User Defined Feature Modelling: Representing Extrinsic Form, Dimensions And Tolerances

A. G. Pedley

Doctor of Philosophy

The University of Edinburgh

1997

To Valentina

Declaration

This thesis has been composed by myself. The work reported in this thesis was undertaken whilst I was a member of a research team. I have been responsible for the work reported except where indicated in the text

Acknowledgements

The completion of this thesis represents the culmination of a journey from student to practising engineer. The voyage has been long and winding, continuously moving from the practical problems faced in industry to the theoretical world of universities, of considering potential and developing reality. The road taken has not been typical of a PhD student and I should like to thank my supervisors, Dr. George Alder and Mr. Frank Mill, for their motivation, guidance and support. I would also like to acknowledge the flexible stance taken by the University of Edinburgh in enabling my studies to continue.

The bulk of the developments reported in the thesis have been made during research on the following two projects, "SESAME" (BREU 4539) and "IMPRESS" (BREU 7049), both funded by the Commission of the European Communities. Earlier background work was undertaken on research projects, "Representation, Reasoning And Decision Making In Process Planning With Complex Components" (GR/D 63101) and "Feature Oriented Design" (GR/F 92312), funded by the UK Science and Engineering Research Council (now EPSRC). This support is gratefully acknowledged. To colleagues who worked for partner companies in the SESAME and IMPRESS projects I thank you for sharing your knowledge.

The work reported has been completed in collaboration with Strässle Informationssysteme AG. I would like to express my thanks to all members of the 3D Feature Modelling and Surface Modelling development teams for the meaningful discussions, functional support and "zusammenarbeit". In particular I would like to thank Hansjörg Denzel for his work on the feature modelling kernel and Peter Wyss for his programming on the user interface.

I would like to thank all colleagues and staff in the Department of Mechanical Engineering for their support and encouragement, and especially members of the Manufacturing Planning Group, Steve Warrington, Jonathan Salmon, and Jane

Naish, for the useful discussions. I would also like to thank Professor Tom Childs for his interest during my short stay in Mechanical Engineering at Leeds University. To past and present colleagues at IFW (Hannover University), Joachim Taiber (ETH Zürich), thanks for the encouragement and the motivation.

The last few months have been particularly stressful and I have been something of a recluse working for Strässle during the day, and on my thesis in the evenings and at the weekends. I would like to thank my parents and brother for their continuing moral support. To friends and flatmates who have put up with me, thanks, and I'll try not to be so manic again. A big thank you to Don Fleming and Andy Clegg for the pep talks.

I feel that I can now have my life back. For your patience, understanding, friendship, support and enthusiasm, to you Valentina, I shall be eternally grateful.

Graham Pedley

Edinburgh, September 25, 1997.

Abstract

The fundamental concept investigated in this thesis is to integrate in a single computer software representation the information required by designers to define a component for analysis and manufacturing, thus enabling software based simultaneous engineering strategies to be implemented. The investigations will examine and advance the understanding of a specific modelling technique used in Computer Aided Design (CAD) and Computer Aided Manufacturing (CAM) software applications known as *feature modelling*. Feature modelling has been developed in response to industry needs to improve competitiveness, particularly in the vital area of reducing product development time. The concept necessitates combining into a single integrated and fully associative model the requirements of 2D drafting for the production of detailed engineering drawings, representing 3D solid geometry containing both analytic and free form surfaces, and, modelling dimensional and geometric tolerances, and surface conditions.

Though feature modelling techniques have been proposed for over a decade, and are accepted as an aid to process integration, further investigations are required to establish if there are deficiencies in the understanding of features, which has resulted in little industrial impact of the technology to date and the continuing lack of design model and manufacturing application integration.

The following original contributions to the knowledge and understanding of feature modelling technology have been made as a result of the investigations:

- Certain features cannot be completely described by their make methods, therefore an important new class of features called *extrinsic form features* has been defined.
- The scope and architectural complexity of feature model data structures has been advanced by creating *multi-dimensional* or *hyper* feature models enabling a wider group of feature classes to be represented including form, dimension, and tolerance features.
- A novel combination of procedural and declarative approaches has been combined into a new, more sophisticated feature definition language, which describes both the dimensions and tolerances of a feature in a 3D manner in the template definition and enables a more intelligent graphical user interface.
- Exchange of feature models with increased information content can be achieved by reducing the flexibility of systems to subsequently change a non-native feature model.

