIS has also been recommended as the geometric modelling kernel for feature–based design systems to alleviate some of the feature–geometry interfacing problems (Sreevalsan and Shah 1992). The following sections outline some features of ACIS utilised in this research.

3.6.2 APPLICATION PROCEDURAL INTERFACE

The Application Procedural Interface (API) is a collection of routines that can be called by applications to create, change or retrieve an ACIS class library. The advantage of the API interface is that all of its calls are stored in a file called a journal and thus a sequence of calls made exclusively through this interface can be replayed. Users can create their own API routines using guidelines and header files, tools, and macros provided. In this research, API functions are incorporated into the C++ programs which build the feature library and are used to create specific feature profiles and manipulate the features through functions such as move, draw and save. Examples on the use of API for these purposes are given in Section 3.6.5.

3.6.3 C++ CLASS STRUCTURES

Another means of accessing internal objects, classes and methods is through the *Direct Interface*. This approach is used to make rapid and efficient inquiries of models and reads and changes data structure entities directly. However ACIS does not store calls to it in a journal and thus it cannot be replayed. It is suitable for read–only access e.g. for graphic output.

ACIS provides five types of classes (as in ACIS version 1.4.1), which are briefly described in the following paragraphs:

i. *Mathematical*

This class represents the concepts of 3D cartesian coordinates, direction vectors, transformation matrices for positioning entities and general 3 x 3 matrices.

ii. *Geometry*

This contains various classes to define geometric curves and surfaces which are not retained permanently in its object data structure. The classes include curves, straight lines and ellipses.

iii. *Entity*

Entity is the class from which all classes representing permanent objects in the ACIS modeller are derived. It represents common data and functionality which must be contained in all classes that represent permanent objects within the modeller. The relationship of the model classes is illustrated in Figure 3.7. The Entity class also covers two categories of classes – topological and data structure.

The above classes are utilised in the development of a feature and assembly model. For example, the transformation matrix from the mathematical class is used to position the parts in an assembly model. From the entity class, the topology part of the class is used for identifying the face on the feature. Two other classes which are not used in this work are the *Miscellaneous* and *Utility* classes. The miscellaneous class consists of two classes – box and interval. The box class provides a method to test the interaction between two entities while the interval class represents a finite range on the real line. Classes in the

Utility class deal with the intersection between curves and surfaces as well as the intersection and relation of an edge to an entity.

All ACIS classes utilise the strength of C++ in an OO environment. Some of these features are:

i. Compile time checking.

ii. Private data which can be accessed only by public methods, thus protecting the data from the application program (see Section 3.2.1).

iii. Public methods which can be accessed by application programs.

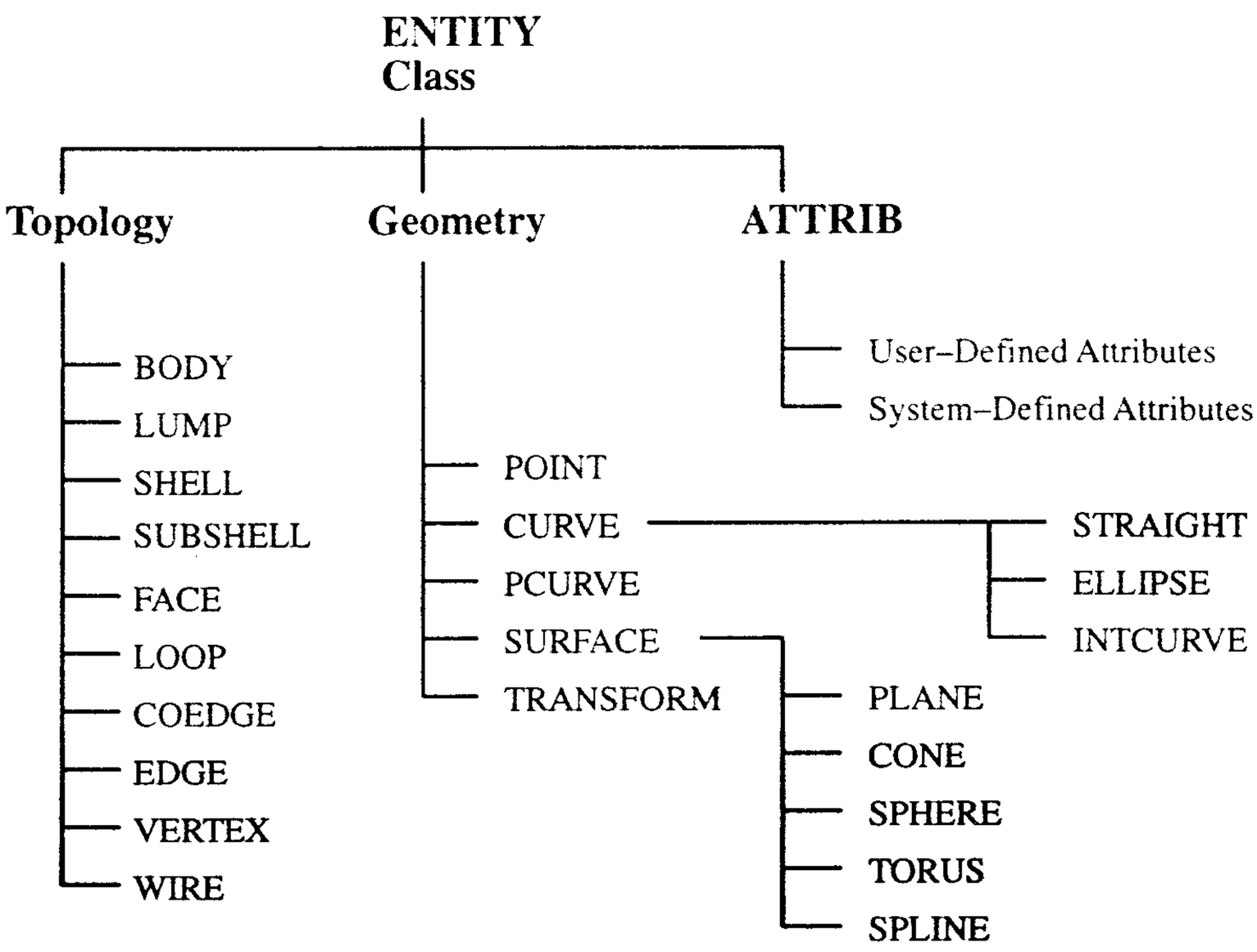


Figure 3.7: Relationships of the ACIS model classes

iv. Constructors and operators are overloaded (see Section 3.2.2). For example, there are three versions of constructors for the position class:

position() defines a position without initialising any coordinates.

position(double, double, double) defines a position with coordinates

x, y and z of double precision.

position(double[ ]) defines a position from an array of three points with double precision.

v. Functions and operators in mathematical classes are overloaded.

vi. Virtual functions are provided. For example, the curve and surface classes each have virtual position and parametric–based evaluator functions. The correct evaluator will be used according to the type of geometric entity called.

3.6.4 THE TEST HARNESS

The Test Harness is a simple application program written in C++ and supports English–like commands entered by the user. It provides a simple vectographic and shaded image output and several different forms of file input and output. The test harness provides an interface to all features of ACIS and is used to test ACIS and for program proving. A model created by an application program is saved in a file and the file can be retrieved from the test harness. The version of the test harness used in this work does not provide a rendering facility, and due to its limited capability, it is not suitable as a Graphical User Interface tool for a professional application system.

3.6.5 EXAMPLE OF AN ACIS PROGRAM

The following C++ code shows an example of how API commands are used to create a component with two features – a base feature of rectangular profile represented by a cuboid of dimensions 100 x 80 x 30 units and a cylindrical boss feature represented by a cylinder of radius 10 units and height of 40 units. First the cuboid is created using api_make_cuboid and the api_find_face function finds its top face. Then a cylinder is created and moved by the application of a transformation function by a distance of 20 units in the z direction. The cylinder is united with the cuboid and the resulting body is saved in a file "cuboid.sat" which is retrieved for display in the test harness, as shown in Figure 3.8.

```
api_start_modeller(TRUE, NULL, 0);
BODY* cuboid;
api_make_cuboid(100, 80, 30, cuboid);

FACE* face;
outcome result = api_find_face(cuboid, unit_vector(0,0,1), face);
if(!result.ok( ))
{
        cout << "failed to find face (error number)\n";
        exit(0);
}

BODY* cyl;
api_make_frustum(40,10,10,10,cyl);
api_apply_transf(cuboid, translate_transf(vector(0,0,20)));
api_unite(cyl, cuboid);

FILE* save_file = fopen("cuboid.sat", "w");
api_save_body(save_file, TRUE, cuboid);
fclose(save_file);
```

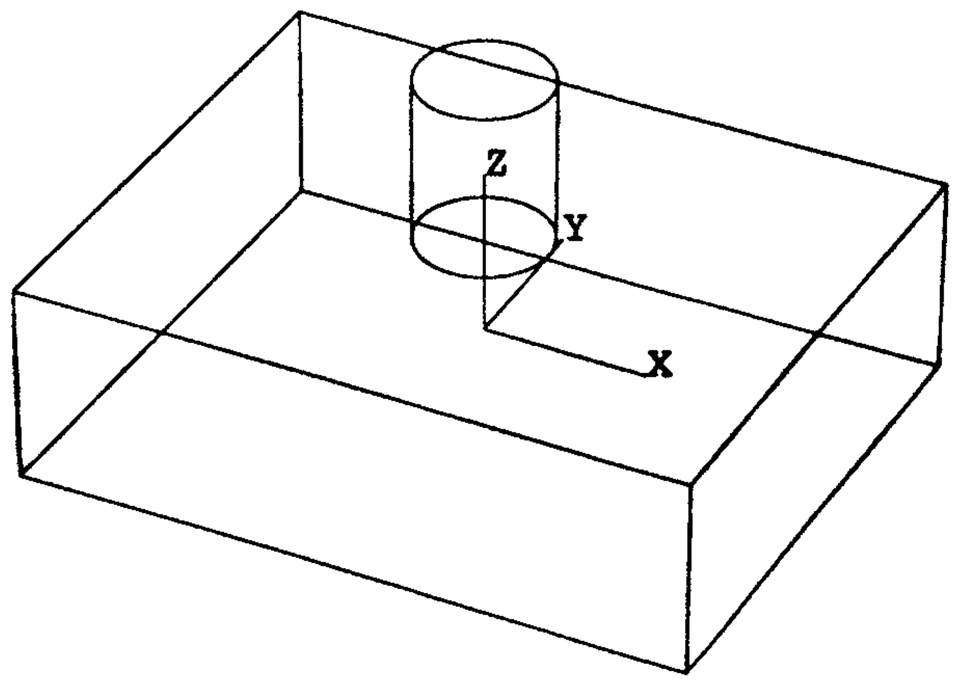


Figure 3.8: Example of ACIS model

3.7 SUMMARY

This chapter has discussed the basic principles of OO programming, its benefits and the OO design approach adopted in this research work. The OO technique introduces many new ideas and involves a different approach to programming. The three main concepts of OO programming – encapsulation, polymorphism and inheritance offer a new and powerful model for writing computer software. The technique offers benefits of faster development, easier maintenance, improved modifiability, more compact code and the opportunity to reuse and recycle large sections of the code. The modular and hierarchical nature of the system provides a natural way of handling the complex relationships between parts in an assembly of products. This justifies the use of this approach in this research work.

The development of a feature–based assembly modelling system is also facilitated with the use of the C++ language and the solid modeller kernel ACIS. A major advantage of using ACIS is its extensibility. This can be done by adding attributes, deriving from the entity class and adding new API functions. The C++ language is a practical language and has the necessary facilities for OO programming and this is utilised in ACIS to provide flexibility in the design.

The next chapter discusses the application of OO approach in the representation of feature while Chapter 5 describes the application in the representation of assembly models.

CHAPTER FOUR
FEATURE REPRESENTATION

4.1 INTRODUCTION

An important part of this research is the development of a feature representation which is capable of incorporating knowledge on applications, particularly assembly modelling. As discussed in Chapter 2, the way a feature is defined and represented affects the scope of its use. The aim of this chapter is to provide a framework for the description of features, their classification and how they are represented in an object–oriented environment. This will be the basis for the subsequent work described in the following chapters. Section 4.2 describes the features used in this research work. The feature taxonomy is discussed in Section 4.3. Section 4.4 describes how feature knowledge is represented in an object–oriented environment. Section 4.5 summarises this chapter.

4.2 FEATURE DESCRIPTION

An important concept in feature–based design and manufacture has been outlined in Chapter 1 – a single feature representation should be capable of supporting a number of different applications. This requires that a feature should incorporate as much knowledge as possible which allows its use in many applications. This requirement can be achieved by extending the knowledge within the feature object in an object–oriented environment, as described later. However, as this research is focussed on the application of features in assembly modelling, the emphasis of knowledge in the feature will be for this application domain. Further, as assembly modelling and other applications such as inspection are carried out after the machining process, process planning knowledge must be included in the feature.

The approach taken in this research is to describe features based on machined shapes, but in a way that is at the same time useful in the design process. Features are defined in terms of volumes enveloped by a set of real and imaginary faces. A real face refers to an actual face which exists on the feature and are typically surfaces from the original part or the

result of manufacturing operations. Imaginary faces can be considered as surfaces required together with the real faces to form an enclosed volume. Figure 4.1 shows a step feature formed by the removal of a volume that is in part enclosed by imaginary faces. In the ACIS solid modeller features are represented in a B–Rep scheme, in which faces are explicitly defined. Information on faces is required to determine the relationship between each feature in an assembly model. The real faces for each feature is shown in Figure 4.6. Volumes removed as a result of machining operations form depression features while protrusion features are volumes to be added to the part or to be left after machining of surrounding regions. Figure 4.2 shows a depression feature in the form of a pocket and a protrusion feature represented by a boss.

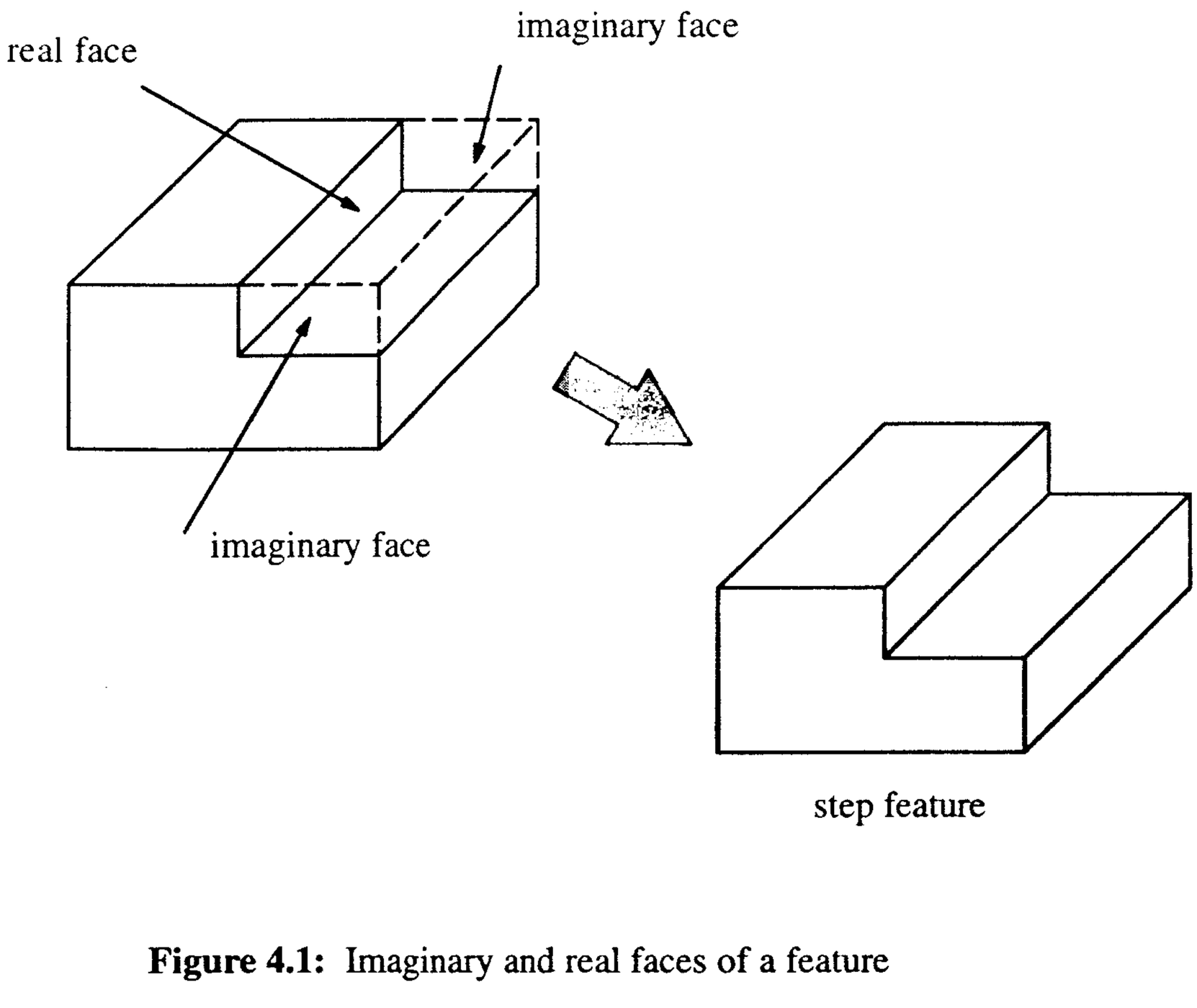


Figure 4.1: Imaginary and real faces of a feature

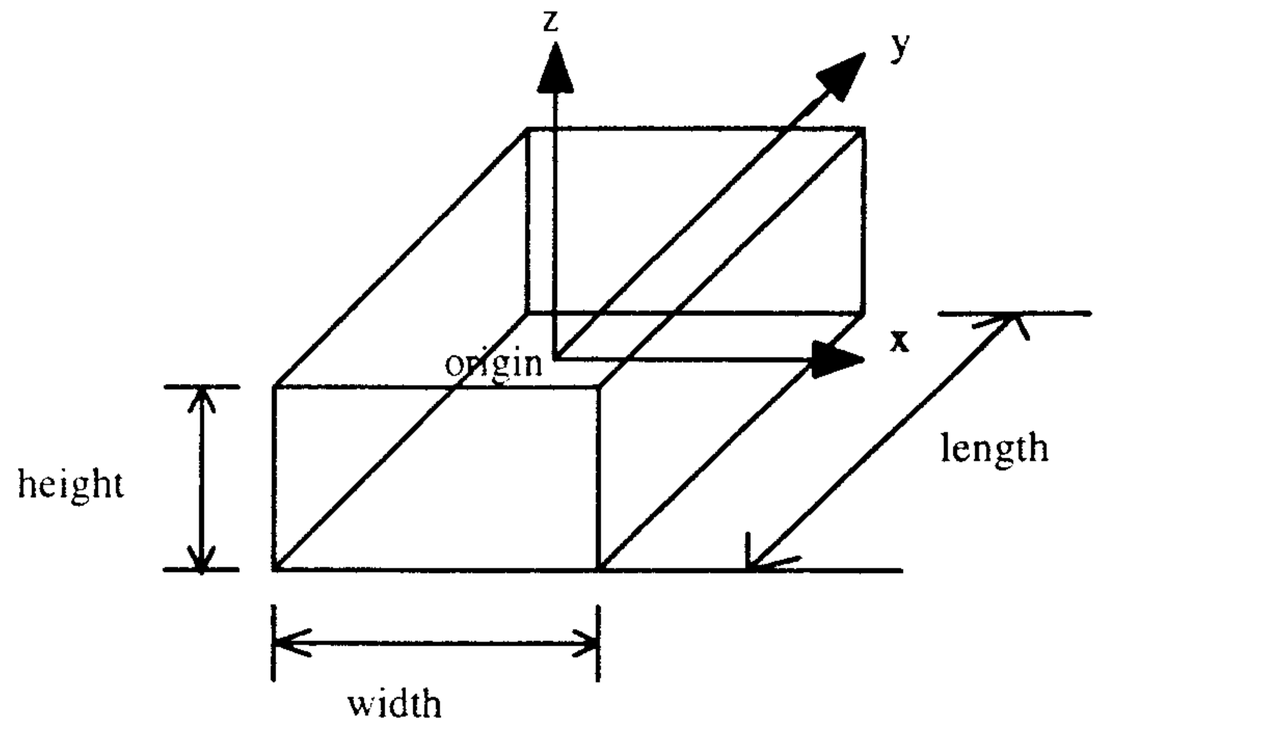
Features are further defined in terms of dimensions such as height, width, length and radius, according to their geometric profiles. Each feature has its own coordinate system attached to it at the point of reference, as shown in Figure 4.3. The origin is at the centre of

the feature body, following the convention used in the ACIS modeller. The orientation of the feature is represented by three independent (Eulerian) angles – θ (rotation about z axis), ψ (rotation about y axis) and ϕ (rotation about z axis). The position and orientation of the feature provide six degrees of freedom – three translational and three rotational and are used as a reference for the placement of features in an assembly as well as establishing positional relationships between two features.

Figure 4.2: Depression and protrusion features

4.3 FEATURE TAXONOMY

The requirement for a feature taxonomy is that knowledge on the application domain can be structured and organised so that features can be used effectively in an application. To achieve this, the taxonomy should organise information on the process planning and assembly modelling that can be conveniently represented in an object–oriented environment. The most natural way is to organise features in a hierarchical structure. The taxonomy scheme is an extension and enhancement of the scheme developed by Gindy (1989) as described in Chapter 2. The scheme was found to satisfy the hierarchical requirements and to be well suited to object–oriented design.



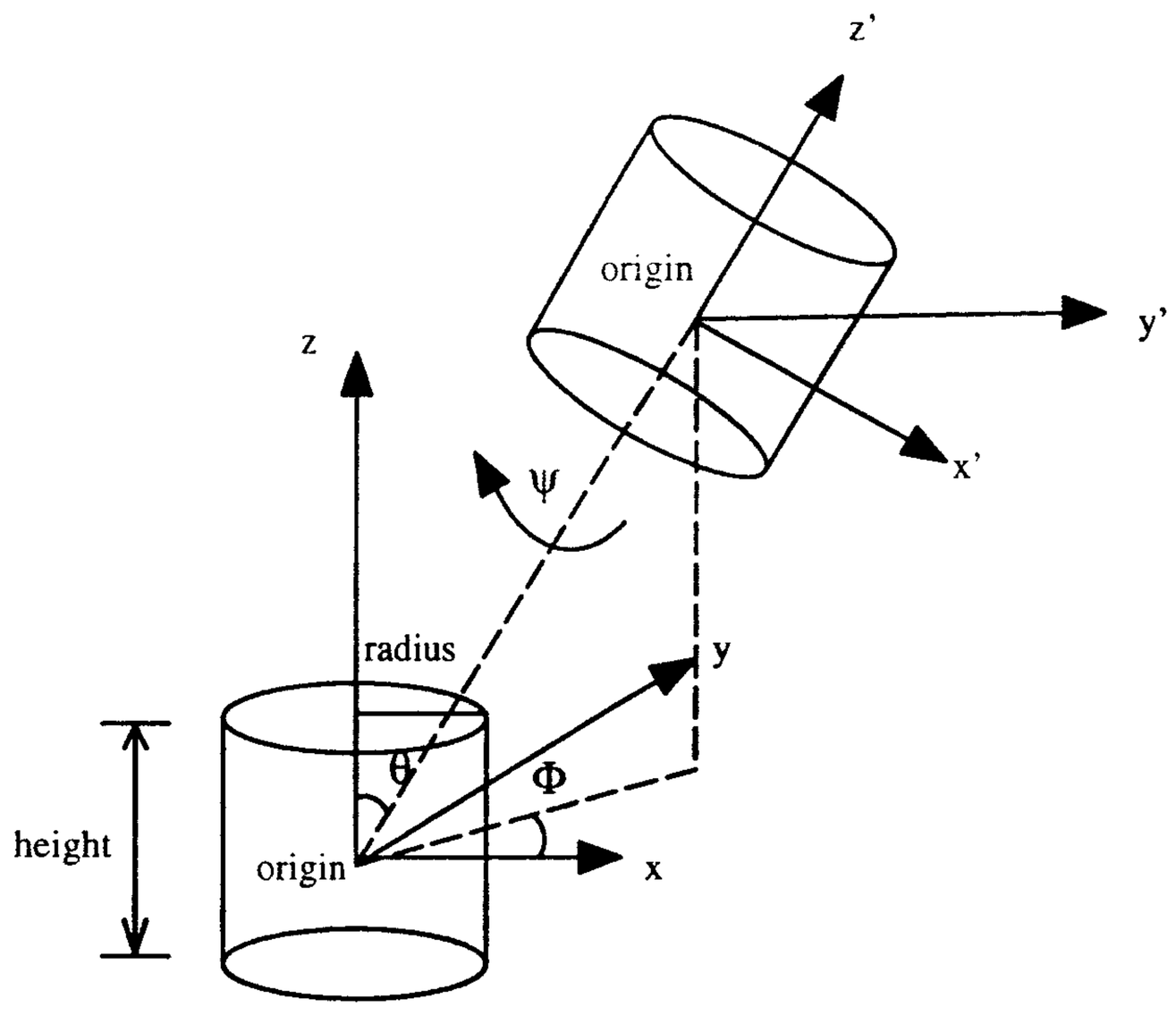


Figure 4.3: Feature dimensions and orientation

In this scheme, features are structured at four levels of classification, as shown in Figure 4.4. At the top level, they are divided into three categories of protrusions, depressions and surfaces. Protrusions are external features of a solid and can only have closed boundaries. Depressions are external or internal features with closed or open boundaries. Surfaces occur when the feature has no depth. A surface will be real when the inside of the boundary is solid and imaginary when the boundary is enveloping an empty area. Surfaces are only included in the taxonomy for completeness of the scheme, as the mating surfaces that are of great significance are represented as faces of features, and not independent features. The next level of classification is the number of orthogonal directions from which the feature volume might be approached. These are known as *External Access Directions* (EADs) and all features have between 0 and 6 EADs, as shown in Figure 4.6. Zero EAD indicates a protrusion, one to four EADs indicate a depression and five and six EADs denote real and imaginary surfaces respectively. The EAD is used in process planning to characterise a face through which a cutting tool can pass in order to machine the feature volume.

Further classification is on the basis of the nature of boundary perimeters – open or closed as shown in Figure 4.5. An open profile has imaginary edges (edges of imaginary faces such as a top face of a slot) while a closed profile has all real edges, such as a hole. The classification results in nine types of feature – boss, pocket, hole, non–through slot, through slot, notch, step, real face and imaginary face. Based on the distinctive shapes of many parts involved in machining and assembly, five common profile types are identified – rectangular, circular, triangular, oblong and semi–circular. The profile shapes are not limited to these five types as other shapes can be defined if necessary. For each feature type, a number of primitive shapes are defined, based on the geometry of the feature profiles as shown in Figure 4.6.

The feature shapes defined above provide convenient building blocks for the assembly modelling as well as process planning applications. In general, the taxonomy provides the opportunity for feature profiles to be extended to suit other manufacturing applications.

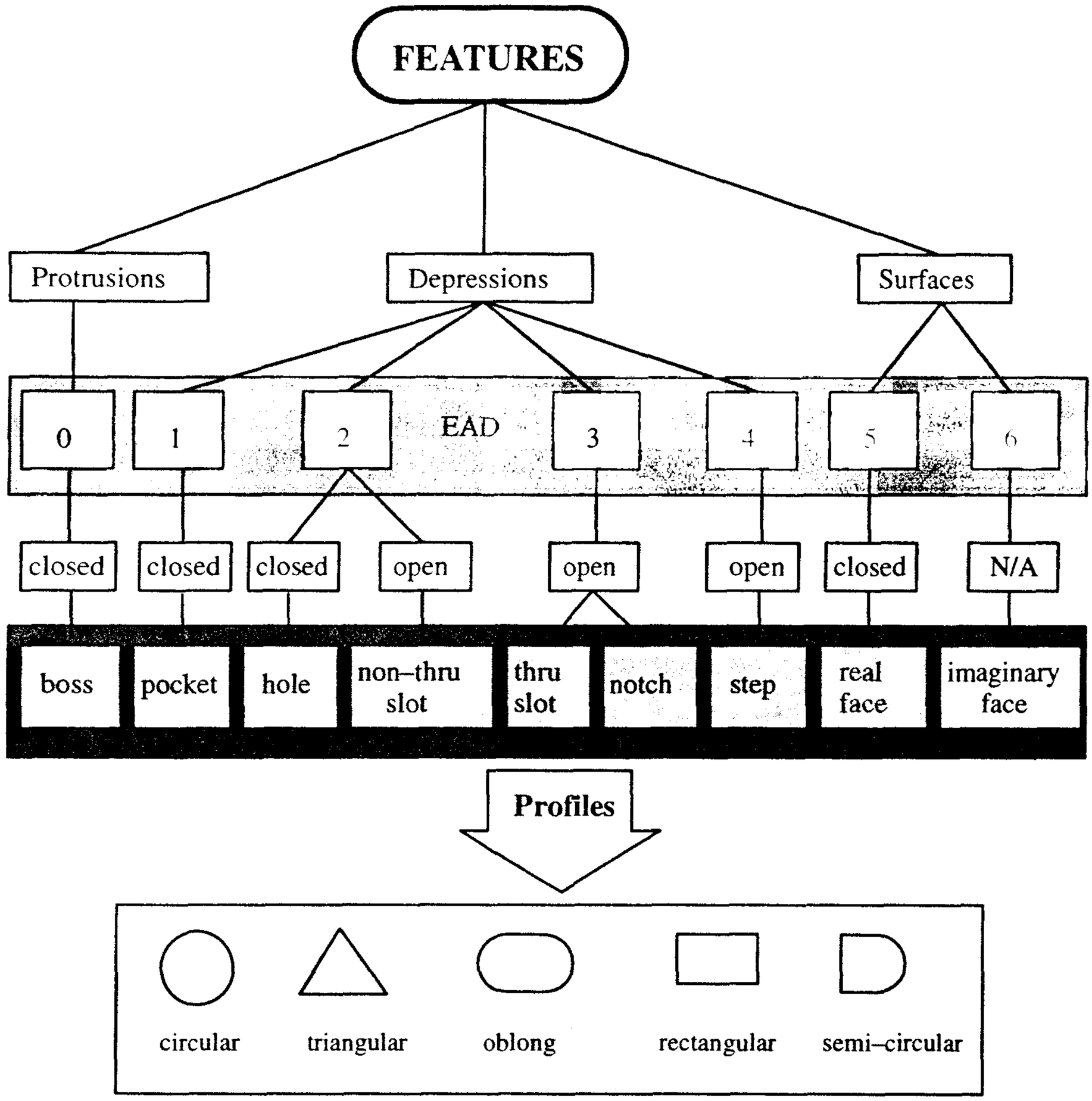


Figure 4.4: Feature hierarchy

Figure 4.5: Open and closed boundaries

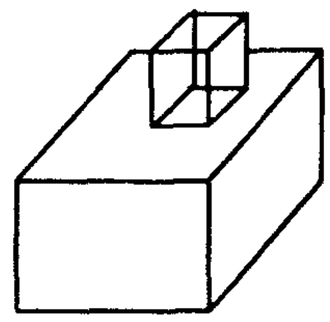
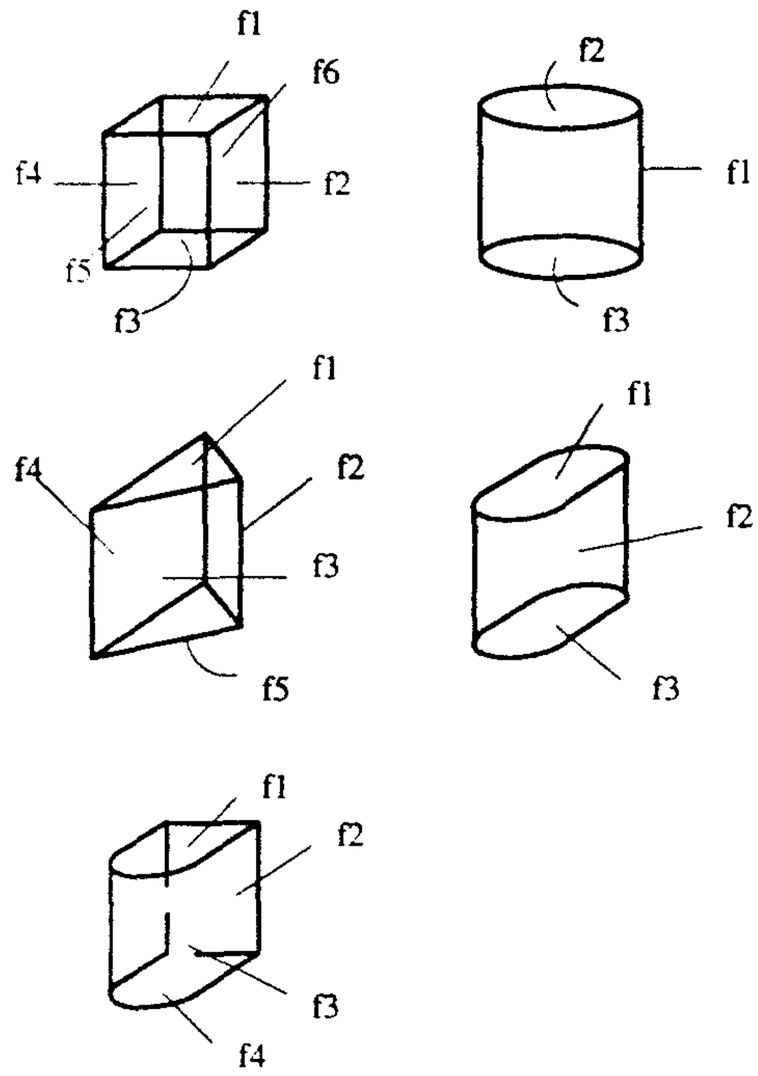
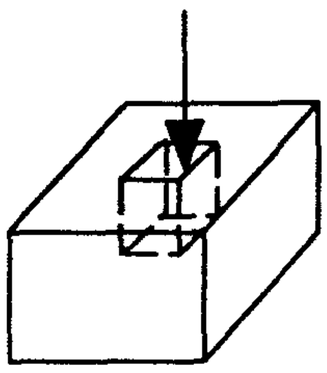
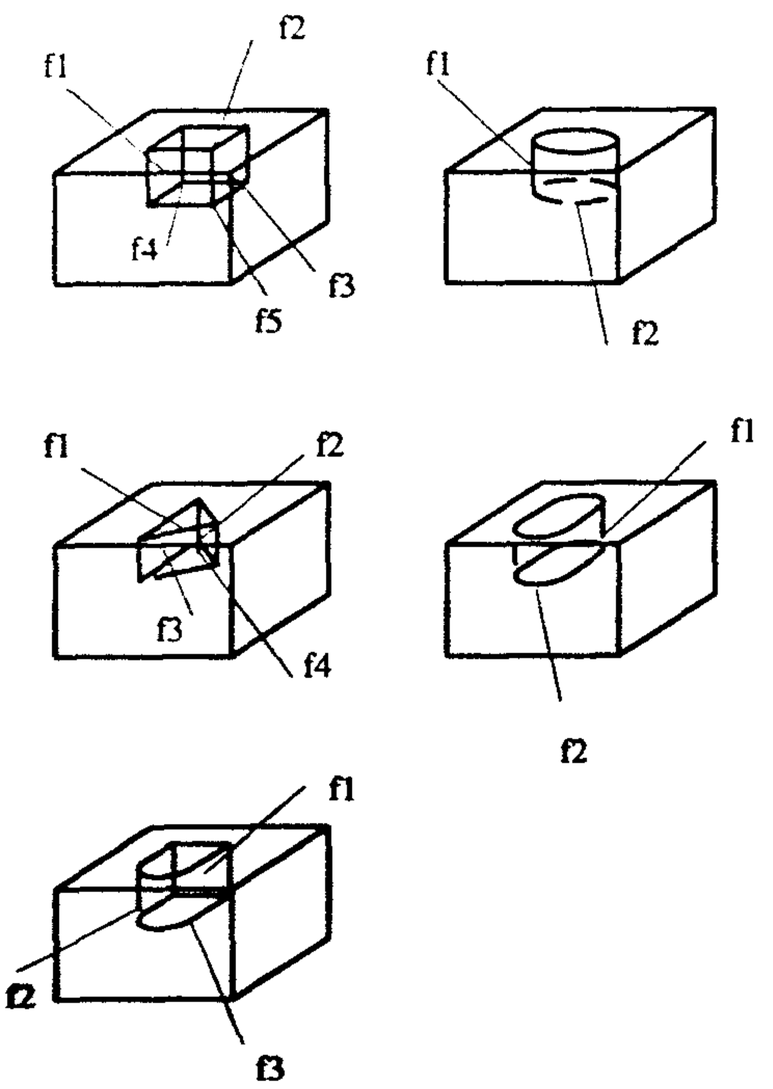
Type	Data	Profiles and Faces
BOSS 	Protrusion EAD: 0	
POCKET 	Depresssion EAD: 1	

Figure 4.6: Feature classification data

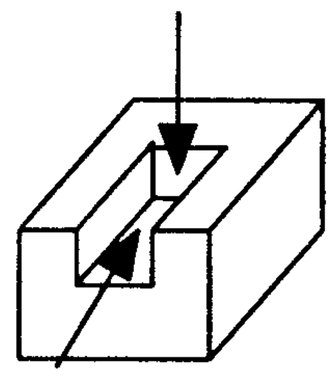
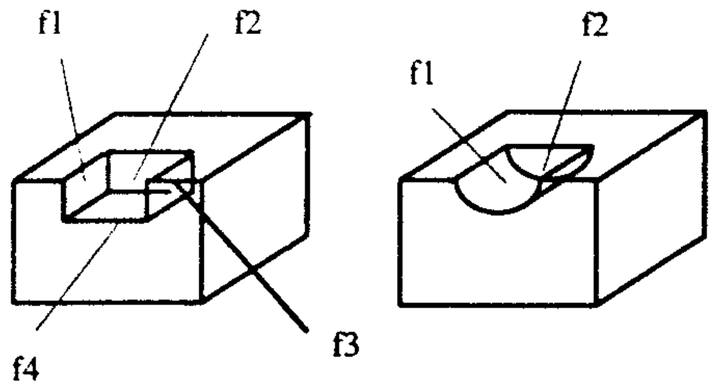
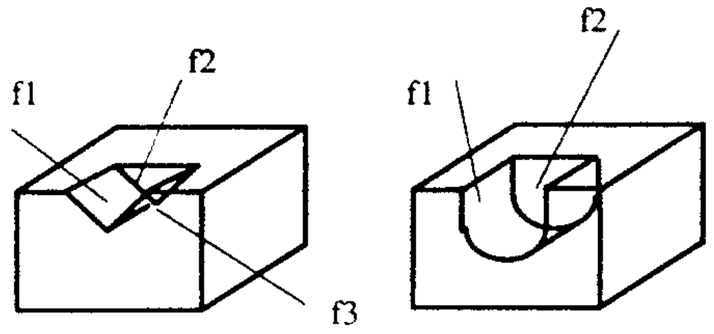
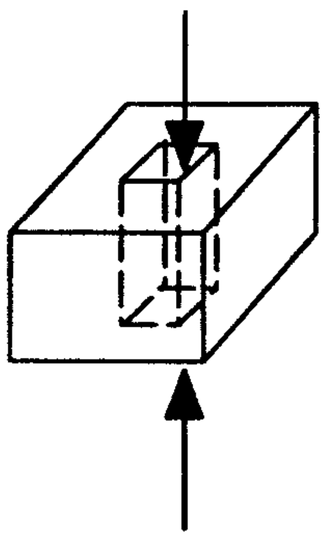
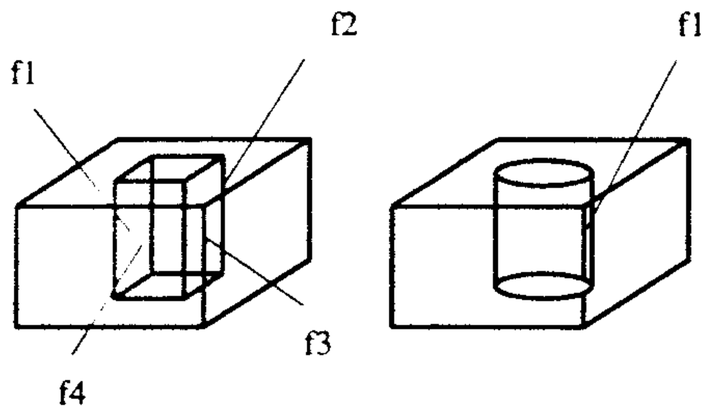
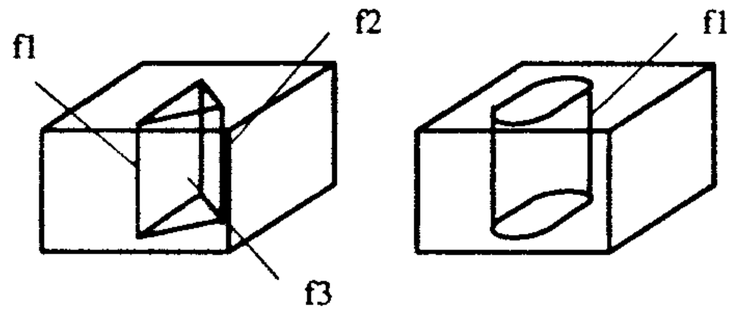
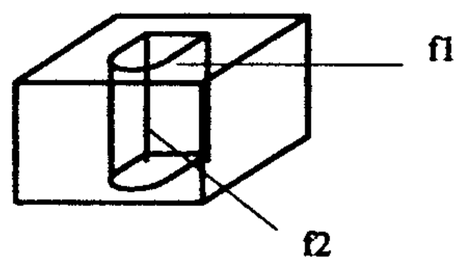
Type	Data	Profiles and Faces
NON–THROUGH SLOT 	Depresssion EAD: 2	 
HOLE 	Depresssion EAD: 2	  

Figure 4.6: Feature classification data (continued)

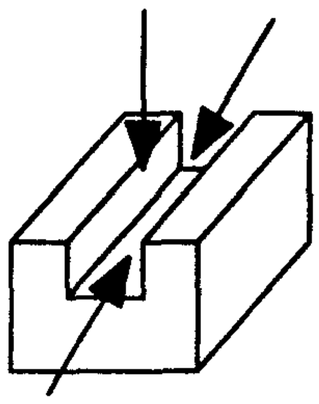
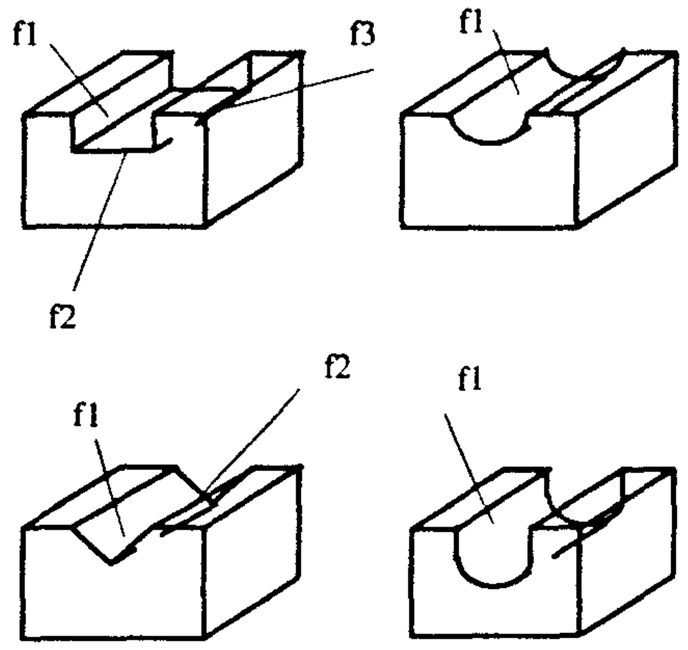
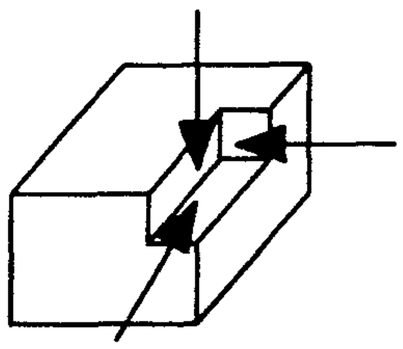
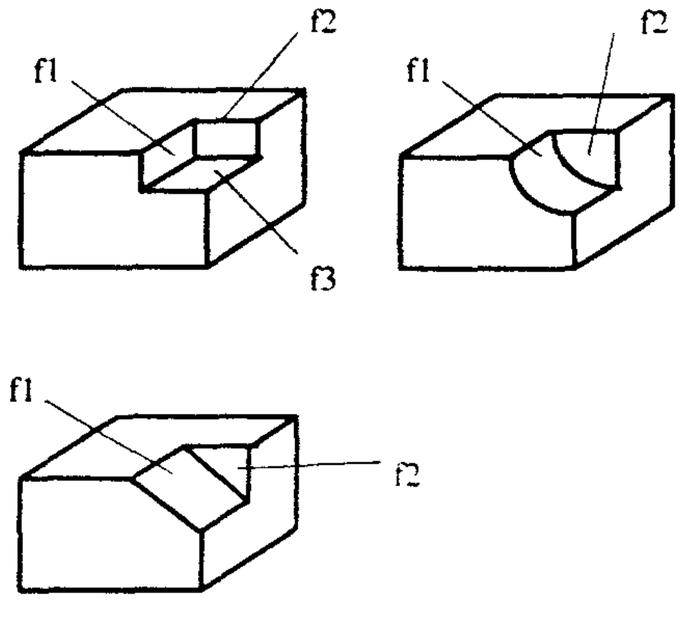
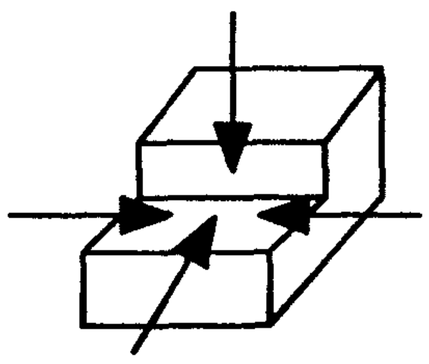
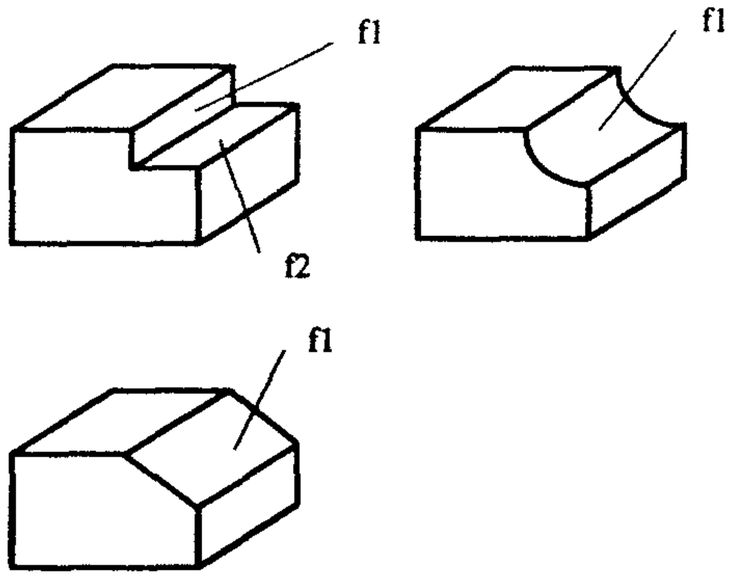
Type	Data	Profiles and Faces
THROUGH SLOT 	Depresssion EAD: 3	
NOTCH 	Depresssion EAD: 3	
STEP 	Depresssion EAD: 4	

Figure 4.6: Feature classification data (continued)

4.4 FEATURE CLASS REPRESENTATION

In an object–oriented approach, a feature is modelled as an object encapsulating various attributes and methods to manipulate the data related to the feature as described in earlier sections. The feature taxonomy described in Section 4.3 provides a convenient way to design a feature class. Using the concept of inheritance, a feature class is defined to be a base class for the seven types of features and five types of profiles. The other three levels in the feature hierarchy – categories, number of EAD's and the nature of the boundary perimeter are not implemented as classes. The number of EAD's, the categories and the nature of the boundary perimeter are attributes of the feature type. The following paragraphs describe the structure of each class, using the notation described in Chapter 3. The class name is shown in bold text such as **boss**.

4.4.1 FEATURE CLASS

All features have common attributes such as position and orientation. There are also common actions which features have to perform such as drawing the shape and saving the entity. These common attributes and methods are defined in a **feature** class. The object diagram for the **feature** class is shown in Table 4.1.

The class **feature** has private members of a pointer to the body of the feature, a pointer to the next feature on the same component, the location with respect to the world coordinate system, the orientation, the height, the number of external access directions (EADs), which is a constant integer (0 to 6), a pointer to the feature face, a pointer to the feature type (boss, hole, etc), a pointer to feature profile and a pointer to the assembly relationship. The role of the pointer to the assembly relationship is elaborated in the following chapter. These attributes can only be accessed through the public member functions described in the following paragraphs.

Feature
Pointer to feature body Pointer to next feature Location Orientation Height EADs Pointer to feature face Pointer to feature type Pointer to feature profile Pointer to assembly relationship
Constructor Destructor Get Dimension Get Location Get Orientation Validate Input Create Feature Select Feature Type Select Profile Type Find Face Move Delete Save

Table 4.1: Feature Class

Feature Constructor

The feature constructor creates a feature instance and initialises its parameters when a feature type is defined. There are three variations of constructors available and the right type is invoked according to the function parameters supplied:

feature() is a default constructor and used to reserve space for a feature instance.

feature(double &h) initialises a feature body from the dimensions provided by the user. The height of the feature (h) is the common dimension for

all types of features. Other dimensions are initialised according to the profile types.

feature(feature const &) is a copy constructor which is invoked when an instance of a feature is copied.

For example, a pointer to the feature instance f, with height of 50 units is created with the expression *feature *f = new feature(50)*.

Feature Destructor

The destructor, denoted as *~feature()* destroys the feature body at the end of the session, to free the memory.

Get Dimension

This function is required to obtain dimensions of the feature from the user. The function is virtual, which means that (for this class), only the height dimension is requested from the user and a similar function name is used by **feature type** and **profile** classes to get other dimensions which are specific to the profile of the feature.

Get Location

Get Location is used to ask the user to specify the location of the feature with respect to the datum of the component. In the case of a base feature (a base for the component in an assembly), the location is given with respect to the local coordinate system. The location is specified in arbitrary units along x, y and z axes.

Get Orientation

This function is used to initialise the orientation of the feature, which is the rotation about x, y and z axes, as described in Section 4.2.

Validate Input

This is a virtual function to validate entries on the location and dimensions of the feature. The function is implemented in the feature **profile** class.

Create Feature

Create Feature is a function to create an instance of the feature body. This is also a

virtual function which is derived by the **feature type** and **profile** classes so that an appropriate feature type is created. For example, when a rectangular boss feature is defined and its parameters are correctly entered, the function to create that particular boss type is invoked. The function utilises API functions from the ACIS library.

Select Feature Type

This function is used by the user to make a selection of a feature. A menu of feature types is provided and the user selects the type required.

Select Profile Type

After selecting the feature type, the user identifies the feature profile from a profile menu. The selection of a particular profile invokes appropriate functions from the profile class. The following expressions represent a part of the code which shows the choice of circular profile:

```
feature* feature::SelectProfile(component C)
{           int ans;
            feature *f;
            ProfileMenu();
            cin >> ans;
            {
            switch(ans)
            {
                        case circ: f = new cylinder(r,h);
                                    f->GetLength(C)->GetHeight();
                                    f->CreateFeature();
                                    break;
                        case rect:  . . . . . .
}
```

Find Face

The function is used to identify a face on the feature for the assembly. Faces are

identified according to the vector directions of the face in the x, y and z directions. This uses the ACIS direct interface function *face()*.

Move

This is a function to move a particular feature within the base feature. The movement is achieved by applying the appropriate transformation function provided by an API from the ACIS library. For example, the following code moves a boss feature *bos* by 10 units in the x direction:

api_apply_transform(bos, translate_trans(vector(10,0,0)));

Delete

This function deletes a feature body from a component or a subassembly. This is done using an API function *api_delent()* which deletes the entity and invokes the feature destructor. For example, the following code deletes a boss feature **bos*:

*api_delent(*bos);*

Save

Save is a function to save a feature to a file. This is also a virtual function, as the actual entity saved depends on a specified feature type and profile.

C++ Codes for the Feature Class

The declaration for the feature class is implemented in C++ as follows:

```
class feature {
protected:
        BODY * feat;
        feature *next;
        char feature_ID [20];
        double pos_x, pos_y, pos_z;
        double angleX, angleY, angleZ;
        double height;
        int EAD;
        feature *type;
```

```
                feature *relationtype;
public:
                feature();
                virtual ~feature();
                virtual void Get_Dim();
                void GetLocation();
                void GetOrientation();
                void Validate();
                void SelectFeatureType();
                void ProfileMenu();
                feature * SelectProfile();
                virtual void Create_Feature();
                virtual void Move();
                virtual void Delete();
                virtual void SaveEntity();
};
```

4.4.2 FEATURE TYPE CLASS

Each of the seven types of feature defined in Section 4.3 is represented by a class which is derived from the **feature** class. They inherit all attributes and functions of the **feature** class. The general content of the class is shown in Table 4.2.

The private member for this class is a pointer to the feature type body (such as *boss) and a feature name. The feature name identifies a feature type such as boss, pocket or hole and its index number such as boss1, hole2, etc. The public member functions are described in the following paragraphs:

Feature Type
Pointer to the feature type body Feature name
Constructor Destructor Create Feature

Table 4.2: Feature Type Class

Feature Type Constructor

The constructor initialises an instance of a feature type. For example, a boss feature is initialised by a constructor in the form of *boss(double &x, double &y, double &z, feature *ptr)*. The x, y, and z values denote the location of the boss. The pointer to the feature is needed to add the feature type to the feature list that makes up the component. The constructor also initialises the feature name and the number of EAD's associated with it.

Feature Type Destructor

The destructor deletes the feature type body when its instance is deleted. For example, for a boss feature, the destructor is denoted by *~boss()*.

Create Feature

This function redefines the virtual function in the **feature** class by creating a specific type of feature given by the user. This ensures that the right function associated with the feature type is invoked whenever it is used. For example, for a boss feature, Create Feature will invoke the following actions:

determine the position of the boss

move the boss to its position

This is represented by the following code:

*z_pos = 0.5*height of base feature + 0.5*feature height*

api_apply_transform(boss, translate_transf(vector(x_pos, y_pos, z_pos)))

where *x_pos*, *y_pos* and *z_pos* are the x, y and z locations of the feature and *api_apply_transform* is the API function to position the boss on the base feature.

C++ Codes for Feature Type Class

Taking a boss as an example, the declaration of the **boss** class in C++ is as follows:

```
class boss:public feature {
        BODY * bos;
        char FeatureType [5];
public:
        boss(double &x, double &y, double &z);
        ~boss();
        void CreateFeature();
};
```

4.4.3 PROFILE CLASS

Each profile type described in Section 4.3 is defined in the profile class . It is derived from the feature class and thus shares common attributes and methods of the class. The general content of the class is shown in Table 4.3.

The private members consist of the pointer to the profile shape (such as *rect for a rectangular profile) and the dimensions of the profile. The dimensions in this case are those required to define the profile shape, as detailed in Table 4.4.

Profile Type
Pointer to profile type Dimensions
Constructor Destructor Get Dimension Validate Input Create Profile Save

Table 4.3: Profile Type Class

Profile	**Dimensions**
Rectangular	length, width
Circular	radius
Triangular	length of first side, length of second side
Oblong	length, radius
Semi–circular	length, width, radius

Table 4.4: Dimensions of profiles

The public methods are described as follows:

Profile Constructor

The constructor creates an instance of the profile and initialises its dimensions. It is in the form of *feature_type(dimension parameters)*. For example, for a rectangular profile, the constructor takes the form of *rectangle(double &l, double &w)*. *l* and *w* being the length and width of the profile respectively.

Profile Destructor

The destructor deletes the profile instance when it is no longer in existence. It takes the

form of *~profile_type()*. For example the destructor for a rectangular profile is *~rectangle()*.

Get Dimension

Get Dimension is a derived function from the **feature** class. It redefines the function according to the type of profile selected. Thus the selection of a rectangular profile ensures that the length and width are required from the user while for a circular profile, only a radius is solicited.

Validate Input

This function provides the implementation of the method defined in the **feature** class. It checks two parameters – the location of the feature and the input dimension of the profile against the dimension of the base feature. A user is asked to enter the value until it is correct.

The validation of the location of a feature is done by checking the x and y positions (Fx and Fy respectively) against the dimensions of the base feature. Referring to Figure 4.7 which shows the dimensions of the base feature, the criteria for validation are shown in Table 4.5:

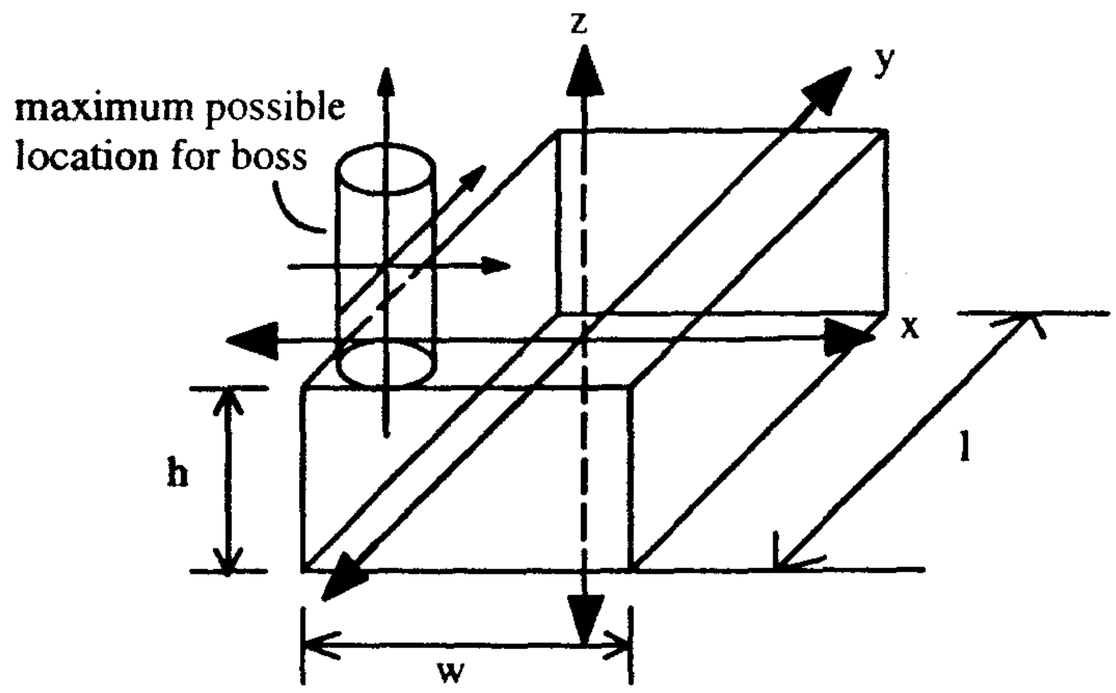


Figure 4.7: Validation of feature location

Profile	Criteria
Circular	$Fx \leq 0.5w - rad_f$, $F-x \geq -0.5w - rad_f$ $Fy \leq 0.5l - rad_f$, $F-y \geq -0.5l - rad_f$
Rectangular	$Fx \leq 0.5w - 0.5F_{lw}$, $F-x \geq -0.5w - 0.5F_{lw}$ $Fy \leq 0.5l - 0.5F_{lw}$, $F-y \geq -0.5l - 0.5F_{lw}$
Triangular	$Fx \leq 0.5w - 0.5F_s$, $F-x \geq -0.5w - 0.5F_s$ $Fy \leq 0.5l - 0.5F_s$, $F-y \geq -0.5l - 0.5F_s$
Oblong	$Fx \leq 0.5w - 0.5F_s$, $F-x \geq -0.5w - 0.5F_s$ $Fy \leq 0.5l - 0.5F_s$, $F-y \geq -0.5l - 0.5F_s$
Semi–circular	$Fx \leq 0.5w - 0.5F_s$, $F-x \geq -0.5w - 0.5F_s$ $Fy \leq 0.5l - 0.5F_s$, $F-y \geq -0.5l - 0.5F_s$

Table 4.5: Criteria for validation of profiles

In Table 4.3, Fx and Fy refer to the x and y positions of the feature, rad_f is the radius of the feature, F_{lw} refers to the length or width of the feature and F_s refers to each of the dimensions of the triangular, oblong and semi–circular profile (see Table 4.6).

The z location of the feature is determined by the type of the feature. For example the boss feature is always on a face of the base feature, so that the z position should not be more than the height of the base feature. This is determined by the CreateFeature function.

The height of depression features (h_f) is checked against the height of the base feature (h_c). The validation of feature height is done according to the criteria listed in Table 4. 6.

As an example, if the height of a pocket or a slot is zero or greater than the height of the base feature, then the entry is considered as invalid and the user is asked to reenter another value.

Feature Type	Criteria
Boss	$h_f > 0$
Pocket	$0 < h_f < h_c$
Hole	$h_f > 0$
Through Slot	$0 < h_f < h_c$
Non–Through Slot	$0 < h_f < h_c$
Notch	$0 < h_f < h_c$
Step	$0 < h_f < h_c$

Table 4.6: Validation of feature height

The validation of other dimensions such as the width, the length and diameter of the features are done according to the criteria shown in Table 4.7.

Profile Type	Criteria
Circular	$0 < dia_f < w_c$ or l_c
Rectangular	$0 < w_f$ or $l_f < w_c$ or l_c
Triangular	$0 < l_{1f}$ or $l_{2f} < w_c$ or l_c
Oblong	$0 < l_f$ or $dia_f < w_c$ or l_c
Semi–circular	$0 < w_f$ or l_f or $dia_f < w_c$ or l_c

Table 4.7: Criteria for validation of profile dimensions

Referring to Table 4.7, dia_f is the diameter of feature, w_f is the width of the feature, l_f is the length of the feature, l_{1f} or l_{2f} are the length of the sides of triangular profile, w_c the width of the base feature and l_c the length of the base feature.

Create Profile

Create Profile uses API functions to create the selected profile of the feature. For example, to create a rectangular profile, the following function is invoked:

api_make_cuboid(length, width, height, rect)

Similarly, the function to create a circular profile is:

api_make_frustum(height, radx, rady, radt, cyl)

api_make_frustum is an API function to create an elliptical cylinder of given height and three radii – x direction at base (*radx*), y direction at base (*rady*) and x direction at top of cylinder (*radt*).

Save

The feature entity with the specified profile can be saved in a file by this function. For example, to save a rectangular feature *rect* in a file *rect.sat*, the following codes are used:

```
FILE* fp = fopen("rect.sat", "w");
if(!fp)
{
        cout << "error saving feature";
        exit(1);
}
 api_save_entity(fp, TRUE, rect);
fclose(fp);
```

C++ Codes for Profile Class

An example of the C++ implementation for the declaration of the rectangular profile type is as follows:

```
class rectangle: public feature {
        BODY * rect;
        double width, depth;
```

```
public:
        rectangle(double &w, double &d, double &h);
        ~rectangle();
        void Get_Dim();
        void Validate();
        void Create_Profile();
        void SaveEntity();
};
```

4.4.4 RELATIONSHIPS AMONG CLASSES

Determining relationships among the classes defined in previous sections helps in organising them in the program. In this case, relationships among various classes have been described by the inheritance property of the object–oriented concept, which involves the sharing of attributes and operations among classes based on hierarchical relationships. The **feature** class is the base class for two general classes – the **feature type** and **profile**. Each class incorporates and inherits all of the properties of its base class and adds its own unique properties. For example, the boss class inherits the properties of the feature class but adds a different method for drawing the boss. Figure 4.8 illustrates the relationship among defined classes using the notation described in Chapter 3.

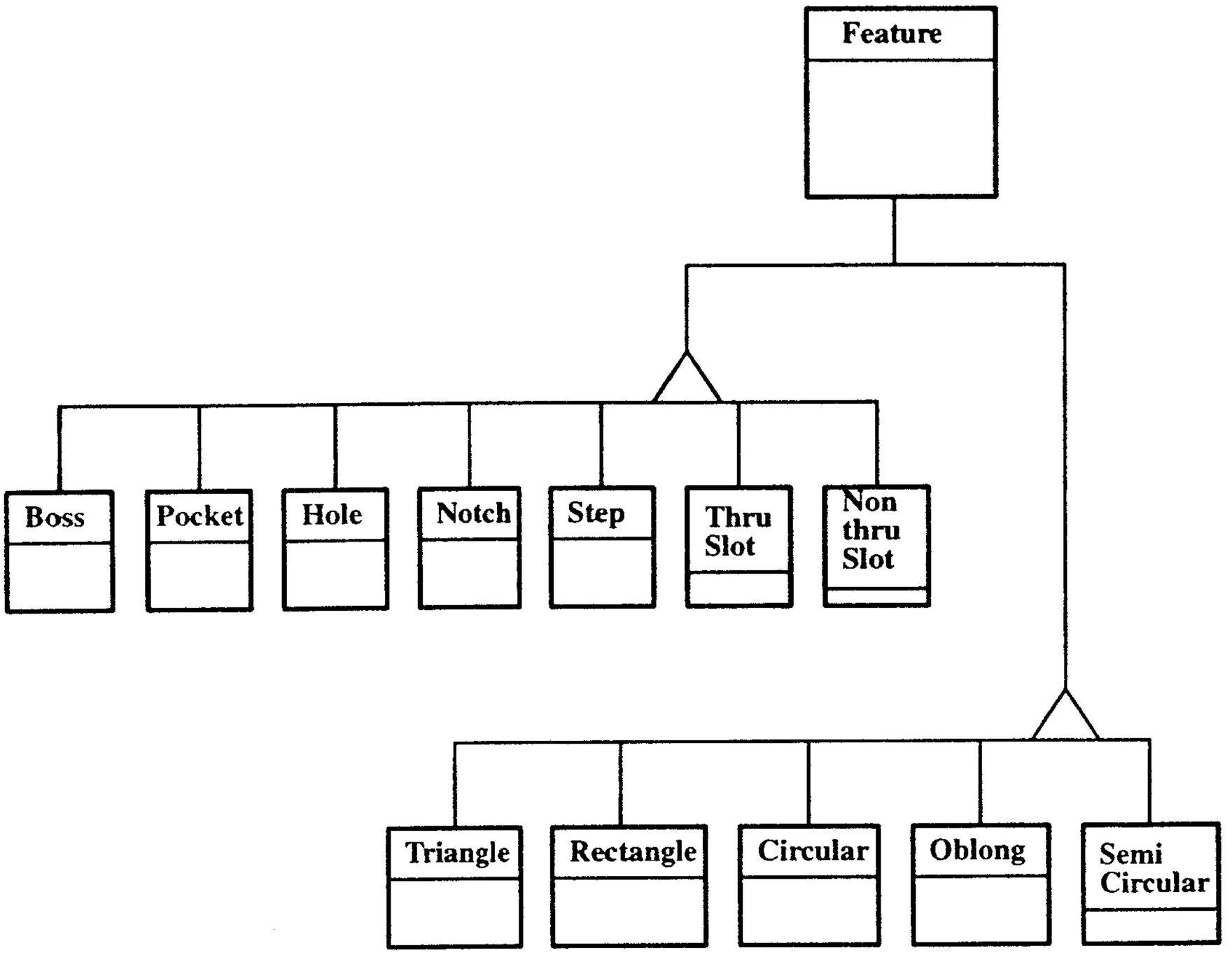


Figure 4.8: Relation Among Classes

4.5 SUMMARY

In this research, a feature–based approach utilising a hierarchical structure for feature definition and classification has been used. Features have been described in terms of machined volumes bounded by real and imaginary faces. A feature taxonomy is adopted and provides the basis for implementation in the object–oriented approach. The feature class defines the attributes and functions common to both the feature types and profiles. This class structure permits the use of inheritance between the object classes for accessing data and using various functions. In addition to the hierarchies defined above, the concept can be extended to create additional class hierarchies to support assembly modelling. This is discussed in the next chapter.

CHAPTER FIVE
EXTENDING FEATURE DEFINITIONS FOR ASSEMBLY MODELLING

5.1 INTRODUCTION

The objective of this research is to extend the knowledge of feature–based product representations by exploring their use as supporting tools for assembly modelling. This is achieved by incorporating assembly knowledge into the feature–based model established in Chapter 4. Section 5.2 discusses basic requirements of modelling an assembly and Section 5.3 outlines the general structure of the assembly and how parts are related in an assembly. An analysis of selected assemblies is presented in Section 5.4, the mating relationships among features in an assembly are defined in Section 5.5 and Section 5.6 shows the data representation in a model database. Section 5.7 outlines the method of inferencing assembly positions from the mating relationships. Section 5.8 describes how the assembly modelling knowledge can be related with the process planning knowledge. The assembly data structure is discussed in Section 5.9 and Section 5.10 gives the implementation of the assembly representation in an object–oriented environment.

5.2 MODELLING REQUIREMENTS

Assembly modelling deals with the interrelations between assembled parts. The general aims of assembly modelling have been defined in Chapter 1 and require the building of an assembly model to describe the part geometry and to define the relationships between parts of the final assembly. This requires a representation of the parts which captures all the information needed for their assembly and a data structure which stores information on how all the parts are connected in an assembly.

Zeid (1991) outlines three requirements necessary for assembly modelling:– modelling of individual parts that make up the assembly, specification of relationships between these parts and specification of the methods of determining the locations and orientations

of the parts in their assembled positions. The first requirement of modelling individual parts has been fulfilled with the use of feature–based geometric modelling as described in Chapter 4. The structure of the relationships between assembled parts is discussed in Section 5.3.

Determination of the correct location and orientation of each part to be assembled is crucial for assembly models. For an assembly of N parts, the goal is to locate and orientate N–1 parts with respect to the base or reference part to arrive at the final assembly. To do this, the position and orientation of each part in conjunction with the other parts in an assembly must be determined. This can be achieved either by assigning a transformation matrix to each part or by specifying mating relationships between assembled parts.

In the first approach, a 4 x 4 homogeneous transformation matrix can be assigned interactively and is used as an input to constrain the location and the orientation of each part in the assembly. The matrix transforms a reference coordinate system into a body coordinate system attached to the part, thus specifying the location and orientation of the part with respect to the reference coordinate system (Lee and Andrews 1985, Zeid 1991).

Although this approach has been used in many traditional CAD applications, there are a number of difficulties. The two principal difficulties are that the assignment of transformation matrices does not represent a natural interface for the designer, and the explicit nature of the representation does not allow for easy manipulation of the relationship during the interactive construction of a design. Thus for example, it is closer to the designer's thinking processes to "insert a bolt in a hole" than it is to define a set of constraints to the six degrees of freedom that would achieve the same result. The use of relationships such as "bolt in hole" also allows the specific (numeric) detail of the transformation to be implied rather than explicitly stated. This has benefits if interactive changes to the design (a dimensional change for example) are introduced as the relationship can remain constant while the derived transformation changes. In addition to these significant difficulties there are many practical problems that have to be overcome. Typical of these would be the solution of the matrix equations, the amount of time

consumed in generating the transformation matrices and the tendency to make errors due to the mathematical complexity (Lee and Gossard 1985).

In the second approach mating relationships between parts are defined and individual part positions and orientations can be automatically derived from these relationships (Lee and Andrews 1985). The orientations and positions can then be stored as transformation matrices. The computation of the transformation matrices from the mating parts can be used to determine whether a given assembly is possible. If no matrix exists which satisfies the mating conditions, then the parts cannot be assembled. This method of defining mating conditions can eliminate the problems resulting from direct assignment of the transformation matrix. This is the approach undertaken in this research work.

Once mating relationships are defined, the way in which the information is conveyed by the mating parts and stored in the database is also important so that it can be useful for later applications of the data. The following section discusses how the above requirements are represented in the feature–based model.

5.3 ASSEMBLY STRUCTURE

The focus of this research is the modelling of mechanical products. Most of these products are compositions of interconnected parts which are individually manufactured components and in this context are typically machined components. The approach adopted is to view a mechanical product as an assembly composed of one or several subassemblies, which themselves may consist of one or more components. Each component is made up of a base part (defined as a feature) and any number of features. Thus from the designer's point of view a feature forms a basic entity in the assembly of the product. In the following discussion, all assembled parts will be referred to as one of the feature types defined in Chapter 4. Thus a shaft is referred to as a boss, a through hole as a hole, a non–through hole as a pocket and so on.

The hierarchical structure of an assembly is represented by an Assembly Graph such as that shown in Figure 5.1. In the figure, the assembly is at the top–most level and features

are at the lowest level in the hierarchy. The dotted box in the figure shows that each feature can be further represented by a series of faces (within the geometric model).

The use of a hierarchical Assembly Graph provides the most efficient means of representation in the design of an object–oriented system, where lower levels in the hierarchy can inherit the properties of higher levels, while adding their own properties, as elaborated in later sections. The graph also reflects the way the designer views the assembly process as the progressive building up of the product from subassemblies and components comprised of individual features.

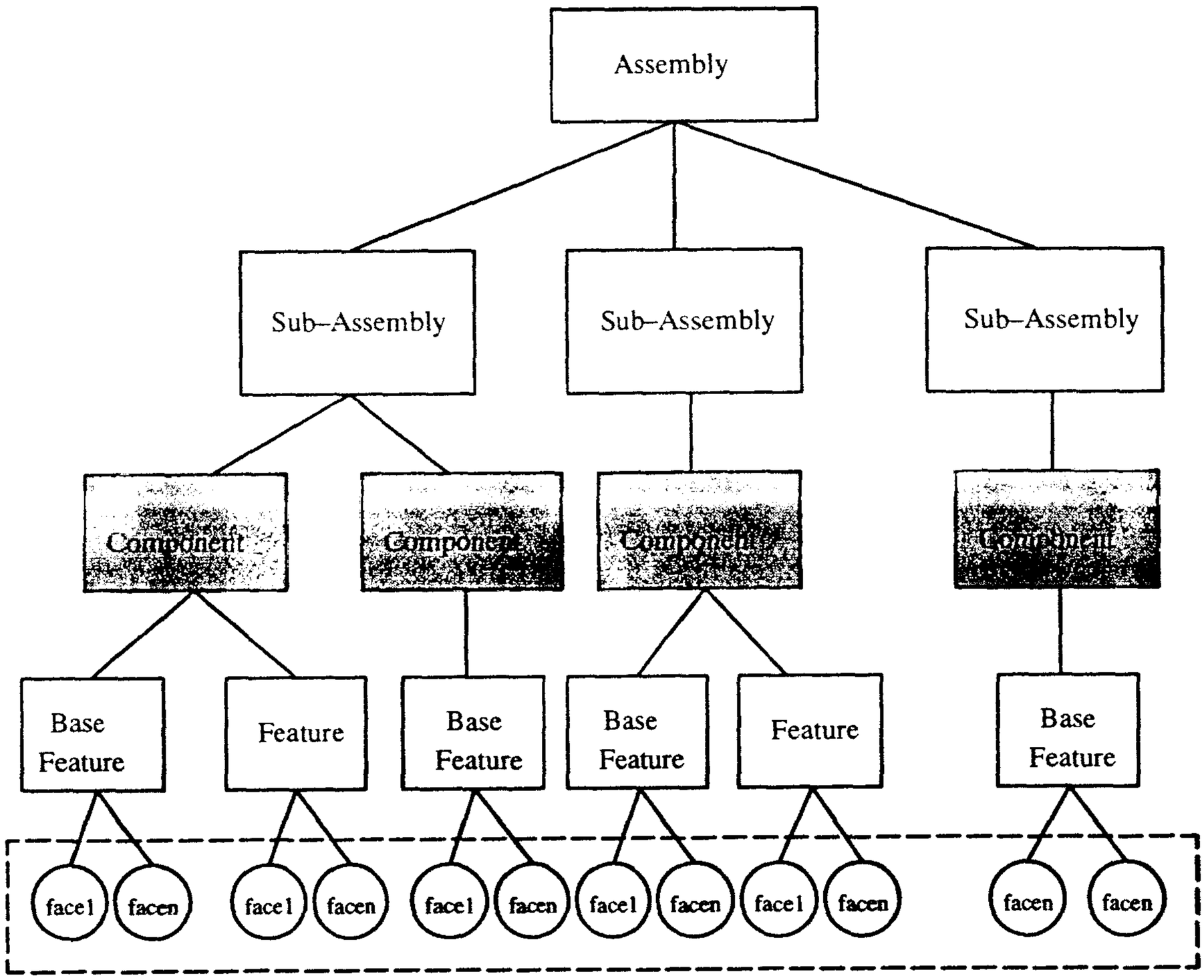


Figure 5.1: Product Assembly Graph

Figure 5.2a shows a lathe tool post assembly and illustrates the application of the hierarchical structure to a mechanical part. The lathe tool post could be viewed as consisting of two subassemblies (Figure 5.2b) – a post and a slide plus two components

(a nut and a washer). The post subassembly comprises two set screws and a tool post while the slide subassembly is comprised of two components – a tee bolt and a top slide. Going down to the feature level, each set screw is made up of two bosses of rectangular and circular profiles while the tool post consists of a boss, three holes and a through slot. The tee bolt which fits the slot in the top slide and passes through the hole of the tool post consists of a rectangular boss, two steps and a cylindrical boss. The top slide is made up of a boss, a through slot and two triangular notches. The features that go to make up the parts are shown in Figure 5.2c, and the Assembly Graph for the assembly is shown in Figure 5.3.

5.4 ANALYSIS OF ASSEMBLY

An analysis has been carried out to determine the type of mating relationships which occur in mechanical assembly and to relate them to the features which constitute the assembly. In order to achieve this, a number of typical assemblies have been selected to exhibit a range of characteristics that are considered to be representative of assemblies in general. The following procedures are carried out for each assembly:

1. The Assembly Graph is constructed.
2. The assembly relationships at the component level are analysed.
3. The assembly relationships at the feature level are analysed.
4. The relationships at the face level of each feature are analysed.

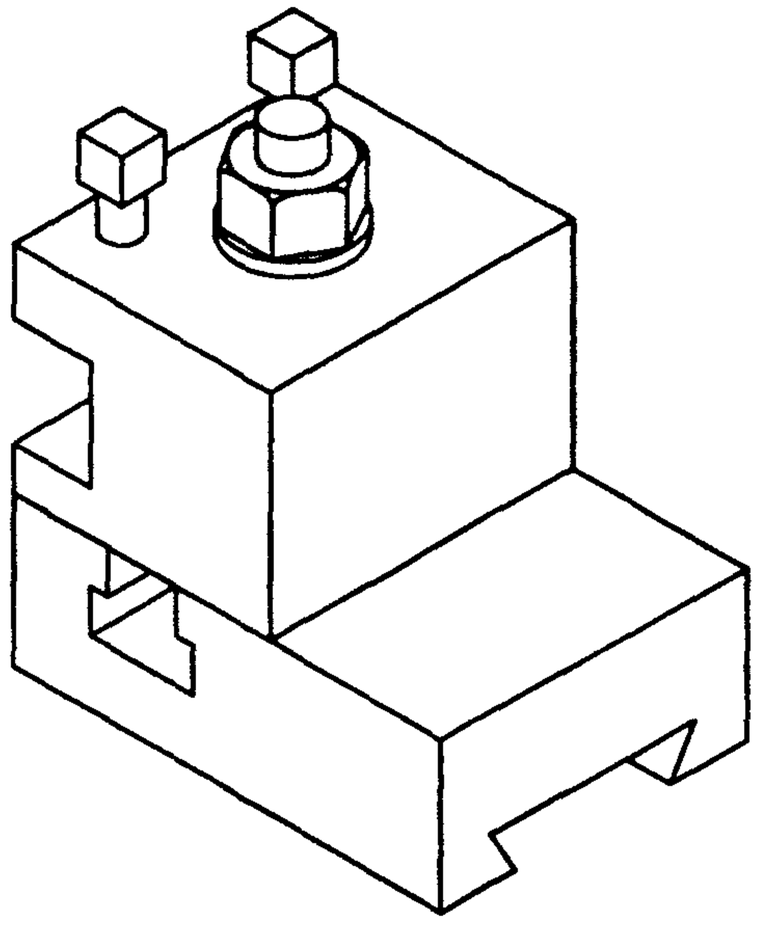


Figure 5.2a: Lathe Tool Post Assembly

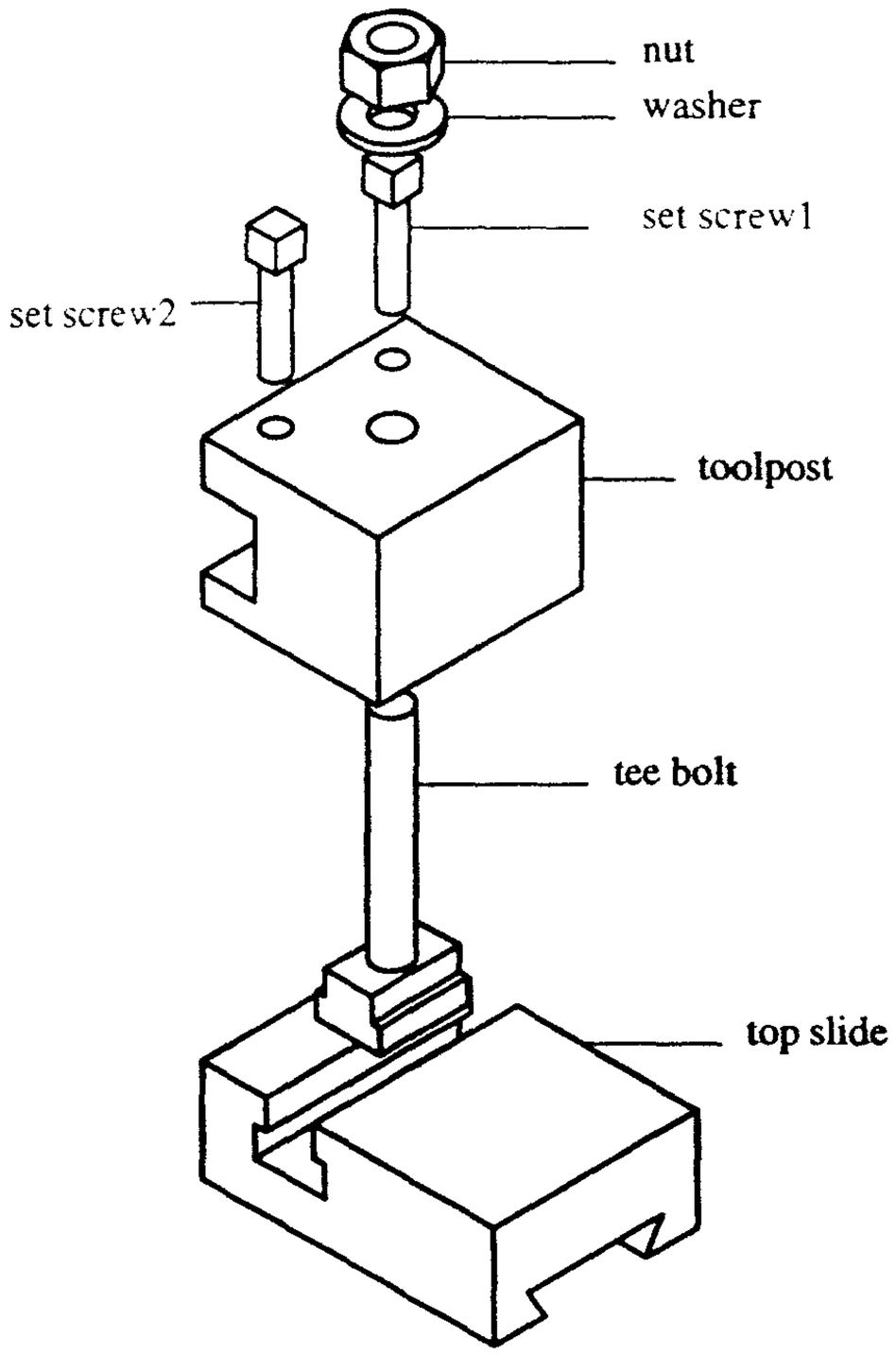


Figure 5.2b: Lathe Tool Post Components

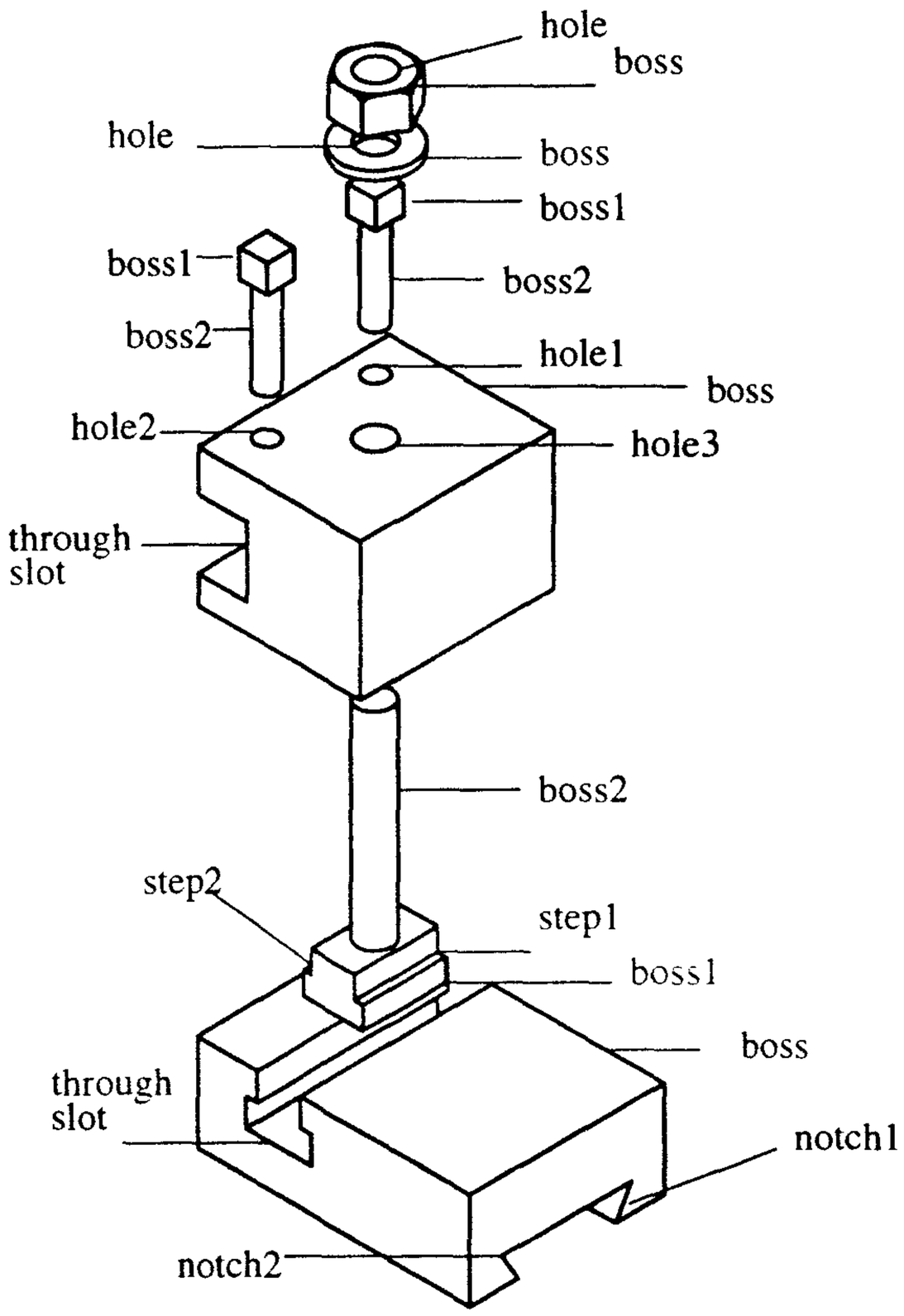


Figure 5.2c: Lathe Tool Post Assembly (features)

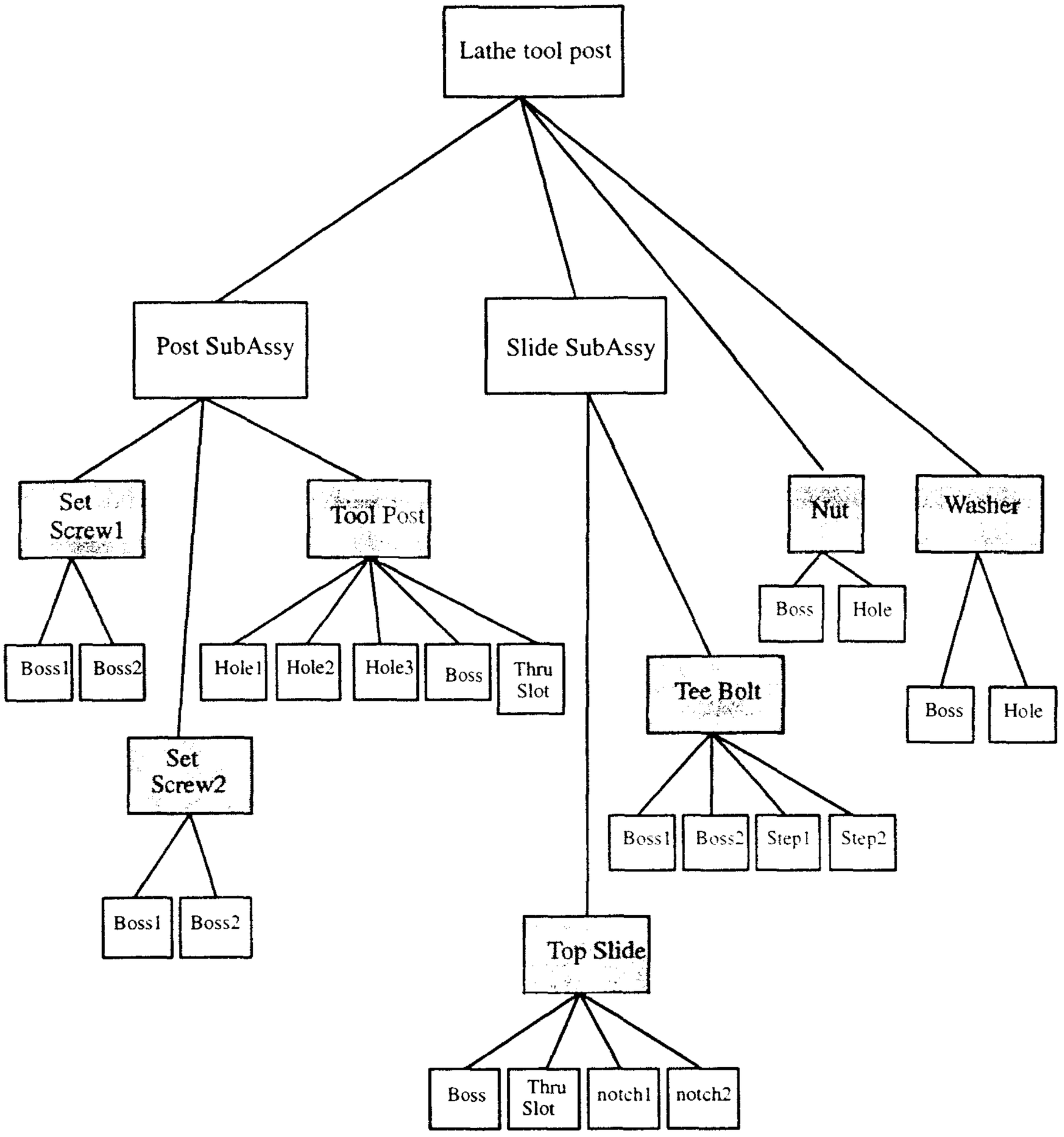


Figure 5.3: Assembly Graph for the lathe tool post

Three types of graph are used to represent the relationships at each level of interaction in the assembly hierarchy.

1. The *Component Relation Graph* shows the assembly relationships at the component level. Each component is represented by a rectangular node and a line (CR) indicates that a relationship exists between the two components. In a particular instance of a Component Relation Graph (e.g. figure 5.8) this single relationship is replaced by one

or more feature–to–feature relationships Rn. An example of a relationship between a bracket and a pulley is shown in Figure 5.4.

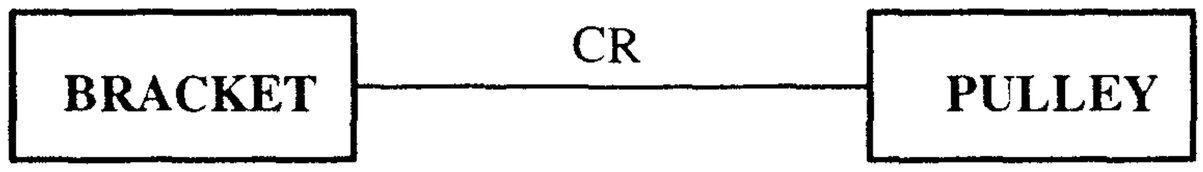


Figure 5.4: Notation for the Component Relation Graph

2. The *Feature Relation Graph* shows the relationship among features in their final assembled state. The graph shows how each feature in a component, represented by a circular node, is related to a feature or features from another component(s). The relationship is indicated by a bold line. A rectangular shaded box shows the components that make up the subassemblies. The thin line shows that the feature is part of the component. Figure 5.5 shows the notation for a relationship between a boss of a tee bolt and and the hole of a nut. Rn is the index number of the relationship.

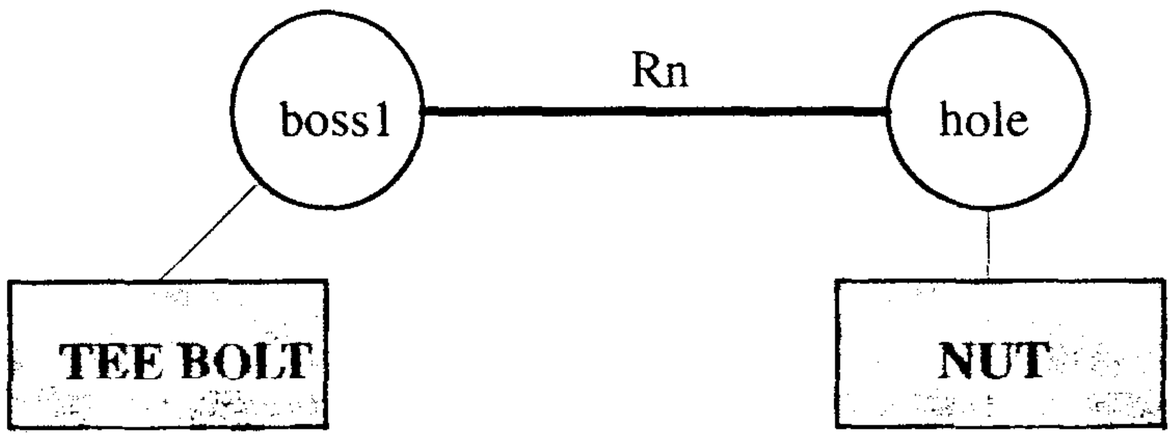


Figure 5.5: Notation for Feature Relation Graph

3. The *Face Mating Graph* shows the interaction at the face level of each feature. A face on a feature is represented by a small circle with a face number. A line indicates that there exists a relationship between two faces. Only real faces are considered in this graph and the number of faces that exist on each feature depends on the profile of the feature, as shown in Figure 4.6. The notation for the Face Mating Graph is shown in Figure 5.6, where face 1 of boss1 has a relationship with face 2 of boss2.