Extrinsic form features have been developed that allow the integration of objects from many different sources within the feature model that can be swept to create free form geometry. Any solid or feature model may have associative dimensions and tolerances applied which are visualised as part of the 3D model. The template definition of a feature has been extended to enable dimensions representing a features size parameters to be described; these dimensions can be used to control the model. Parallel geometric tolerances have been found to behave as 3D constraints.

The work has clearly demonstrated the feasibility of providing homogeneous form feature, dimension and tolerance models. Working completely in a 3D environment eliminates the need for a separate 2D model to produce engineering drawings. Integration of the design model with down stream manufacturing applications has been improved. Providing persistent object identity is a problem. Only partial solutions to the exchange of feature models have been investigated. Further work is required to improve the persistency of object identity, extend the functionality to cover all geometric tolerancing and associated constraint problems, and to develop variational modelling and analysis capabilities.

Table Of Contents

List Of Figures

Abbreviations

CAE	Computer Aided Engineering
CAD	Computer Aided Design
CAM	Computer Aided Manufacture
CIM	Computer Integrated Manufacturing
CAPP	Computer Aided Process Planning
NC	Numerical Control
APT	Automatically Programmed Tools
FE	Finite Element
BREP	Boundary Representation
CSG	Constructive Solid Geometry
Transform	Transformation Matrix In Homogeneous Coordinates
API	Application Procedural Interface
GUI	Graphical User Interface
SGUI	Strässle Graphical User Interface
SME	Small To Medium Size Enterprise
MCL+	Macro Command Language (with Object Oriented Characteristics)
NLS	National Language Support

Glossary Of Terms

Feature	A feature is an information unit, geometric or otherwise, that is used to define, or help the understanding of, a component model.
Feature Model	A data structure and computer representation of a component constructed from a pre-defined set of features.
Product model	A computer representation of the shape, technology, processes and operations required to realise a product. A feature model defines a partial product model.
Form Feature	A feature that describes a geometric shape such as a hole or slot.
Feature Library	The set of features descriptions that are known by the feature modeller from which the user selects examples to create the desired model.
User Defined Feature	A feature specified, defined, implemented and added to the system library by the user of the feature modeller rather than the developer.
Implicit Representation	The description of the feature stored in the library, usually incorporating default values. Sometimes referred to as the feature prototype.
Explicit Representation	The description of the feature in the model after being instanced from the library and having had its parameters explicitly instantiated (set) by the user.

Template	The programming term used to describe the implicit representation of the feature, i.e. the feature prototype. New (user defined) features are described by completing a form or template. The attributes and parameters of the feature are described within the template but not how the feature behaves.
Procedural definition	The rules that define how the feature behaves are described by a set of high level procedures formed in a programming language. The procedure names are defined in the feature template. Each feature requires its own unique procedures to be written.
Declarative definition	The rules that define how the feature behaves are described by a set of low level constraints. The feature modeller incorporates a constraint solver. Only applicable to form features which generate geometry. The set of constraints is defined in the feature template. Each feature requires its own unique set of constraints to be constructed.
Build / control procedures	The procedures declared in the template definition are used to describe procedurally to the feature modeller how the feature behaves or what it does. For instance, the build procedure of a form feature will use the values of the parameters to generate the solid body of the feature.
Intrinsic definition	The geometric shape of a form feature is completely defined within the build procedures.

Extrinsic Definition	The geometric shape of a form feature is not, or can only be partially defined by the build procedures. The exact resultant form of the feature cannot be pre-determined in the build procedure.
Model evaluation	The process of combining all the instances of features in the model to produce a result.
Geometric primitive	The pre-defined solid representation of a shape used by a geometric modeller. For example, a cylinder, a cube, a prism, etc.
Feature handle / origin	A point which can be defined in the implicit, intrinsic description of a form feature which is useful as an aid to positioning the feature.
Feature Mapping	Features are application or view specific. Feature mapping is the process of generating a new feature model, representing a different view, from an existing set of features. For instance, a component may be described by design features representing the view of the designer. A mapping may be defined which allows the same geometric model to be described by a different set of features that represent the view of the manufacturing engineer.
Simultaneous engineering	The integrated design of both product and manufacturing process performed simultaneously in order to shorten time to market. The term is frequently used interchangeably with concurrent engineering.