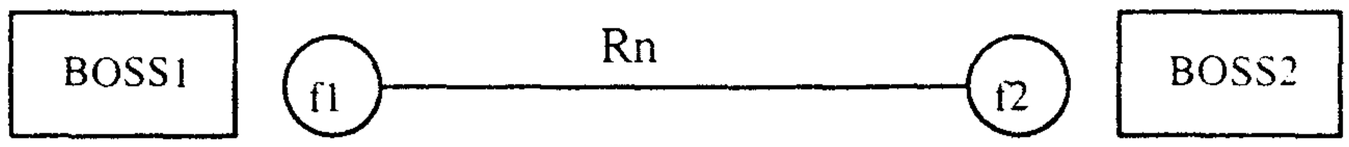


Figure 5.6: Notation for Face Mating Graph

These graphs help to visualise the relationships for each level of the assembly hierarchy and they also form the basis for constructing an appropriate class hierarchy and content. Their application in the analysis of assembly interfaces are shown by the examples described in the following sections.

5.4.1 THE LATHE TOOL POST

The lathe tool post, as shown in Figure 5.2 has been described in Section 5.3, and the Assembly Graph is shown in Figure 5.3. The assembly involves many types of features defined in Chapter 4 (Figure 5.2c). It also involves multiple components which results in many assembly interactions. Figure 5.7 shows a cross sectional view of the assembly. The existence of interactions between the parts in the assembled state are identified below and shown in figure 5.8. These are identified in section 5.6 as feature–to–feature relationships (illustrated in figure 5.9).

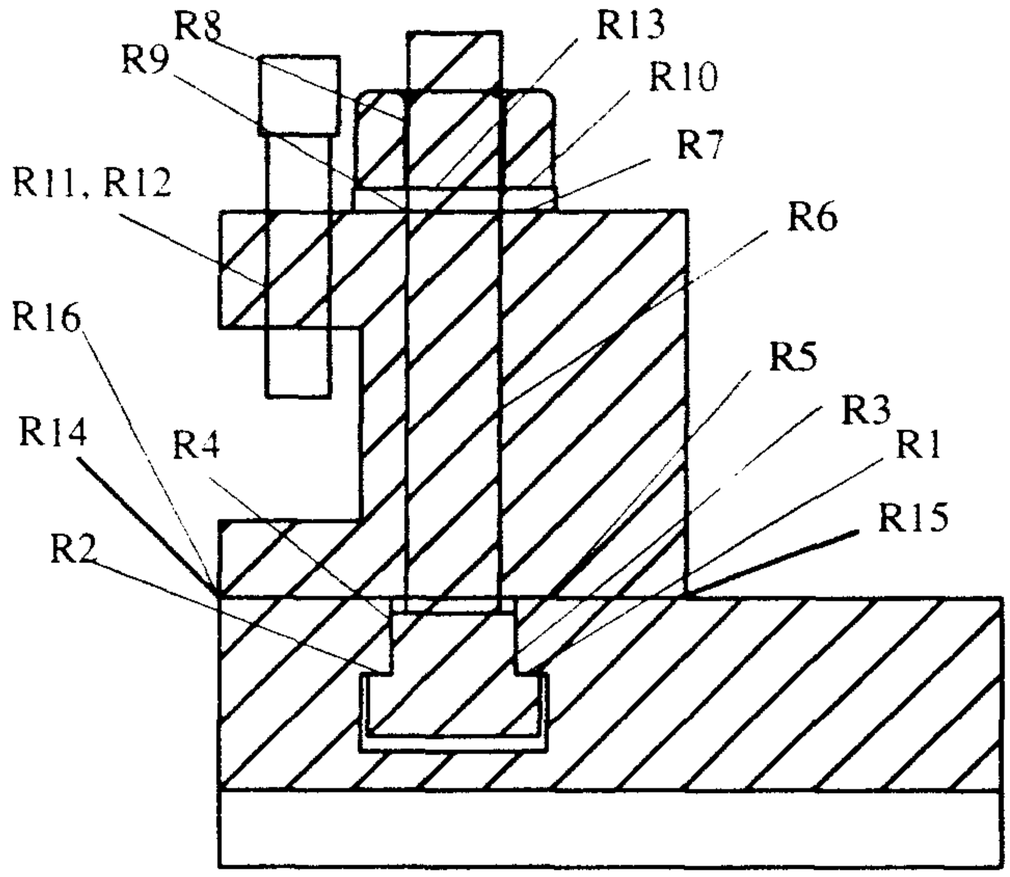


Figure 5.7: Cross sectional view of lathe tool post assembly

R1: step of tee bolt (right) and through slot of top slide

R2: step of tee bolt (left) and through slot of top slide

R3: vertical side of tee bolt (right) and through slot of top slide

R4: vertical side of tee bolt (left) and through slot of top slide

R5: bottom face of tool post and top face of top slide

R6: tee bolt and hole of tool post

R7: bottom face of washer and top face of tool post

R8: tee bolt and hole of the nut

R9: tee bolt and hole of the washer

R10: top face of washer and bottom face of nut

R11: shaft of set screw 1 and hole of tool post

R12: shaft of set screw 2 and hole of tool post

R13: hole of nut and hole of washer

R14: left side of tool post and left side of top slide

R15: front side of tool post and front of top slide

R16: back side of tool post and back of top slide

The relationships at the component level are represented by the Component Relation Graph as shown in Figure 5.8.

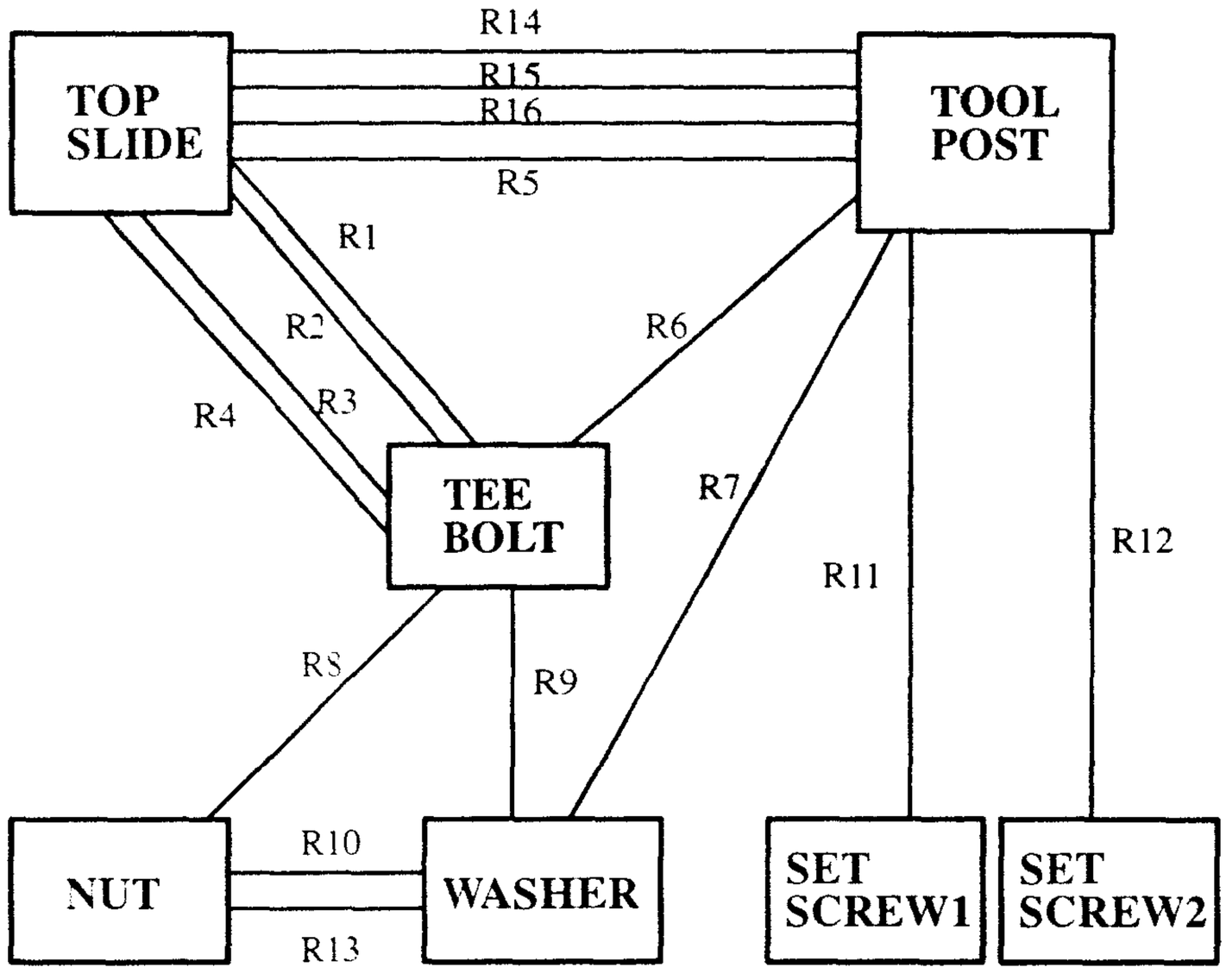


Figure 5.8: The Component Relation Graph for the lathe tool post assembly

As components are made up of features, the mating relationships at the feature level are examined using the Feature Relationship graph, as shown in Figure 5.9. Figure 5.10 shows the Face Mating Graph determining the existence of relationships at the face level for each interacting feature. The nature of these relationships is identified in section 5.6.

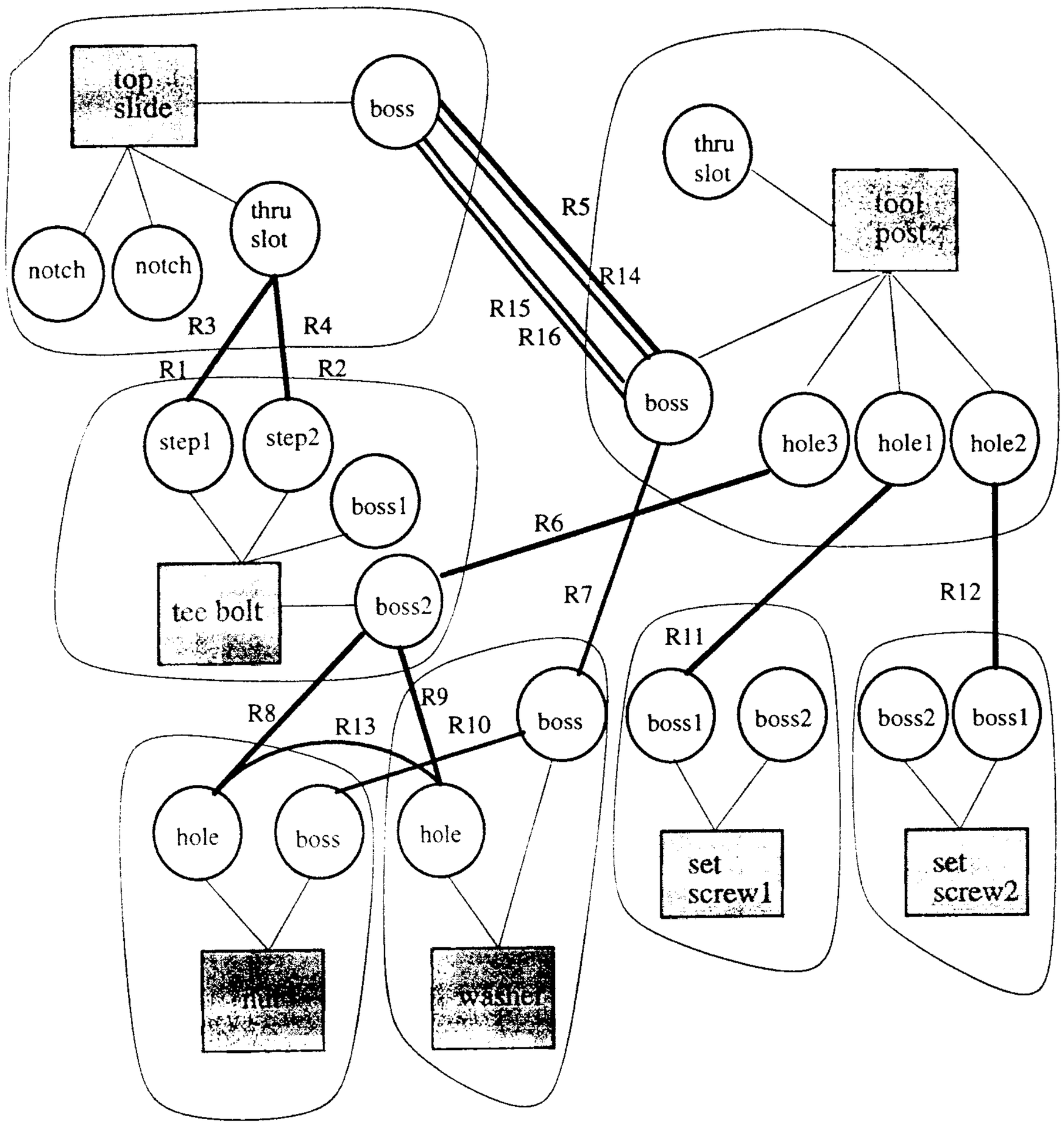


Figure 5.9: Feature Relation Graph for lathe tool post assembly

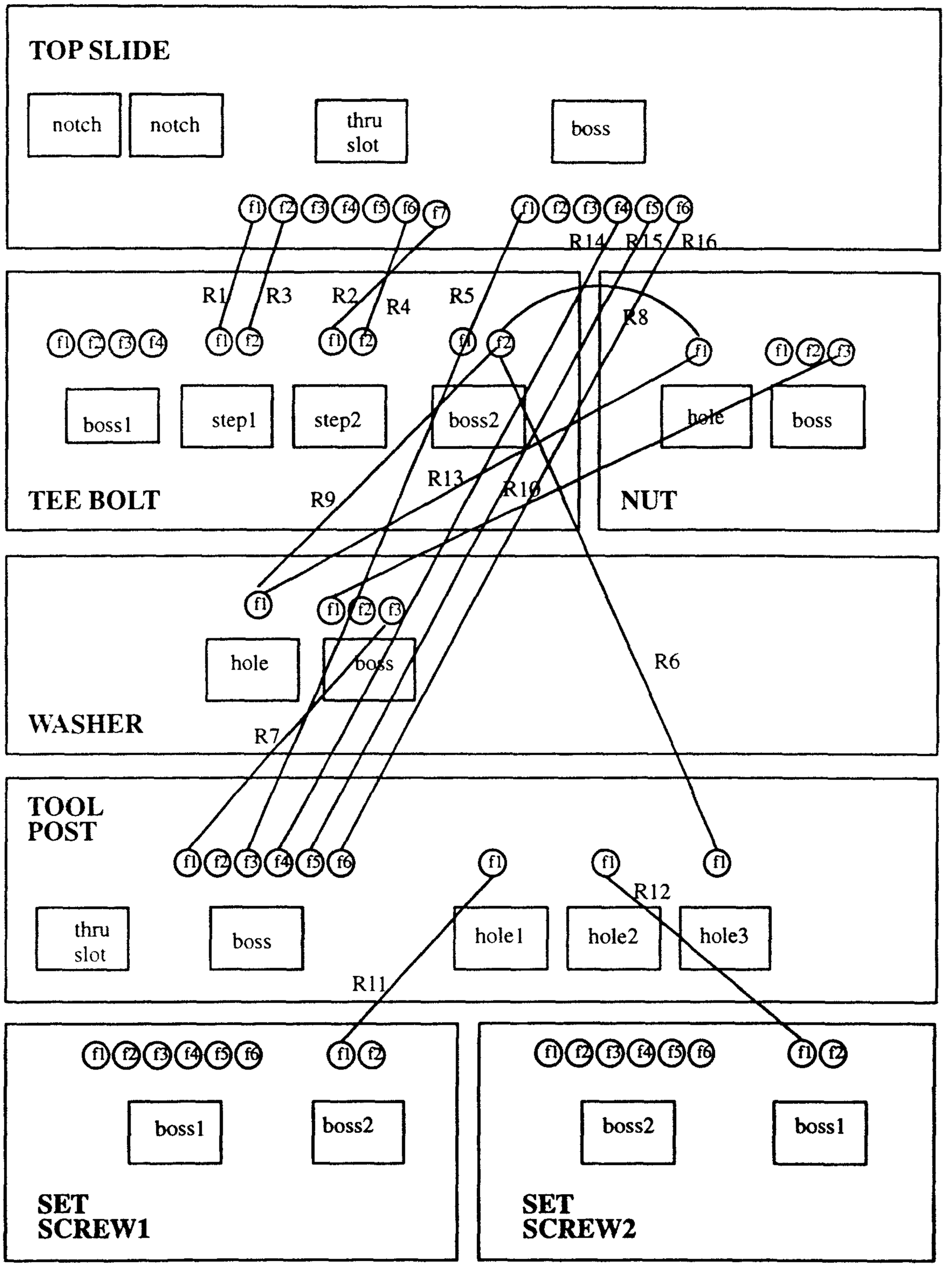


Figure 5.10: Face Mating Graph for lathe tool post assembly

5.4.2 BRACKET AND PULLEY ASSEMBLY

The bracket and pulley assembly, as shown in Figure 5.11a exemplifies the assembly of three cylindrical components and a key (Figure 5.11b). A shaft is assembled to a bracket and held by a key at one end and is assembled to a pulley at the other end. The Assembly Graph is shown in Figure 5.12.

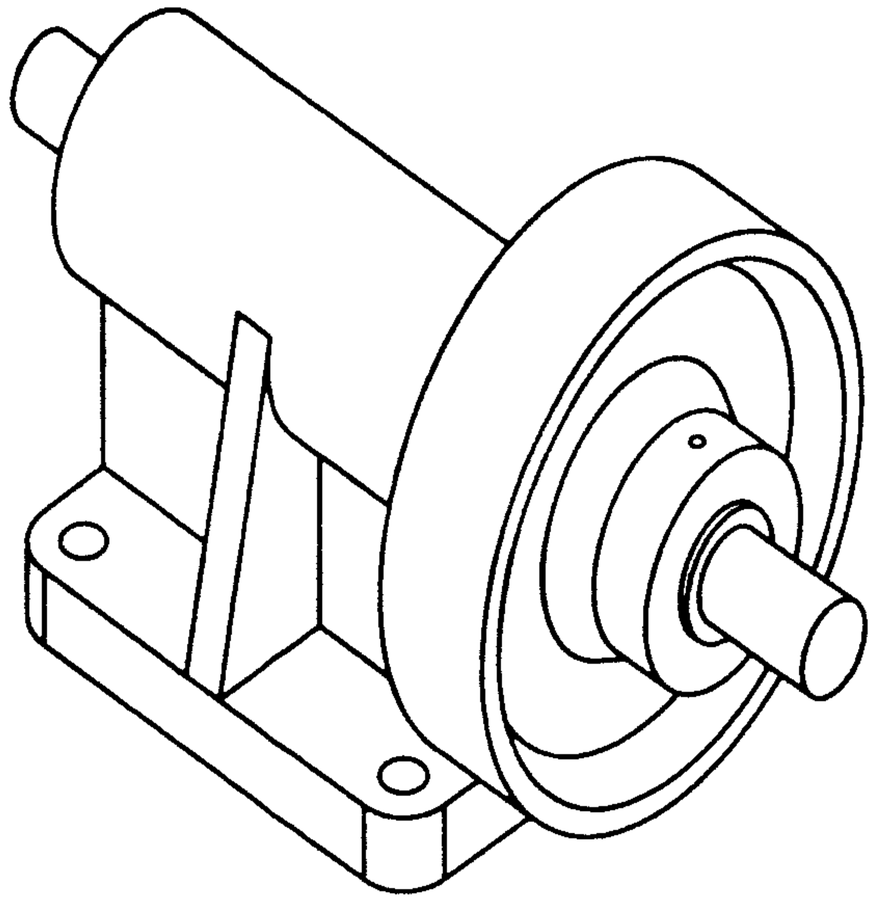


Figure 5.11a: Bracket and pulley assembly

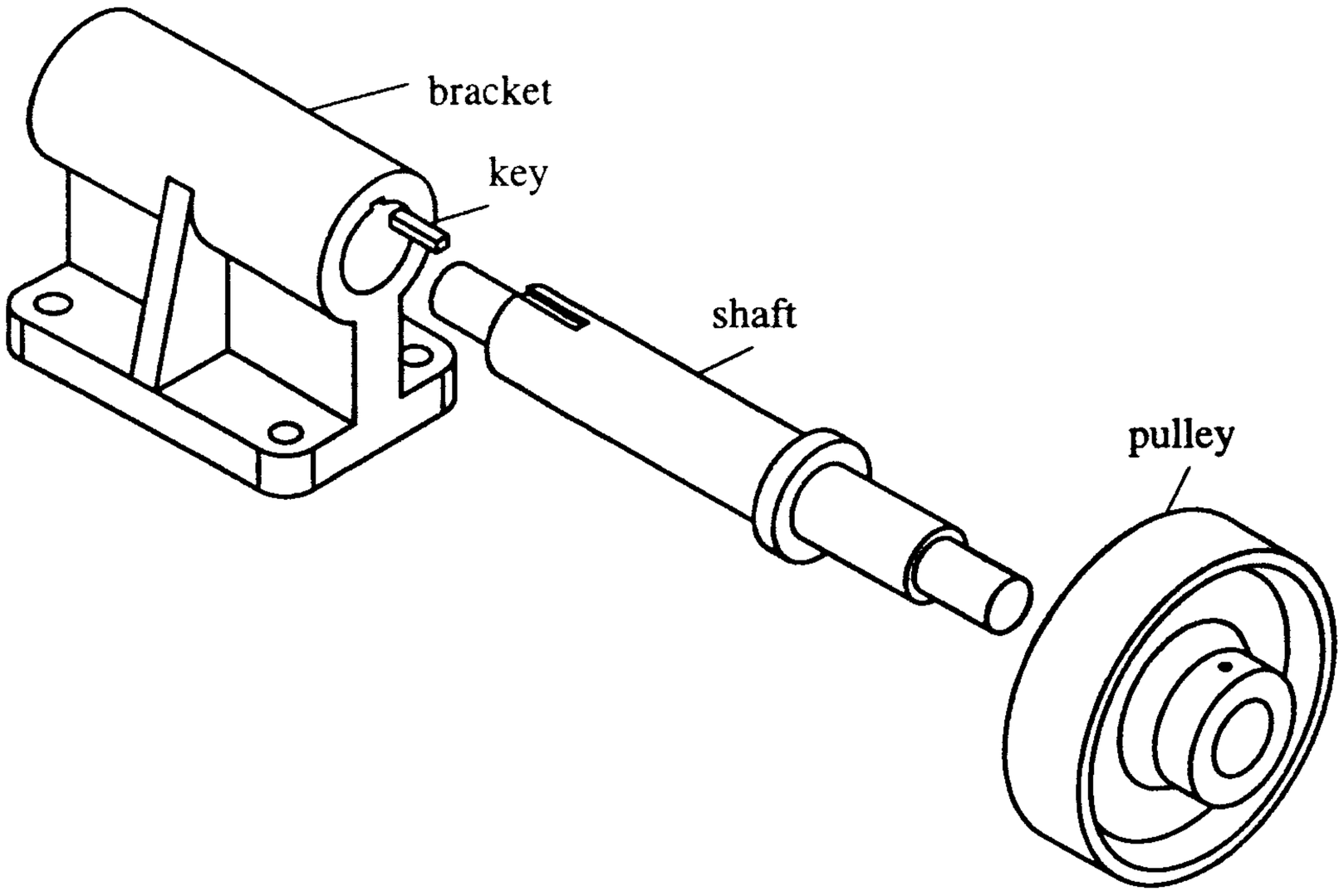


Figure 5.11b: Bracket and pulley components

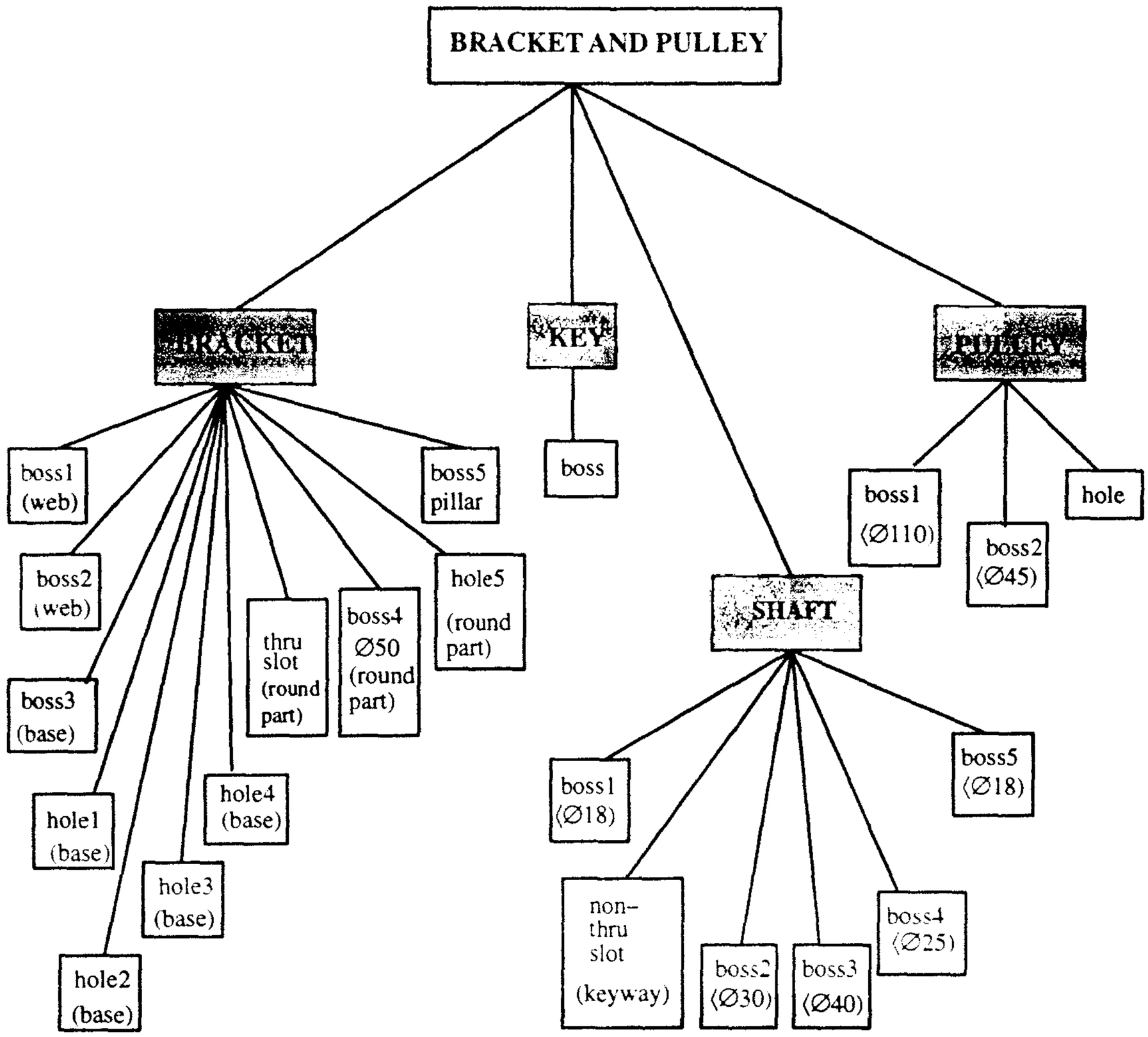


Figure 5.12: Assembly Graph for bracket and pulley

The mating relationships among the parts in the assembly are shown in a cross sectional view (Figure 5.13) and each pair of mating parts is listed in the following paragraph:

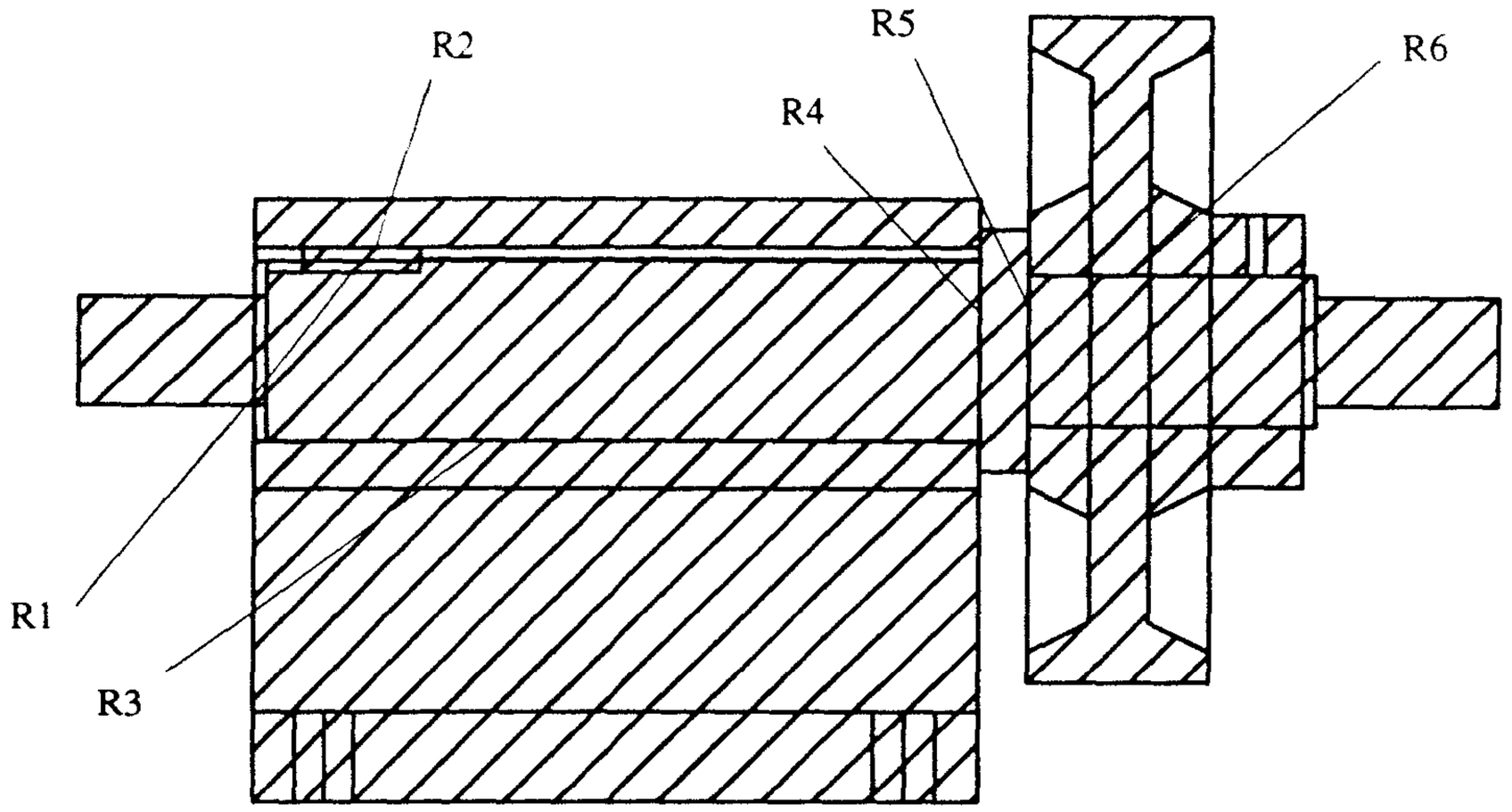


Figure 5.13: Assembly of bracket and pulley

R1: bottom face of key and keyway on shaft

R2: top face of key and keyway on bracket

R3: shaft ∅30 and hole of bracket

R4: flange ∅40 and face of cylinder ∅50

R5: flange ∅40 and face of pulley front ∅45

R6: shaft ∅25 and hole of pulley

R7: side of key and side of keyway on shaft

R8: longer side of key and side of keyway on shaft

R9: longer side of key and side of keyway on shaft

R10: longer side of key and side of keyway on bracket

R11: longer side of key and side of keyway on bracket

The Component Relation Graph for the bracket and pulley assembly is shown in Figure 5.14, while Figures 5.15 and 5.16 show the interactions at feature and face levels respectively.

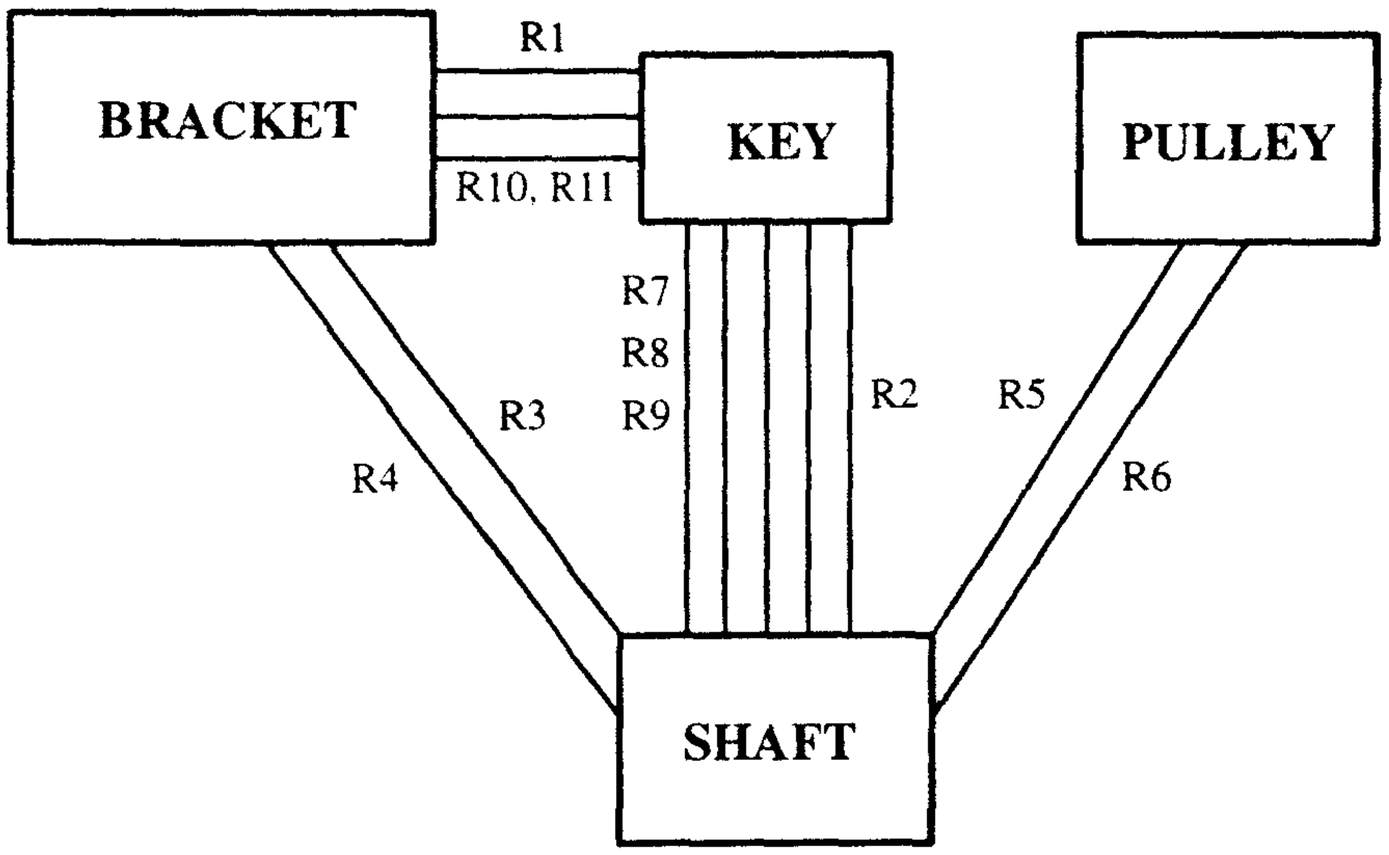


Figure 5.14: Component Relation Graph for bracket and pulley assembly

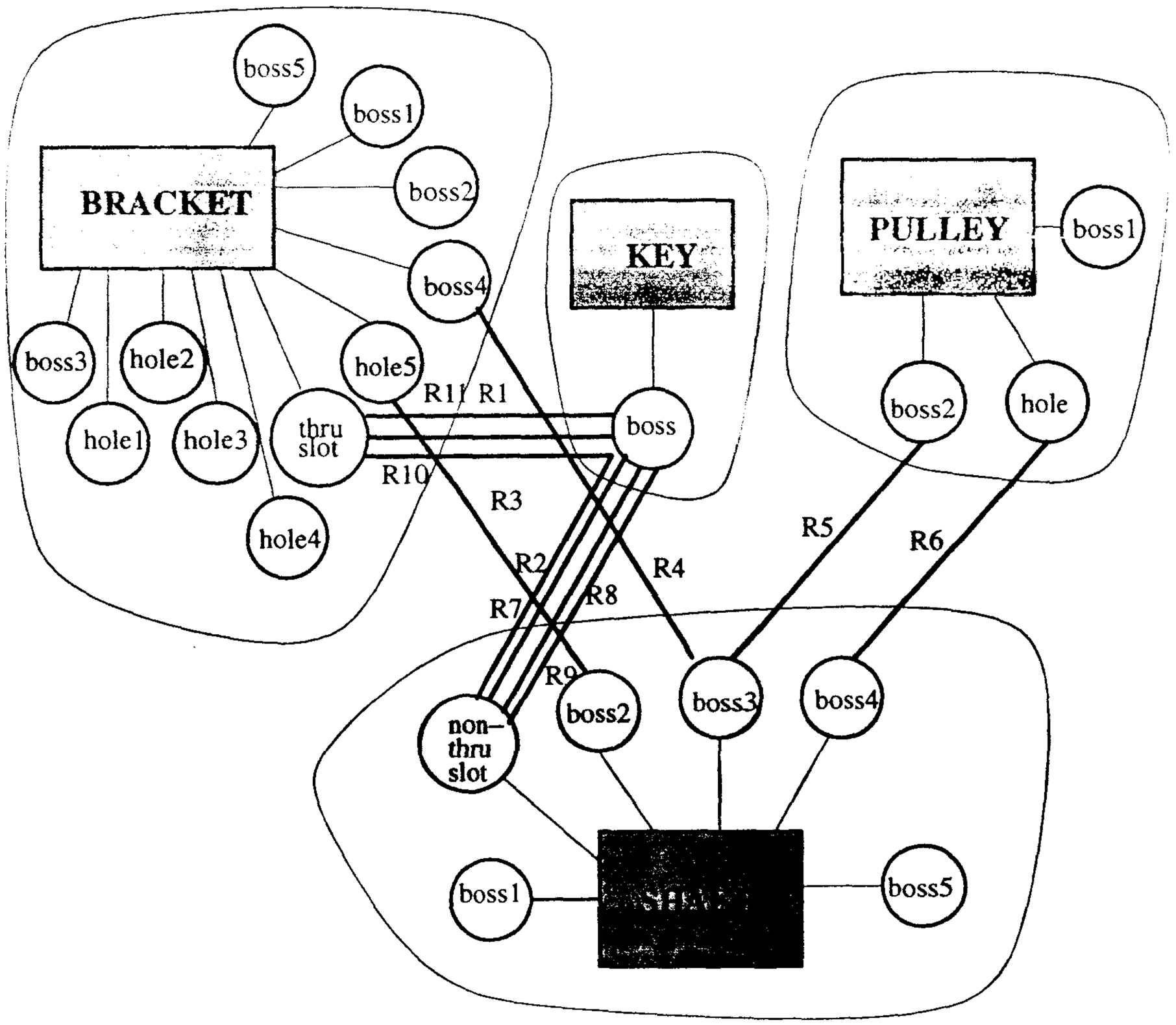


Figure 5.15: Feature Relation Graph for bracket and pulley assembly

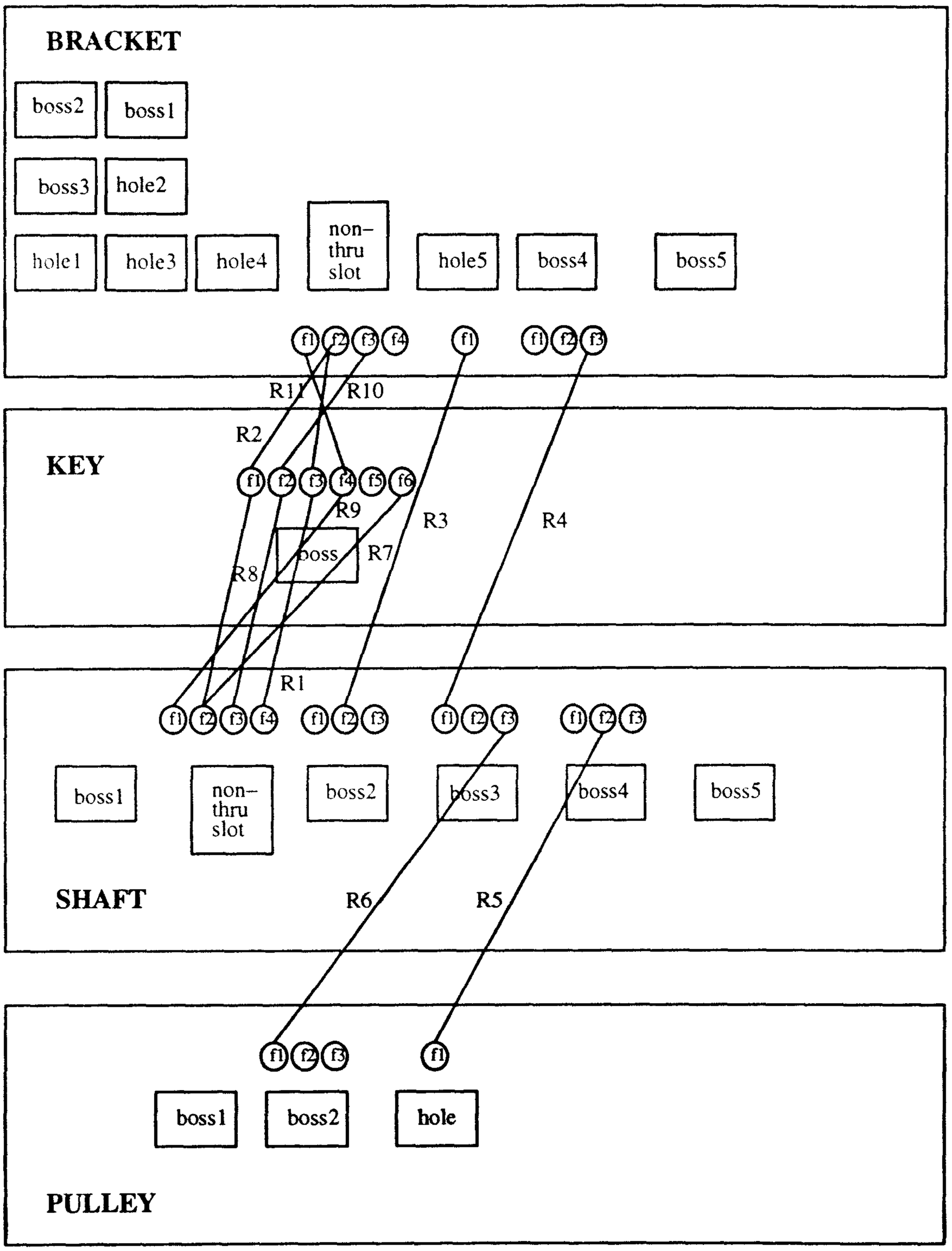


Figure 5.16: Face Mating Graph for bracket and pulley assembly

5.4.3 VALVE SUBASSEMBLY

Figures 5.17a and 5.17b shows a butterfly valve subassembly which consists of two housings, body1 and body2, fastened together by three nuts and bolts. The Assembly Graph is shown in Figure 5.18. Each body is made up of four bosses including a base feature of a cylindrical boss and four holes including three bolt holes. Each nut is made of a boss and a hole feature while each bolt is made up of two cylindrical bosses.

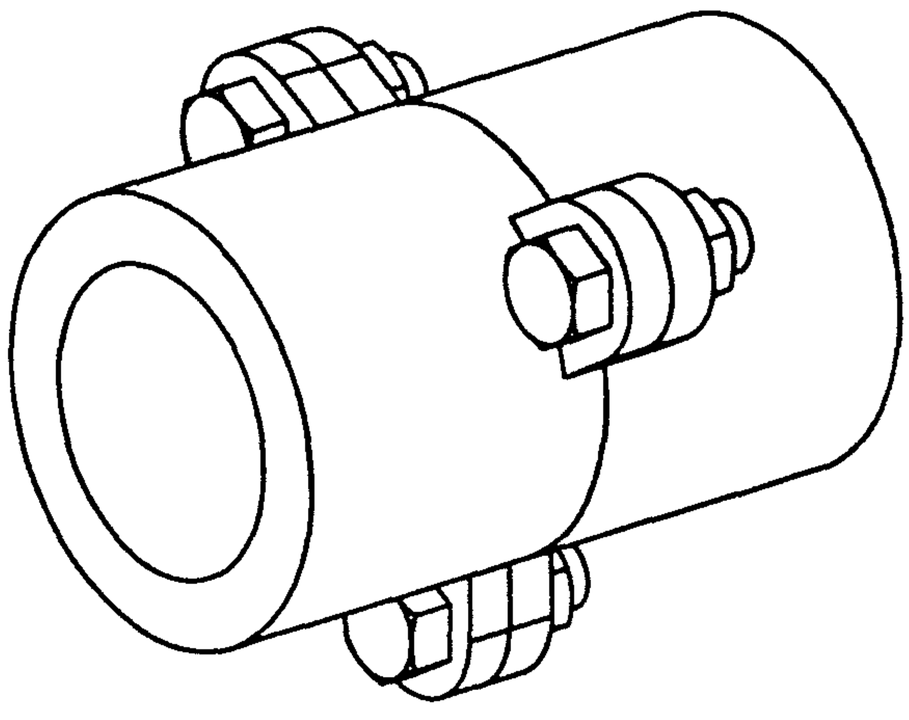


Figure 5.17a: Valve subassembly

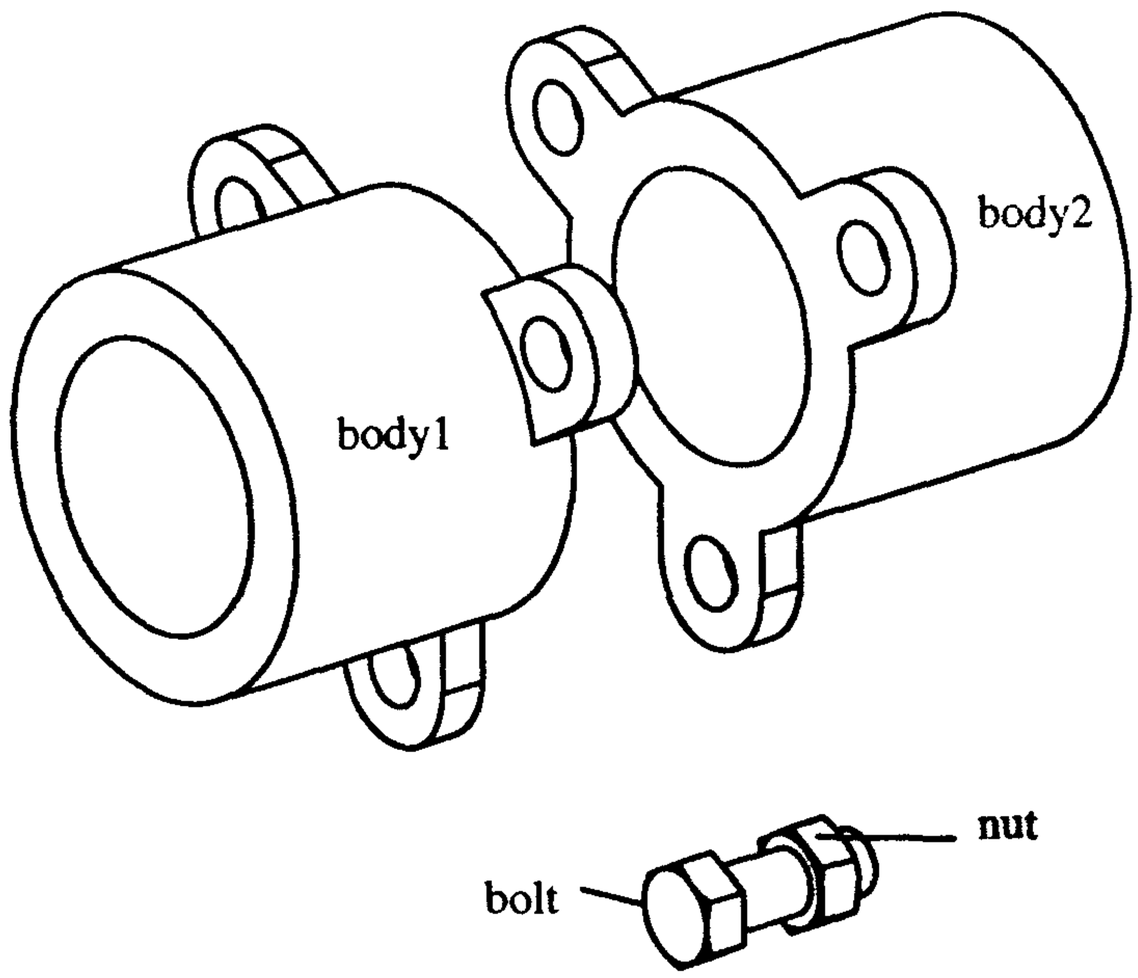


Figure 5.17b: Valve subassembly components

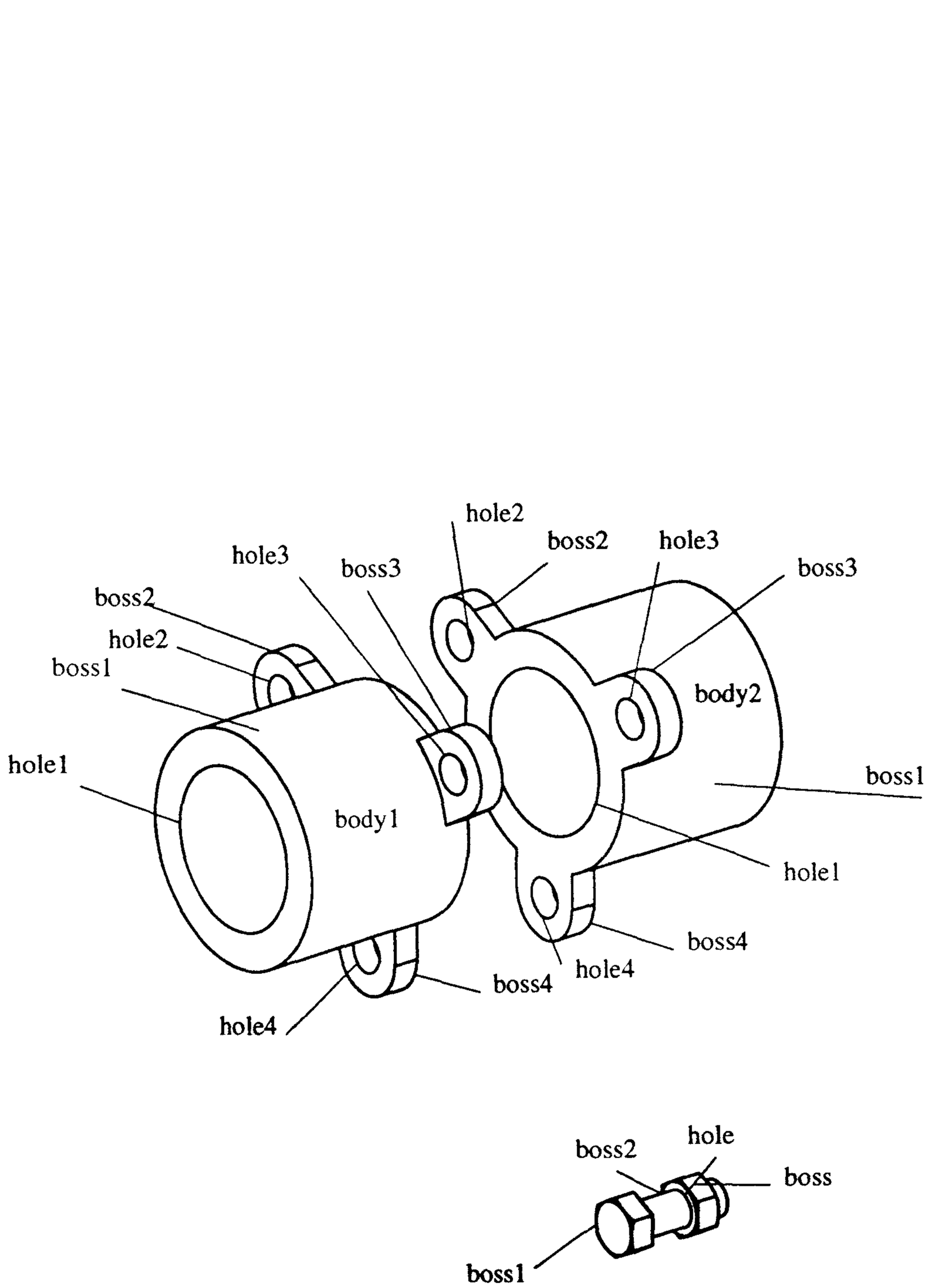


Figure 5.17c: Valve subassembly (features)

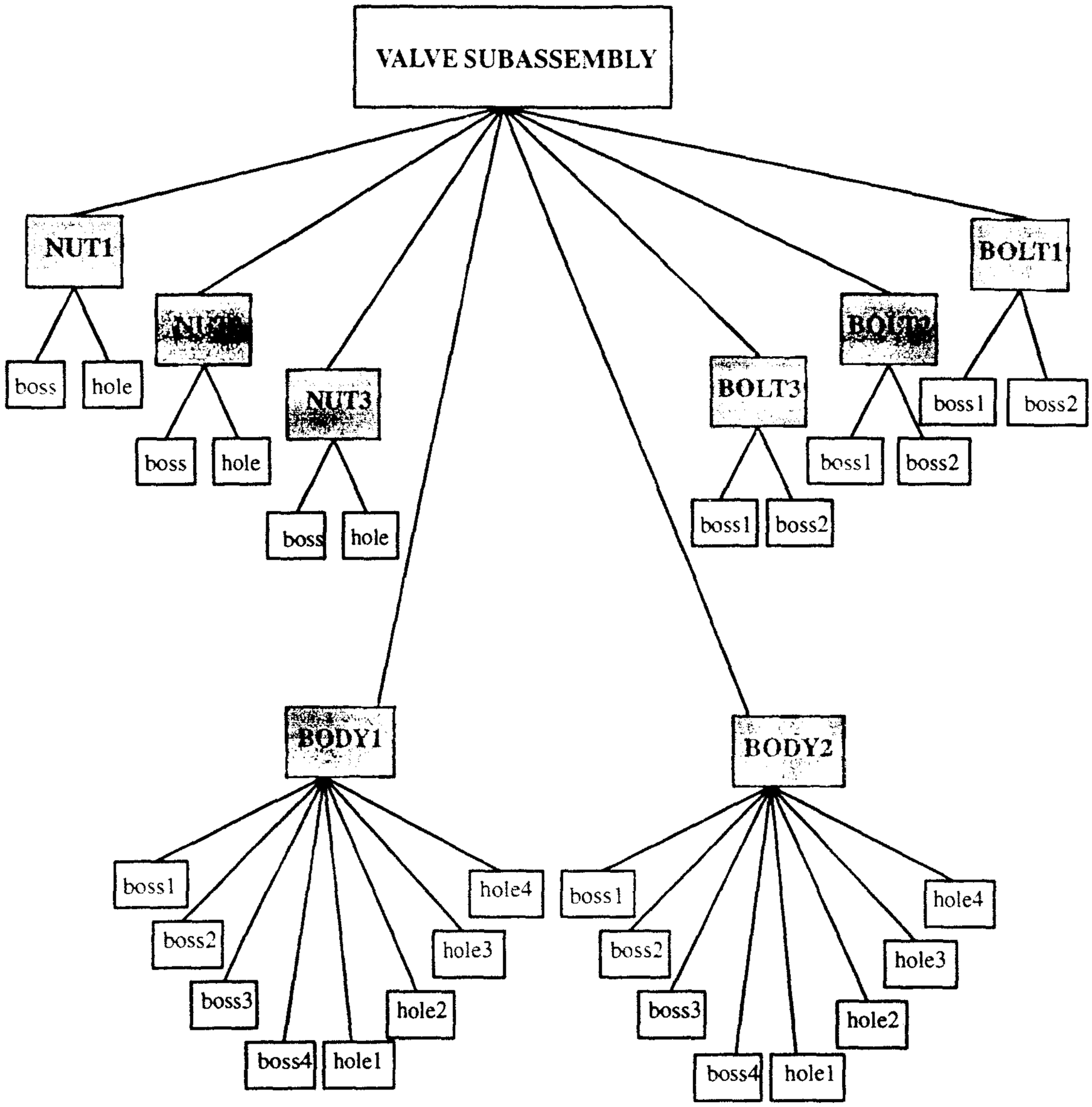


Figure 5.18: Assembly Graph for valve subassembly

Figure 5.19 shows a cross sectional view of some of the mating interactions which occur in the valve subassembly. The interactions are shown for the main bodies and one of the nuts and bolts. Similar interactions are repeated for the other two nuts and bolts.

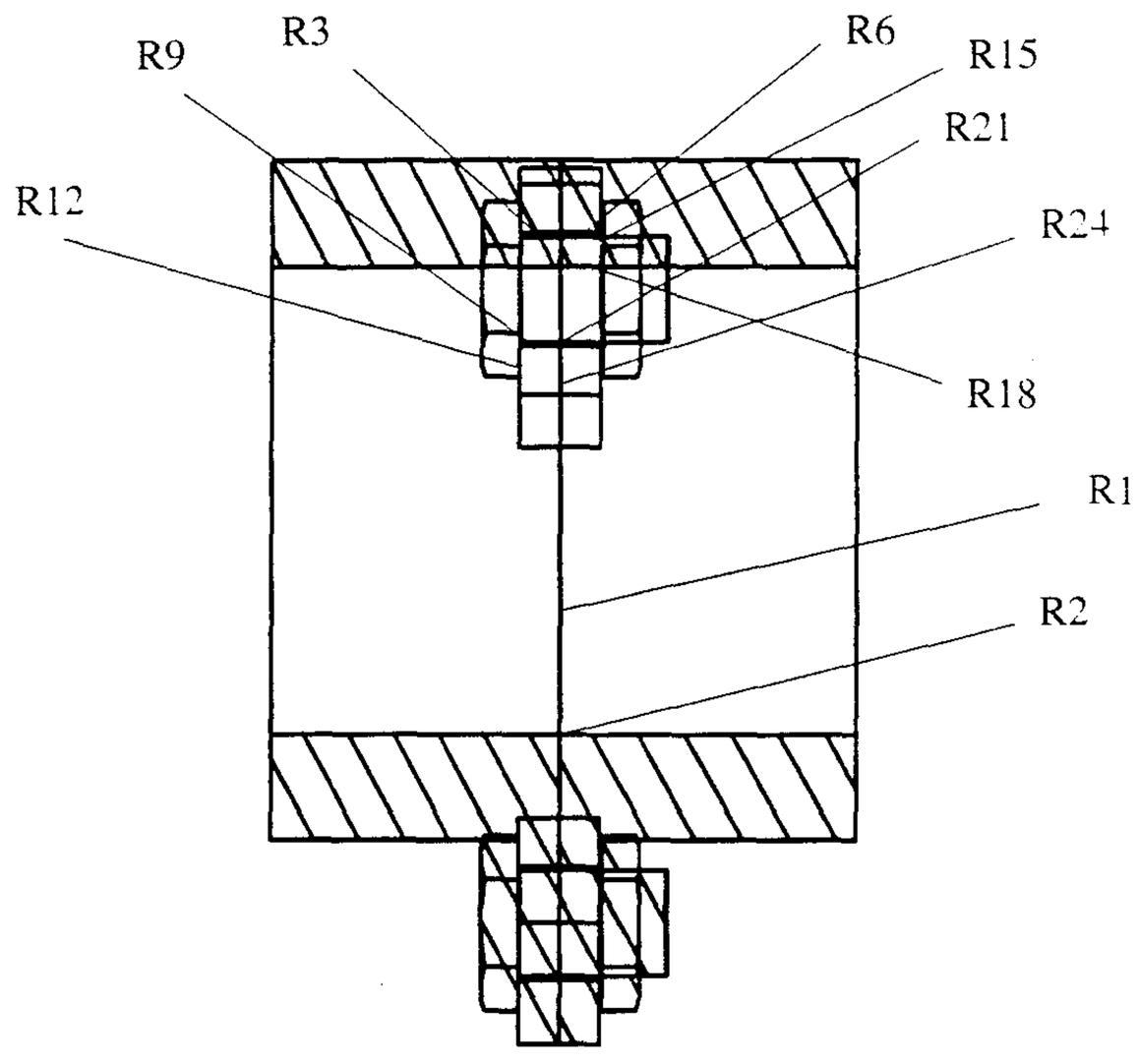


Figure 5.19: Cross sectional view of valve subassembly

The following mating relationships can be established from the above figure:

R1: face of body1 and face of body2

R2: hole1 of body1 and hole1 of body2

R3: bolt1 and hole2 of body1

R4: bolt2 and hole3 of body1

R5: bolt3 and hole4 of body1

R6: bolt1 and hole2 of body2

R7: bolt2 and hole3 of body2

R8: bolt3 and hole4 of body2

R9: bolt1 and hole of nut1

R10: bolt2 and hole of nut2

R11: bolt3 and hole of nut3

R12: head of bolt1 and face of boss2 of body1

R13: head of bolt2 and face of boss3 of body1

R14: head of bolt3 and face of boss4 of body1

R15: face of nut1 and face of boss2 of body2

R16: face of nut2 and face of boss3 of body2

R17: face of nut3 and face of boss4 of body2

R18: hole of nut1 and hole2 of body2

R19: hole of nut2 and hole3 of body2

R20: hole of nut3 and hole4 of body2

R21: hole2 of body1 and hole2 of body2

R22: hole3 of body1 and hole3 of body2

R23: hole4 of body1 and hole4 of body2

R24: face of boss2 of body1 and face of boss2 of body2

R25: face of boss3 of body1 and face of boss3 of body2

R26: face of boss4 of body1 and face of boss4 of body2

These relationships are shown by the Component Relation Graph in Figure 5.20. The interactions at the feature and face levels are shown in Figures 5.21 and 5.22 respectively.

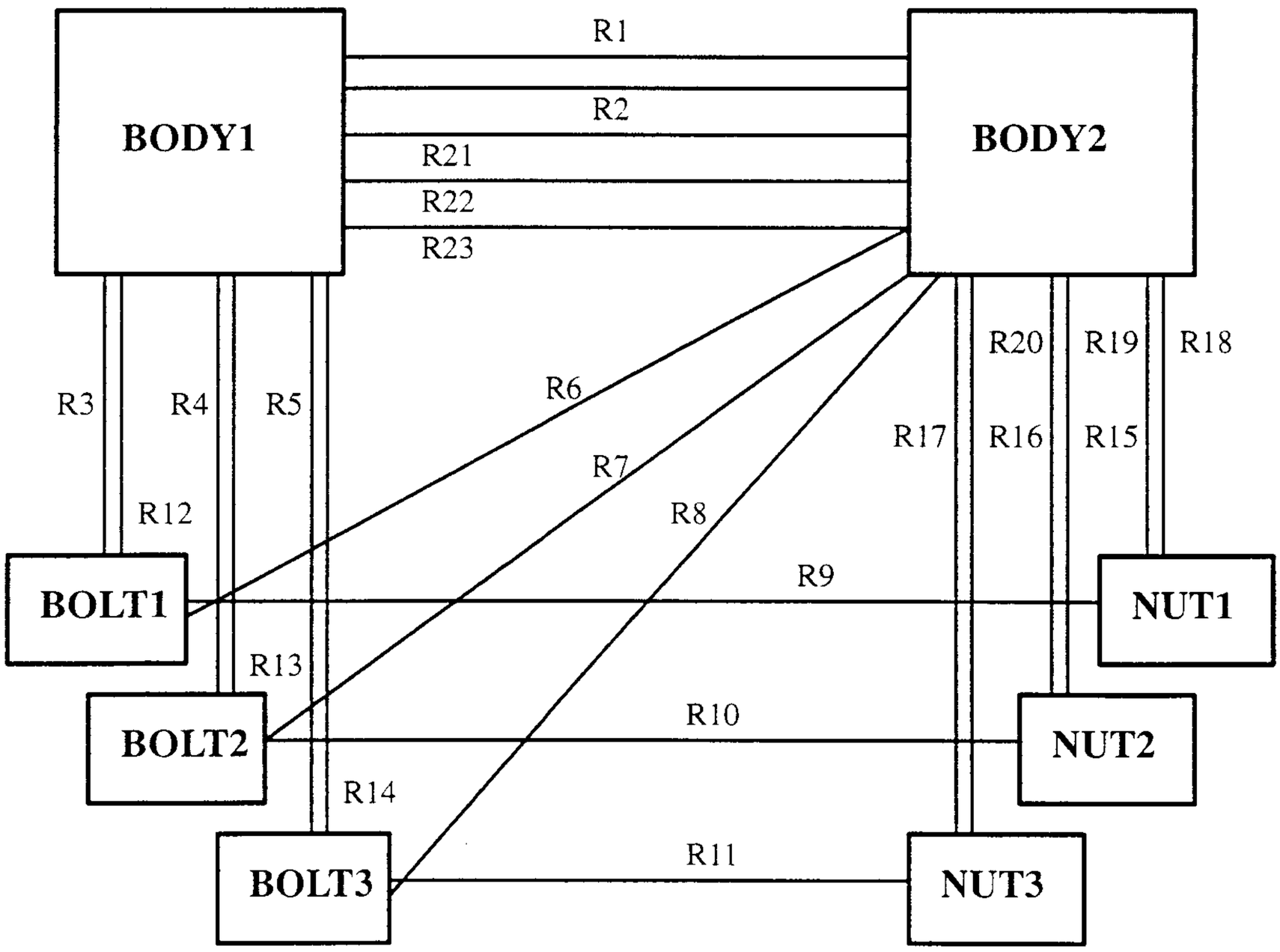


Figure 5.20: Component Relation Graph for valve subassembly

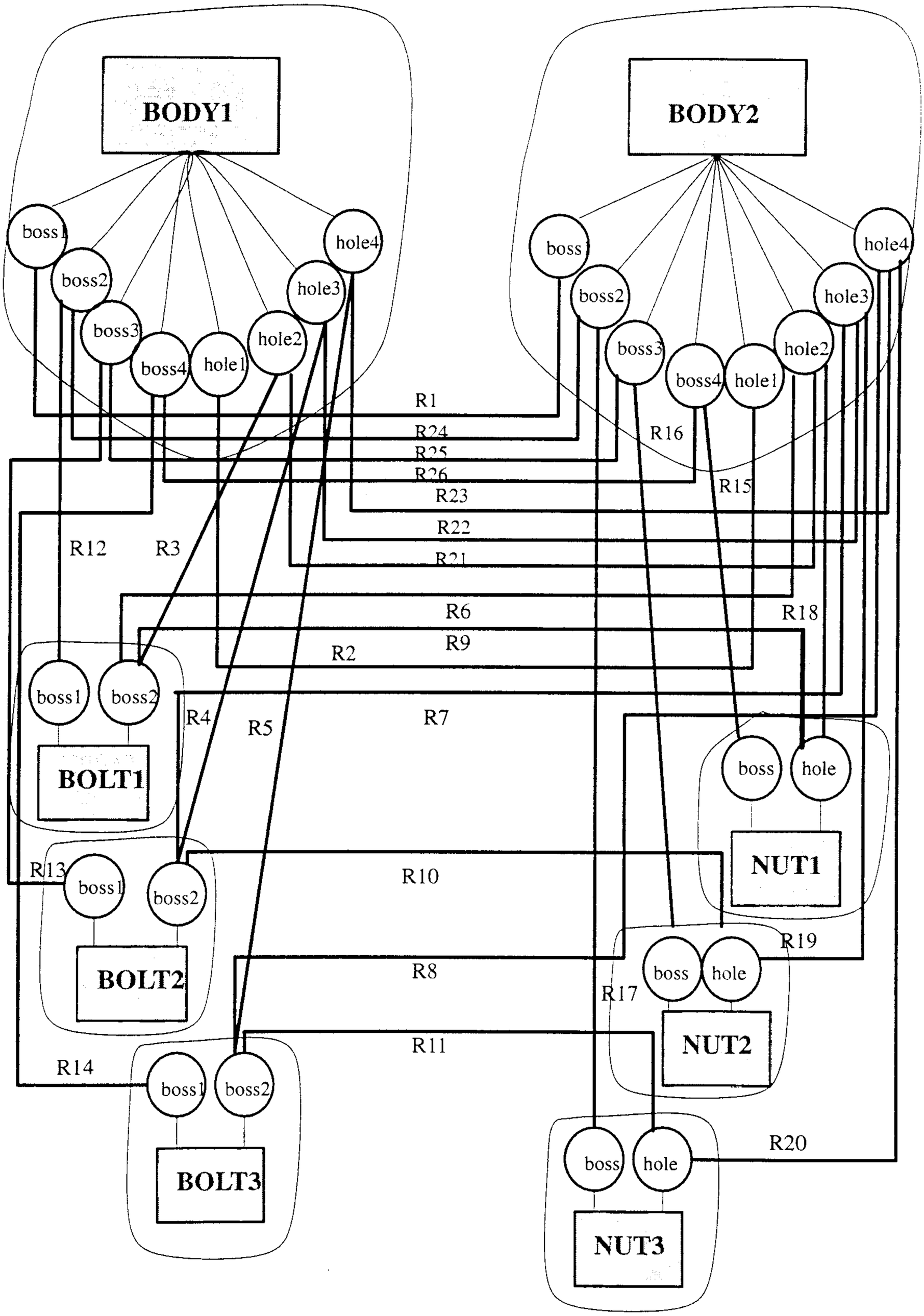


Figure 5.21: Feature Relation Graph for valve subassembly

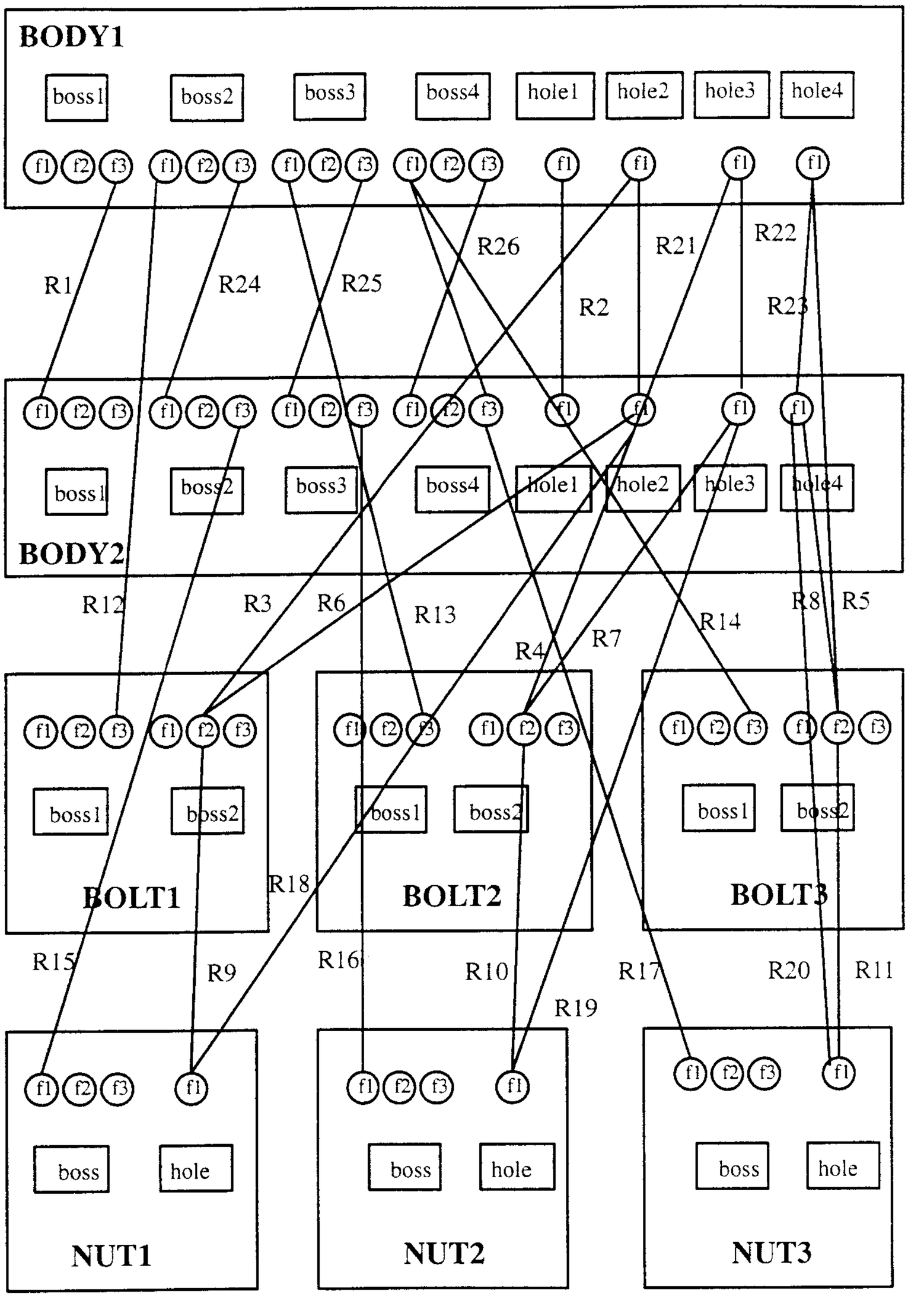


Figure 5.22: Face Mating Graph for valve subassembly

From the analysis, the following observations are made:

1. Assembly relationships exist at three levels:

   i. the component level identifies assembly interactions at the highest level from which it should be possible to determine potential methods of creating alternative sets of subassemblies. This would be of use in assembly planning.

   ii. the feature level presents a useful way for the designer to define the assembly methods in some detail, and also provides a valuable link with feature–based process planning. It has been shown that all features defined for process planning (except for pocket and notch, which are not available in the examples) have assembly interactions with other features.

   iii. the face level represents the level normally contained within the geometric model and allows sufficient information to be included to fully constrain the assembly.

2. More than one assembly interaction can occur at the component, feature and face levels. Examples of multiple interactions for the component level are:

   tool post and top slide

   key and shaft

   body1 and body2

   At the feature level, more than one interaction occur at the following features:

   boss top slide

   boss key

   hole2 of body1 of valve subassembly

   At the face level, multiple interactions occur at

   f2 of boss2 of tee bolt

   f2 of non–through slot of bracket

   f1 of boss4 of body1

3. Assembly interactions occur at the face level as well as for two holes having collinear centre points. The assembly analysis, as shown in Figures 5.10, 5.16 and 5.22

indicates that contacts between features involve one of the following pairs of compatible surfaces:

- between two planar faces
- between a shaft and a hole
- between a hole and another hole
- between two planar faces aligned in the same plane

These are summarised in Figure 5.23.

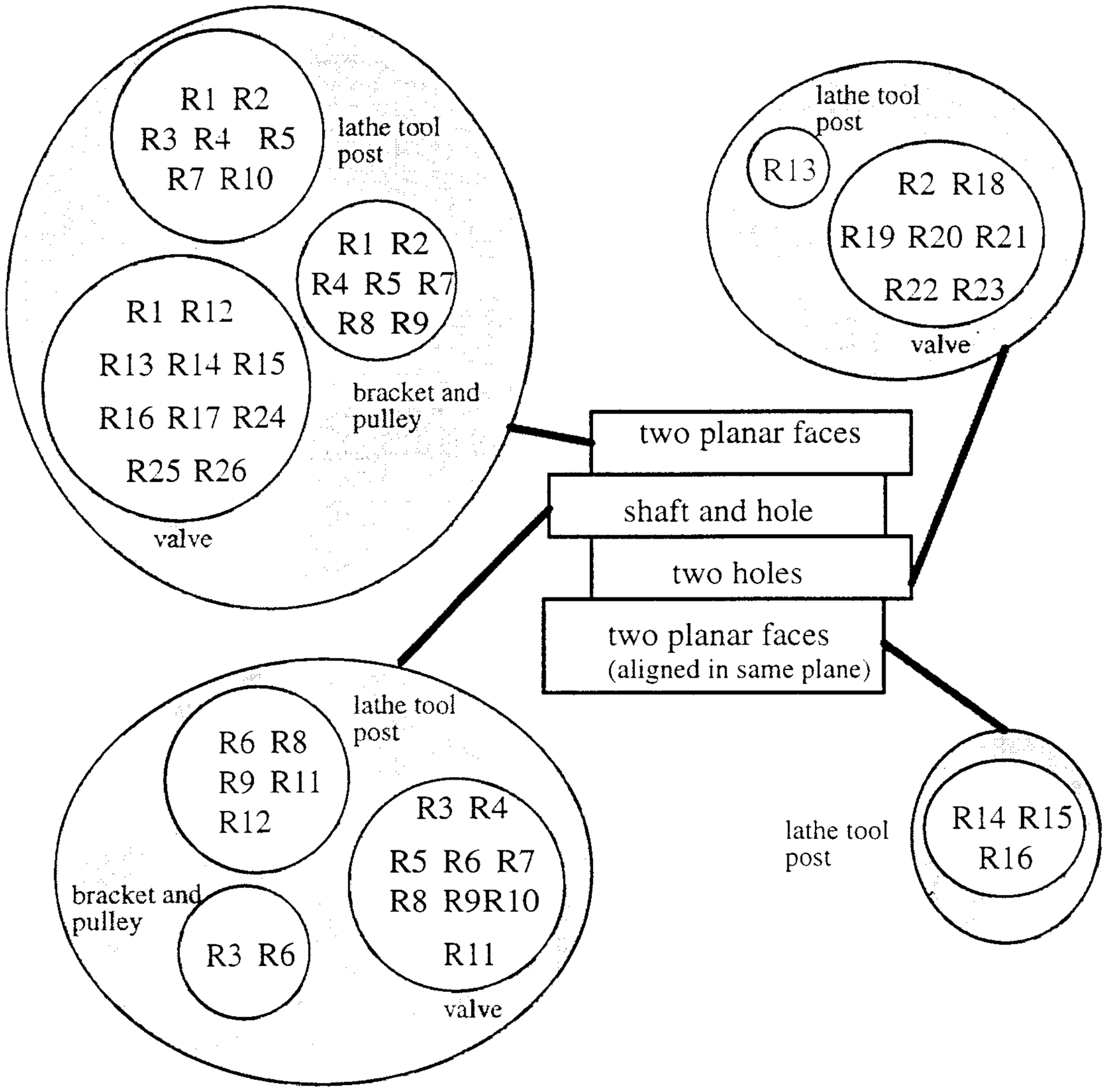


Figure 5.23: Categories of assembly interactions

The main purpose of the analysis was to categorise assembly interactions into groups such that distinguishing characteristics of the group could be identified and used as the basis of specifying Feature Mating Relationships. This aspect is considered in the next section.

5.5 FEATURE MATING RELATIONSHIPS

Features in an assembly are said to have a mating relationship whenever they have one or more faces in physical contact with another feature, although there are occasions such as magnetic fixing where this is not strictly true but these situations are not considered here. This requires the definition of possible mating relationships for each feature and the representation of these relationships in a form suitable for assembly modelling.

From the analysis of Section 5.4, three basic mating relationships can be defined – *against, fits* and *align*. These relationships are defined based on established terms used by various researchers in assembly modelling, such as Lee and Gossard (1985) and Ko and Lee (1987) and are explained in the following paragraphs:

i. *Against*

This is a mating relationship between two planar faces or between a planar face and a cylindrical face. The condition exists when two or more features are either stacked on top of one another or they are placed adjacent to each other with at least one of their faces touching. The *against* condition can be specified along any of the three major axes (x, y and z axes) together with the two adjoining faces and the direction of the contact. Figure 5.24a shows the *against* condition between two rectangular bosses placed adjacent to each other with faces f1 and f2 to be mated. The *against* condition is satisfied by forcing the normal vectors to faces to be in opposing directions and establishing contact between the two faces. Figure 5.24b shows an *against* condition between a rectangular boss and a cylindrical boss, a situation that is not commonly found but which is included to maintain the completeness and generality of the feature representation. Examples of planar surface *against* relationships are between the bottom of the tool post and the top of the top slide

(R5) of the lathe tool post assembly and between the end face of body1 and the end face of body2 (R1) of the valve subassembly.

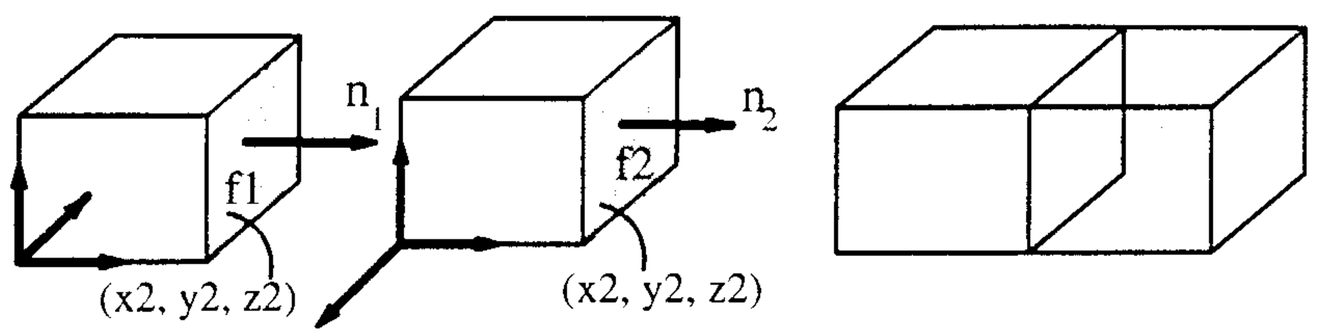


Figure 5.24a: Against condition for two rectangular bosses

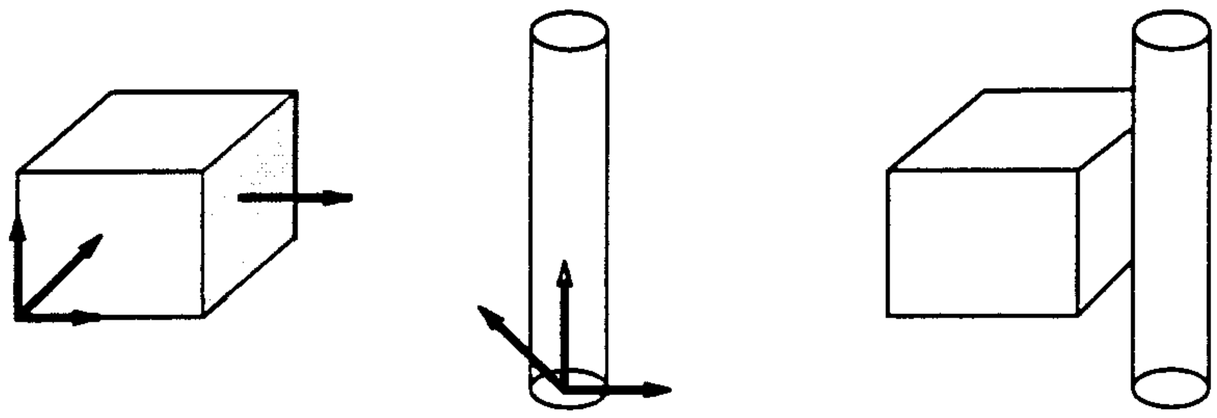


Figure 5.24b: Against condition for rectangular and cylindrical bosses

ii. *Fits*

Fits is a mating relationship occurring when two features are required to fit together with clearance or interference. The condition holds between a shaft (boss) cylindrical face and a hole cylindrical face or between a polyhedral shaft (boss) and polyhedral hole. In the typical cylindrical case it allows both rotational and translational freedom of movement between the mating features. Non–cylindrical fits result in a single translational degree of freedom. This requires the centrelines of each feature to be collinear. The *fits* condition between a hole and a cylindrical boss is shown in Figure 5.25.

Fits can be further classified according to the degree of difficulty of assembly and the method used to assemble the parts. Some of the types of *fits* relevant for the assembly examples shown in earlier sections are *tight fit*, which is an interference fit, *screw fit* which involves assembly of threaded items and *clearance fit*. For example the assembly

of shaft ∅30 and hole of bracket (R3 for pulley and bracket) can be considered as a *tight fit*.

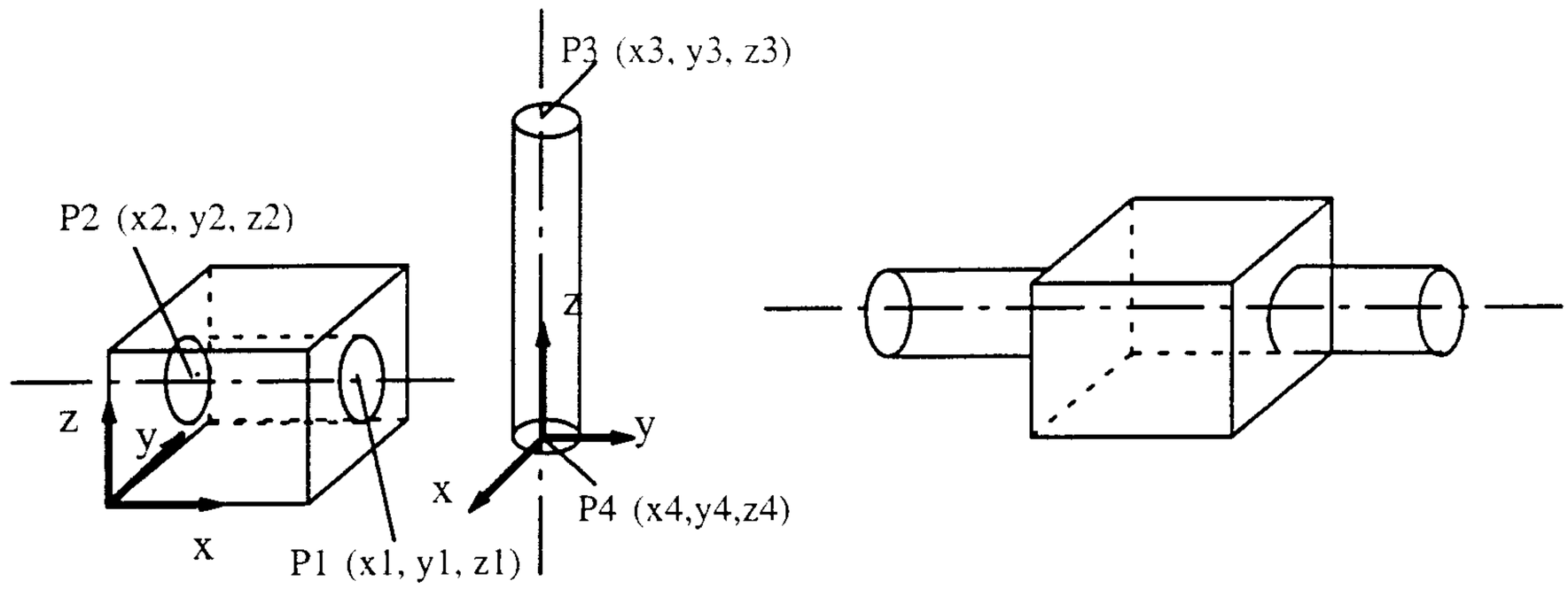


Figure 5.25: A Fits condition for a cylindrical boss and a hole

iii. *Align*

Align is a mating relationship which exists in two situations – between two holes and between two planar faces. In the former situation, it requires the centre line of one hole to coincide with the centre line of another hole. In Figure 5.26a, in order to achieve the *align* condition, point P1 on hole1 should be coincident with point P2 on the hole2. Examples of *align* relationships are between holes of the valve subassembly bodies (eg R2) and between the washer and nut holes (R13) in the lathe tool post assembly. An *align* relationship between two planar faces exists when the faces (f1 and f2) lie on the same surface as shown by Figure 5.26b. An example of such a relationship is that between the faces of the boss of top slide and the boss of tool post (R14, R15 and R16) in the lathe tool post assembly.

In an assembled position, mating could occur over one or more faces that may or may not be adjacent. Thus each feature needs to be checked for the possibility of mating with every other feature. As a general guide to the possible types of mating relationship occurring between one feature and another, a *Feature Relation Table* is developed, as

shown in Table 5.1. In this table, each feature type is assigned a possible relationship with each other feature type.

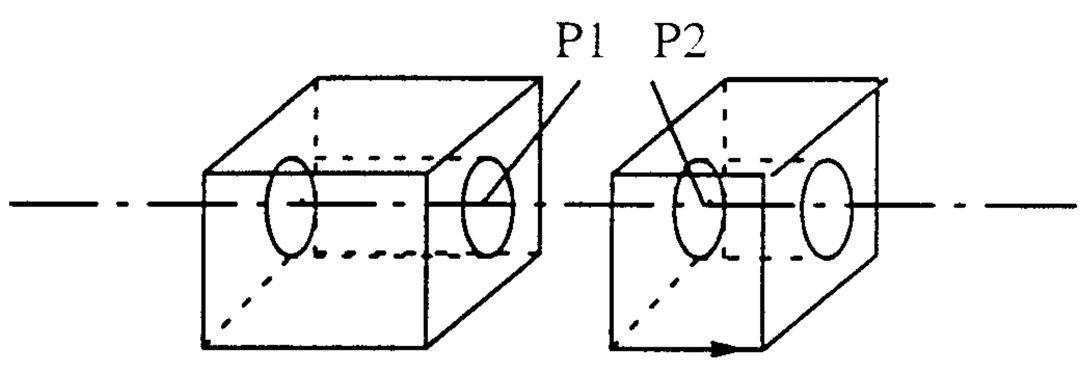


Figure 5.26a: Hole alignment

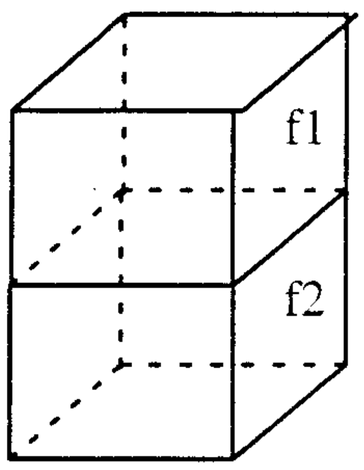


Figure 5.26b: Alignment of planar faces

In the table, letters denote the type of relationship defined earlier: A for *against*, F for *fits*, and L for *align*. X denotes that there is no possible relationship between the feature types. This data is used as an input to the relationship database in the feature relationship class described in Section 5.10.

The above definition of mating relationships is sufficiently general to encompass the class of mechanical assemblies using all types of features defined in Chapter 4. The inference of the location and orientation of a part in an assembly from the mating relationships above requires the computation of its transformation matrix from these conditions. The matrix relates the feature's local coordinate system to the global coordinate system of the assembly. The approach is to infer the position of a part in an assembly from a mating condition based on the work by Lee and Andrews (1985), and is given in Section 5.7.

	boss	**hole**	**pocket**	**thru slot**	**non–thru slot**	**notch**	**step**	**surface**
boss	A, L	F	F, L	A	A	A	A	A
hole	F	L	L	X	X	X	X	X
pocket	F,L	L	L	X	X	X	X	X
thru slot	A	X	X	L	X	X	A	A
non–thru slot	A	X	X	X	X	X	A	A
notch	A	X	X	X	X	A	X	A
step	A	X	X	X	A	X	A,L	A,L
surface	A	X	X	A	A	A	A	A,L

Table 5.1: Feature Relation Table

5.6 REPRESENTATION OF MATING RELATIONSHIPS

In order for the information on the mating relationships to be useful in assembly modelling, it has to be associated with each feature and readily accessible when two features are to be assembled to form a component, a subassembly or an assembly. To achieve this, the relationships are established in the form of expressions.

The general form of the expression representing the mating relationship between two features is created by specifying the two features that mate and the mating condition type in a relation, as follows:

$$component1.feature1_n - mating\ relationship - component2.feature2_n$$

where n denotes the feature index number in the assembly. For example, using the above expressions, the assembly relationships for the lathe tool post can be represented in the following forms:

R1: *tee_bolt.step1–against–top_slide.thru_slot*

R2: *tee_bolt.step2–against–top_slide.thru_slot*

R3: *tee_bolt.step1–against–top_slide.thru_slot*

R4: *tee_bolt.step2–against–top_slide.thru_slot*

R5: *tool post.boss–against–top_slide.boss*

R6: *tee_bolt.boss2–fits–tool_post.hole3*

R7: *washer.boss–against–tool_post.boss*

R8: *tee_bolt.boss2–fits–nut.hole*

R9: *tee_bolt.boss2–fits–washer.hole*

R10: *washer.boss–against–nut.boss*

R11: *setscrew1.boss1–fits–tool_post.hole1*

R12: *setscrew2.boss1–fits–tool post.hole2*

R13: *nut.hole–align–washer.hole*

R14: *tool_post.boss–align–top_slide.boss*

R15: *tool_post.boss–align–top_slide.boss*

R16: *tool_post.boss–align–top_slide.boss*

The relationship expressions for the bracket and pulley assembly are as follows:

R1: *key.boss–against–shaft.non_thru_slot*

R2: *key.boss–against–bracket.thru_slot*

R3: *shaft.boss2–fits–bracket.hole5*

R4: *shaft.boss3–against–bracket.boss4*

R5: *shaft.boss3–against–pulley.boss2*

R6: *shaft.boss4–fits–pulley.hole*

R7: *key.boss–against–shaft.non_thru_slot*

R8: *key.boss–against–shaft.non_thru_slot*

R9: *key.boss–against–shaft.non_thru_slot*

R10: *key.boss–against–bracket.thru_slot*

R11: *key.boss–against–bracket.thru_slot*

The expressions for the valve subassembly are as follows:

R1: *body1.boss1–against–body2.boss1*

R2: *body1.hole1–align–body2.hole1*

R3: *bolt1.boss2–fits–body1.hole2*

R4: *bolt2.boss2–fits–body1.hole3*

R5: *bolt3.boss2–fits–body1.hole4*

R6: *bolt1.boss2–fits–body2.hole2*

R7: *bolt2.boss2–fits–body2.hole3*

R8: *bolt3.boss2–fits–body2.hole4*

R9: *bolt1.boss2–fits–nut1.hole*

R10: *bolt2.boss2–fits–nut2.hole*

R11: *bolt3.boss2–fits–nut3.hole*

R12: *bolt1.boss1–against–body1.boss2*

R13: *bolt2.boss1–against–body1.boss3*

R14: *bolt3.boss1–against–body1.boss4*

R15: *nut1.boss–against–body2.boss2*

R16: *nut2.boss–against–body2.boss3*

R17: *nut3.boss–against–body2.boss4*

R18: *nut1.hole–align–body2.hole2*

R19: *nut2.hole–align–body2.hole3*

R20: *nut3.hole–align–body2.hole4*

R21: *body1.hole2–align–body2.hole2*

R22: *body1.hole3–align–body2.hole3*

R23: *body1.hole4–align–body2.hole4*

R24: *body1.boss2–against–body2.boss2*

R25: *body1.boss3–against–body2.boss3*

R26: *body1.boss4–against–body2.boss4*

In a feature–based design system, these expressions can be automatically derived for each pair of features, based on the data in the Feature Relation Table.