Implicit Feature Interaction	An interaction between features, or feature elements, which is implicit in the model. For example, proximity, obstruction, and the like which could potentially cause manufacturing problems.
Explicit Feature Interaction	An interaction between features, or feature elements, which is explicitly stated by the constructor of the model - typically dimensional and geometric tolerances.
Hyper-Feature Model	Also referred to as a Multi-Dimensional Feature Model. A single computer data structure which consists of a number of feature trees. Each tree contains only one major class of feature: form, dimensions, tolerances, etc. The trees are related by links from parameters in one feature tree to features or resulting geometry in another.
Parameter Dimension	A dimension feature which explicitly represents a single length, diameter, radius or angle parameter of a unique feature parameter. It is defined in the feature template and is represented as a graphical 3D symbol by the system. It is used to change the model interactively.

Chapter 1

Introduction

1.1 Introduction

1.1.1 Chapter Overview

The fundamental concept investigated in this thesis is to integrate in a single computer software representation the information required by designers to define a component for analysis and manufacturing, thus enabling software based simultaneous engineering strategies to be implemented. The investigations will examine and advance the understanding of a specific modelling technique used in Computer Aided Design (CAD) and Computer Aided Manufacturing (CAM) software applications known as *feature modelling*. Feature modelling has been developed in response to industry needs to improve competitiveness, particularly in the vital area of reducing product development time.

In this chapter (chapter 1) the principles of feature modelling will be introduced. The historical background and industrial needs which have led to its development will be described. The deficiencies of current systems will be detailed, and the aims and objectives of the work reported in this thesis will be established in order to address these problems. The software implementation reported forms refinements and additions to an existing commercial feature modelling system which will be

introduced. An overview of the thesis will be provided. In chapter 2 CAD and CAM strategies, software technologies and relevant standards will be reviewed with respect to their influence on feature modelling techniques and the developments to be reported in this thesis.

1.1.2 Chapter Structure

Section 1.2 will introduce the concept of feature modelling. An overview of the historical development of manufacturing strategy, and consequently design and manufacturing software support, will be given in section 1.3. The need for the investigations undertaken originates from a number of deficiencies with feature modelling systems which will be described in section 1.4. In section 1.5 the aims and objectives of the work will be stated in order to address these problems; the contribution of this work to the knowledge and advancement of feature modelling techniques will be stated. A commercial system will provide the baseline for the developments reported in this thesis, it will be introduced in section 1.6. The structure of the thesis will be described in section 1.7. A summary of the chapter will be provided in section 1.8.

1.2 Feature Modelling Concepts

Features are typically used to describe the geometric shape of an object. A form feature may be broadly described as "a geometric form or entity" [LUB86] that has some engineering meaning, a hole, a pocket, or a slot, for instance. Feature models use a tree like data structure to maintain the feature information and ordering of the model which is separate from the geometry [CHA90]. A conventional geometric (usually solid) modelling engine is used to represent the geometry of the individual features and combine each feature under the control of the feature modelling engine to generate the desired model, Figure 1.1. Although this structure appears at first similar to that of a *constructive solid geometry* (CSG) solid modeller, there are

significant differences. Changes may be made to the feature model by setting new values for the parameters which define the form of any feature, followed by re-evaluation of the model, rather than by the addition and removal of further primitive shapes.

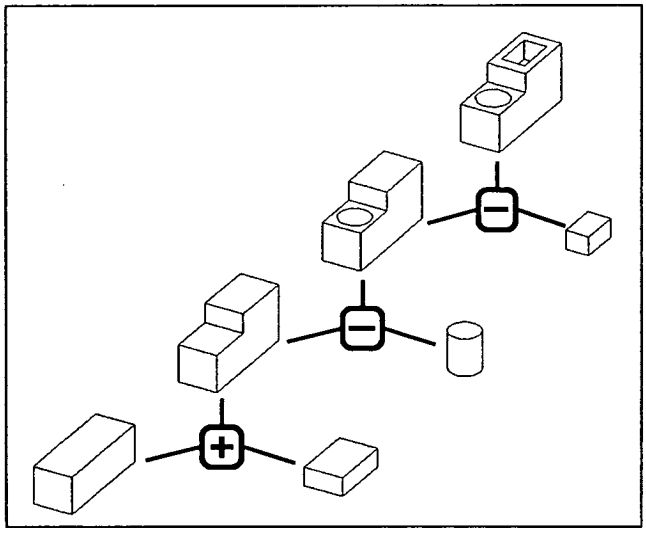


Figure 1.1 Feature Model Structure

The feature data structure may also be used to support information related to the form of a model, such as tolerances or other manufacturing information, in which case other classes of feature are introduced, accuracy features, for example. Features represent a functionally specific view of a component, design compared to manufacturing, for example, Figure 1.2.

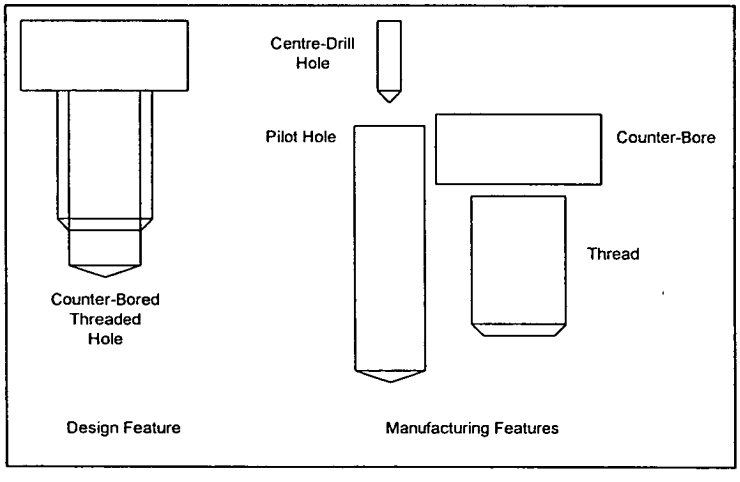


Figure 1.2 Functionally Specific Features

The manufacturing feature model may be obtained from the design feature model using a top down approach known as feature mapping, or a bottom up approach called feature recognition. If a set of features in a model represent the manufacturing engineer's view of the component, manufacturing processes and operations may be associated with them, leading to the development of the manufacturing process model simultaneously as the design progresses. If tolerances and other information may be supported within a feature modelling structure and associated with the features, process, tool and operation selection may be more easily automated. It is the possibility of feature technology to represent different views of a component (design, manufacturing, etc.) combined with non-geometric information that offers the potential to improve CAD and CAM integration.

Feature Modelling Systems are provided with pre-defined sets of features. However, an infinite set would be required to functionally model all possible geometric needs from the specific point of view of the user. For example, a designer modelling a car body panel has very different needs from a designer of a milling machine, Figure 1.3.

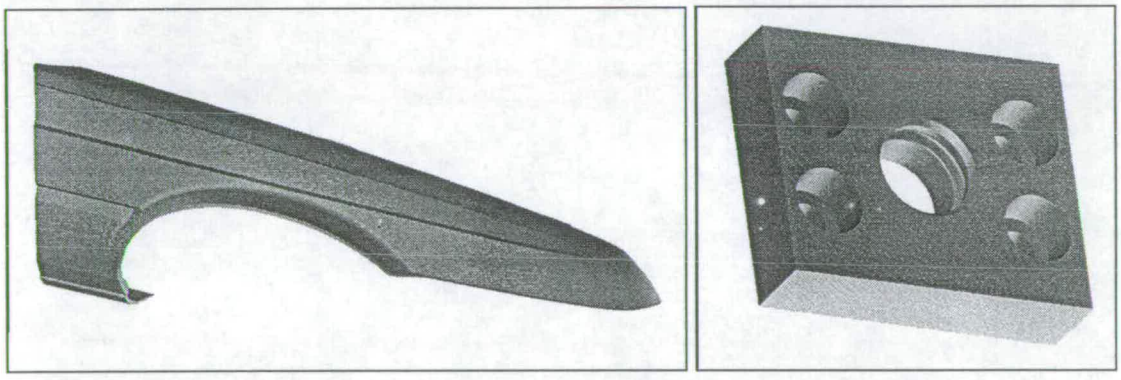


Figure 1.3 (a) Car Wing (b) Connecting Plate

Therefore it is desirable to extended feature modelling techniques to provide a flexible data structure in which the user may define additional features (*user defined features*) that represent the specific needs of the user's business environment. The ability to parametrically pre-define shapes and to have manufacturing know-how associated with the features used to construct a model is seen as a means of more

rapidly designing both the product and the manufacturing process, hence reducing development time.

Feature Modelling offers the potential to build more highly developed, information rich models than geometric modelling methods allow, but has yet to achieve the wide spread industrial acceptance of well established techniques such as surface and parametric modelling.

1.3 Historical Developments

The advent of industrialisation has led to people becoming specialists in a particular area: design, manufacturing or sales, for example. There are very few people today who both conceive, realise and sell a product, as was the case with a blacksmith. This specialisation and division of labour has led to compartmentalisation of business functions and a sequential flow of processes from design, through production planning to manufacturing. Generally the approach taken has been one of "over the wall" from one department to the next with surprisingly little interaction, Figure 1.4. Factors such as hierarchical status, training and particularly non-existence of interdisciplinary aids and support tools have contributed to a maintenance of the *status quo*.