5.7 INFERENCE OF POSITIONS

The inference of the location and orientation of a part in an assembly from mating relationships requires the computation of its transformation matrix from these conditions. The matrix relates the part's local coordinate system to the global coordinate system of the assembly.

For the *against* condition shown in Figure 5.24a, each face where the two parts mate is specified by a unit normal vector (n) and a point (x, y, z) described in the local coordinate system of its corresponding part. To satisfy the *against* condition, the normals are constrained to be parallel and point in opposite directions. Also, the points are required to lie in the same plane. The numerical values of the normals and the points are stored with respect to the body coordinate systems attached to the corresponding parts. Before the *against* equations can be written, the values of the points and the normals must be transformed to a reference coordinate system. This creates a group of secondary variables which can then be used to construct the *against* equations.

To create the secondary variables, let $[T_1]$ and $[T_2]$ be the transformation matrices from the $X_1Y_1Z_1$ and $X_2Y_2Z_2$ coordinate systems respectively to the global coordinate system of the assembly. The unit normals and the two points specifying the mating conditions can be expressed in terms of the XYZ system as follows:

$$\begin{bmatrix} n^a_{1x} \\ n^a_{1y} \\ n^a_{1z} \\ 0 \end{bmatrix} = [T_1] \begin{bmatrix} n_{1x} \\ n_{1y} \\ n_{1z} \\ 0 \end{bmatrix} \quad \text{(Eq 1)}$$

$$\begin{bmatrix} x^a_1 \\ y^a_1 \\ z^a_1 \\ 1 \end{bmatrix} = [T_1] \begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{bmatrix} \quad \text{(Eq 2)}$$

$$\begin{bmatrix} n^a_{2x} \\ n^a_{2y} \\ n^a_{2z} \\ 0 \end{bmatrix} = [T_2] \begin{bmatrix} n_{2x} \\ n_{2y} \\ n_{2z} \\ 0 \end{bmatrix} \quad \text{(Eq 3)}$$

$$\begin{bmatrix} x^a_2 \\ y^a_2 \\ z^a_2 \\ 1 \end{bmatrix} = [T_2] \begin{bmatrix} x_2 \\ y_2 \\ z_2 \\ 1 \end{bmatrix} \quad \text{(Eq 4)}$$

In the above equations, n_{1x} and n_{2x} are the normal vectors of the planar faces and (x_1, y_1, z_1) and (x_2, y_2, z_2) are the points on the planar faces with respect to each body coordinate system. The superscript *a* indicates assembly. The *against* condition requires the directions of the two unit normals to be equal and opposite as expressed by these equations:

$$n^a_{1x} = -n^a_{2x} \quad \text{(Eq 5)}$$

$$n^a_{1y} = -n^a_{2y} \quad \text{(Eq 6)}$$

$$n^a_{1z} = -n^a_{2z} \quad \text{(Eq 7)}$$

and the two points to lie in the same plane, expressed in the following equation:

$$[\, n^a_{1x} \quad n^a_{1y} \quad n^a_{1z} \quad 0\,] \left\{ \begin{bmatrix} x^a_1 \\ y^a_1 \\ z^a_1 \\ 1 \end{bmatrix} - \begin{bmatrix} x^a_2 \\ y^a_2 \\ z^a_2 \\ 1 \end{bmatrix} \right\} = 0 \qquad \text{(Eq 8)}$$

Hence four equations (5 – 8) are required for each *against* condition.

The *fits* condition requires that the centrelines of the boss and the hole be co–linear, as shown in Figure 5.25. The equations of the centrelines, of say the hole can be written as:

$$\frac{x - x^a_1}{x^a_2 - x^a_1} = \frac{y - y^a_1}{y^a_2 - y^a_1} = \frac{z - z^a_1}{z^a_2 - z^a_1} \qquad \text{(Eq 9)}$$

If the shaft axis is co–linear with the hole centreline, points P_3 and P_4 defining the axis should satisfy equation 9. The points must first be transformed using $[T_2]$ to the assembly global coordinate system. The constraint equations required for each *fits* condition can be written as:

$$\frac{x^a_3 - x^a_1}{x^a_2 - x^a_1} = \frac{y^a_3 - y^a_1}{y^a_2 - y^a_1} = \frac{z^a_3 - z^a_1}{z^a_2 - z^a_1} \qquad \text{(Eq 10)}$$

$$\frac{x^a_4 - x^a_1}{x^a_2 - x^a_1} = \frac{y^a_4 - y^a_1}{y^a_2 - y^a_1} = \frac{z^a_4 - z^a_1}{z^a_2 - z^a_1} \qquad \text{(Eq 11)}$$

Each of the above equations yields three combinations of equations resulting in a total of six equations for each *fits* condition. In general, two of these equations are redundant because equations 10 and 11 each yields only two independent equations instead of three. However, it is necessary to carry all three to cover the case where the centreline passing

through points P_1 and P_2 is parallel to any of the global coordinate axes. For example, if the centreline is parallel to the X axis as shown in Figure 5.5, equation 10 becomes:

$$\frac{x_3^a - x_1^a}{x_2^a - x_1^a} = \frac{y_3^a - y_1^a}{0} = \frac{z_3^a - z_1^a}{0} \qquad \text{(Eq 12)}$$

which gives the following two equations only:

$$(y_3^a - y_1^a)(x_2^a - x_1^a) = 0 \qquad \text{(Eq 13)}$$

$$(z_3^a - z_1^a)(x_2^a - x_1^a) = 0 \qquad \text{(Eq 14)}$$

Hence, it can be seen that all three equations must be carried so that at least two independent equations can be written for all cases, although this introduces redundancy in the system of equations.

The determination of transformation for the *align* mating relationship are very similar to the centreline coincidence used in the fit of a boss into a hole.

For each *against* condition, 16 equations can be written, 12 are provided by equations 1 – 4 and the other four are equations 5 – 8. For each *fits* condition, 18 equations can be written, 12 are provided by equations 1 –4 and the other six are equations 10 and 11. Additional constraint equations are needed for free rotation bodies such as bolts and pins and other parts where it is not desired to fully describe the mechanism which constraints rotation (such as a key and keyway). If the rotation axis to be constrained is coincided with an axis of the part's local coordinate system, the additional constraint equations can be written as:

$o_z = a_y = 0$ for a component rotating around its x axis

$n_z = a_x = 0$ for a component rotating around its y axis

$n_y = o_x = 0$ for a component rotating around its z axis

Thus there are two equations generated for each rotational part.

Because the 12 elements in a transformation matrix are treated independently of each other, the following equations have to be included to satisfy the properties of a transformation matrix. To satisfy the unit length requirement on the rotation axis, the following equations are required:

$$n_x^2 + n_y^2 + n_z^2 = 1 \qquad \text{(Eq 15)}$$

$$o_x^2 + o_y^2 + o_z^2 = 1 \qquad \text{(Eq 16)}$$

To satisfy orthogonality of the rotation axes, four additional equations are needed:

$$n_x o_x + n_y o_y + n_z o_z = 0 \qquad \text{(Eq 17)}$$

$$a_x = n_y o_z - n_z o_y \qquad \text{(Eq 18)}$$

$$a_y = o_x n_z - n_x o_z \qquad \text{(Eq 19)}$$

$$a_z = n_x o_y - n_y o_x \qquad \text{(Eq 20)}$$

The unit length measurement of (a_x, a_y, a_z) is automatically satisfied by equations 18 –20.

The number of equations generated for an assembly of N components (RP rotational components), with MA *against* condition and MF *fits* conditions, is given by the following formula:

Number of equations

= 6 (N – 1)	from matrix properties
+ 16 MA	from *against* equations
+ 18 MF	from *fits* equations
+ 2 RP	from rotational parts

The number of variables for the assembly is:

= 12 (N – 1)	transformation matrix elements
+ 12 (MA + MF)	secondary variables

In the above equations, the number of equations is always equal to or larger than the number of variables.

The method used to remove the redundant equations is to solve the equations with a Newton–Raphson iteration method and the use of an algorithm to search for groups of equations which contain a linear dependency. This will result in an equal number of equations and variables. The remaining equations will be a linearly independent set and the Newton–Raphson iteration method can be used in the normal way.

The ACIS modeller gives access to these transformation matrices through the Application Procedural Interface (API) and this facility has been utilised in the assembly **relationship** class described later in the chapter.

5.8 ASSEMBLY AND PROCESS PLANNING FEATURES

Three types of mating conditions have been identified in Section 5.5. These mating conditions associate pairs of machining features. As features presented in this research are machining features that have been used for process planning (Gindy et. al. 1993), knowledge on process planning has been associated with each feature. In order to find the relation between the assembly relationships and the process planning information, assemblies at the face level for selected parts in the three products described in Section 5.4 are re–examined, but before doing so a brief description of the aims of process planning is given.

The overall objective of process planning is to devise a method of manufacture that is optimal with respect to a set of criteria. Typically these criteria will be concerned with the economy and quality of manufacture and the optimisation takes place with a knowledge of the capability and availability of appropriate manufacturing processes. There is much discussion as to exactly what constitutes process planning and what might be considered to be in other fields of manufacturing engineering such as production planning and NC part programming. However, some of the key aspects can be stated as:

1. Overall Process Selection i.e. is the component to be machined, formed or fabricated. It could be argued that these decisions are taken at the design rather than process planning stage, but in any case this work is focussed upon machined components.

2. Specific Machining Process Selection. Many geometric forms can be generated by a variety of machining processes (turning, milling, grinding, etc) and the choice will be made on the differing processes' capabilities in quality terms (precision, surface finish, etc) taking into account availability and relative costs.

3. Machine Selection. Machines of the same general type (lathes for example) have different capabilities and associated costs. There is also a connection here with production planning as a machine has to be available at the required time of manufacture.

4. Set–up Determination. The number of set–ups required to machine a component is a very significant determinant of the total cost of manufacture, and can also have technological implications in terms of maintenance of tolerances, etc.

5. Operations Sequence Planning. Within a particular set–up it is necessary to determine the sequence in which manufacturing operations will be carried out. Partly this is concerned with the feasibility of different sequences (it may be necessary to machine one feature to give access for machining a subsequent feature) and partly it is concerned with optimising machining time through minimising tool changes and non–cutting motions.

6. Detailed Process Planning. This involves the determination of cutter paths, selection of tools, fixturing, feeds, speeds etc and is more normally considered as NC part programming rather than process planning.

Thus the process planning task requires a considerable amount of information about the parts to be manufactured. In a feature–based process planning system, information is inferred from the feature model data. The process planning information which is required for each feature and relevant to this work is listed in the following paragraphs:

1. The dimensional and geometric tolerances are important to ensure that parts will function correctly, be interchangeable and can be manufactured economically. Dimensional tolerances, marked **d** in the subsequent diagrams, are used to communicate ranges of dimensions that are acceptable in meeting functionality. Geometric tolerances, such as parallelism (marked // in the diagrams), circularity (**C**) and flatness (**F**) further refine the specification for manufacturing to meet functional requirements.

2. The imaginary faces, represented as i_n determine the external access directions (EAD's) as explained in Chapter 4. These are potential tool approach directions in machining, and can be used in set–up determination.

3. The surface finish attribute can be used in determining suitable manufacturing process. In the three examples shown in Figures 5.27, 5.28 and 5.29, the surface finish for the mating parts is not critical in assembly.

4. The parent–child relationship determines the machining precedence. It also affects the tool access directions, operation sequencing and set–up strategy. A parent–child relationship exists if one feature can be defined with respect to another feature. The former is called a child while the latter is a parent feature. For example, with countersunk hole, the hole might be the child of the countersink.

These items of information are added to the selected parts in each assembly and its relation with the assembly relationship is examined. These are shown in the Component Connectivity Graphs shown as Figures 5.26, 5.27 and 5.28.

In Figure 5.27, there is an interaction between the base of tee bolt and the through slot of the top slide. From Figure 5.7, faces f1, f2 of step1 and step2 of the tee bolt and faces f1, f2, f6 and f7 of the through slot of top slide are critical in the assembly. These surfaces are required to be parallel and should have dimensional tolerances. The step can be assembled to the imaginary faces i1 and i3 and these can be considered as potential assembly directions (PAD).

The assembly of a key to a shaft is shown in Figure 5.28. For this assembly, the edges of the key (f2 and f4) are critical and should have parallelism and dimensional tolerances. Faces f1 and f3 of the non–through slot of the shaft should have similar requirements. The imaginary faces of the non–through slot form the potential assembly directions to the key, as shown.

Figure 5.29 shows the interactions between four parts in the valve subassembly. The two faces of the bosses (f3 of body1 and face1 of body2) must be flat and the holes should be defined with a cylindricity tolerance. Other dimensions and tolerances are not critical. The imaginary faces of the nut and body holes (i1) become the potential assembly directions for the face of the bolt.

From the above analysis, the following observations are inferred and summarised in Figure 30:

1. The External Access Direction (EADs) of each feature can be viewed both as a potential machining direction and a potential direction in which another feature can be assembled to it. The latter direction is referred to as Potential Assembly Direction (PAD) and occurs between an imaginary face of one feature and a real face of a mating feature.

2. Each mating face has its own process planning data attached to it. This data either relates to the feature itself, such as the cylindricity of a hole or it is a relationship between two features such as the parallelism of the sides of the features.

3. Some of the process planning data are relevant to the assembly modelling. For example, the dimensional tolerance will determine the type of fit between two features. For example, the parallelism of the faces are important if two faces are to have sliding contact, as shown by the example of the assembly of steps to the through slot in the lathe tool post assembly. Other information such as parent–child relationships are not relevant to the assembly modelling, and are only used to determine intermediate configurations of the component during machining.

4. The Component Connectivity Graphs show a clear relationship between process planning and assembly information. Thus for example the functional assembly requirement of the Tee Bolt to mate with the through slot of the Top Slide (Figure 5.27) generates the assembly information describing *against* conditions between the faces of the components. Process planning information in the form of dimensional and geometrical tolerances on the faces of individual features of each component are then required to ensure this assembly functionality.

To realise the benefits of combined assembly and process planning knowledge it is necessary to represent it by a data structure in a feature–based model, and this is discussed in the next section.

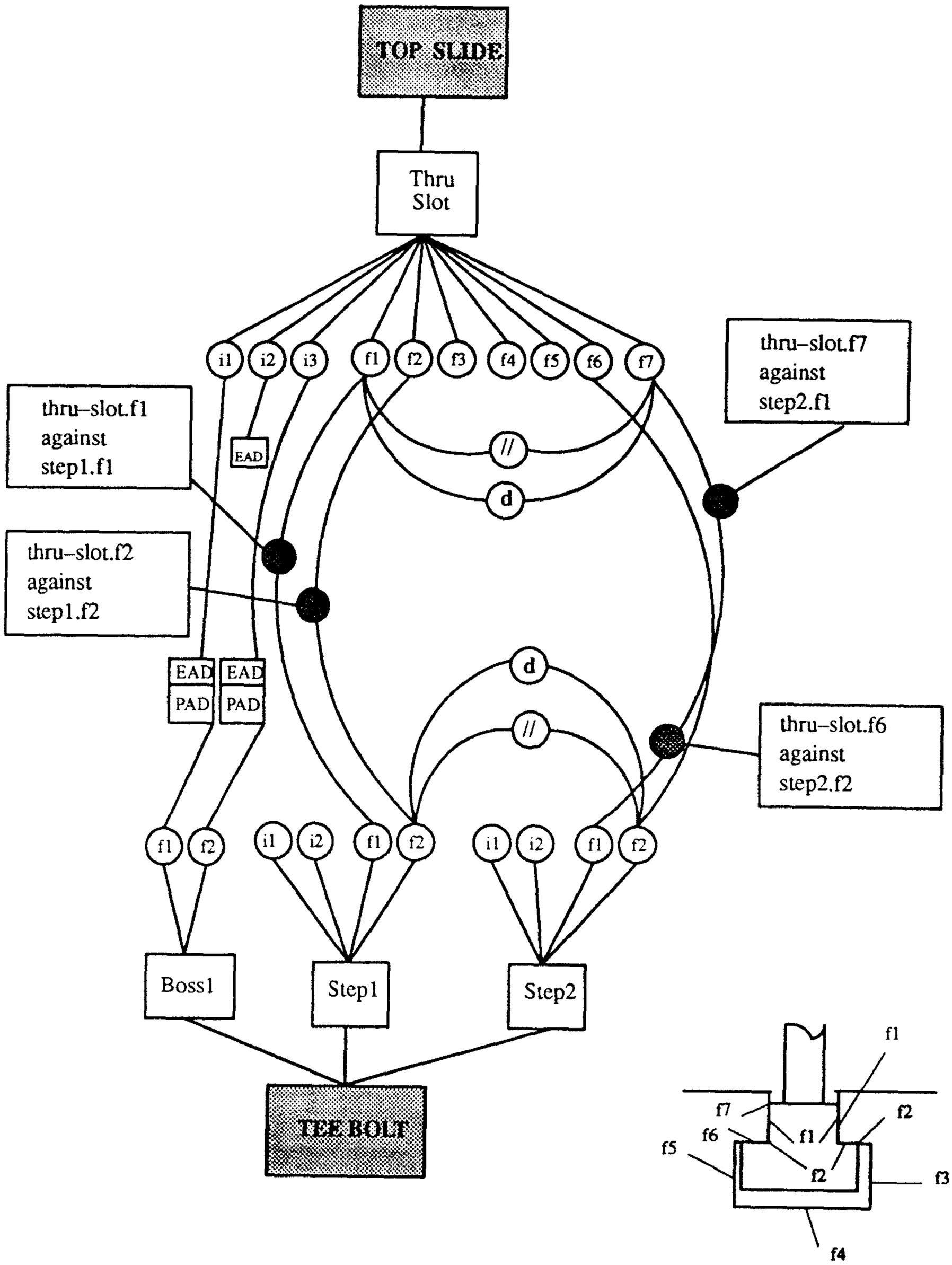


Figure 5.27: Component Connectivity Graph for Lathe Tool Post Assembly

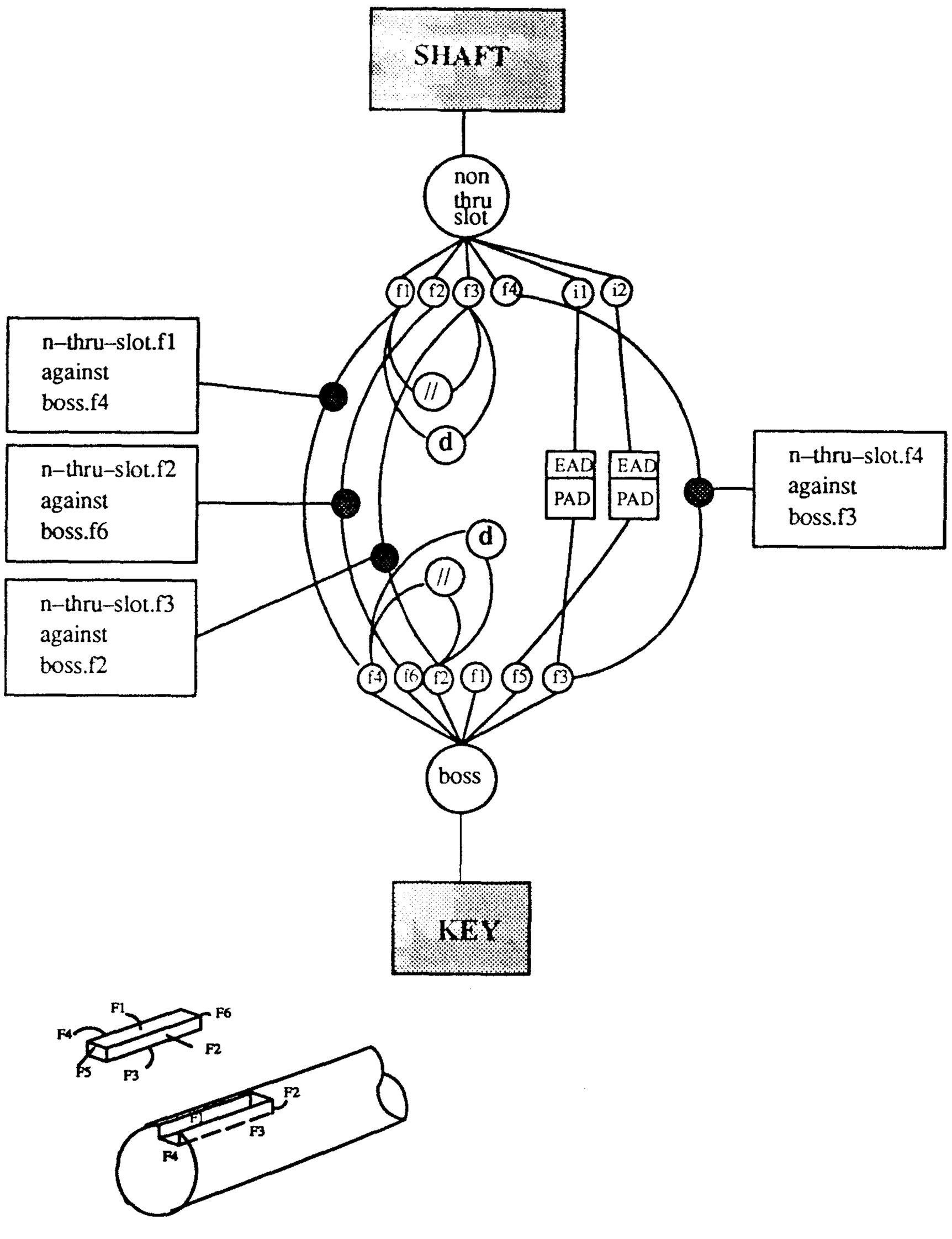


Figure 5.28: Component Connectivity Graph for Key and Keyway

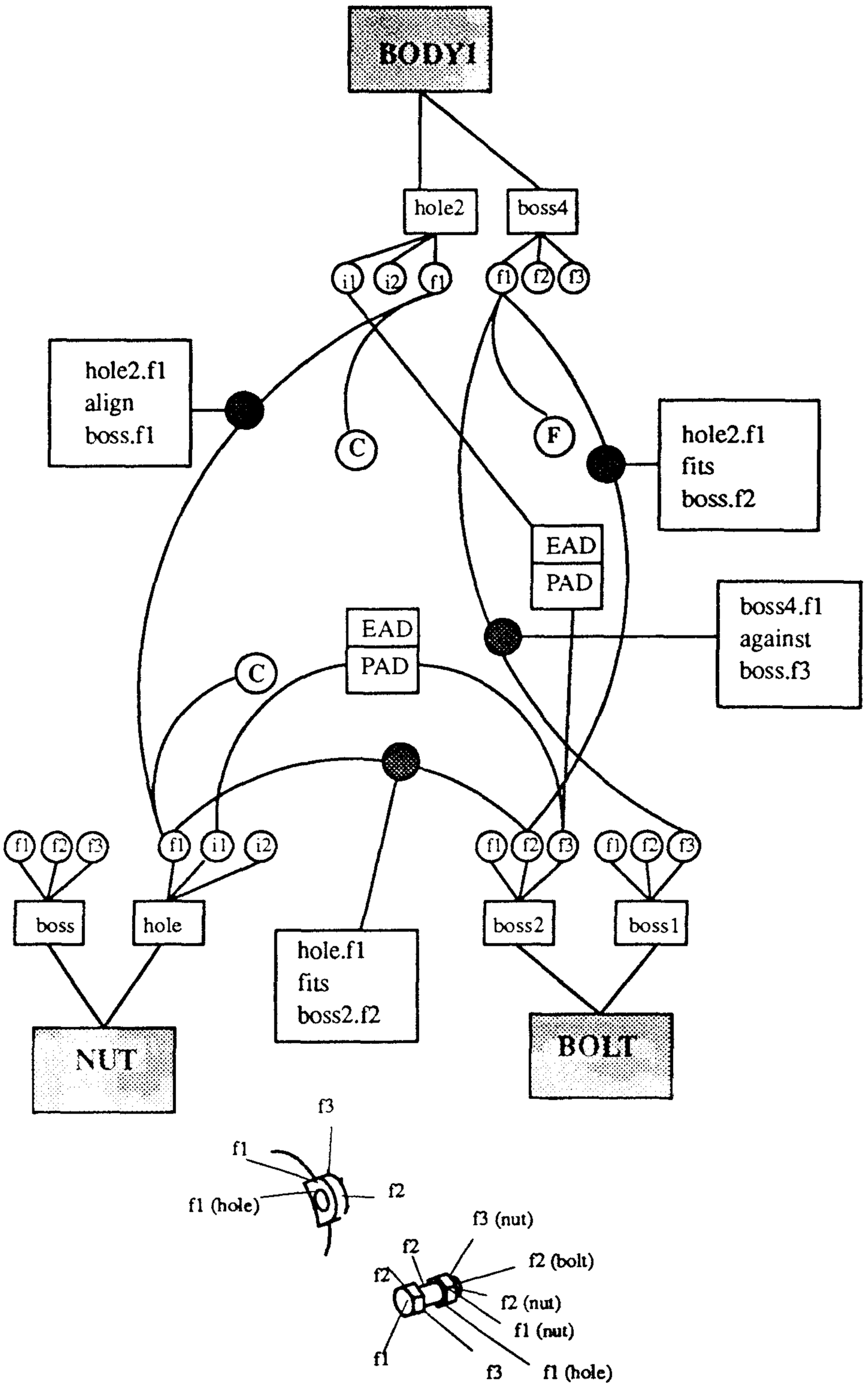


Figure 5.29: Component Connectivity Graph for Valve Subassembly

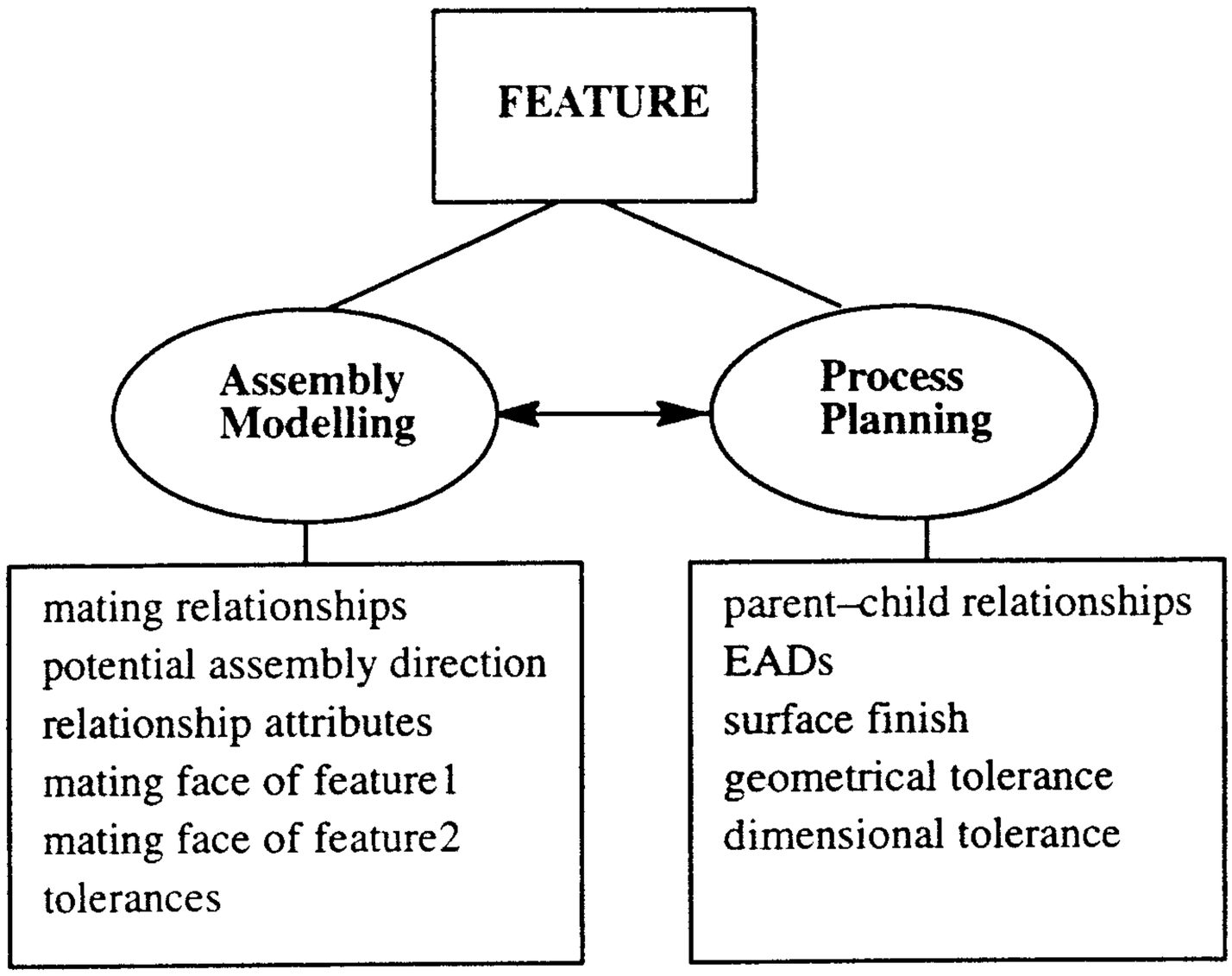


Figure 5.30: Relation between assembly and process planning knowledge in a feature

5.9 ASSEMBLY DATA STRUCTURE

Data structure is a very important aspect of a database and an assembly data structure provides a link between the assembly database and the database of its assembled parts so that when any part (a feature, a component or a subassembly) is modified, the corresponding instance in the assembly is updated automatically. Linked lists are one of the basic elements of C++ programming and offer several advantages over other structures such as arrays. Lists do not have predefined size and they can be formed, reorganised or destroyed dynamically, object by object using defined pointers. This is useful in modelling the assembly situation where features are added, moved or deleted from the components or subassemblies. Lists are also claimed to be fast and fit the object–oriented way of thinking (Soukup 1994).

Using the hierarchical data structure of Section 5.3, an assembly can be considered as a list of sub–assemblies. Each sub–assembly is a list of components, a component is a list of features and a feature is a list of faces in the geometric model. This implies the use of

linked lists to represent each level of the assembly. A data structure of the assembly is shown in Figures 5.31 and 5.32. Figure 5.33 shows the assembly parts in a linked list structure. In this case, a forward pointer ring structure is used in which the last element in the list points back to the first element of the next highest level of assembly, instead of being a NULL pointer.

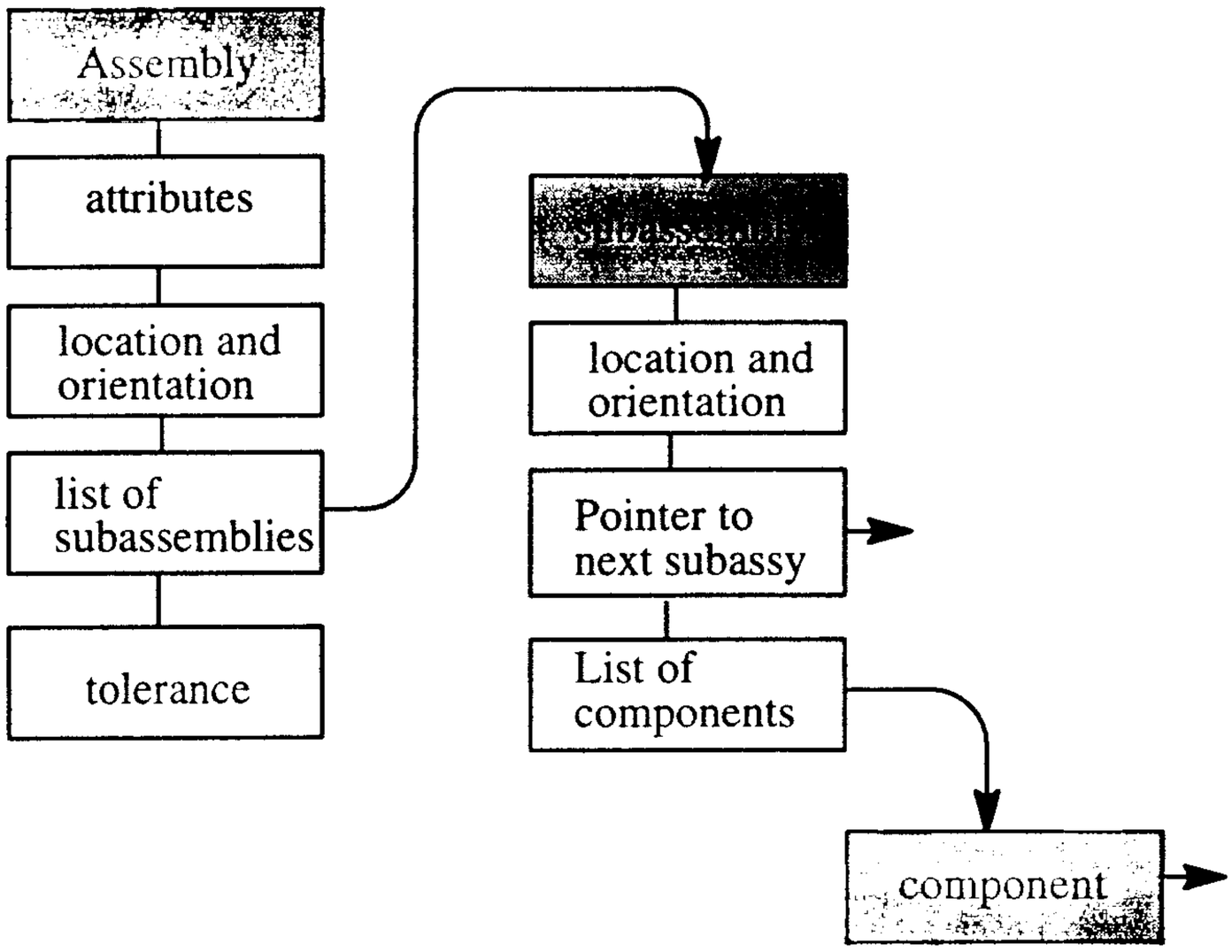


Figure 5.31: Data structure for assembly and subassembly levels

A general representation of a structure for the assembly as shown in Figure 5.31 consists of the following information:

Name of assembly (such as the lathe tool post)
Product attributes, which could be the the mass of the assembly or other relevant information
Location and orientation of the assembly with respect to the World Coordinate System
Pointer to a list of subassemblies
Overall product tolerance

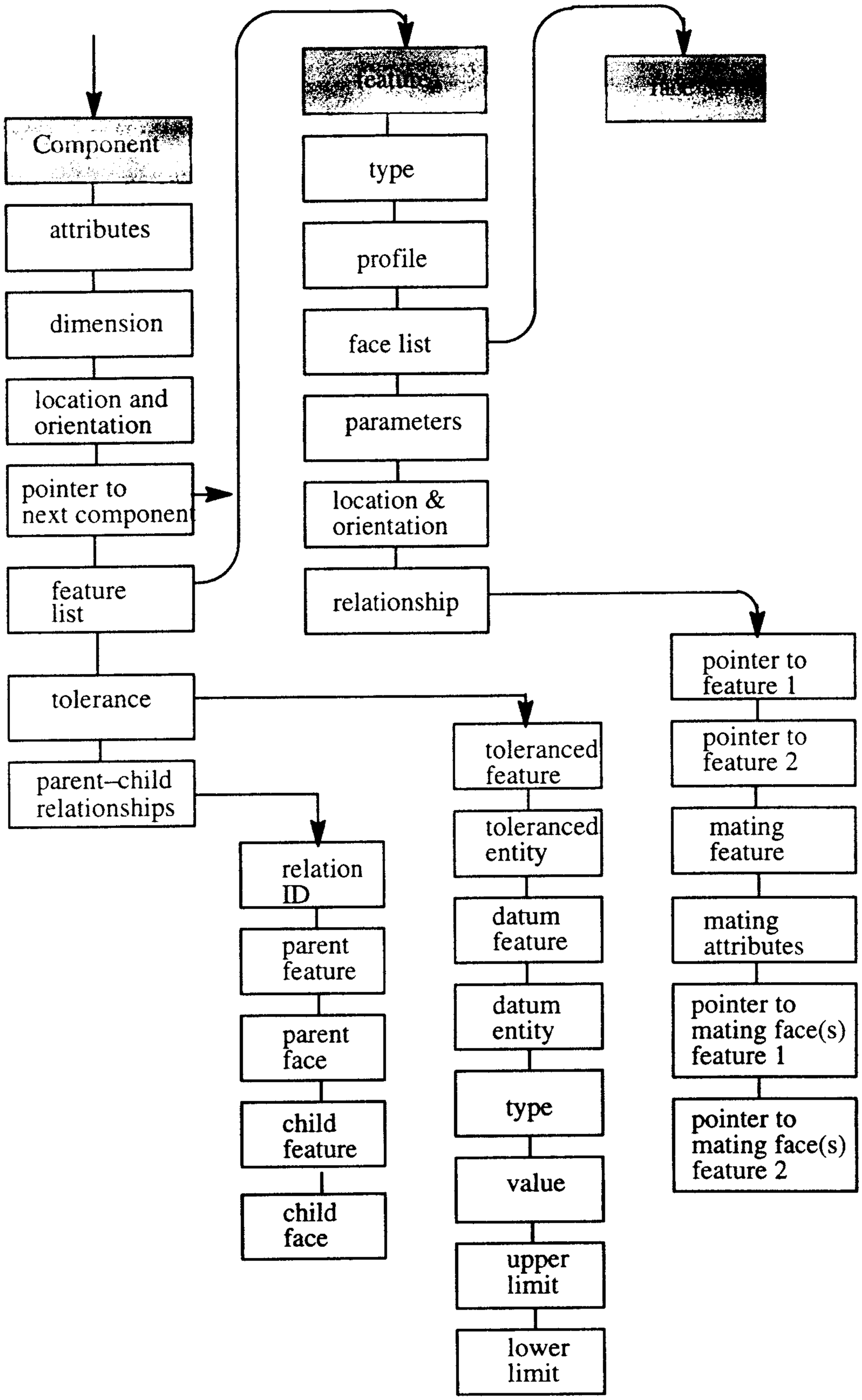


Figure 5.32: Data structure for component and feature levels

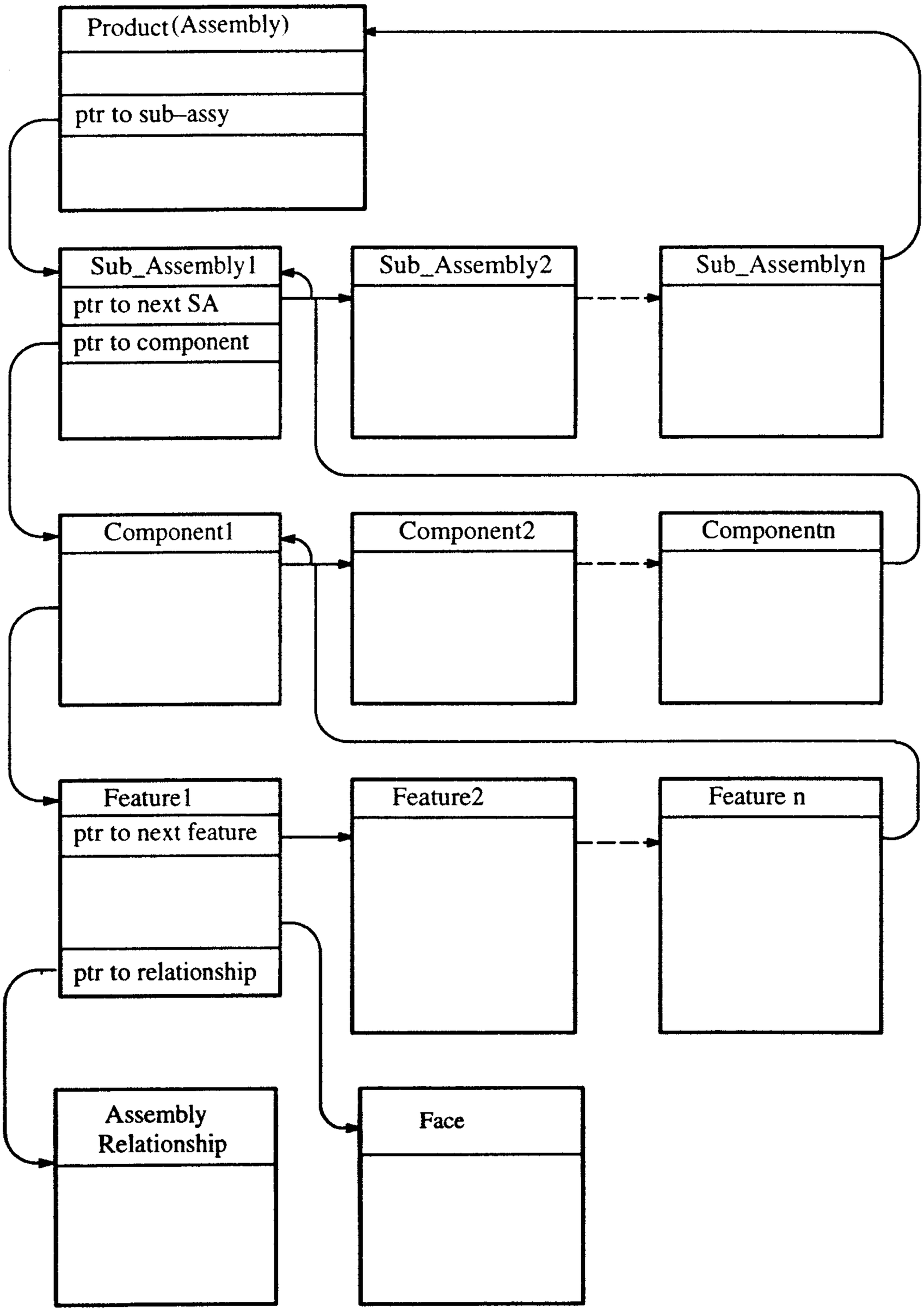


Figure 5.33: Assembly Data Structure

The data for the subassembly consists of the following information:

Name of subassembly (such as top slide)
Location and orientation of the subassembly with respect to the World Coordinate System
Pointer to the next subassembly that is also part of the assembly
Pointer to a list of components that constitute the subassembly

Referring to Figure 5.32, the data structure for the Component consists of the following information:

Component name (such as bolt)
Component attributes (such as mass, material)
Overall dimensions of the component
Location and orientation with respect to the World Coordinate System
Pointer to next component in the assembly
Pointer to list of features that constitute the component
Pointer to tolerance relationships
Pointer to parent–child relationships

For an individual feature, the parameters refer to the dimensions and the number of EADs. The face list refers to the list of mating faces which is accessible from the geometric model created in ACIS.

The data structure for the mating relationships contains the following information:

pointer to feature 1
pointer to feature 2
type of mating relationship (mating feature)
relationship attributes (a refinement of the mating relationships described in Section 5.5. Examples of such attributes are screw fit and sliding fit).
pointer to the mating face of feature 1
pointer to the mating face of feature 2

In the above structure, a mating between a pair of features is represented by a pointer to the feature and a pointer to the mating face. In the event of one feature being removed

from the assembly or a new feature being added to mate with an existing feature, the pointer will be reset to point to the new object.

This data structure is implemented in a feature–based environment using an object–oriented approach as described below.

5.10 IMPLEMENTATION

The application of the object–oriented approach to the modelling of assembly involves combining the information in the data structure with the appropriate methods to manipulate each part within the assembly. Using this approach, each level in the assembly hierarchy is defined in a class with the **assembly** class as the base feature. Other classes inherit the attributes of the class that is immediately above it. The **feature** class which is a base class for **feature type** classes and **profile** classes, has been discussed in Chapter 4. The following sections describe the assembly, subassembly, component and feature relationship classes. The declarations for classes described in this chapter can be found in Appendix A.

5.10.1 ASSEMBLY CLASS

The **assembly** class, shown in Table 5, represents the assembly of parts which is the highest level in the assembly hierarchy. It contains attributes and methods for the product assembly. The class has the following attribute data – the name of the assembly which is the product name, a pointer to the body of the assembly, the body's location and orientation and a pointer to a list of subassemblies. The location and orientation of the assembly corresponds to the world coordinate system and becomes the reference coordinate system for other assembled parts.

The methods for the assembly class are described in the following paragraphs:

Assembly Constructor

The constructor function creates a new instance of **assembly** object and initialises its parameters whenever the object is declared. An instance of assembly, *assy* is created by the following expression:

*assembly *assy = new assembly(assyname, x, y, z, angleX, angleY, angleZ)*

where *assyname* is the name of the assembly; *x*, *y* and *z* are the location of the assembly and *angleX*, *angleY* and *angle Z* are its orientation.

Assembly
Name of assembly Pointer to Body Location Orientation List of subassemblies
Constructor Destructor Add Subassembly Draw Save

Table 5.2: Assembly Class

Assembly Destructor

The destructor deletes the assembly body when it is no longer in use or at the end of the modelling session. This is denoted by *~assembly( )*.

Add Subassembly

This method adds a subassembly to the list of subassemblies that make up the assembly. A subassembly is assembled to another subassembly by the process of matching features which mate with one another. This is described in Section 6.6.

Draw

The assembly can be displayed on the screen using this method by recalling all entities that have been created and saved in the modelling process.

Save

This method saves an instance of assembly body in a file, to be retrieved for display.

5.10.2 SUB ASSEMBLY CLASS

The **subassembly** class, presented in Table 5.3 defines the attributes and methods for the subassembly level. The class inherits properties of the assembly class. The types of attributes of the class are similar to those in the **assembly** class, except that the class also contains a list of components that makes up the subassembly as well as a pointer to the next subassembly. The methods of the class are explained as follows:

Sub–Assembly
Name of Subassembly Pointer to sub assy Body Location Orientation List of Components Pointer to next subassembly
Constructor Destructor Add Component Draw Save

Table 5.3: SubAssembly Class

Subassembly Constructor

The constructor function creates a new instance of subassembly class and initialises its parameters. An instance of subassembly, *subassy* is created by the following expression:

*subassembly *subassy = new subassembly(x, y, z)*

where *x*, *y* and *z* are the location of the subassembly

Subassembly Destructor

The destructor releases the memory occupied by the subassembly body when it is deleted or at the end of the modelling session. It is denoted by *~subassembly( )*.

Add Component

This method adds a component to the list of components that makes up the subassembly. The process of joining two components is similar to the process of building up the assembly which is described in Section 6.6.

Draw

This method draws the subassembly body on the screen. This is done by recalling all entities that have been created and saved in the modelling process.

Save

This method saves an instance of a subassembly body in a file.

5.10.3 COMPONENT CLASS

The component class, presented in Table 5.4 describes the component level in the assembly hierarchy. The class contains a pointer to the component body, the component name, its dimensions, location, orientation, a pointer to the next component in the subassembly and a pointer to a list of features. The dimensions of the component in this case are the dimensions of the base feature since one of the criteria for choosing the base feature is to select the largest feature (see Section 6.5).

The methods for the class are described as follows:

Component Constructor

The constructor function creates a new instance of component and initialises its parameters. An instance of the component is created by the following expression:

*component *comp = new component(compname, l, w, h)*

where *compname* is the component name, *l* is the length, *w* the width and *h* the height of the component.

Component
Pointer to component body Component name Dimensions Location Orientation Pointer to next component Pointer to list of features
Component Constructor Component Destructor Get Dimension Add Feature Draw Save

Table 5.4: Component Class

Component Destructor

The destructor, denoted by *~component( )* releases the memory when the component body is deleted or at the end of the modelling session.

Get Dimension

This method is used to get dimensions of the component. Since the component is assumed to be a rectangular block, dimensions required from the user are the length, width and height.

Add Feature

This method is called to add a feature to the list of features that makes up the component. The user identifies the feature type and profile and this generates

functions associated with the particular feature such as Get Location, Get Orientation, and Save Entity.

```
b= new boss(x,y,z,b)
b->SelectProfile()->GetLocation()->GetOrientation();
b->DrawBoss->SaveEntity()
```

Draw

This method is used to draw the component by recalling all the features that have been created and saved for the particular component.

Save

A component is saved in a file using this method.

5.10.4 FEATURE RELATIONSHIP CLASS

The feature relationship class, listed in Table 5.5 is created to hold the information that defines a relationship between two features. The class has five attributes – the name of mating relationship (*against, fits* and *align*), pointers to the first and second features and pointers to mating faces of both features. Methods for this class are:

Input Feature

The method Input Feature is invoked to ask the user to input two features to be mated. The user enters the names of the feature to be mated such as boss1 for the first feature and hole1 for the second feature. These input are validated with the list of features in the database and an invalid input will be notified.

Find Mating Relation

This method is used to identify a suitable mating relationship when two mating features are identified from the Input Feature method. For example, when a boss feature is instanced and it is to be assembled to a hole, the function searches for a suitable mating relationship from a database, which is an input from the Feature Relation Table. In this case a *fit* or *tight–fit* relationship is identified. If no mating

condition exists, the function returns a NULL pointer and indicates that the assembly is not possible.

Feature Relationship
Pointer to first mating feature Pointer to second mating feature Name of Relationship Relationship attributes (type) Pointer to mating face of first feature Pointer to mating face of second feature
Input Feature Find Mating Relation Transform List Relation

Table 5.5: Feature Relationship Class

Transform

The transform function is used to locate a feature (existing in a component or a subassembly) in the assembled position. The function uses a transformation matrix that defines the relationship between the feature's coordinate system and the world coordinate system. The method uses an API function which can be expressed as follows:

api_apply_transform(feature, translate_trans(vector(0,0,dist)))

dist is the distance the feature is moved to its assembled position.

List Relation

This method lists the relationship(s) for the identified pair of features and displays it on the screen.

5.10.5 LINK CLASS

The **link** class consists of two classes, **objectlink** which defines the nature of the objects that are stored in the list and **assylist** which implements the linked list mechanism. This is implemented as a *template* class which is a generic class that can take as input any type of data. In this case, the data is in the form of objects from **feature type**, **profile**, **component**, **subassembly** and **assembly** classes. The use of a template class is advantageous as the same codes can be used for different data without having to create new functions or separate classes. The generic class decouples the algorithms that maintain a linked list from the data actually stored in the list.

5.10.5.1 OBJECTLINK CLASS

The class defines the nature of each element in the list. All members are defined as public and are described in the following paragraphs and shown in Table 5.6.

Objectlink
Data type Pointer to next object Pointer to previous object Constructor Get Next Object Get Previous Object

Table 5.6: Content of objectlink class

Data Type

This is the generic data type which is used as the type specifier for data stored in the **objectlink** class. The data can be a feature or a component. This type is replaced by the actual type specified when an object is created.

Constructor

The constructor initialises the pointer to the next and previous objects to NULL.

Get Next Object

This function is used to return the next object in the list.

Get Previous Object

This function return previous object in the list.

5.10.5.2 ASSYLIST CLASS

Assylist class implements the linked list mechanism. It inherits the **objectlink** class and operates on the object of the type held by the **objectlink** class. The data members of the class are shown in Table 5.7. It contains two pointers – one to the start of the list and another to the end of the list. The public members of this class are described in the following paragraphs:

Assylist
Pointer to the start of the list Pointer to the end of the list
Constructor Store Remove Display Forward Display Backward Get Start Get End

Table 5.7: Content of assylist class

Constructor

The constructor initialises the pointer to the start of the list and the pointer to the end of the list to NULL, when the list is first created.

Store

This function stores an item such as a feature or a component in the list.

Remove

This function removes an item from the list.

Display Forward

This function displays the list from the beginning.

Display Backward

This function displays the list from the end.

Get Start

Returns a pointer to the start of the list.

Get End

Returns a pointer to the end of the list.

5.10.6 RELATIONSHIPS AMONG CLASSES

Figure 5.34 shows the relationships among classes described above. The **subassembly** class inherits the properties of the **assembly** class and in turn becomes a base class for the **component** class. The **feature** class is derived from the **component** class and thus inherits the properties of all classes above its hierarchy. The **feature relationships** class and the **link** class use data from the **assembly, subassembly** and **component** classes. The **link** class also uses data from the **feature relationships** class and they are linked to each other as shown in the diagram.

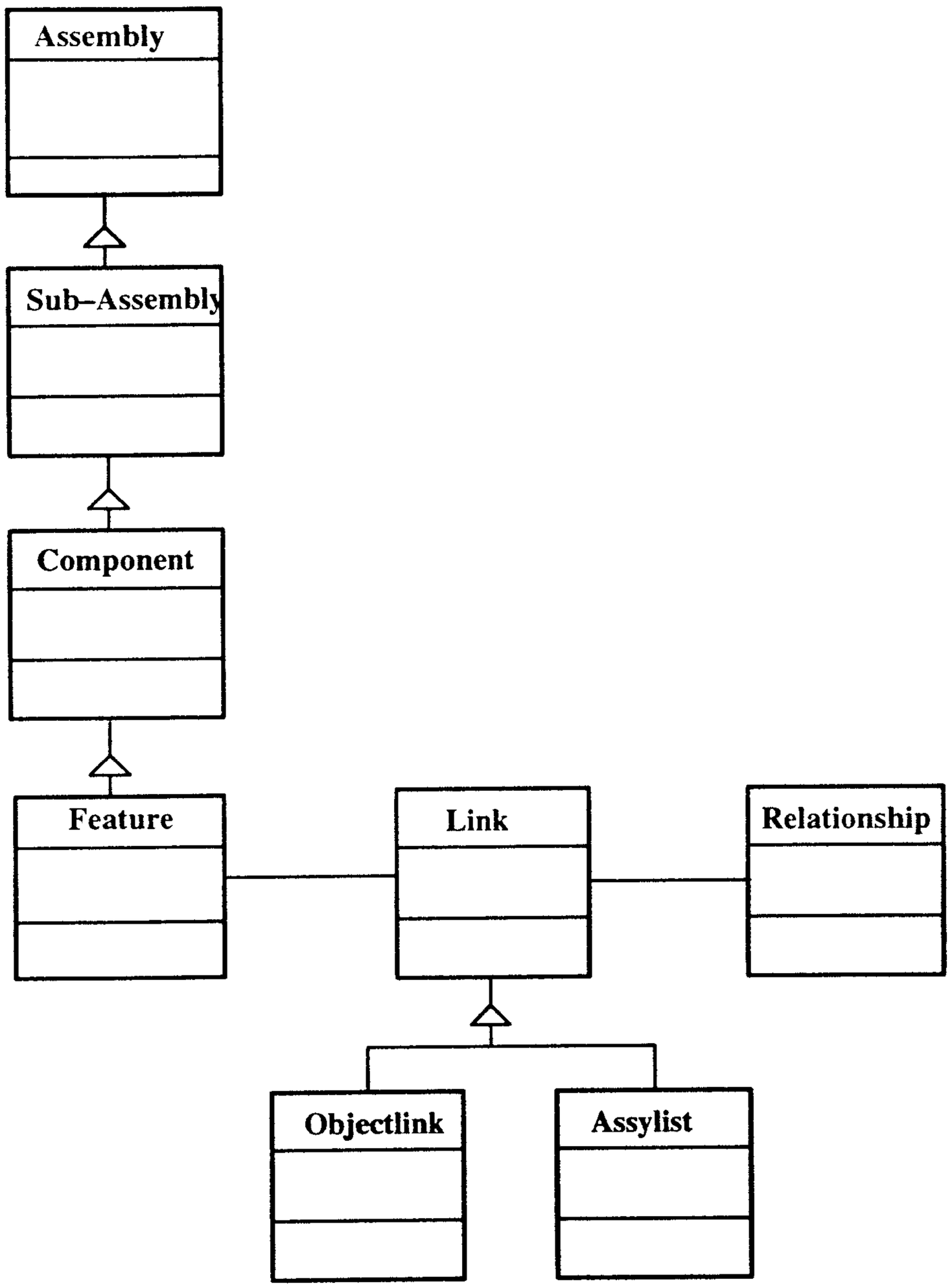


Figure 5.34: Relationship among classes

5.11 SUMMARY

In this chapter, the concept of designing with features for the purpose of process planning has been extended to incorporate the specification of an assembly. An assembly is modelled as an Assembly Graph that captures the hierarchy of subassemblies, components and features. Features form basic entities in the assembly, and the Feature Relation Graph forms a basis for representing and modelling the mating relationships among the features. Effective data representation requires that the interactions between the faces of two mating features be modelled and this is achieved through the Face Mating Graph. The Component Connectivity Graph unites the assembly information with process planning information and highlights areas of interdependency between the two.

Analysis of typical assemblies shows that feature interactions occur in three situations. Three mating relationships have been defined and implemented – *against*, *fits* and *align*. Each feature is assigned a possible type of mating relationship with each other feature type. This is then represented in an expression that describes the relationship between the two assembled features. A Feature Relation Table has been established to provide an aid in identifying mating relationships occurring among the features in an assembly. The assembly data structure provides links between the assembly database and the feature database. Each assembly level is implemented as a class in an object–oriented system. These classes are further supported by relationships and linked list classes.

The assembly model described in this chapter is implemented in a feature–based design system that supports the interactive modelling of assemblies. The application of this approach is discussed in Chapter 6.

CHAPTER SIX
IMPLEMENTING A FEATURE–BASED ASSEMBLY MODELLING SYSTEM

6.1 INTRODUCTION

Representations for features and assemblies have been presented in previous chapters. In this chapter the implementation of a feature–based modelling system and the procedures for modelling assemblies are described. A prototype system is developed as a proof of the concept presented in this thesis and used as an example to illustrate how the ideas mentioned previously can be put together. The design of the system, its structure and the implementation are addressed. Section 6.2 describes the overall design of a prototype feature–based modelling system. Section 6.3 describes an overall approach to the creation of the assembly model. Sections 6.4 and 6.5 discuss procedures for creating individual features and forming them into a component. Section 6.6 discusses the procedure for creating an assembly model. Section 6.7 describes how assembly data is presented. To illustrate the above procedures, examples involving simple mechanical products are included in Sections 6.8 and 6.9.

6.2 A PROTOTYPE FEATURE–BASED DESIGN SYSTEM

One of the requirements of a system to support the design of mechanical assemblies is the availability of methods that allow designers to work with abstract conceptual levels and geometry, specify functions in terms of relationships and define a system hierarchy. A design by features modelling approach can provide the platform to achieve these aims by storing assembly information during the design process so that the application can be considered from the early stages of the design process. This capture of design intent is not possible using the alternative method of feature recognition, and hence a design by features approach is adopted in this research. A prototype feature–based design system (FBDS) is developed to provide a platform for modelling individual features and an assembly. Features defined in Chapter 4 are stored in a library to be instanced during the

modelling process. The emphasis of the implementation is on the assembly modelling and thus some of the structures presented in Chapter 5 such as the tolerance and parent–child relationships are not implemented. They have been implemented in the process planning work (Case et. al. 1993)

FBDS is a prototype system aimed at testing the ideas proposed in this research. The structure of the system, shown in Figure 6.1 is designed to fulfil the requirement for an integrated data representation. The main engine of the system is the ACIS solid modeller (the version used in this research is 1.4.1) which provides methods and classes for creating and manipulating features, through a direct interface and the API functions, as described in Chapter 3. These are accessed by the methods in the **feature** and **assembly** classes defined in earlier chapters. The utility class consists of a collection of supporting programs which provide facilities such as the main menu and the manipulation of files. The application program, listed in Appendix B is the main program which creates and connects objects in various class libraries as well as acting as an interface to the ACIS test harness. The latter acts as a platform for testing and validating the program by providing an interface to all features of ACIS and provides a wireframe display of the model created during a session with the user. There are two types of files created by the system, an ACIS file format (an example for the pin and block assembly is shown in Appendix C) which is used by the test harness and a file used to store data once an assembly is created. The user interacts with the system through the application program. The operating environment for the system is UNIX running on a Sun workstation.

The system offers four options in the main menu:

[1] *Create Feature/Component*

This option is used to invoke the creation of an individual feature or a component.

[2] *Create Assembly*

This option allows the user to find mating relationships between two features and to invoke the assembly operations.

3 *Test Harness*

The test harness option allows the user to test the model created using option 1 or 2, by retrieving an ACIS file to be displayed on the screen.

4 *Print Data*

This option is used to display the data on the assembly, as mentioned in Section 6.7.

5 *Quit*

This option ends the modelling session and quits the system.

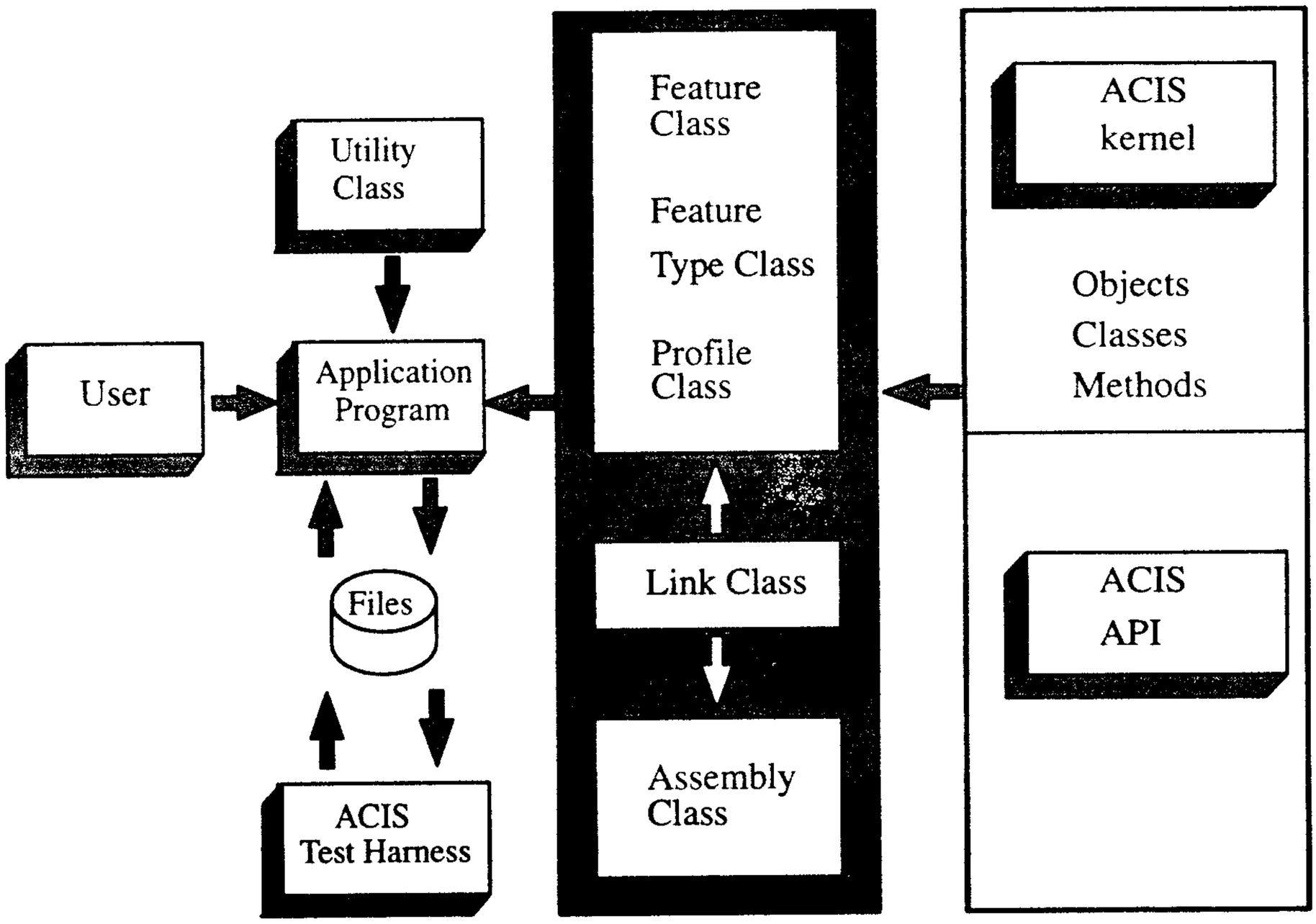


Figure 6.1: Structure of Feature–Based Design System

6.3 MODEL CREATION PROCEDURES

The general approach used to derive an assembly model is a bottom up approach that involves building up the assembly from individual features, as outlined in the following steps:

i. Create individual features, starting with a base feature

ii. Assemble features to form a component

iii. Specify all pairs of features to be mated

iv. Identify if relationships exist between the feature pairs

v. Assemble components to form a subassembly

vi. If more than one subassembly exists, repeat steps iii to v

vii. Combine subassemblies to form a final assembly

The steps are shown diagrammatically in Figure 6.2 and elaborated in the following sections.

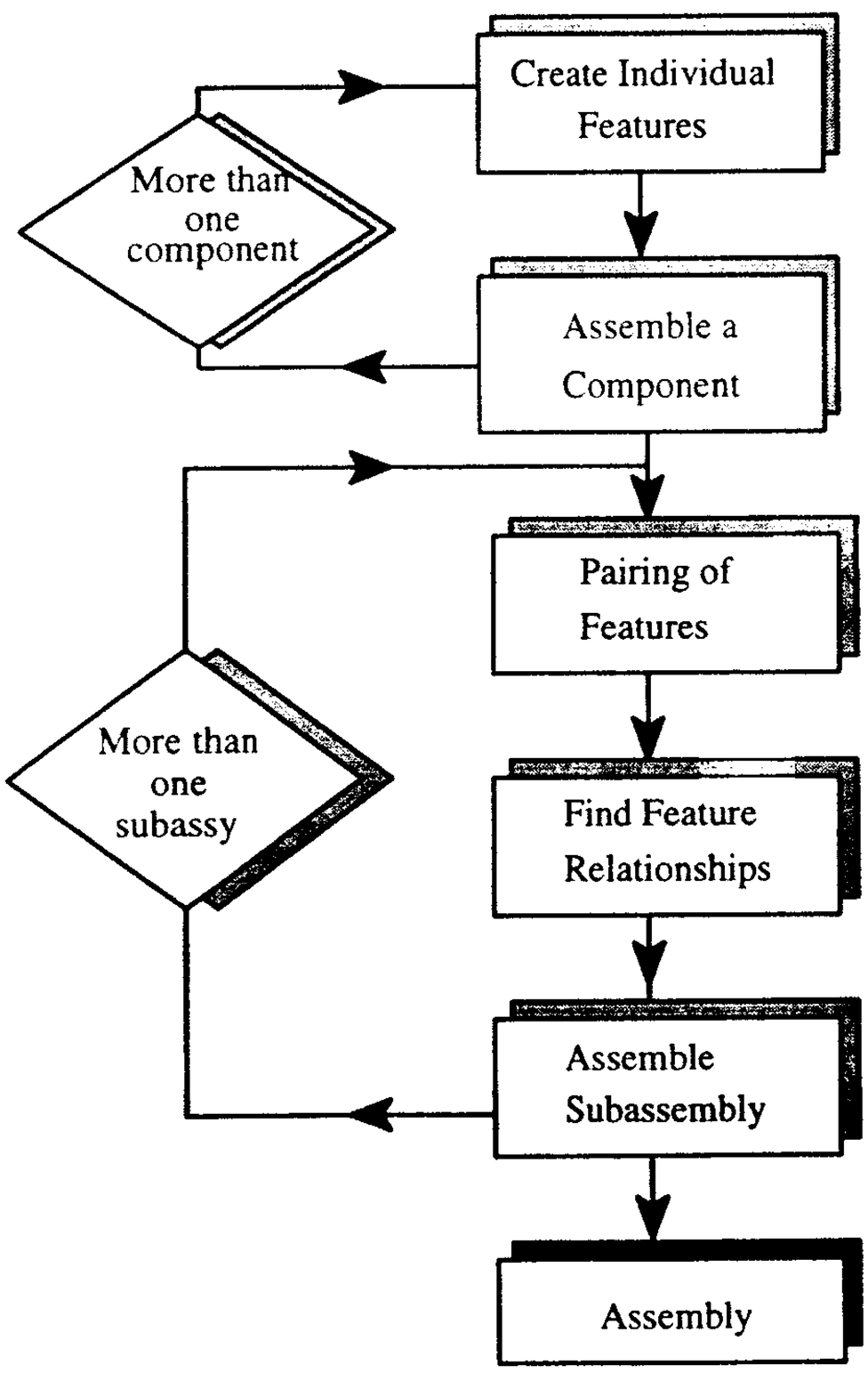


Figure 6.2: Model creation procedures

6.4 FEATURE CREATION

An individual feature is created in an interactive session with the system. The user inputs the type of feature, the profile type for that particular feature, the dimensions, and the location and orientation of the feature with respect to the base feature. Each input for the dimension and location is validated against the dimensions of the base feature. If there is an incorrect entry, the user is asked to re–enter the value. All options are presented in a form of a menu, as shown in Figure 6.3.

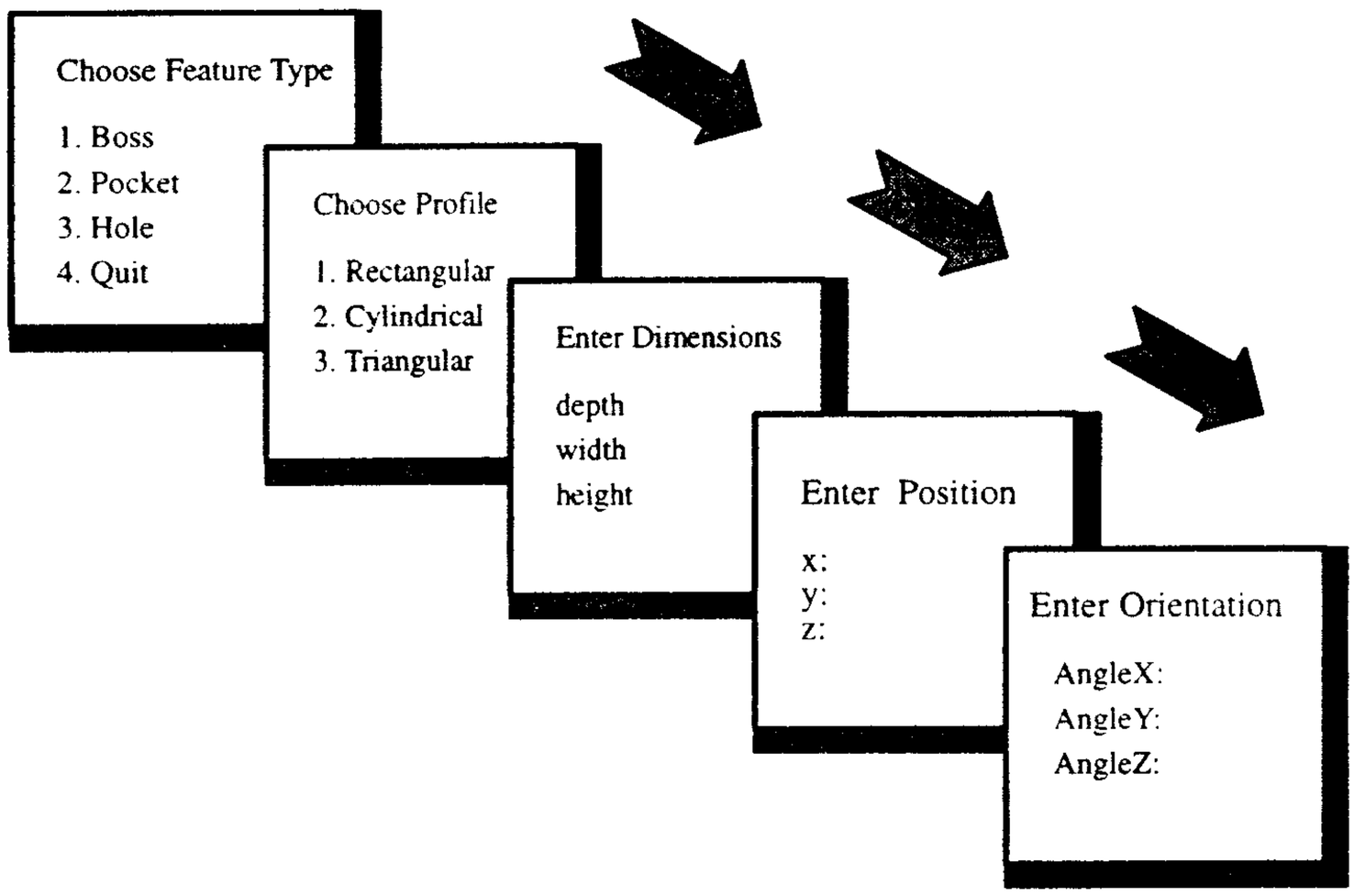


Figure 6.3: Menu for creation of a feature

6.5 COMPONENT MODEL

A component is made up of a base feature plus any number of other features. The process of creating a component model is shown in Figure 6.4 and described as follows:

i. An instance of a base feature is created. The criteria for choosing a base feature can either be the largest feature, the easiest to fix or the feature which provides the most assembly points. These criteria are based on heuristics and concur with the common

practice in assembly. The user gives the name of the component to be developed, its dimensions, position and orientation.

ii. Other features that make up the component are created one at a time according to the procedures described in Section 6.4. They are positioned and orientated with respect to the base feature.

iii. Using the Boolean operations provided by ACIS, the feature is either united to or subtracted from the base feature to form a component model. For example, the boss feature is united with the base feature while the hole or pocket feature is subtracted from it.

iv. The resultant component is saved in an ASCII file which has the suffix .sat, to indicate that it is an ACIS file.

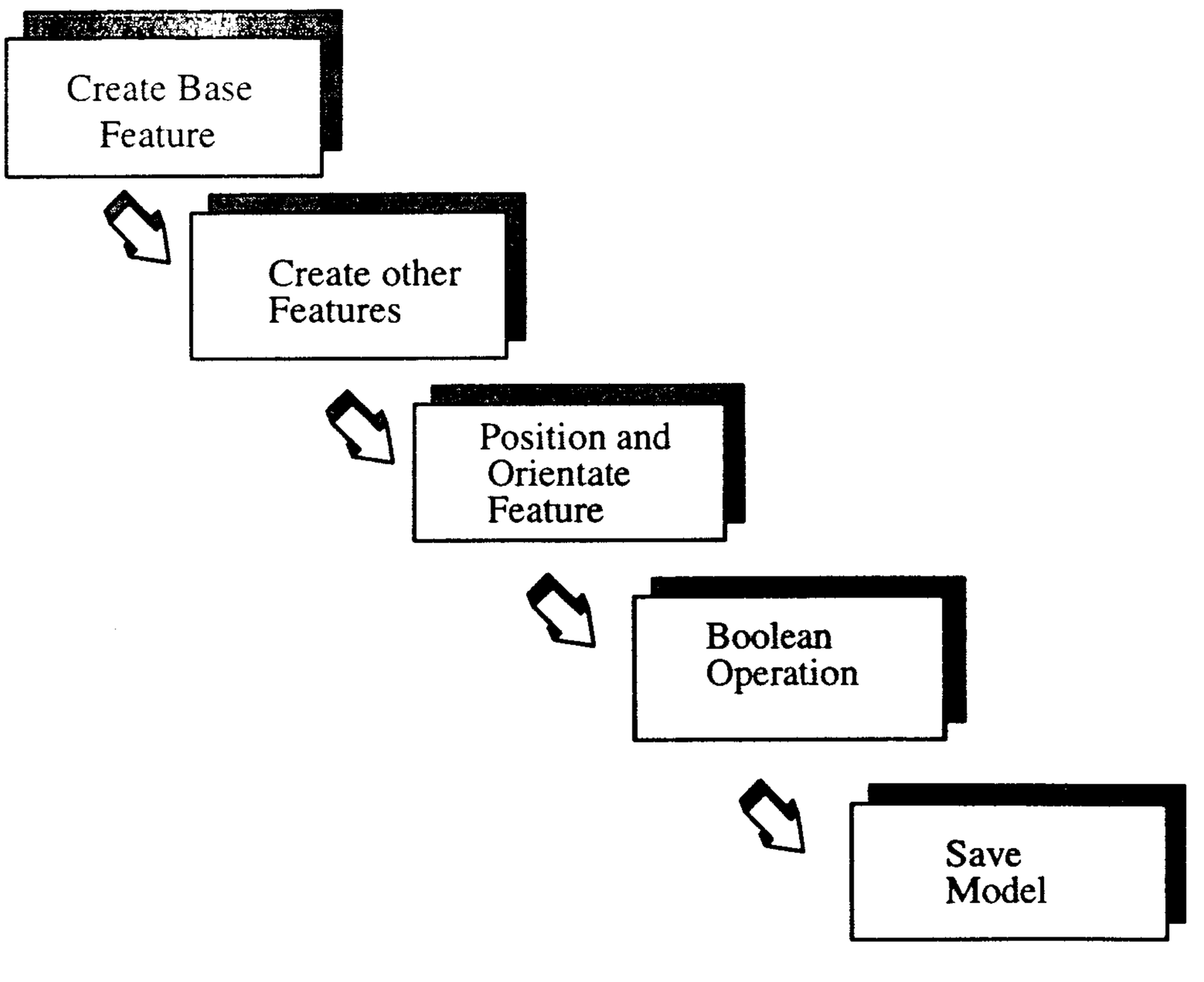


Figure 6.4: Steps in the creation of components

6.6 CREATION OF AN ASSEMBLY MODEL

The final assembly is made up of one or more subassemblies. The creation of a subassembly involves joining two or more components. This is done according to the following procedures and shown in Figure 6.5:

i. Identify a feature on the first component and a mating feature on the second component.

ii. Check for the existence of a mating relationship defined for the features.

iii. Validate dimensional and shape compatibility between the two features.

iv. If all conditions are met, the assembly is recognised to be valid and related functions to assemble the feature are generated.

v. Steps 1 to 4 are repeated until all features are assembled to form a subassembly.

vi. The final assembly is created by the same procedures, except that pairs of features within the subassemblies are checked for mating relationships.

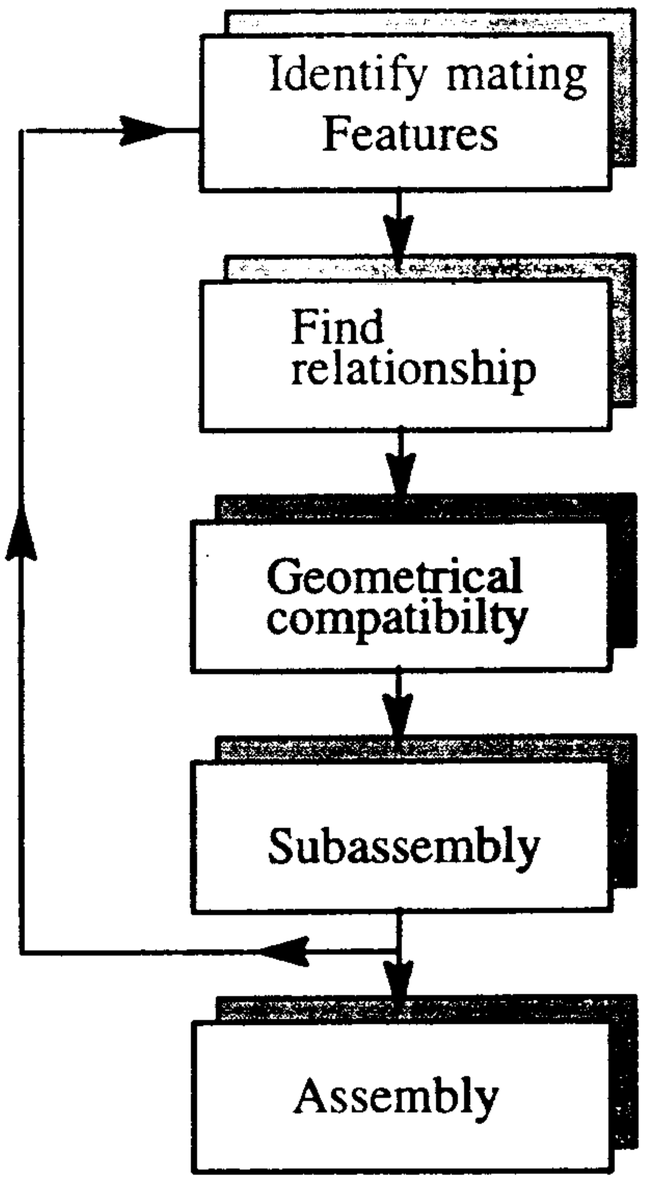


Figure 6.5: Steps in the creation of assembly

6.7 ASSEMBLY DATA

The assembly data provides information on the assembly, based on the Assembly Graph presented in Section 5.3. This is generated by the system after the creation of the assembly and stored in a file. The contents of the assembly data are:

name of assembly
location
orientation
list of subassemblies
list of components
list of features

For example the assembly data for the lathe tool post assembly illustrated in Figure 5.2 is as follows:

lathe tool post
0,0,0
0,0,0
post, pin, tool_post
slide, top_slide, tee_bolt_pin
pin, boss, boss
tool_post, hole, hole, hole, thru_slot
top_slide, thru_slot, notch, notch
tee_bolt_pin,boss,step,step

6.8 EXAMPLE 1 – PIN AND BLOCK

In this section, the FBDS is used to model a simple assembly which consists of two components – a pin and a block, illustrated in Figure 6.6 with the dimensions of each feature. The assembly involves a mating process between two pairs of features – a boss and a hole and two faces of the rectangular bosses. The Assembly Graph is shown in Figure 6.7. The creation of the assembly follows the procedures described in earlier sections. Letters in italics denote data input by the user.

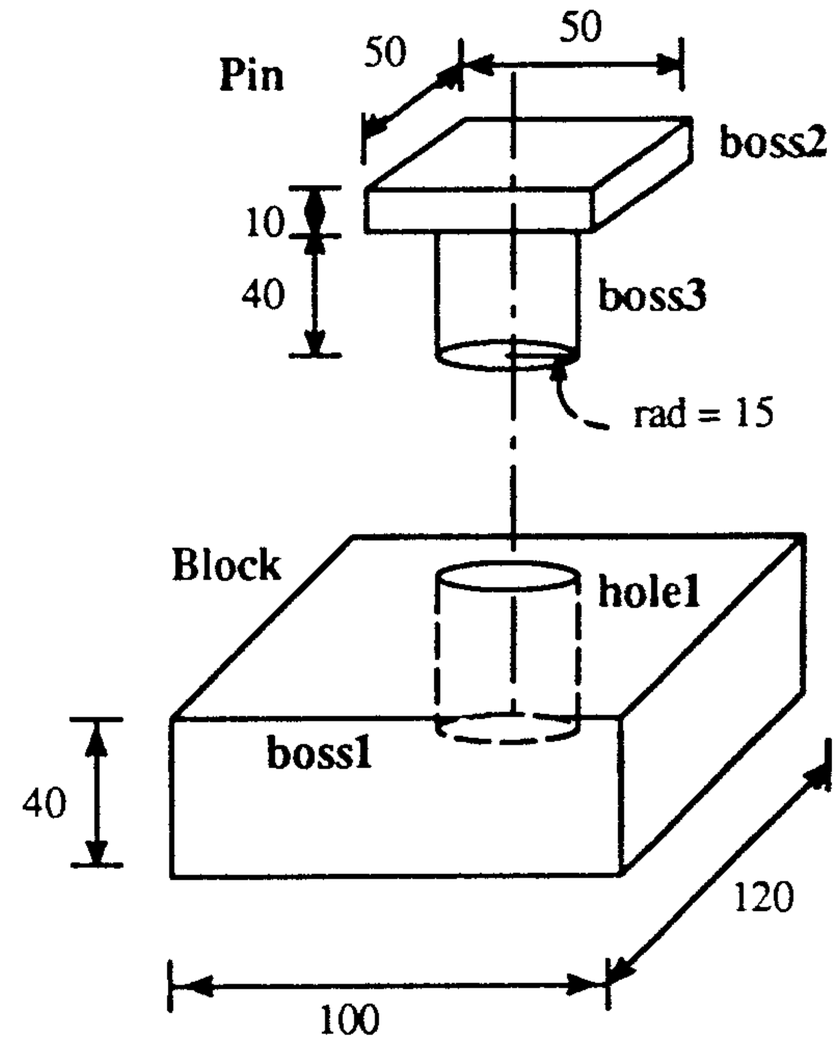


Figure 6.6: A pin and block assembly

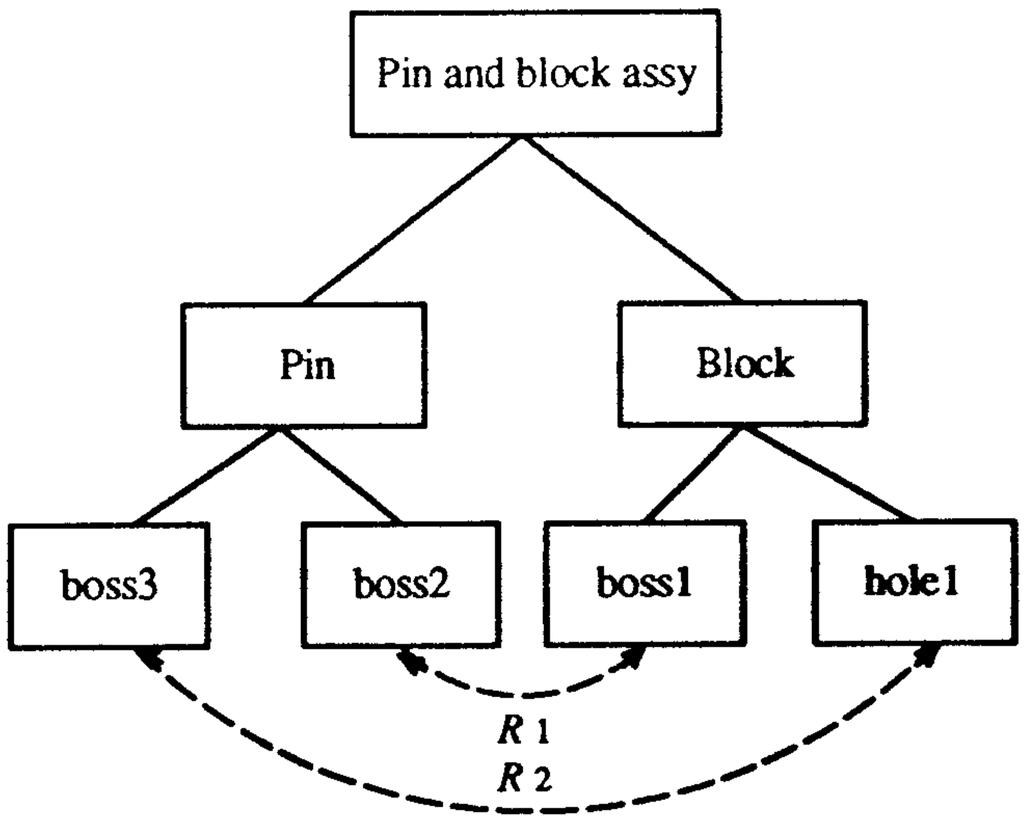


Figure 6.7: Assembly structure for pin and block

The first step is to create each component of the assembly. The block consists of a base feature which is a rectangular boss and a cylindrical hole is attached to it. The input data for the boss is shown in Figure 6.8:

Component Name:	*block*
Location x:	*0*
y:	*0*
z:	*0*
Orientation:	*0, 0, 0*
Feature type:	*boss*
Profile:	*rectangular*
Length:	*100*
Width:	*120*
Height:	*40*

Figure 6.8: Input for a rectangular block component

A hole feature is then created and attached to the base feature. The input for the hole feature is shown in Figure 6.9.

Feature type:	*hole*
Profile:	circular
Radius:	*15*
Height:	*40*
Location x:	*0*
y:	*0*
Orientation:	*0, 0, 0*

Figure 6.9: Input for a cylindrical hole feature

The hole feature is then subtracted from the base feature to create the block component. Next, the pin component is created in the same manner. First, a rectangular boss is created as a base feature and then a cylindrical boss is attached and united to form the pin

component. The data for these features are shown in Figures 6.10 and 6.11.

Component Name: *pin*
Location: x: *0*
y: *0*
z: *150*
Orientation: *0, 0, 0*

Feature Type: *boss*
Profile: *rectangular*
Length: *50*
Width: *50*
Height: *10*

Figure 6.10: Input for a pin component

Feature type: *boss*
Profile: *circular*
Radius: *15*
Height: *40*
Location x: *0*
y: *0*
z: *150*
Orientation: *0, 0, 0*

Figure 6.11: Input for a cylindrical boss

To confirm the model created so far, the user goes to the test harness and views the model. This is shown in Figure 6.12.

To assemble a boss feature to a hole feature, the user goes to the Create Assembly option. The inputs to the system are shown in Figure 6.13 and the assembled model is viewed in the test harness and shown in Figure 6.14.

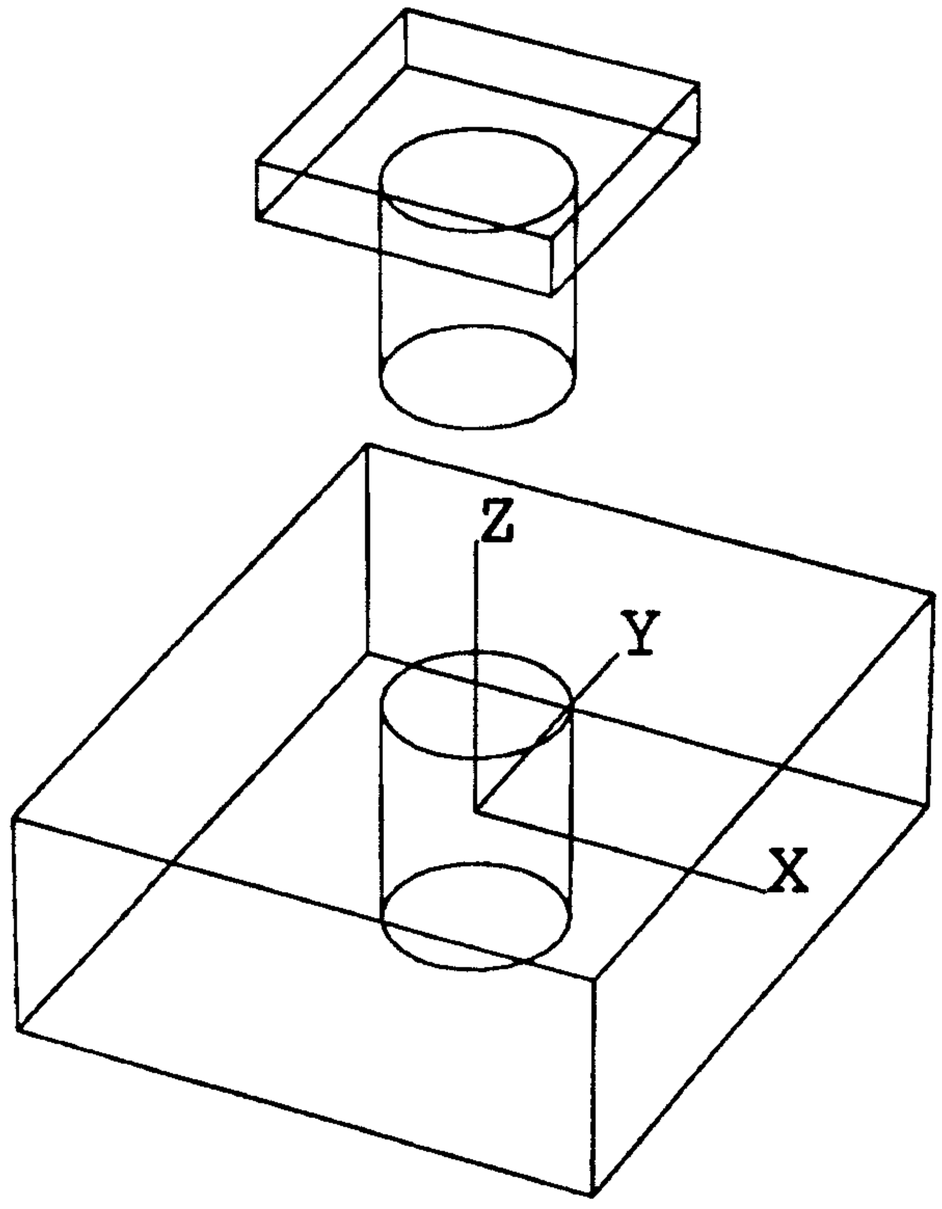


Figure 6.12: ACIS model for pin and block

Features to assemble:
Feature 1: *hole*
Feature 2: *boss3*

The system responds:

pin.boss3–fits–block.hole1

Other features to assemble y/n:*y*

Feature 1: *boss*1
Feature 2: *boss2*

The system responds:

pin.boss1–against–block.boss2

Other feature to assemble y/n: *n*

Create assemble y/n? *y*

Figure 6.13: Input for assembly relationship

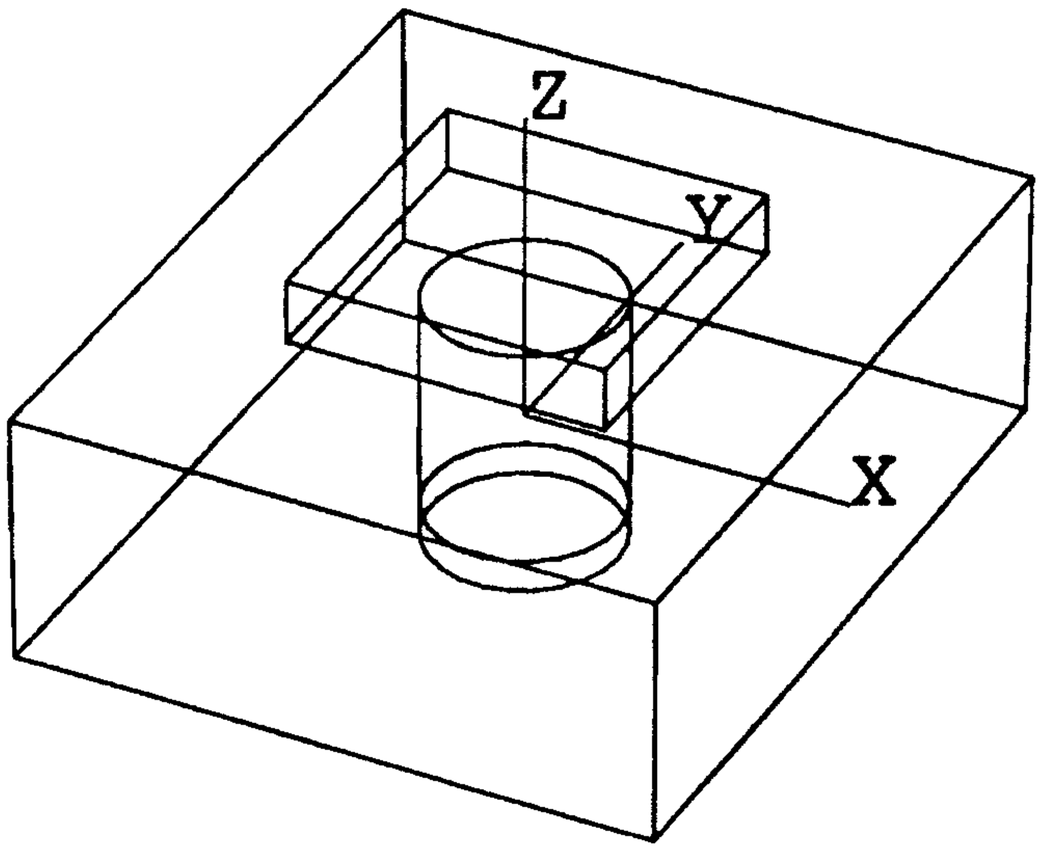


Figure 6.14: Model of pin and block assembly

In this assembly, the component and the subassembly are the same. The assembly data for the example is as follows:

pinblock

0,0,0

0,0,0

block, pin

block, boss, hole

pin, boss, boss

The data structure for the block and pin assembly is shown in Figure 6.15.