The development and application of computer technology has mimicked this division of labour whereby applications are of a specialist nature, they rarely take into consideration the needs of other applications, generally use different data structures and modelling techniques, and communicate poorly with each other. As a consequence, providing feedback of problems through the software chain is almost impossible and feeding through design changes is fraught with difficulties, additional costs and lengthening of product development periods.

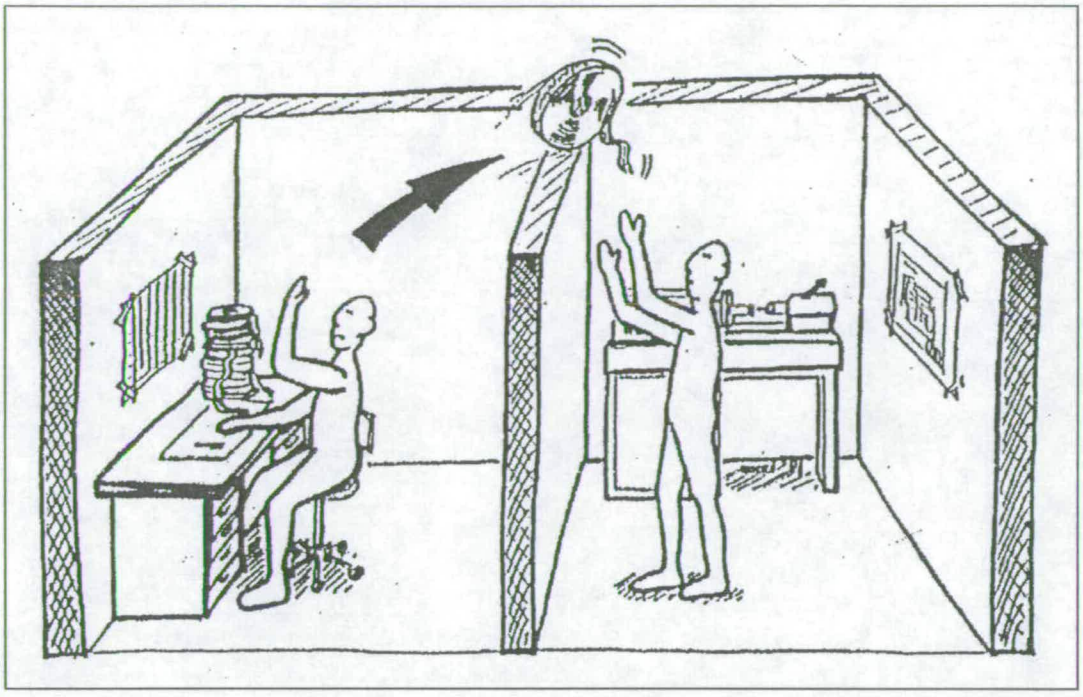
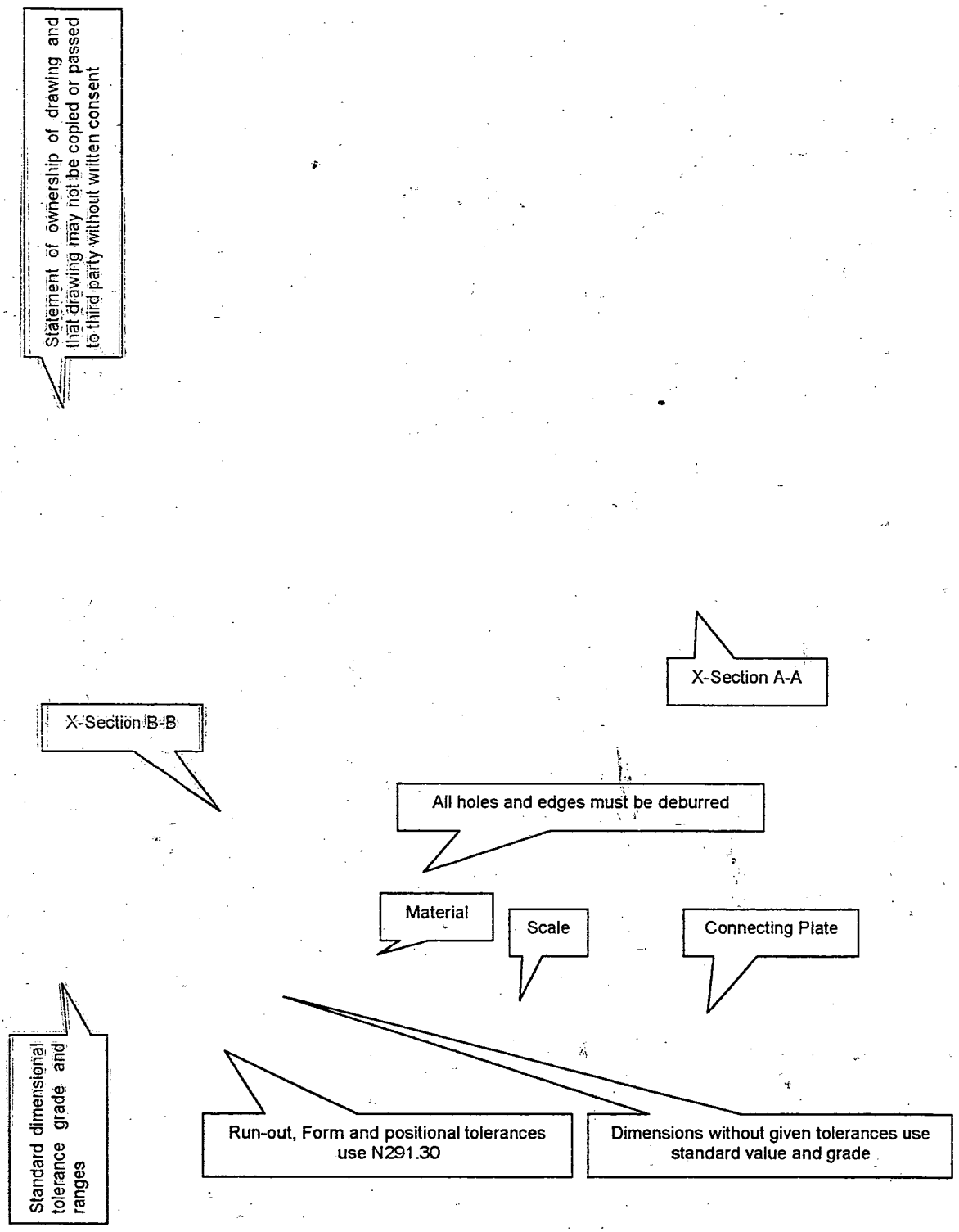


Figure 1.4 The "Over the Wall" Approach

Computers were first applied to support the manufacturing process during the 1950's with the development of the programming language APT (Automatically Programmed Tools) [GRO84]. This language enabled tool cutter paths of the then recently developed numerically controlled (NC) milling machines to be defined independently from the machine itself. It was not until 1963, however, that graphical representations of a component model were able to be created on a computer screen and modified. The Sketch Pad system from Sutherland [SUT63] could only produce two dimensional (2D) wire frame models, the generation and display of complex geometry was not possible. However it was possible to automate draughting and simple NC part programming via two dimensional contours. Mathematical modelling of complex geometry also began in the early sixties with DeCasteljau at Citroen and Bezier at Renault separately developing the same method to describe free form curves. The curves became known as Bezier Curves because of wide publication and use of the software system UNISURF [ZEI91]. It is interesting to note that it was demand from manufacturing engineering rather than design needs that pushed the early developments of computer support for product development.


Statement of ownership of drawing and that drawing may not be copied or passed to third party without written consent
X-Section A-A
X-Section B-B
All holes and edges must be deburred
Material
Scale
Connecting Plate
Standard dimensional tolerance grade and ranges
Run-out, Form and positional tolerances use N291.30
Dimensions without given tolerances use standard value and grade

Today 2D engineering drawings, Figure 1.5, albeit frequently produced with the aid of computer software applications, are still accepted as the standard method of defining a component.