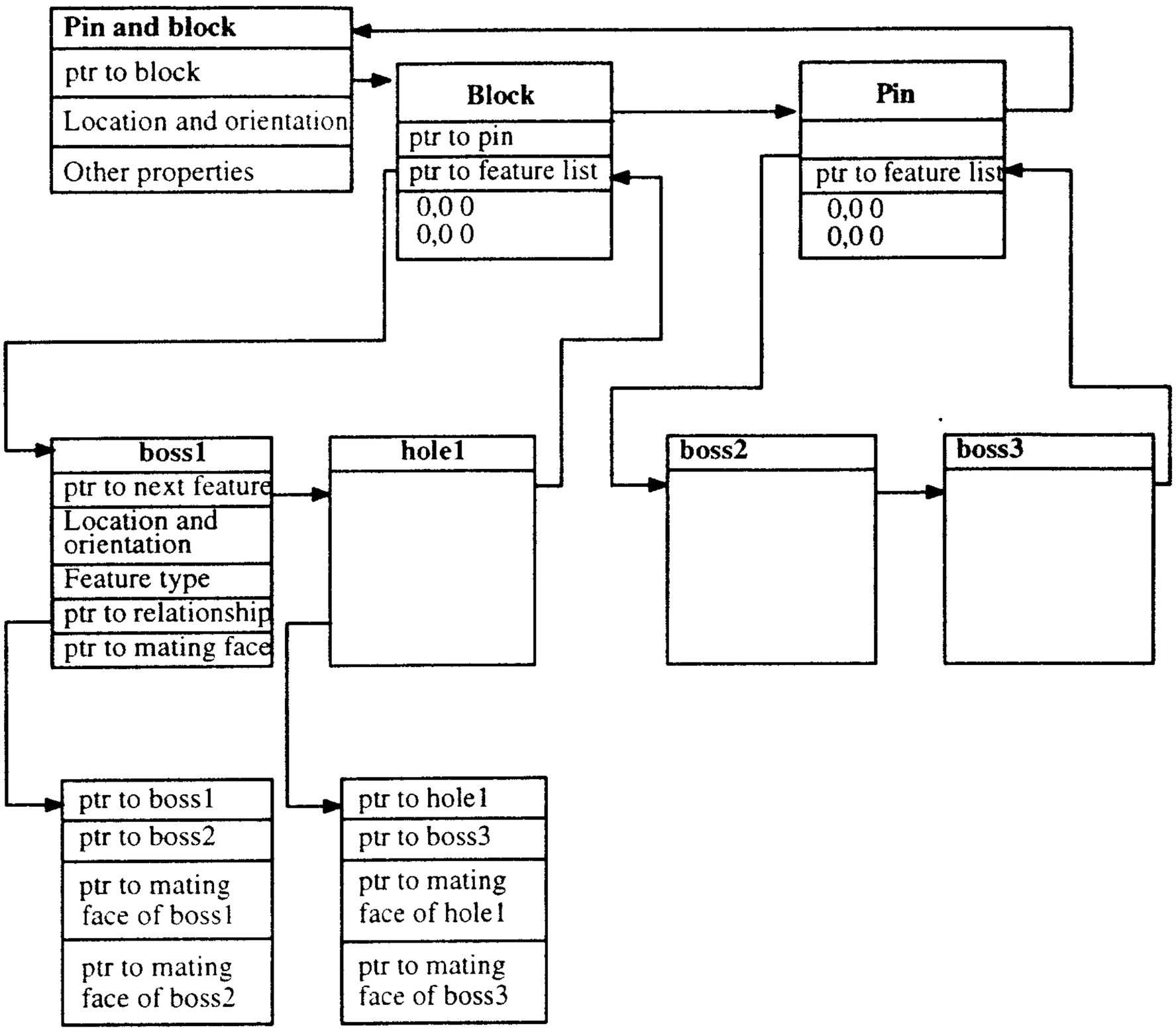


Figure 6.15: Data structure for the block and pin assembly (abbreviated)

6.9 EXAMPLE 2 – EJECTOR PLATE ASSEMBLY

Figure 6.16 illustrates ejector parts of a typical injection mould assembly. The ejector plate is assembled to the rear clamping plate. An ejector retainer plate is assembled on top of the ejector plate. The ejector plate and the retainer plate are held against each other by four pins which pass through the respective holes. The dimensions of each part are given. Other features present on the plates such as threaded holes are omitted. The aim of this exercise is to show an assembly process which involves the simultaneous interaction of more than one pair of features.

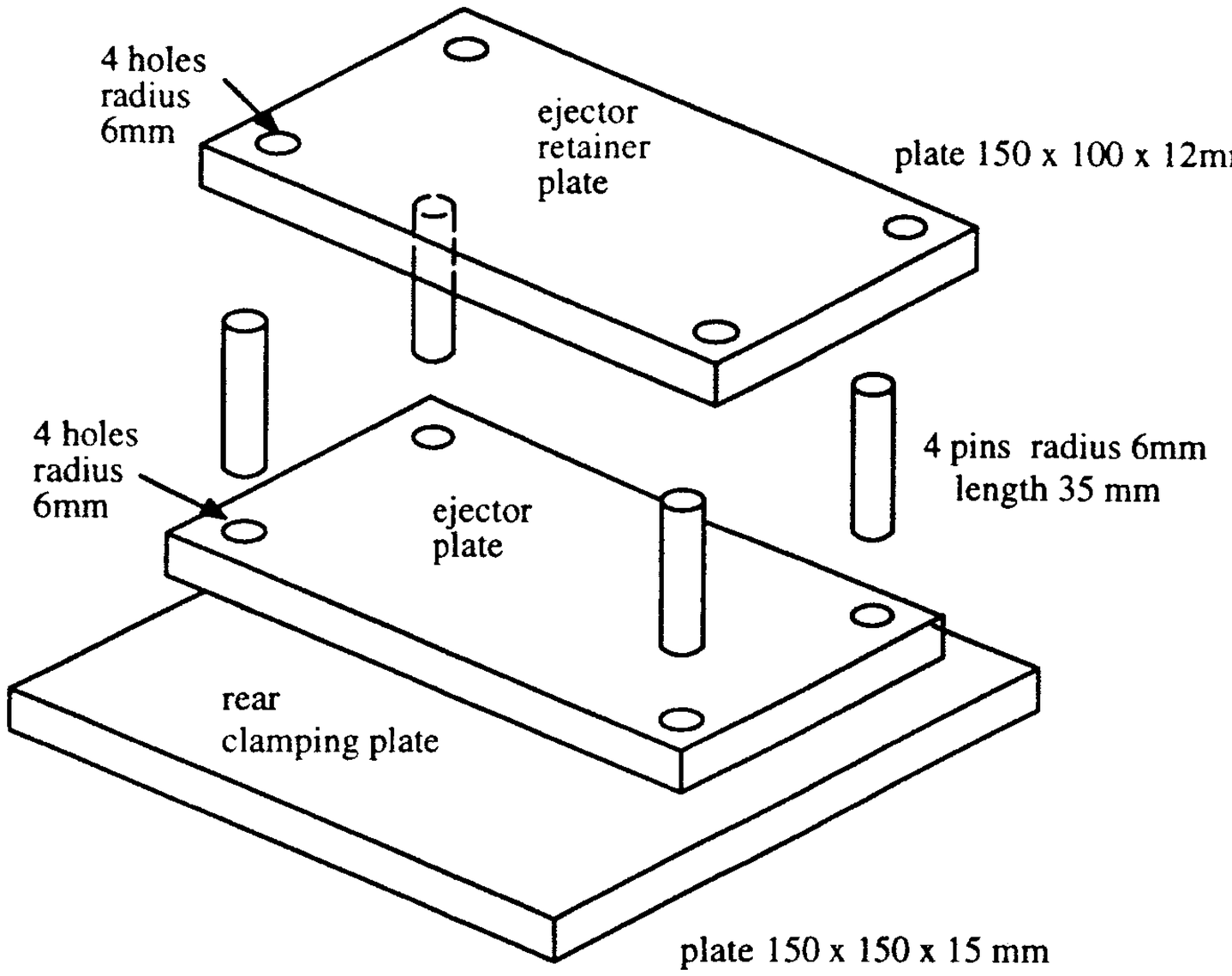


Figure 6.16: Ejector plate assembly

The assembly consists of seven components each of which consists of one or more features, as shown in Figure 6.17. The rear clamping plate is simply made up of a rectangular boss, while the ejector plate and ejector retainer plate each has a rectangular boss as its base feature and four holes as additional features. The four pins are simply cylindrical boss base features. The rear clamping plate is created first and its location and orientation become a reference for subsequent components. The pins are first assembled to the holes of the ejector plate and then to the holes of the ejector retainer plate. The process of creation of the assembly is shown in Figure 6.18. The ACIS model for the components created by the system is shown in Figure 6.19.

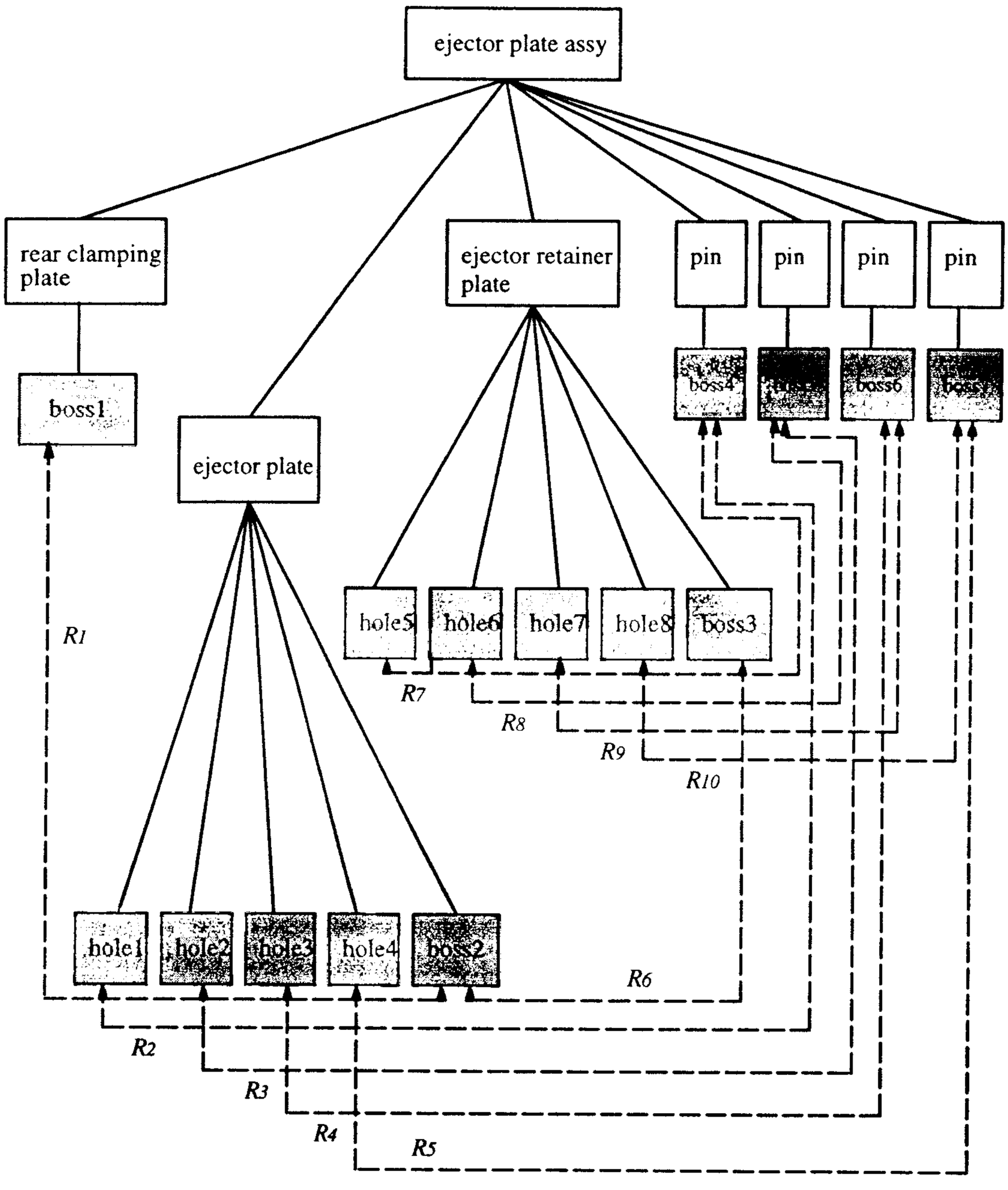


Figure 6.17: Assembly Graph for ejector plate assembly

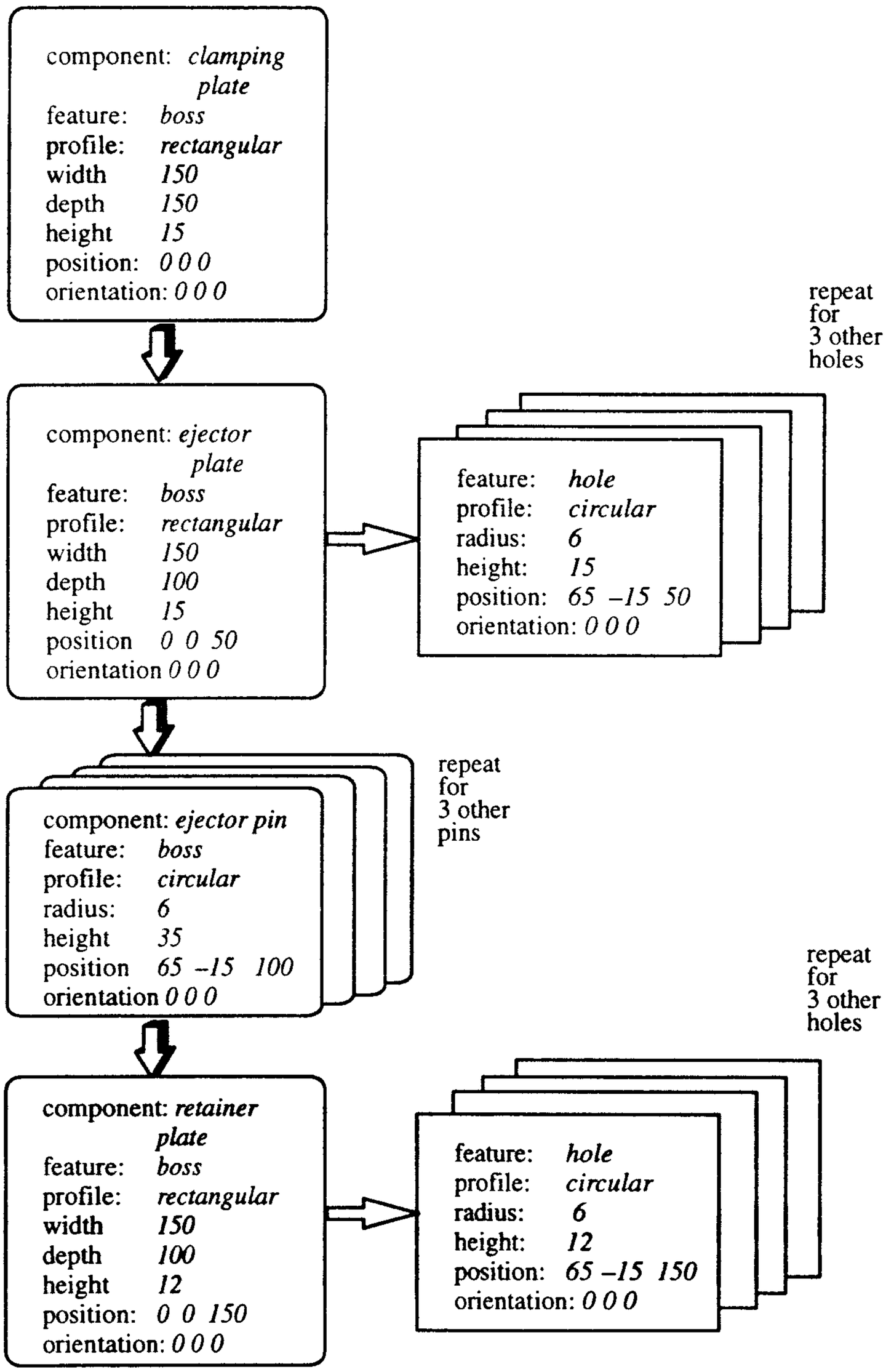


Figure 6.18: Data input for ejector plate assembly

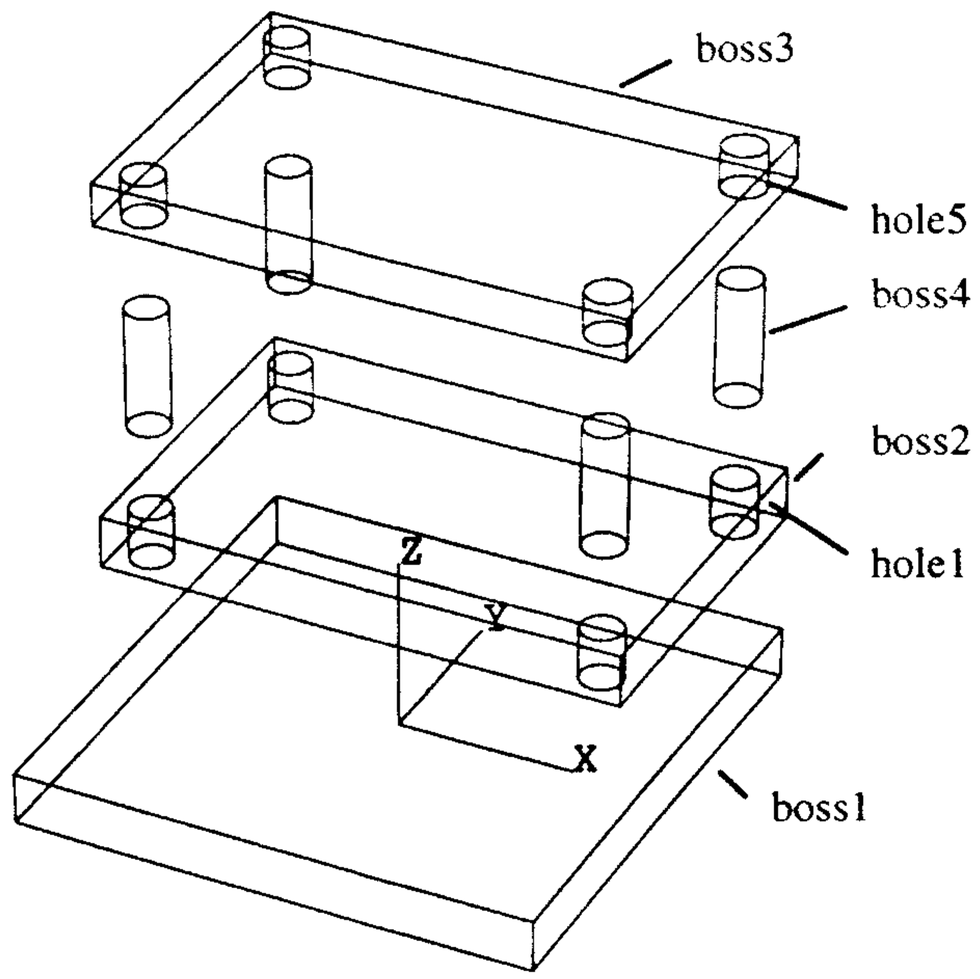


Figure 6.19: Model of ejector plate features

The assembly procedure involves the pairing of features which mate with each other. There are ten pairs of features to be mated. The ejector plate, represented by boss2 mates with the clamping plate (boss1) and the retainer plate (boss3). Each pin has a *fit* relationship with two holes, one on the ejector plate (holes 1 to 4) and another one on the retainer plate (holes 5 to 8). The mating data for some of these features are shown in Figure 6.20.

The relationship expressions are generated as shown:

rear_clamp_plate.boss1–against–eject_plate.boss2

eject_plate.hole1–fits–pin.boss4

eject_plate.hole2–fits–pin.boss5

eject_plate.hole3–fits–pin.boss6

eject_plate.hole4–fits–pin.boss7

eject_plate.boss2–against–retain_plate.boss3

retain_plate.hole5–fits–pin.boss4

retain_plate.hole6–fits–pin.boss5

retain_plate.hole7–fits–pin.boss6

retain_plate.hole8–fits–pin.boss7

Mating Features:
Feature 1: *boss1*
Feature 2: *boss2*

Feature 1: *hole1*
Feature 2: *boss4*

Feature 1: *boss2*
Feature 2: *boss3*

Feature 1: *boss4*
Feature 2: *hole5*

Figure 6.20: Input for mating features

The assembled model is shown in Figure 6.21.

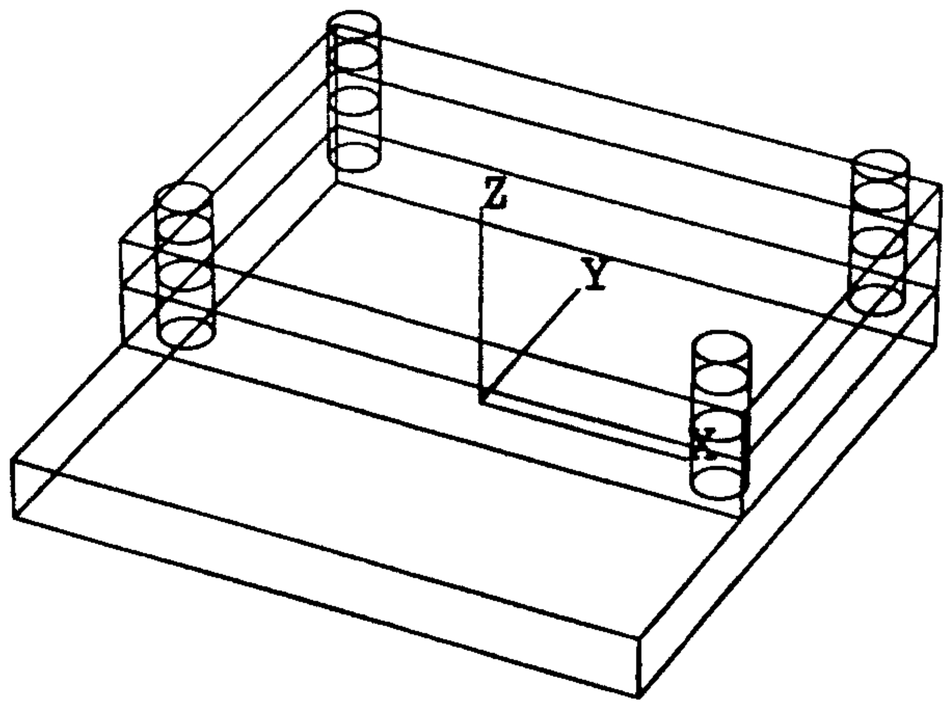


Figure 6.21: Assembly model for the ejector plate

The data for the assembly is as follows:

ejector plate assembly

0 0 0

0 0 0

clamping_plate, ejector_plate, pin, pin, pin, pin, retainer_plate

clamping_plate, boss

ejector_plate, boss, hole, hole, hole, hole

pin, boss

pin, boss

pin, boss

pin, boss

retainer_plate, boss, hole, hole, hole, hole

The data structure for the ejector plate assembly is shown in Figure 6.22.

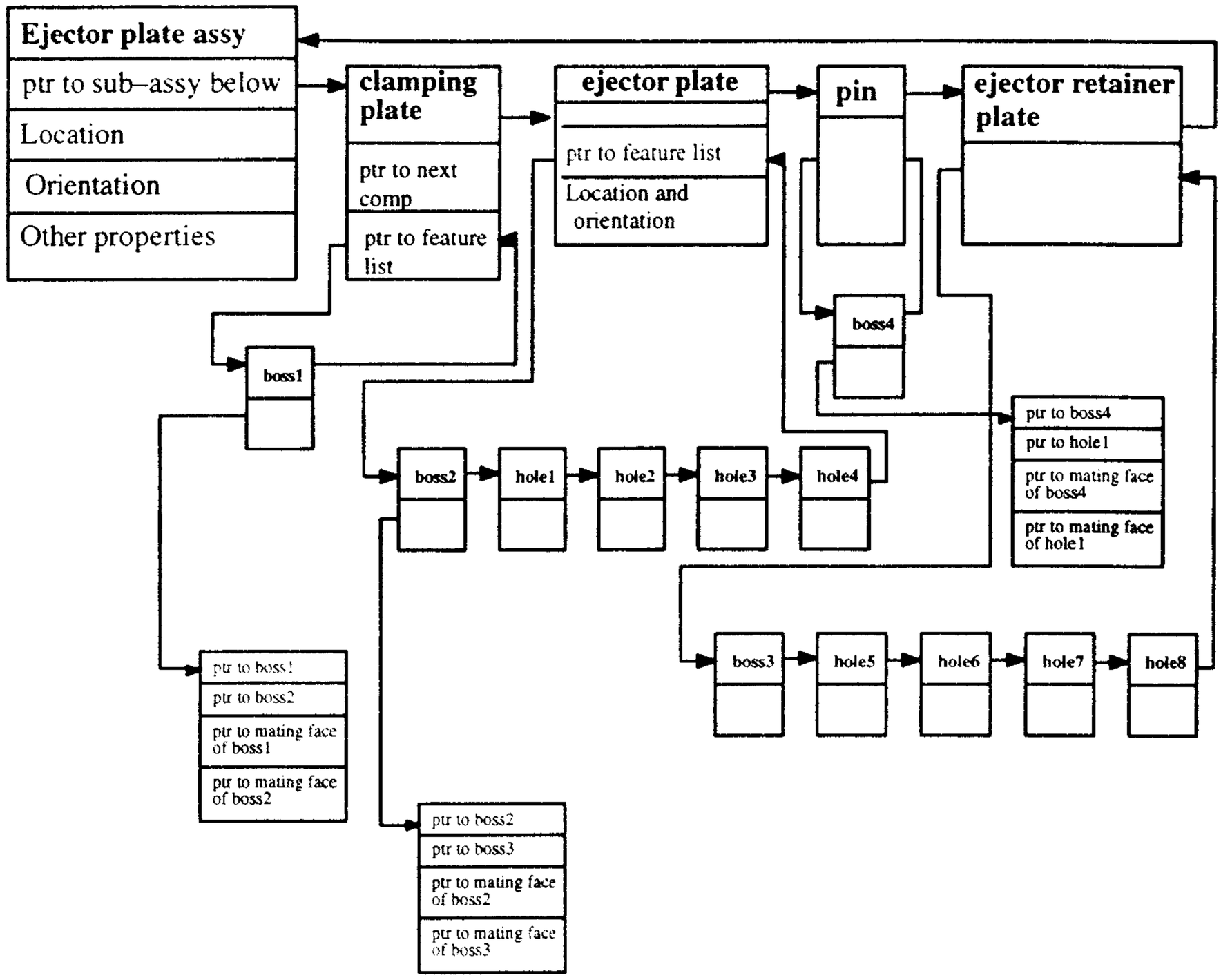


Figure 6.22: Data structure for ejector plate assembly (abbreviated)

6.10 SUMMARY

A prototype feature–based assembly modelling system has been presented which is based on the design by features approach. The system consists of a library of features which interface with ACIS methods and classes for the generation of models. The system provides interactive input of the feature data as well as features to be mated. The procedures for creating features, components, subassemblies and an assembly have been described, and the applicability of the approach has been tested by modelling two examples of mechanical assembly. The outputs from the modelling session have been presented. The outcome of this exercise and the limitations of the system are discussed in the next chapter.

CHAPTER SEVEN
DISCUSSION

7.1 INTRODUCTION

This chapter reviews work on feature representations, assembly representations and the implementation described in the three preceding chapters. The overall approach adopted is critically analysed and the limitations are identified. Section 7.2 summarises the research methodology. Discussions on the representation of features and assembly are covered in Sections 7.3 and 7.4 respectively. Comments on the use of an object–oriented approach are given in Section 7.5. Section 7.6 discusses the practical implementation of the system, with regards to the development of a prototype feature–based modelling system.

7.2 REVIEW OF THE METHODOLOGY

This thesis has described the methodology for representation of assembly modelling in a feature–based environment using an object–oriented approach. The methodology starts with the creation of features which form basic entities in the assembly. The fundamental approach has focussed on:–

i. Defining a set of features within a suitable taxonomy.

ii. Analysing typical assemblies to identify the assembly interactions and incorporating the knowledge on assembly representation into the feature representation.

iii. Establishing techniques for the representation and modelling of a range of features and assembly knowledge.

iv Incorporating process planning knowledge into the assembly representation.

v. Applying object–oriented concepts of abstraction, encapsulation and inheritance to significantly reduce the quantity of software and the creation time while at the same time allowing the knowledge and domain to be extensible and flexible.

vi. Designing a Design by Feature system that allows incremental construction and evolution.

The intention has been to establish a data model for features that is capable of representing assembly and process planning information in addition to the geometrical and topological details and which also handles information concerning the functional requirements of related features in an assembly of parts.

7.3 FEATURE REPRESENTATION

There is no doubt that the feature–based approach provides a very convenient way of representing geometrical as well as non–geometrical information. However, for features to be useful in the integration of product life cycle activities, it is preferable to have a single unified representation. It is envisaged that this aim can be achieved by defining a generic feature. However, in practice this is not possible due to the range of complexity of products such as sculptured and sheet metal products. Thus the approach taken in this research of defining a range of the most common shapes of machined features is more practical and supports many applications such as process planning, assembly planning and inspection. The taxonomy established is comprehensive enough to include most shapes used in the process planning and assembly of machined parts. This is justified since a large part of components machined in industry consists of simple shapes produced by operations such as milling, turning and drilling (HMSO statistics reported in Case and Acar 1989).

A feature representation which is based on a similar approach has proved to be effective in the process planning and process capability modelling applications (Case 1994). In this work, features were defined in a feature library and implemented in a B–Rep solid modeller, Imaginer. A design by feature user interface to the solid modeller was developed to allow designers to generate components using feature primitives and to store attributes in a feature–based data structure, which is separate from the database of the geometric modeller. The way in which features relate to each other on the same component have been defined for this purpose. This earlier work has been incorporated

into an overall scheme of data representation that includes assembly knowledge to establish that a single feature representation can be useful across a range of key manufacturing applications. The feature representation not only covers a single component, as in the process planning work, but also supports interaction between components at the assembly level. The process planning knowledge can be redesigned using the object–oriented approach and combined with the assembly knowledge in the feature to provide a multiple representation within a single feature definition.

The feature model defines representations for various items of knowledge that have not been fully implemented in the prototype system. In particular, Geometric and Dimensional Tolerances are excluded whereas they clearly have an important role to play in assembly modelling. This deliberate omission was a consequence of the scale and the complexity of the tolerances issue that it was felt could eclipse the main issue of representing assembly knowledge within the feature. However, it is felt that the use of object–oriented concepts and techniques has provided a framework for the future inclusion of tolerance aspects.

7.4 ASSEMBLY REPRESENTATION

The representation of the assembly focuses on the role of the feature as the basic unit in the assembly and emphasises the relationships between features. This is consistent with the goal of the design by feature approach and the way in which designers visualise mechanical assemblies. The hierarchical model organises the relationships in the assembly and provides a more realistic representation of the role of the features in the assembly, since many assemblies are designed sequentially. Assembling in this way can confine attention to relationships between a pair of features at one time. The hierarchical structure fits well with the object–oriented approach.

Much research in assembly modelling and assembly planning utilises the mating relationship approach to specify spatial relationships among assembled parts. The three mating relationships defined in this research represent the most common types and they are the most suitable for the range of features used and the static nature of the assembly,

where components are assembled onto a static base feature. These mating conditions are well suited to the types of features defined. A possible criticism of using this approach is that these mating conditions may not fully represent the assembly, as highlighted by Shah and Rogers (1993) and Baxter et. al. (1992). Thus it is possible to define mating conditions for specific applications in the way for example that Baxter et. al. (1992) proposed conditions which accommodate the mating of two gears.

Certain limitations are a natural consequence of the way in which geometric models are constructed in solid modelling CAD systems (feature–based or otherwise). Hence fastening details such as threads are usually considered as secondary features that can only exist on a base feature, and do not normally have a direct representation in the geometric modeller. This type of attachment is used only to modify the type of relationships defined and can be incorporated in the assembly knowledge of the feature by the inclusion of appropriate attributes without disturbing the basic structure of the information.

Another issue is that a single feature may mate with more than one feature in general orientations within the assembly and not just along the three major axes. However, the mating relationship expressions are independent of this factor, and mating is restricted to the linear orientation only so as not to complicate the prototype implementation with well–known but mathematically complex methods. Furthermore there is some practical justification for this as seventy five percent of assembly involves only linear assembly (Delchambre 1991) and it is in line with the objectives of Design for Assembly techniques of reducing occurrences of non–orthogonal assembly directions.

7.5 USE OF THE OBJECT–ORIENTED APPROACH

The advantage of using OO data models for building the knowledge environment for assembly is the straightforward integration with OO programs. The power of the OO technique, as outlined in Chapter 3 is in the knowledge representation and manipulation. The OO approach was found to be more capable than the conventional method of addressing the problems of representation of features. The hierarchical nature of the

feature taxonomy and the assembly structure is well–suited to an OO implementation. It also allows effective manipulation of features, by providing convenient ways to extend feature functionalities as well as feature types and profiles. The approach provides easy maintenance of the system through its modular design and the addition of attributes and functions which are independent of each other.

In this case, the feature library is defined independently of the solid modeller. This allows for future expansion of the system or the transfer of the feature library into another solid modeller.

The flexibility of the system is enhanced by the use of the inheritance approach where a derived class can share the common methods of the base class while at the same time define its own set of attributes and methods. This is found to be useful in extending the program to provide additional features and functionalities.

The use of the C++ programming language is also well–suited to the whole framework of the system by providing a convenient method of programming, and it is envisaged that the task would have been more time–consuming using a more conventional approach.

7.6 PRACTICAL IMPLEMENTATION ISSUES

The design of the prototype feature–based design system presented in Chapter 6 had the aim of testing the idea presented for a small range of feature types. As a prototype system, it has limited practical application. The base feature is limited to a rectangular shape. A more practical system should address a wide range of possible shapes for a base feature and a complete set of feature types and profiles. In order to achieve this goal, large resources in terms of programming times and skills are required.

In implementing the system, two approaches of creating solid models are possible – using a solid modeller or a kernel modeller. The former method involves using a CAD/CAM system such as Unigraphics, whereby features are defined and stored in a library and called during the design session. The advantage of using this approach is the availability of a good user interface and powerful graphics on a single system. The system is usually easy to use and the user interface can be customised. However, depending on the

language with which the software is developed, the programming may be difficult and limited and frequently does not allow adequate access to low level entities. Usually a program is required to interface the feature library and the solid model. Further development may also be restricted by the proprietary nature of the system, and there is also a likely problem in extending the system for other applications such as assembly planning, due to difficulties encountered in accessing low levels of the (topological) data structure.

The second approach of using an open architecture kernel modeller, which is the basis of this research, offers a more flexible approach. ACIS provides a library of functions which act as building block components which can be used by application developers. The user has to choose the right function to integrate with the feature library. A major advantage of using ACIS is its extensibility. Any further development of the system such as developing a fully–fledged assembly modelling system or design for assembly system is relatively easier, as discussed in Chapter 3. The capability of the system can be enhanced by creating new API functions to complement the available ones. This can provide a consistent interface with ACIS.

The disadvantage of this approach is the programming aspect. The capability of the prototype system can be greatly enhanced by good programming skill. Although the use of OO has alleviated the programming aspect, the experience of this research shows that the level of programming knowledge required to produce an application system is very high and much of the time is required for programming.

The role of the user interface to the system is very important in the assembly modeller. The aim is to minimise the interaction with the user and provide an interactive display of the model being assembled. In this system, the user interacts with the system by inputting the data on the screen, and the use of the ACIS test harness provides only a limited user interface An improvement to the system could be made if data on the dimensions, location and orientation of the feature could be automatically generated when the feature is created and if the user could click on the pair of features to be mated, with the system identifying the consequent mating relationships. These kinds of interaction may be

possible using a more powerful graphical user interface engine such as MOTIF (OSF 1989), but were not considered to be an essential research aspect of this work.

During the course of this work substantial improvements have been made to kernel modellers in general and ACIS in particular. It is not possible to track all developments as they occur, and thus more recent versions of ACIS (version 1.6) offer many additional features which enhance the modelling and the user interface in ways that are important to assembly modelling. Similarly, the last few years has seen the development of a number of husks which, had they been available at the start of the research, would have proved useful. A partial solution to this problem is to utilise an ACIS–based solid modeller such as Bentley's MicroStation (Bentley 1995), whereby the user is presented with a user interface system and at the same time can develop feature and application libraries using ACIS functions.

7.7 SUMMARY

This chapter has highlighted some salient points in the whole framework of the thesis, especially on the feature representation, the assembly representation, the use of the object–oriented approach and the design of a feature–based design system. Limitations of the system have also been discussed. This work is significant as the approach can be extended to other upstream applications such as assembly planning and Design for Assembly. The summary of the work covered in this thesis and how it contributes to manufacturing knowledge are outlined in the next chapter.

CHAPTER EIGHT
SUMMARY AND CONCLUSIONS

8.1 INTRODUCTION

This chapter concludes the thesis by summarising discussions in previous chapters, outlining the contributions to the research area and proposing further work that can be pursued in this area. Section 8.2 summarises the work discussed in previous chapters. The research contributions is discussed in Section 8.3. Potential areas for future research are discussed in Section 8.4. Section 8.5 concludes the thesis.

8.2 SUMMARY OF THE THESIS

This research set out to address the problem of the lack of a unified definition for features and to determine an assembly representation as an integral part of a feature–based representation. These issues have been highlighted by a review of relevant literature on features and assembly modelling. The problems arising from a feature being defined for a specific application have been recognised by various researchers, and current trends in the development of feature–based systems have been identified. The need to arrive at a single feature definition for multiple applications has been recognised and in this context, assembly modelling is deemed important as a complement to the well–established process planning activity.

The focus of this research is on the representation of assembly knowledge within a feature–based model in order to show that a single feature representation can support multiple applications, particularly process planning and assembly modelling.

Features used in this research have been defined as machined volumes and a suitable hierarchical taxonomy has been defined in detail to cover common feature types and profiles that represent the general machined features used for assembly. A feature class hierarchy has been established that uses the concept of inheritance for ease of development and maintenance of the system. Some typical assemblies have been

analysed to identify mating relationships which occur among features. This results in the definition of three mating relationships of *against*, *fits* and *align*. The knowledge on these relationships are combined with the process planning data to identify common information in the feature.

The representation of assembly knowledge as part of a feature model has been defined and detailed. Feature definitions have been substantially enhanced to include knowledge on assembly in the form of the logical position of features within an assembly structure and the interactions between pairs of features. The interactions have been defined in terms of mating relationships which are represented by binary expressions with mating conditions as operators and features as operands. Using the inheritance concept of the object–oriented technique, classes for assembly and feature relationships have been defined. The class definitions also include the process planning knowledge.

The essential development tools for the research and the benefits of using them have been identified. The tools, in the form of the C++ programming language and the ACIS kernel modeller, are based on the technique of object–orientation which is currently seen as being the most appropriate and effective method for handling the complexities found in modern CAD/CAM systems. This methodology has been used to implement and test a simplified prototype feature–based system that combines the feature and assembly classes with existing ACIS classes to create the model. Two examples involving a simple two–part product and a more complicated multi–part one have been presented for verification in the modelling environment. Although the prototype system can successfully model the assembly, it requires some improvements in the form of a better user interface.

8.3 RESEARCH CONTRIBUTIONS

The representation of assembly information is considered to be an essential prerequisite to the generation of CAD/CAM systems that are capable of optimising product design. Such a representation can form the basis of design improvement techniques such as design for assembly (DFA) and manufacturing planning such as assembly planning. It

can also be used to support other related applications such as tolerance analysis of assemblies and inspection planning.

The main objective of assembly planning is to improve the efficiency of the assembly process in terms of time to assemble, cost and quality of finished products. One of the outputs of assembly planning is the generation of assembly sequences. These sequences are determined by various factors, the mating relationships being the most important. The mating conditions can be organised in the form of a mating graph which is similar to the Feature Relationship Graph described in Chapter 5. The assembly sequence is generated with the aid of interference checking between mating features.

This research has provide several contributions to the area of CAD/CAM, in particular to features technology and in general to the Computer Integrated Manufacturing environment. These are outlined below:

i. An object–oriented representation of a set of features which comprise knowledge useful for multiple applications such as process planning and assembly modelling has been developed. The design of the knowledge in the features allows appropriate extensions to be provided within the features, through the inheritance property of object–oriented technique, to support the needs of other applications.

ii. A hierarchical feature taxonomy has been adopted which caters for a range of feature types and profiles useful for manufacturing applications. The hierarchy is particularly suited to implementation using the object–oriented methodology.

iii. Effective representation of assembly knowledge in the feature data has been achieved, with the use of mating relationships between features. The data has been combined with relevant process planning knowledge and this provides a high level interface with the designer for the creation and modelling of assemblies. The representation is considered to be useful for assembly modelling and for other subsequent tasks such as assembly planning.

iv. An assembly data structure based on a linked list of subassemblies, components and features has been established and implemented and is considered to be useful for extended applications such as assembly planning and design for assembly.

v. A prototype feature–based modelling system has been implemented based on the ideas and methodologies presented in the thesis. This implementation uses an object–oriented kernel modeller, which is extensible and allows future development by adding appropriate functions to the existing classes or by the development of new classes.

8.4 RECOMMENDATIONS FOR FUTURE WORK

There are several areas where further investigations can be pursued, based on the ideas presented in this research and the discussion in Chapter 7. These are outlined in the following paragraphs:

8.4.1 ADDITION OF FEATURE ATTRIBUTES

The possibility of extending the feature definition to include feature attributes such as threads is deemed important as most mechanical assemblies include these parts. The approach proposed is to include an additional attribute in the class definition of the features. This can be in the form of a Boolean representation such as (thread, no_thread). A thread would be further defined by additional attributes such as the thread pitch and type. Thus a boss and a hole can be threaded to represent a bolt and a nut respectively.

The inclusion of additional attributes is also necessary for the feature–based system to be used in a wider product modelling environment. An example of this would be the inclusion of locations on the feature that could be used as inspection points. The approach of defining attributes separately from the feature prevents the feature from being associated with a particular application. This is consistent with the concept of defining features for multiple applications.

8.4.2 APPLICATION IN ASSEMBLY PLANNING

The logical extension to this work is to investigate the practicality of the approach in assembly planning, particularly in assembly sequencing. This requires the acquisition and processing of assembly knowledge. The data on the mating relationships could be used as input to this application and there are various approaches available to achieve this aim. One approach deemed suitable for the representation used in this thesis is to use an assembly graph which shows the connections between all features, in a way that is similar to the Feature Relationship Graph introduced in Chapter 5. With the help of the graph, assembly could be split into autonomous sub–assembly groups by decomposing the graph into smaller parts. The algorithm proposed by Wang and Li (1991) could be used to group the features and components. Further algorithms and heuristics would be required to order the components and features to generate a list of ordered pairs of features in an assembly. The mating relationships data could then be transformed into a connectivity matrix, as shown in Figure 8.1. Each entry in the matrix shows whether the features are connected and other algorithms proposed by Wang and Li (1991) could then be used to sequence the features in the assembly.

F / F	1	2	3	4	5	6
1	0	1	0	0	0	0
2	0	0	0	0	0	0
3	1	0	0	1	0	0
4	0	0	0	0	1	0
5	0	0	0	0	0	1
6	1	0	0	0	0	0

Figure 8.1: Mating Relationship Matrix

8.4.3 VALIDITY CHECKING

The representation of assembly knowledge in features has been tested on simple products. In order to be fully confident that this representation is applicable in manufacturing environments, the validity of the approach needs to be checked. This could be achieved by testing the approach on a range of real products which consist of

parts within the domain of the features described in this research. A product would be modelled and then assembled according to the procedures described in Chapter 6.

The approach could also be tested for its applicability in a manufacturing environment by conducting controlled experiments involving use of the system by designers and engineers. Objective measures of performance in terms of time, accuracy and quality of assembly information could be obtained and compared with similar data from manual or alternative computer approaches. At the same time subjective measures of the suitability and acceptability of the method could be obtained using survey and observational techniques. However, for such studies to be meaningful there would be a need to develop the prototype system to a level where its user interface was comparable with commercial systems. This case study could also form the basis for the improvement of the system or the general approach.

8.4.4 INTEGRATION WITH CAD/CAM SYSTEMS

An assembly modelling system will be useful if it can be integrated with other CAD/CAM systems. The use of ACIS has made it possible to transfer files to many popular CAD/CAM systems or to use data translators to import from or export to other systems. Many CAD/CAM systems such as the recent versions of AutoCAD solid modeller and Microstation can read files from ACIS. ACIS can also be interfaced to many other CAD/CAM systems using IGES or STEP translators which are commercially available.

Data transfer could be an alternative to the development of a graphical user interface within the feature–based modelling system. Models created by the system could be transferred to the CAD/CAM system for further analysis and manipulation, and thus enhance the capability of the feature–based assembly modelling system. Transfer of data between a general CAD/CAM system and an assembly modelling system could be beneficial, and imitates the way in which many design systems gain access to design analysis methods. However, true integration can only be achieved by implanting the data structure and methods of assembly modelling within a general CAD/CAM system. The

use of ACIS, a commonly used kernel, and the object–oriented technique makes this a feasible approach.

8.4.5 OTHER MANUFACTURING APPLICATIONS

There are opportunities to extend the object–oriented features approach to other manufacturing applications such as inspection planning. To achieve this, features in an assembly can be created and then analysed using a CAD/CAM system which can communicate information to a Coordinate Measuring Machine. Feature attributes required for inspection planning include the geometry and topology, shape and precision attributes, relations between features in the assembly hierarchy and relevant technological data (ElMaraghy and ElMaraghy 1994). With the exception of the precision attributes (tolerance information), this information is available in the feature representation presented in this thesis.

The application of the features to other manufacturing areas should provide further evidence of the extensibility of the object–oriented approach and that the feature representation can support multiple applications.

8.5 CONCLUSIONS

The potential application of features in geometric modelling has been demonstrated by much research and industrial work. The research presented here reinforces the idea that features can be used in multiple applications and that the object–oriented approach assists in moving towards a unified definition for features. Features which have previously been used for process planning have been used in an enhanced form to represent an assembly. The use of features for assembly modelling provides a natural representation, since in assembly operations it is the feature that dictates the way in which parts are assembled. Features technology, combined with the object–oriented technology form a powerful means to represent manufacturing knowledge.

The approach adopted provides a design tool for designers by allowing them to create a mechanical assembly in terms of features, which is applicable for subsequent

manufacturing planning activities. The feature representation methodology implemented is suitable for the concurrent representation of knowledge on process planning and assembly modelling. Clearly, this does not conclusively establish that *all* aspects of design and manufacturing can be encapsulated in a single representation, but it goes some way to confirm the feasibility of the idea.

Future CAD/CAM systems will be more heterogeneous in nature. A number of database requirements must be considered to control and support design, manufacturing, assembly and related applications. This research shows that features have much to offer in effectively fulfilling the requirements of a Simultaneous Engineering environment.