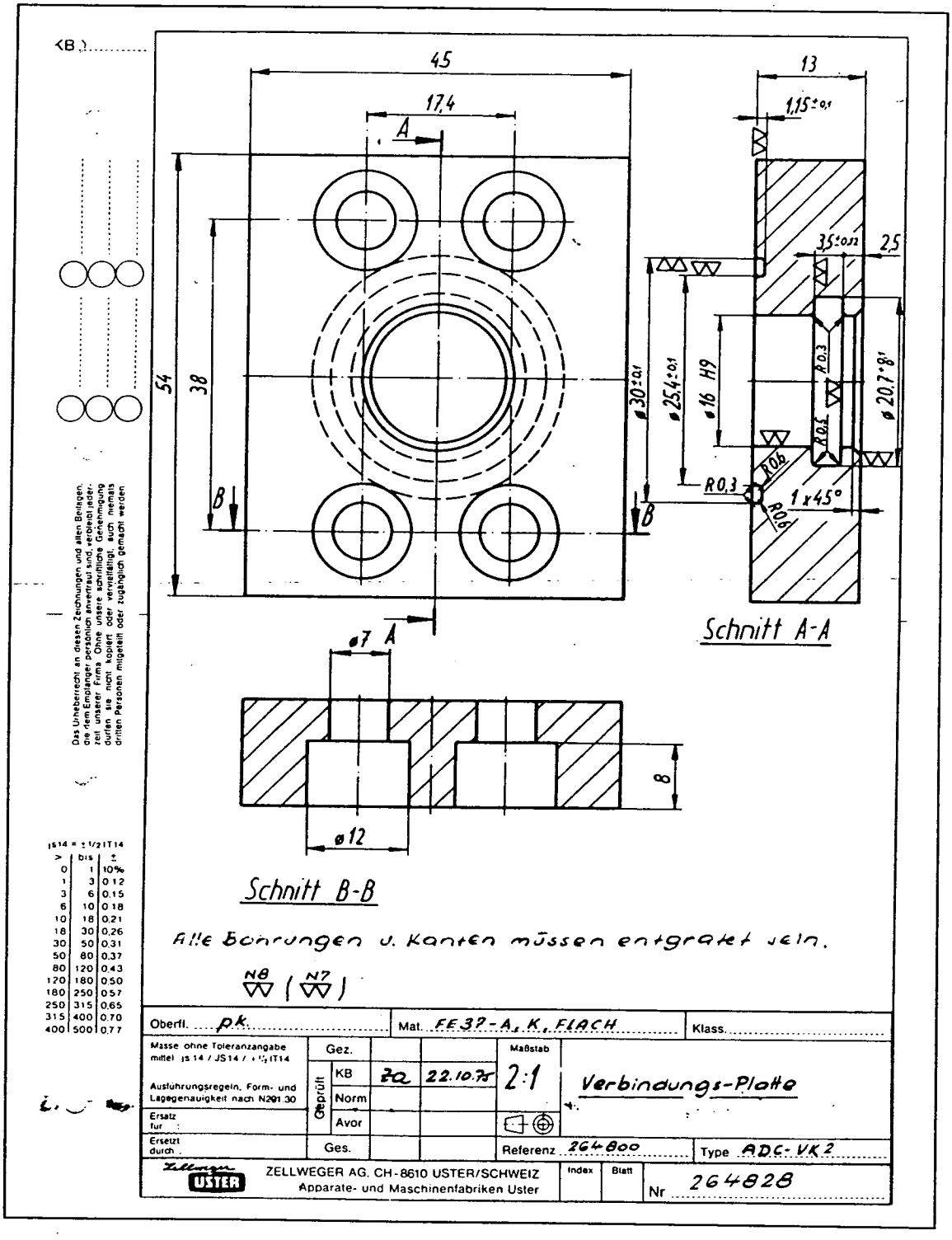


Figure 1.5 Traditional 2D Engineering Drawing

2D engineering drawings are of little use in efforts to advance computer integration of design and manufacturing, nor do 2D data models meet the requirements for product modelling seen as a core element of *simultaneous engineering* activities. Simultaneous engineering is the concurrent design of both the product and the process by which it will be realised and should consider all aspects of a product's life cycle. A model that is used to support such activities is called a *product model*. It is clear that many design and manufacturing analysis tasks are three-dimensional (3D) in nature and therefore cannot be supported by 2D drawings. A 3D geometric model is an essential element of product modelling systems.

The simplest form of 3D geometric model is the wire frame model which was developed in the early 70's. A wire frame model is an edge representation of an object which contains no information about surfaces or what is solid. Much is left to visual interpretation, Figure 1.6.