REFERENCES

AL–ASHAAB, A. H. S. and **YOUNG, R. I. M.**, Design for Injection Moulding in a Manufacturing Model Environment, *Journal of Design and Manufacturing*, No 5, 1995, pp 45 – 54

ALLADA, V. and **ANAND, S.**, Feature–Based Modelling Approaches For Integrated Manufacturing: State–of–the–Art Survey and Future Research Directions, *Int Journal Computer Integrated Manufacturing*, Vol. 8, No. 6, 1995, pp 411 – 440

BAXTER, J. E., JUSTER, N. P. and **de PENNINGTON, A.**, An Assessment of Assembly Mating Conditions In the Context of a Product Model, *Computers in Engineering*, Vol 1, ASME, 1992, pp 421 – 428

BEDWORTH, D. D., HENDERSON, M. R. and **WOLFE, P. M.**, *Computer–Integrated Design and Manufacturing*, McGraw–Hill International Editions, 1991

BENTLEY SYSTEMS, INC., *MicroStation Modeler*, Product Summary Brochure, 1994

BOND, A. H. and **CHANG, K. J.**, Feature–Based Process Planning for Machined Parts, P*roc ASME Computers in Engineering Conf*, S.Fransisco, Jul/Aug 1988, pp 571 – 576

BONNEY, M. C., TAYLOR, N. K. and **CASE, K.**, Using Computer–Aided Design and Expert Systems for Human Workplace Design, in *Geometric Reasoning*, Woodwark, J. (ed), Clarendon Press, Oxford, 1989, pp 269 – 282

BOOCH, G., *Object–Oriented Design With Applications*, Benjamin/Cummings, California, 1991

BOOTHROYD, G. DEWHURST, P. and **KNIGHT, W.**, *Product Design for Manufacture and Assembly*, Marcel Dekker, Inc., New York, 1994

BROONSVORT, W. F. and **JANSEN, F. W.**, Feature Modelling and Conversion–Key Concepts to Concurrent Engineering, *Computers in Industry*, vol 21, 1993, pp 61 –86

BUTTERFIELD, W. R., **GREEN, M. K.**, **SCOTT, D. C**. and **STOKER, W.J.**, Part Features for Process Planning, *Report C–85–PPP–03, CAM–I Inc*, Arlington, Texas, 1986

CASE, K. and **ACAR, B. S.**, Learning Studies in the Use of Computer Aided Design Systems for Discrete–Parts Manufacture, *Behaviour and Information Technology*, Vol 8, No 5, 1989, pp 353 – 368

CASE, K. and **GINDY, N.**, Future Directions in Features Research, *Internal Document, LUT*, Nov 1991

CASE, K. and **GAO, J.**, Feature Technology – An Overview, *Int Journal Computer Integrated Manufacturing*, Vol. 6, No 1 & 2, 1993, pp 2 –12

CASE, K., **GAO, J.** and **GINDY, N.**, LUT–FBDS: A Feature–Based Design System, *Project Final Report, SERC Grant No: GR/G 35657*, Dept of Manufacturing Eng, LUT, 1993

CASE, K., Using a Design by Features CAD System for Process Capability Modelling, *Computer Integrated Manufacturing Systems*, Vol 7, No 1 1994, pp 39 – 49

CATANIA, G., Form Features for Mechanical Design and Manufacturing, *Journal of Engineering Design*, Vol 2, No 1, 1991, pp 21 – 43

CHAMBERLAIN, M. A., **JONEJA, A**. and **CHANG, T. C.**, Protrusion–Features Handling in Design and Manufacturing Planning, *Computer–Aided Design*, Volume 25, Number 1, January 1993, pp 19 – 28

CHANG, T. C., *Expert Process Planning For Manufacturing*, Addison Wesley, USA, 1990

CHANG, T. C. and **WYSK, R. A.**, *An Introduction to Automated Process Planning Systems*, Prentice–Hall, Inc, New Jersey, 1985

CHEN, Y. M., MILLER, R. A. and **VEMURI, K. R.**, A Framework for Feature–Based Part Modelling, *Computers in Engineering* – Vol. 1, ASME 1991, pp 357 – 365

CHEN, C., SWIFT, F., LEE, S., EGE, R. and **SHEN, Q.**, Development of a Feature–Based and Object–Oriented Concurrent Engineering System, *Journal of Intelligent Manufacturing*, No. 5, 1994, pp 23 –31

CHOI, B. K., BARASH, M. M. and **ANDERSON, D. C.**, Automatic Recognition of Machined Surfaces From a 3D Solid Model, *Computer Aided Design*, Vol 16, No 2, 1984

CHUNG, J. C. H., COOK, R. L., PATEL, D. and **SIMMONS, M. K.**, Feature–Based Geometry Construction for Geometric Reasoning, *Proc ASME Computers in Engineering Conf*, S.Fransisco, Jul/Aug 1988, pp 497 – 504

CLARK, A.L. and **SOUTH, N.E.**, Feature–Based Design of Mechanical Parts, *Proceedings, AUTOFACT '87*, Michigan, SME, 1987

COMPUTER AIDED DESIGN REPORT, CAD/CAM Publishing Inc, San Diego, USA, Vol 11, NO. 4, April 1991

CORBETT, J. and **WOODWARD, J. P. J.**, A CAD–Integrated Knowledge–Based System for the Design of Die Cast Components, *Annals of the CIRP*, Vol 40/1/91, 1991, pp 103 – 105

DeFAZIO, T. L., et. al., A Prototype of Feature–Based Design for Assembly, *Advances in Design Automation* – Volume 1, DE–Vol 23–1, ASME, 1990, pp 9 – 16

DELCHAMBRE, A., *Computer–Aided Assembly Planning*, Chapman and Hall, Great Britain, 1992

DONG, X. and **WOZNY, M. J.**, Instantiation of User Defined Features on a Geometric Model, *Product Modelling for Computer–Aided Design and Manufacturing*, Turner et. al. (eds), Elsevier Science Publishers, IFIP, 1991 pp 183 – 195

DONG, X., DeVRIES, W. R. and **WOZNY, M.J.**, Feature–Based Reasoning in Fixture Design, *Annals of the CIRP*, Vol 40/1/91, 1991, pp 111 – 114

DUAN, W., ZHOU, J. and **LAI, K.**, FSMT: A Feature Solid Modelling Tool for Feature–Based Design and Manufacturing, *Computer–Aided Design*, Volume 25, Number 1, Jan 1993, pp 29 – 38

ELMARAGHY, H. A. and **ELMARAGHY, W. H.,** Computer–Aided Inspection Planning (CAIP), in *Advances in Feature–Based Manufacturing*, Shah, J. J., Mantyla, M. and Nau, D. S. (eds), Elsevier, 1994, pp 363 – 396

FALCIDIENO, B. and **GIANNINI, F.**, Neutral Format Representation of Feature–Based Models in Multiple Viewpoint Context, *Product Modelling for Computer–Aided Design and Manufacturing*, Turner et. al. (eds), Elsevier Science Publishers, IFIP, 1991 pp 165 – 182

FAUX, I. D., Reconciliation of Design and Manufacturing Requirements For Product Description Data Using Functional Primitive Part Features, *Final Report, Computer–Aided Manufacturing–International,* Inc, Texas, 1986

FU, Z., De PENNINGTON, A. and **SAIA, A.**, A Graph Grammar Approach to Feature Representation and Transformation, *Int Journal Computer Integrated Manufacturing*, Vol 6, Nos 1 & 2, 1993, pp 137 – 151

GERO, J.S., Knowledge–Based Computer–Aided Design, *Computer Applications in Production and Engineering*, Kimura F. and Rostaldas A. (eds), Elsevier Science Publishers, IFIP, 1989, pp 13 – 20

GIACOMETTI, F. and **CHANG, T. C.**, Object–Oriented Design for Modelling Parts, Assemblies and Tolerances, *Proceedings, Second International TOOLS 2*, Bezivin, J. et. al. (eds), Paris, 1990

GINDY, N. N. Z., A Hierarchical Structure For Form Features, *Int Journal of Production Research,* Vol 27, No. 12, 1989, pp 2089 – 2103

GINDY, N. N. Z., HUANG, X. and **RATCHEV, T. M.**, Feature–Based Component Model For Computer–Aided Process Planning Systems, *Int Journal Computer Integrated Manufacturing*, Vol 6, Nos 1 & 2, 1993, pp 20 – 26

GORLEN, K.E., An Object–oriented Class Library for C++ Programs, *Software–Practice and Experience*, Vol 17, NO.12, 1987, pp 899 – 922

GUI, J. K. and **MANTYLA, M.**, Functional Understanding of Assembly Modelling, *Computer–Aided Design*, Vol 26, No. 6, 1994, pp 435 – 451

HENDERSON, M. R. and **ANDERSON, D. C.**, Computer Recognition and Extraction of Form Features: a CAD/CAM Link, *Computers in Industry*, Vol 5, 1984, pp 329 – 339

HENDERSON, M. R. and **CHANG, G. J.**, FRAPP: Automated Feature Recognition and Process Planning From Solid Model Data, *Proc ASME Computers in Engineering Conf*, S. Fransisco, Jul/Aug 1988, pp 529 – 536

HENSON, B. W., BAXTER, J. E. and **JUSTER, N. P.**, Assembly Representation Within a Product Data Framework, *Proceedings, ASME Design and Technical Conference*, Sept 1993

HUANG, T., XIANG, W., ZHOU, J. and **YU, J.**, Tolerance Analysis and Synthesis Based on Feature Modeling, Proceedings, *International Conf on Engineering Design ICED93*, The Hague, August 1993

ISO 10303: Product Data Representation and Exchange – Part 41: Integrated Generic Resources: Fundamentals of Product Description and Support

JONES, R., MITCHELL S. and **NEWMAN, S.**, Feature–Based Systems for the Design and Manufacture of Sculptured Products, *Int Journal Production Research*, Vol 31, No. 6, 1993, pp 1441 – 1452

JOSHI, S. and **CHANG, T.C.**, Graph–Based Heuristics For Recognition of Machined Features From a 3D Solid Model, *Computer Aided Design*, Vol 20, No 2, 1988, pp 58 – 66

JOSHI, S., VISSA, N. N. and **CHANG, T.C.**, Expert Process Planning System With Solid Model Interface, *Int Journal of Production Research*, Vol 26, No 5, 1988, pp 863 – 885

JOSHI, S., CHANG, T.C. and **LIU R.**, Process Planning Formalisation in an AI Framework, *Artificial Intelligence*, Vol 1, No 1, 1986, pp 45 –53

JURI, A. H., SAIA, A. and **DE PENNINGTON, A.**, Reasoning About Machining Operations Using Feature–Based Models, *Int Journal Production Research*, Vol 28, No. 1, 1990, pp 153 – 171

KARRA, C. and **PHELPS, T. A.**, Tool Approach Directions For Machining Protrusion and Depression Features In An Object, *Advances in Design Automation*, Vol 1, 1990, ASME, pp 193 – 200

KIM, Y. S., Form Feature Recognition by Convex Decomposition, *Computers in Engineering*, Vol 1, ASME, 1991, pp 61 – 69

KIM, S. H. and **LEE, K.**, An Assembly Modelling System For Dynamic and Kinematic Analysis, *Computer Aided Design*, Vol 21, No 1, 1989, pp 2 – 11

KO, H. and **LEE, K.**, Automatic Assembling Procedure Generation From Mating Conditions, *Computer Aided Design*, Vol 19, No 1, 1987, pp 3 – 10

KORAH, J., *Object–Oriented Methodology: A Primer*, SME Blue Book Series, CASA/SME, Michigan, 1994.

KRAUSE, F. L., ULBRICH, A. and **VOSGERAU, F. H.**, Feature–Based Approach for the Integration of Design and Process Planning Systems, *Product Modelling for Computer–Aided Design and Manufacturing*, Turner et. al. (eds), Elsevier Science Publishers, IFIP, 1991

KRAUSE, F. L., KIMURA, F., KJELLBERG, T. and **LU, S. C. Y.**, Product Modelling, *Annals of the CIRP*, Vol 42/2/1993, pp 695 – 706

KUTTNER, B., Manufacturing Expert System. An Object–Oriented Approach to Feature Driven Process Automation, *Journal of Manufacturing Systems*, Vol 7, No 1, 1988

LAAKKO, T. and **MANTYLA, M.**, Feature Modelling By Incremental Feature Recognition, *Computer Aided Design*, Vol 25, No 8, 1993, pp 479 – 492

LATIF, M. N. and **HANNAM, R. G.**, An Investigation of Object–Oriented Concepts in Feature–Based Design, *Proceedings, 29th International MATADOR Conference*, Manchester, April 1992, pp 3 –11

LAWLOR–WRIGHT, T. and **HANNAM, R. G.**, A Feature–Based Design for Manufacture: CADCAM Package, *Computer–Aided Engineering Journal*, Dec 1989, pp 215 – 220

LENAU, T. and **MU, L.**, Features in Integrated Modelling of Products and Their Production, *Int Journal Computer Integrated Manufacturing*, Vol 6, Nos 1 & 2, 1993, pp 65 – 73

LEE, I. B. H., LIM, B. S. and **NEE, A. Y. C.**, Knowledge–Based Process Planning System For The Manufacture of Progressive Dies, *Int Journal of Production Research,* Vol 31, No 2, 1993, pp 251 – 278

LEE, K. and **GOSSARD, D. C.**, A Hierarchical Data Structure For Representing Assemblies: Part 1, *Computer Aided Design*, Vol 17, No 1, 1985, pp 15 – 19

LEE, K. and **ANDREWS, G.**, Inference of the Positions of Components in an Assembly: Part 2, *Computer Aided Design*, Vol 17, No 1, 1985, pp 21 – 24

LI, R. K. and **HUANG, C. L.**, Assembly Code Generation From a Feature–Based Geometric Model, *Int Journal of Production Research*, Vol 30, No. 5, 1992, pp 627 – 646

LIBARDI, E.C., DIXON, J. R. and **SIMMONS, M. K.**, Computer Environment For the Design of Mechanical Assemblies: A Research Review, *Engineering With Computers*, No 3, 1988, pp 121 – 136

LIM, S. S., LEE, I. B. H., LIM, L. E. E. and **NGOI, B. K. A.**, Computer–Aided Concurrent Design of Product and Assembly Processes: A Literature Review, *Journal of Design and Manufacturing*, No 5, 1995, pp 67 – 88

LIN, A. C. and **CHANG, T. C.**, An Integrated Approach to Automated Assembly Planning for Three–Dimensional Mechanical Products, *Int Journal of Production Research*, Vol 31, No. 5, 1993, pp 1201 – 1227

LUBY, S. C., DIXON, J. R. and **SIMMONS, M. K.**, Creating and Using a Features Data Base, *Computers in Mechanical Engineering*, Vol 5, no 3, November 1986, pp 25 – 33

MANTYLA, M., Directions for Research in Product Modelling, *Computer Applications in Production and Engineering*, Kimura F. and Rostaldas A. (eds), Elsevier Science Publishers, IFIP, 1989

MANTYLA, M., A Modeling System For Top–Down Design of Assembled Products, *IBM Journal of Research and Development*, Vol 34, No. 5, 1990, pp 636 – 658

MAREFAT, M., MALHOTRA, S. and **KASHYAP, R. L.**, Object–Oriented Intelligent Computer–Integrated Design, Process Planning and Inspection, *Computer*, March 1993, pp 55 – 65

MASUKI, H., NOMURA, N., IMAMURA, S. and **KOJIMA, T.**, Object–Oriented Modelling to Grasp Form Features, *Computer Applications in Production and Engineering*, Kimura F. and Rostaldas A. (eds), Elsevier Science Publishers, IFIP, 1989, pp 87 – 94

MEERAN, S. and **PRATT, M. J.**, Automated Feature Recognition From 2D Drawings, *Computer–Aided Design*, Volume 25, Number 1, January 1993, pp 7 – 17

MILL, F., RIEKEN, H. R., SALMON, J. C. and **WARRINGTON, S. W.**, Feature Oriented Engineering in UK Industry and Academia, *Report, Manufacturing Planning Group,* Department of Mechanical Engineering, Univ of Edinburgh, April 1996

MITCHELL, R. J., *C++ Object–Oriented Programming*, MacMillan, Gt. Britain, 1993

MOLLOY, E., YANG, H. and **BROWNE, J.**, Feature–Based Modelling in Design for Assembly, *Int Journal Computer Integrated Manufacturing*, Vol 6, Nos 1 & 2, 1993, pp 119–125

MORTENSON, M.E., *Geometric Modeling*, John Wiley, N. York, 1985

NEE, A.Y.C. et. al., Feature–Based Classification Scheme for Fixtures, *Annals of CIRP*, Vol 41/1/1992, pp 189 – 192

NIEMINEN, J. and **TUOMI, J.**, Design With Features for Manufacturing Cost Analysis, *Product Modelling for Computer–Aided Design and Manufacturing*, Turner et. al. (eds), Elsevier Science Publishers, IFIP, 1991 pp 317 – 330

NITSCHKE, D. R., CHEN, Y. M. and **MILLER, R. A.**, A Feature Extraction Interface for CAD Part Models, *Computers in Engineering*, Vol 1, ASME 1991

NOF, S.Y., Critiquing The Potential of Object Orientation in Manufacturing, *Int J. Computer Integrated Manufacturing*, Vol. 7, No. 1, 1994, pp 3 – 16

OH, J. H. and **LEE, K.**, Automatic Dimensioning From 3D Solid Model With Feature Extraction, *Advances in Design Automation*, Vol 1, ASME, 1990, pp 115 –119

OPEN SOFTWARE FOUNDATION, *OSF/Motif, Programmer's Reference*, Release 1.2, Prentice Hall, USA, 1989

PARAMETRIC TECHNOLOGY CORPORATION, *Pro/Engineer Product Brochure*, USA, 1993

PATEL, R.M. and **McLEOD, A. J.**, The Implementation of a Mechanical Engineering Design Interface Using Engineering Features, *Computer–Aided Engineering Journal*, Dec 1988, pp 241 – 246

PORCHET, M. and **ZHANG, G.**, Assembly Modeling For Tolerancing In CAD: An Approach of Oriented Functional Relationship Graph, *Proceedings, International Conf on Engineering Design ICED93*, The Hague, August 1993

PRABHAKAR, S. and **HENDERSON, M.R.**, Automatic Form Feature Recognition Using Neural Network–Based Techniques on Boundary Representations of Solid Model, *Computer–Aided Design*, Volume 24, Number 7, July 1992 pp 381 – 392

PRATT, M.J. and **WILSON, P. R.**, Requirements For Support of Form Features In a Solid Modeling System, *Final Report, Computer–Aided Manufacturing–International, Inc*, Texas, June 1985

PRATT, M. J., Application of Feature Recognition in the Product Life–Cycle, *Int Journal Computer Integrated Manufacturing*, Vol 6, Nos 1 & 2, 1993, pp 13 – 19

REMBOLD, U., NNAJI, B.O. and **STORR, A.**, *Computer Integrated Manufacturing and Engineering*, Addison–Wesley, 1993

REQUICHA, A. A. G., Representations for Rigid Solids: Theory, Methods and Systems, *ACM Computing Surveys*, 12(4), 1980, pp 437 – 464

ROCHELEAU, D. N. and **LEE, K.**, System for Interactive Assembly Modelling, Computer Aided Design, Vol 19, No 2, 1987, pp 65 – 72

ROSARIO, L. M. and **KNIGHT, W. A.**, Design for Assembly Analysis: Extraction of Geometric Features From a CAD System Data Base, *Annals of the CIRP,* Vol 38/1, 1989, pp 13 – 16

ROSEN, D. W., Feature–Based Design: Four Hypotheses for Future CAD System, *Research in Engineering Design*, No 5, 1993, pp 125 – 139

ROY, U. and **LIU, C. R.**, Feature–Based Representational Scheme of a Solid Modeler for Providing Dimensioning and Tolerancing Information, *Robotics and Computer–Integrated Manufacturing*, Vol 4, Nos 3/4, 1988, pp 335 – 345

RUMBAUGH, J., BLAHA, M., PREMERLANI, W., EDDY, F. and **LORENSEN, W.**, *Object–Oriented Modeling and Design*, Prentice Hall, New Jersey, 1991

SAKURAI, H. and **GOSSARD, D. C.**, Recognising Shape Features in Solid Models, *IEEE Computer Graphics and Applications*, Sept 1990, pp 22 – 32

SAKURAI, H. and **GOSSARD, D. C.**, Geometric Modelling in Setup Planning and Fixture Design, *Product Modelling for Computer–Aided Design and Manufacturing*, Turner et. al. (eds), Elsevier Science Publishers, IFIP, 1991 pp 299 – 313

SALOMONS, O. W., van HOUTEN, F. J. A. M. and **KALS, H. J. J.**, Review of Research in Feature–Based Design, *Journal of Manufacturing Systems*, Vol 12, No 2, 1993, pp 113 – 132

SEKIGUCHI, H., KOJIMA, T. and **INOUE, K.**, Study on Automatic Determination of Assembly Sequence, *Annals of CIRP*, Vol 32, No 1, 1983, pp 371 – 374

SHAH, J. J., Feature Transformations Between Application–Specific Feature Spaces, *Computer–Aided Engineering Journal*, December 1988, pp 247 – 255

SHAH, J. J., Assessment of Features Technology, *Computer–Aided Design*, Volume 23, Number 5, June 1991, pp 331 – 343

SHAH, J.J. and **HSIAO, D. W. C.,** A Meta Knowledge Base For Machining Process Selection, *Computers in Engineering*, Vol 1, ASME, 1991, pp 77 – 84

SHAH, J. J., MANTYLA, M. and **NAU, D. S.,** (eds) *Advances in Feature Based Manufacturing*, Elsevier, 1994

SHAH, J. J. and **MATTHEW, A.**, Experimental Investigation of the STEP Form Feature Information Model, *Computer Aided Design*, Volume 23, number 4, May 1991, pp 282 – 296

SHAH, J. J. and **ROGERS, M.**, Functional Requirements and Conceptual Design of the Feature–Based Modelling System, *Computer–Aided Engineering Journal*, Vol 5, No 1, 1988, pp 9 – 15.

SHAH, J.J. and **ROGERS, M.**, Assembly Modeling as an Extension of Feature–Based Design, *Research in Engineering Design*, No 5, 1993, pp 218 – 237

SHAH, J.J., SREEVALSAN, P., ROGERS, M., BILLO, R. and **MATHEW, A.**, Current Status of Features Technology, *Revised Report, Computer–Aided Manufacturing–International*, Inc, Texas, Nov 1988

SHAH, J. J. and **TADEPALLI, R.**, Feature–Based Assembly Modeling, *Computers in Engineering*, Vol 1, ASME, 1992, pp 253 – 260

SHARP, R., A Solid Future, *CADCAM*, October 1993, p 15

SMITH, N., *The Design of a Feature–Based Pre–Processor*, M.Eng Thesis, LUT, April 1993

SME (Society of Manufacturing Engineers), *PDES: The Enterprise Data Standard*, SME Blue Book Series, CASA/SME, 1989

SODHI, R. and **TURNER, J. U.**, Representing Tolerance and Assembly Information In a Feature–Based Design Environment, *Advances in Design Automation* – Vol 1, ASME, 1991 pp 101– 108

SODHI, R. and **TURNER, J. U.**, Towards Modelling of Assemblies for Product Design, *Computer–Aided Design,* Volume 26, No. 2, 1994, pp 85 – 97

SOHLENIUS, G., Concurrent Engineering, *Annals of the CIRP*, Vol 41/2/1992 pp 645 – 655

SOUKUP, J., *Taming C++, Pattern Classes and Persistence for Large Projects*, Addison Wesley Publishing Co, 1994

SPATIAL TECHNOLOGY INC, *ACIS Geometric Modeler, Programmers Manual*, 1993

SPATIAL TECHNOLOGY EUROPE, *News Notes*, Vol 1, No 1, 1993

SREEVALSAN, P. C. and **SHAH, J. J.**, Unification of Form Feature Definition Methods, in *Intelligent Computer Aided Design*, Brown et. al. (eds), Elsevier, IFIP 1992, pp 83 – 106

STROUSTRUP, B., *The C++ Programming Language*, 2nd Edition, Addison Wesley, New Jersey, 1991

STURGES, R. H. and **KILANI, M. I.**, Towards an Integrated Design For An Assembly Evaluation and Reasoning System, *Computer Aided Design,* Vol 24, No. 2, 1992, pp 67 – 79

SWIFT, K. G., Knowledge–Based Design for Manufacture, Kogan Page, 1987

UNGER, M.B. and **RAY, S. R.**, Feature–Based Process Planning in the AMRF, *Proc ASME Computers in Engineering Conf*, S. Fransisco, Jul/Aug 1988, pp 563 – 569

USHER, J. M., An Object–Oriented Approach to Product Modelling for Manufacturing Systems, *Computers and Industrial Engineering*, Vol 25, Nos 1–4, 1993, pp 557 – 560

VAN EMMERIK, M. J. G. M., An Interactive Graphical Approach to Feature Modeling Using Halfspaces and Geometric Constraints, *Advances in Design Automation* – Volume 1, DE–Vol 32–1, ASME 1991, pp 97 – 105

VAN EMMERIK, M. J. G. M. and **JANSEN, F. W.**, User Interface for Feature Modelling, *Computer Applications in Production and Engineering*, Kimura F. and Rostaldas A. (eds), Elsevier Science Publishers, IFIP, 1989, pp 625 – 632

VAN HOUTEN, F. J. A. M., Manufacturing Interfaces, *Annals of the CIRP*, Vol 41/2/1992, pp 699 – 710

VAN'T ERVE, A. H. and **KALS, H. J. J.**, XPLANE, A Knowledge–Based Driven Process Planning System, *Proc. 2nd Int Conf on Computer–Aided Production Engineering*, 1987

WANG, H. P. and **LI, J. K.**, *Computer–Aided Process Planning*, Elsevier, Netherlands, 1991

WANG, M. T., An Object–Oriented Feature–Based CAD/CAPP/CAM Integration Framework, *Advances in Design Automatio*n – Volume 1, DE–Vol 32–1, ASME 1991, pp 109 – 116

WANG, N. and **OZSOY, T. M.**, Automatic generation of Tolerance Chains From Mating Relations Represented in Assembly Models, *Advances in Design Automation*, Vol 1, 1990, ASME, pp 227 – 233

WARMAN, E. A., Object–Oriented Programming and CAD, *Journal of Engineering Design*, Vol 1, No.1, 1990, pp 37 – 46

WESLEY, M. A., LOZANO–PEREZ, T., LIEBERMAN, L. I., LAVIN, M. A. and **GROSSMAN, D. D.**, A Geometric Modeling System for Automated Mechanical Assembly, *IBM Journal of Research and Development*, Vol 24, No 1, 1980, pp 64 – 74

WIERDA, L. S., Linking Design, Process Planning and Cost Information by Feature–Based Modelling, *Journal of Engineering Design*, Vol 2, No 1, 1991, pp 3 – 19

WOLF, W., Object–Oriented Programming for CAD, *IEEE Design & Test of Computers*, Vol 8, No 1, 1991, pp 35 – 42

WOODWARK, J. R., Shape Models in Computer–Integrated Manufacture, *Computer Integrated Manufacturing*, NATO ASI Series F49, Turksen, B. (ed), Springer–Verlag, 1988

XUE, D. and **DONG, Z.**, Automated Concurrent Design Based on Combined Feature, Tolerance, Production Process and Cost Models, *Advances in Design Automation*, ASME, Vol 2, 1993, pp 199 – 210

YOUNG, R. I. M. and **BELL, R.**, Design by Features: Advantages and Limitations in Machine Planning Integration, *Int Journal Computer Integrated Manufacturing*, Vol 6, Nos 1 & 2, 1993, pp 105 – 112

ZEID, I., *CAD/CAM Theory and Practice*, McGraw–Hill, USA, 1991

ZHANG, Y., GU, P. and **NORRIE, D. H.**, Object–Oriented Product Model for Integrated Manufacturing, *Proc, International Conf on Object–Oriented Manufacturing Systems (ICOOMS '92)*, May 1992, Alberta, Canada

ZHOU, M., GREENWELL, R. and **TANNOCK, J.**, Object–Oriented Methods for Manufacturing Information Systems, *Computer Integrated Manufacturing Systems*, Vol 7, No 2, 1994, pp 113 – 121

APPENDIX A
ASSEMBLY CLASS DECLARATIONS

```
/*The following codes represent declarations of classes defined in Chapter 5.*/

#ifndef _FBDC_H
#define _FBDC_H
#include <fstream.h>
#include <string.h>

class feature;
class component;
class subassy;

//ASSEMBLY CLASS
//Declaration for assembly class

class assembly {
                BODY *assy;
                char assy_name[20];
                assembly *next;
                subassy *s;
                double assy_posX;
                double assy_posY;
                double assy_posZ;
                double assy_AngleX;
                double assy_AngleY;
                double assy_AngleZ;

public:
                assembly();
                ~assembly();
                void addsubassy();
                virtual void draw();
                void save();
};

//SUBASSEMBLY CLASS
//Declaration for subassembly class

class subassy {
                BODY *suba();
                char subassy_name[20];
```

```
face *subassyface;
subassy *next;
component *Comp;
double sub_posX;
double sub_posY;
double sub_posZ;
double subassy_AngleX;
double subassy_AngleY;
double subassy_AngleZ;
public:
        subassy();
        ~subassy();
        void addcomponent();
        void findface();
        void draw();
        void save();
};
```

//COMPONENT CLASS

```
//Declaration for component class

class component {
protected:
        BODY* comp;
        FACE* compface;
        char comp_name[20];
        component *next;
        feature *ft;
        double comp_width;
        double comp_length;
        double comp_height;
        double comp_posX;
        double comp_posY;
        double comp_posZ;
        double comp_angleX;
        double comp_angleY;
        double comp_angleZ;
public:
        component();
        component(char* cn, double &w, double &l, double &h);
        virtual ~component();
        void GetDimension();
```

```
void GetPosition();
void GetOrientation();
char CName() {return comp_name[20];}
double CWidth();
double CLength();
double CHeight();
double CPosX() {return comp_posX;}
double CPosY() {return comp_posY;}
double CPosZ() {return comp_posZ;}
        void Draw_Comp();
        void FindFace();
        void SaveBody();
        void AddFeature(component &comp);
};

#endif  /* _FBDC_H */
```

// ASSEMBLY RELATIONSHIP CLASS

```
class relations: {
        char mating_relation[8];
        feature *first;
        feature *second;
        face *face1;
        face *face2;
public:
        void AssyUserInput();
        char IdentifyRelation(char fea1[5], char fea2[5]);
        void Transform(component *c);
        void ListRelation();
};
```

//LINK CLASS

//this class defines the nature of the objects that will be stored in the list

//OBJECTLINK CLASS

```
template <class DataT> class objectlink
{
public:
        DataT type;             //type of data
        objectlink<DataT> *next;
        objectlink<DataT> *prior;
        objectlink();
```

```
        objectlink<DataT> *getnext() {return next;}
        objectlink<DataT> *getprior() {return prior;}
};

//ASSYLIST CLASS
//this class inherits the above class
//actually implements the double linked list mechanism

template <class DataT> class assylist:public objectlink
{
        objectlink<DataT> *start, *end;
public:
        assylist() {start = end = NULL;}
        void store(DataT *c);
        void remove(objectlink<DataT> *ob);        //delete entry
        void frwdlist();                           //display list from beginning
        void bkwdlist();                           //display list from the end
        objectlink<DataT> *getstart() {return start;}
        objectlink<DataT> *getend() {return end;}
};
```

APPENDIX B
APPLICATION AND MAKE FILES

```
//Partial listing for main file fbds.cc

#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
#include <logical.h>
#include <string.h>

//Include ACIS class files

#include "acis.hxx"
#include "vector/transf.hxx"

#include "vector/vector.hxx"
#include "vector/unitvec.hxx"
#include "data/debug.hxx"
#include "data/entity.hxx"
#include "top/alltop.hxx"

#include "top/body.hxx"
#include "top/face.hxx"

#include "api/api.hxx"
#include "api/journal.hxx"
#include "api/routines.hxx"

#include "fbdclass.cc"
#include "comp.cc"
#include "save.cc"
#include "dblink.cc"
#include "relation.cc"
#include "utility.cc"

//Path for the Test Harness
char path[80] = "/home/samson/acis_1.4/acis1.4/acis1.4_x_demo.sun4";

void Create_Component();
void Welcome();                 //Welcome message – defined in utility.cc
void UserInput();
void MainMenu();                //Displays main menu
void Assy();                    //Interface with functions in relation.cc
```

```
void Output();                    //Outputs data – defined in utility.cc


main()
{
        Welcome();
        MainMenu();
}

void MainMenu()
{
        int mchoice;

        cout << "\n\n SYSTEM MENU \n";
        cout << " ==========\n";
        cout << "\n [1] CREATE FEATURE/COMPONENT\n";
        cout << "\n [2] CREATE ASSEMBLY\n";
        cout << "\n [3] TEST HARNESS\n";
        cout << "\n [4] PRINT DATA\n";
        cout << "\n [5] EXIT\n";

        cout << "\nEnter selection [1–5] : ";
        cin >> mchoice;
        while(mchoice != 6)
        {
                switch(mchoice)
                {
                        case 1: UserInput();
                                break;

                        case 2: Assy();
                                break;

                        case 3: system(path);
                                break;

                        case 4: Output();
                                break;

                        case 5: cout << "\nExit FBDS\n\n";
                                exit(0);

                default : cout << "\nError. Enter Selection [1 – 5]\n";
}
```

```
                MainMenu();
                cin >> mchoice;
        }
}

void UserInput()
{
                char ans;
                Create_Component();
                cout << "\nCreate another component? (y/n) ";
                cin >> ans;
                if (ans == 'y')
                        UserInput();
                else
                        ( cout << "Back to Main Menu\n");
}

void Create_Component()
{

                char name[20];
                double l1,w1,h1;
                dllist list;

                outcome result = api_start_modeller(TRUE, "journal", 0);
                outcome_check(result, "error initialising modeler");

                //component is actually a base feature

                component *comp = new component(name, l1, w1, h1);
                comp->GetPosition();
                comp->GetOrientation();
                comp->GetWorkSize();
                comp->Draw_Comp();
                comp->FindFace();
                comp->AddFeature(*comp);
                list.store(comp);
                comp->PrintData();
                comp->SaveBody();
                combine_file(comp);
}
```

MAKE FILE

```
//This is the makefile for compiling FBDS.CC

model:      fbds.o
            CC –g –o fbds \
            fbds.o \
            /home/samson/acis_1.4/acis1.4/error/obj.sun4/find_message.o  \
            –L/home/samson/acis_1.4/acis1.4/lib.sun4 \
            –lspline –lsg_husk –lkernel –lspline –lkernel –lspline \
            /home/samson/acis_1.4/aglib1.5/lib.sun4/libaglib.a \
            –lm

fbds.o:     fbds.cc comp.cc fbdclass.cc relations.cc save.cc
            CC –c –g \
            –I/home/samson/acis_1.4/acis1.4  \
            –I/home/samson/acis_1.4/acis1.4/kernel \
            –I/home/samson/acis_1.4/acis1.4/spline  \
            fbds.cc
```

APPENDIX C
SAMPLE ACIS FILE

This file contains data generated from the pin and block assembly shown in Figure 6.6. The file is retrieved in the Test Harness to display the model, as shown in Figure 6.12.

```
//Data for boss feature of the block
104 86 0 0
body $–1 $1 $–1 $2 #
lump $–1 $–1 $3 $0 #
transform $–1 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 #
shell $–1 $–1 $–1 $4 $1 #
face $–1 $5 $6 $3 $–1 $7 0 #
face $–1 $8 $9 $3 $–1 $10 1 #
loop $–1 $–1 $11 $4 #
plane–surface $–1 0 0 20 0 0 1 1 0 0 0 #
face $–1 $12 $13 $3 $–1 $14 1 #
loop $–1 $–1 $15 $5 #
plane–surface $–1 0 0 –20 0 0 1 1 0 0 0 #
coedge $–1 $16 $17 $18 $19 0 $6 $–1 #
face $–1 $20 $21 $3 $–1 $22 1 #
loop $–1 $–1 $23 $8 #
plane–surface $–1 0 –50 0 0 1 –0 –0 0 1 0 #
coedge $–1 $24 $25 $26 $27 0 $9 $–1 #
coedge $–1 $28 $11 $29 $30 0 $6 $–1 #
coedge $–1 $11 $28 $31 $32 0 $6 $–1 #
coedge $–1 $33 $34 $11 $19 1 $35 $–1 #
edge $–1 $36 $37 $18 $38 0 #
face $–1 $39 $40 $3 $–1 $41 1 #
loop $–1 $–1 $42 $12 #
plane–surface $–1 –60 0 0 1 0 0 0 0 –1 0 #
coedge $–1 $43 $31 $44 $45 0 $13 $–1 #
coedge $–1 $46 $15 $43 $47 0 $9 $–1 #
coedge $–1 $15 $46 $48 $49 0 $9 $–1 #
coedge $–1 $34 $33 $15 $27 1 $35 $–1 #
edge $–1 $50 $51 $26 $52 0 #
coedge $–1 $17 $16 $53 $54 0 $6 $–1 #
coedge $–1 $55 $56 $16 $30 1 $40 $–1 #
edge $–1 $37 $57 $29 $58 0 #
coedge $–1 $23 $59 $17 $32 1 $13 $–1 #
edge $–1 $60 $36 $31 $61 0 #
coedge $–1 $26 $18 $59 $62 0 $35 $–1 #
coedge $–1 $18 $26 $55 $63 1 $35 $–1 #
loop $–1 $–1 $33 $39 #
vertex $–1 $19 $64 #
vertex $–1 $19 $65 #
straight–curve $–1 60 0 20 0 1 0 #
face $–1 $–1 $35 $3 $–1 $66 1 #
```

loop $–1 $–1 $55 $20 #
plane–surface $–1 0 50 0 0 –1 0 0 0 –1 0 #
coedge $–1 $67 $53 $56 $68 0 $21 $–1 #
coedge $–1 $59 $23 $24 $47 1 $13 $–1 #
coedge $–1 $53 $67 $23 $45 1 $21 $–1 #
edge $–1 $60 $69 $44 $70 0 #
coedge $–1 $25 $24 $67 $71 0 $9 $–1 #
edge $–1 $51 $69 $43 $72 0 #
coedge $–1 $56 $55 $25 $49 1 $40 $–1 #
edge $–1 $73 $50 $48 $74 0 #
vertex $–1 $27 $75 #
vertex $–1 $62 $76 #
straight–curve $–1 60 0 –20 0 –1 0 #
coedge $–1 $42 $44 $28 $54 1 $21 $–1 #
edge $–1 $57 $60 $53 $77 0 #
coedge $–1 $48 $29 $34 $63 0 $40 $–1 #
coedge $–1 $29 $48 $42 $68 1 $40 $–1 #
vertex $–1 $30 $78 #
straight–curve $–1 0 50 20 –1 0 0 #
coedge $–1 $31 $43 $33 $62 1 $13 $–1 #
vertex $–1 $54 $79 #
straight–curve $–1 0 –50 20 1 0 0 #
edge $–1 $36 $51 $33 $80 0 #
edge $–1 $37 $50 $34 $81 0 #
point $–1 60 –50 20 #
point $–1 60 50 20 #
plane–surface $–1 60 0 0 –1 0 0 0 –0 1 0 #
coedge $–1 $44 $42 $46 $71 1 $21 $–1 #
edge $–1 $57 $73 $56 $82 0 #
vertex $–1 $71 $83 #
straight–curve $–1 –60 –50 0 0 0 –1 #
edge $–1 $69 $73 $67 $84 0 #
straight–curve $–1 0 –50 –20 –1 0 0 #
vertex $–1 $49 $85 #
straight–curve $–1 0 50 –20 1 0 0 #
point $–1 60 50 –20 #
point $–1 60 –50 –20 #
straight–curve $–1 –60 0 20 0 –1 0 #
point $–1 –60 50 20 #
point $–1 –60 –50 20 #
straight–curve $–1 60 –50 0 0 0 –1 #
straight–curve $–1 60 50 0 0 0 –1 #
straight–curve $–1 –60 50 0 0 0 –1 #
point $–1 –60 –50 –20 #
straight–curve $–1 –60 0 –20 0 1 0 #
point $–1 –60 50 –20 #

//Data for hole feature of block
104 26 0 0
body $–1 $1 $–1 $2 #
lump $–1 $–1 $3 $0 #

```
transform $–1 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 #
shell $–1 $–1 $–1 $4 $1 #
face $–1 $5 $6 $3 $–1 $7 0 #
face $–1 $8 $9 $3 $–1 $10 0 #
loop $–1 $11 $12 $4 #
cone–surface $–1 0 0 0 0 0 0 1 15 0 0 1 0 1 0 #
face $–1 $–1 $13 $3 $–1 $14 0 #
loop $–1 $–1 $15 $5 #
plane–surface $–1 0 0 –20 0 0 –1 –1 0 0 0 #
loop $–1 $–1 $16 $4 #
coedge $–1 $12 $12 $15 $17 1 $6 $–1 #
loop $–1 $–1 $18 $8 #
plane–surface $–1 0 0 20 0 0 1 1 0 0 0 #
coedge $–1 $15 $15 $12 $17 0 $9 $–1 #
coedge $–1 $16 $16 $18 $19 1 $11 $–1 #
edge $–1 $20 $20 $15 $21 0 #
coedge $–1 $18 $18 $16 $19 0 $13 $–1 #
edge $–1 $22 $22 $18 $23 0 #
vertex $–1 $17 $24 #
ellipse–curve $–1 0 0 –20 0 0 –1 15 0 0 1 #
vertex $–1 $19 $25 #
ellipse–curve $–1 0 0 20 0 0 1 15 0 0 1 #
point $–1 15 0 –20 #
point $–1 15 0 20 #

//Data for rectangular boss feature of the pin
104 86 0 0
body $–1 $1 $–1 $2 #
lump $–1 $–1 $3 $0 #
transform $–1 1 0 0 0 1 0 0 0 1 0 0 100 1 0 0 0 #
shell $–1 $–1 $–1 $4 $1 #
face $–1 $5 $6 $3 $–1 $7 0 #
face $–1 $8 $9 $3 $–1 $10 1 #
loop $–1 $–1 $11 $4 #
plane–surface $–1 0 0 5 0 0 1 1 0 0 0 #
face $–1 $12 $13 $3 $–1 $14 1 #
loop $–1 $–1 $15 $5 #
plane–surface $–1 0 0 –5 0 0 1 1 0 0 0 #
coedge $–1 $16 $17 $18 $19 0 $6 $–1 #
face $–1 $20 $21 $3 $–1 $22 1 #
loop $–1 $–1 $23 $8 #
plane–surface $–1 0 –25 0 0 1 –0 –0 0 1 0 #
coedge $–1 $24 $25 $26 $27 0 $9 $–1 #
coedge $–1 $28 $11 $29 $30 0 $6 $–1 #
coedge $–1 $11 $28 $31 $32 0 $6 $–1 #
coedge $–1 $33 $34 $11 $19 1 $35 $–1 #
edge $–1 $36 $37 $18 $38 0 #
face $–1 $39 $40 $3 $–1 $41 1 #
loop $–1 $–1 $42 $12 #
plane–surface $–1 –25 0 0 1 0 0 0 0 –1 0 #
coedge $–1 $43 $31 $44 $45 0 $13 $–1 #
```

```
coedge $-1 $46 $15 $43 $47 0 $9 $-1 #
coedge $-1 $15 $46 $48 $49 0 $9 $-1 #
coedge $-1 $34 $33 $15 $27 1 $35 $-1 #
edge $-1 $50 $51 $26 $52 0 #
coedge $-1 $17 $16 $53 $54 0 $6 $-1 #
coedge $-1 $55 $56 $16 $30 1 $40 $-1 #
edge $-1 $37 $57 $29 $58 0 #
coedge $-1 $23 $59 $17 $32 1 $13 $-1 #
edge $-1 $60 $36 $31 $61 0 #
coedge $-1 $26 $18 $59 $62 0 $35 $-1 #
coedge $-1 $18 $26 $55 $63 1 $35 $-1 #
loop $-1 $-1 $33 $39 #
vertex $-1 $19 $64 #
vertex $-1 $19 $65 #
straight-curve $-1 25 0 5 0 1 0 #
face $-1 $-1 $35 $3 $-1 $66 1 #
loop $-1 $-1 $55 $20 #
plane-surface $-1 0 25 0 0 -1 0 0 0 -1 0 #
coedge $-1 $67 $53 $56 $68 0 $21 $-1 #
coedge $-1 $59 $23 $24 $47 1 $13 $-1 #
coedge $-1 $53 $67 $23 $45 1 $21 $-1 #
edge $-1 $60 $69 $44 $70 0 #
coedge $-1 $25 $24 $67 $71 0 $9 $-1 #
edge $-1 $51 $69 $43 $72 0 #
coedge $-1 $56 $55 $25 $49 1 $40 $-1 #
edge $-1 $73 $50 $48 $74 0 #
vertex $-1 $27 $75 #
vertex $-1 $62 $76 #
straight-curve $-1 25 0 -5 0 -1 0 #
coedge $-1 $42 $44 $28 $54 1 $21 $-1 #
edge $-1 $57 $60 $53 $77 0 #
coedge $-1 $48 $29 $34 $63 0 $40 $-1 #
coedge $-1 $29 $48 $42 $68 1 $40 $-1 #
vertex $-1 $30 $78 #
straight-curve $-1 0 25 5 -1 0 0 #
coedge $-1 $31 $43 $33 $62 1 $13 $-1 #
vertex $-1 $54 $79 #
straight-curve $-1 0 -25 5 1 0 0 #
edge $-1 $36 $51 $33 $80 0 #
edge $-1 $37 $50 $34 $81 0 #
point $-1 25 -25 5 #
point $-1 25 25 5 #
plane-surface $-1 25 0 0 -1 0 0 0 -0 1 0 #
coedge $-1 $44 $42 $46 $71 1 $21 $-1 #
edge $-1 $57 $73 $56 $82 0 #
vertex $-1 $71 $83 #
straight-curve $-1 -25 -25 0 0 0 -1 #
edge $-1 $69 $73 $67 $84 0 #
straight-curve $-1 0 -25 -5 -1 0 0 #
vertex $-1 $49 $85 #
straight-curve $-1 0 25 -5 1 0 0 #
```

point $–1 25 25 –5 #
point $–1 25 –25 –5 #
straight–curve $–1 –25 0 5 0 –1 0 #
point $–1 –25 25 5 #
point $–1 –25 –25 5 #
straight–curve $–1 25 –25 0 0 0 –1 #
straight–curve $–1 25 25 0 0 0 –1 #
straight–curve $–1 –25 25 0 0 0 –1 #
point $–1 –25 –25 –5 #
straight–curve $–1 –25 0 –5 0 1 0 #
point $–1 –25 25 –5 #

//Data for circular boss feature of the pin
104 26 0 0
body $–1 $1 $–1 $2 #
lump $–1 $–1 $3 $0 #
transform $–1 1 0 0 0 1 0 0 0 1 0 0 125 1 0 0 0 #
shell $–1 $–1 $–1 $4 $1 #
face $–1 $5 $6 $3 $–1 $7 0 #
face $–1 $8 $9 $3 $–1 $10 0 #
loop $–1 $11 $12 $4 #
cone–surface $–1 0 0 0 0 0 1 15 0 0 1 0 1 0 #
face $–1 $–1 $13 $3 $–1 $14 0 #
loop $–1 $–1 $15 $5 #
plane–surface $–1 0 0 –20 0 0 –1 –1 0 0 0 #
loop $–1 $–1 $16 $4 #
coedge $–1 $12 $12 $15 $17 1 $6 $–1 #
loop $–1 $–1 $18 $8 #
plane–surface $–1 0 0 20 0 0 1 1 0 0 0 #
coedge $–1 $15 $15 $12 $17 0 $9 $–1 #
coedge $–1 $16 $16 $18 $19 1 $11 $–1 #
edge $–1 $20 $20 $15 $21 0 #
coedge $–1 $18 $18 $16 $19 0 $13 $–1 #
edge $–1 $22 $22 $18 $23 0 #
vertex $–1 $17 $24 #
ellipse–curve $–1 0 0 –20 0 0 –1 15 0 0 1 #
vertex $–1 $19 $25 #
ellipse–curve $–1 0 0 20 0 0 1 15 0 0 1 #
point $–1 15 0 –20 #
point $–1 15 0 20 #