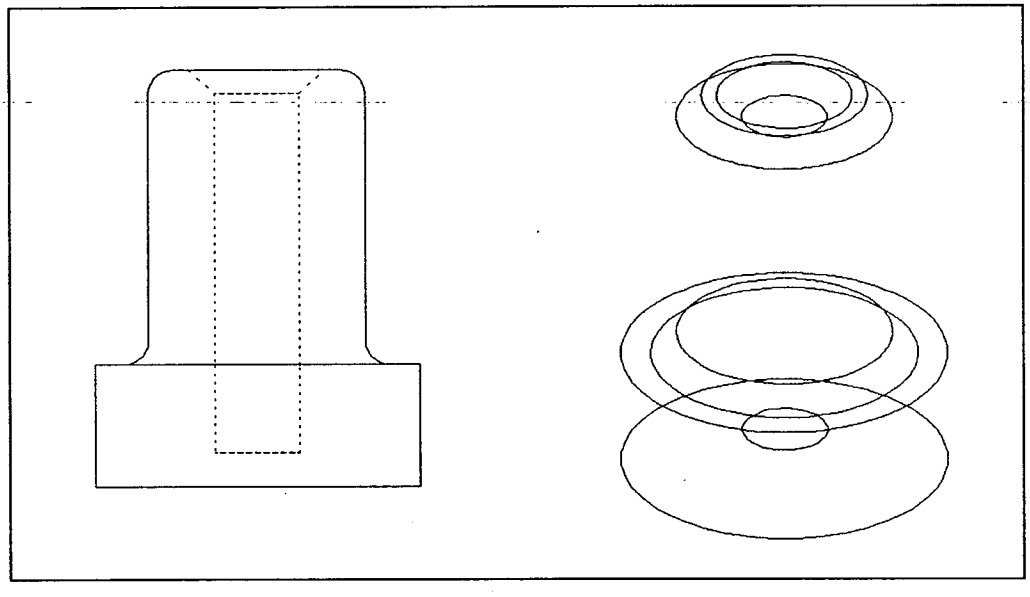


Figure 1.6 Interpretation Problems Of Wire Frame Models

3D surface modelling pioneered by Bezier provides a description of the surfaces of an object but not how the surfaces are connected together. This type of representation is used to model the smooth continuously changing surfaces commonly found in the aerospace and automotive industries where the techniques were pioneered, Figure

1.7. This precludes geometric reasoning but advanced techniques have been successfully developed to generate 3, 4, and 5 axis NC part programmes from the models. Generation of the model is generally very inefficient.

Complete three dimensional descriptions of a component are provided by solid modellers. Also referred to as volume modellers, they have typically been used to model 3D geometry defined using planar or analytic surfaces. More recently much effort has centred on introducing and integrating the synthetic geometry of surface modellers.

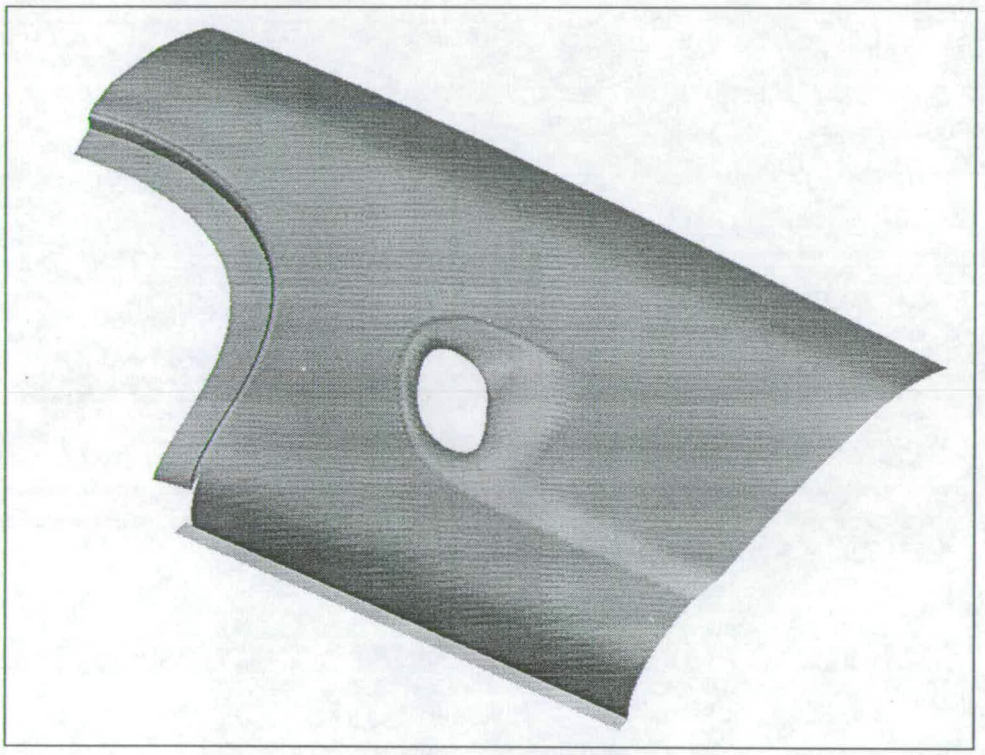


Figure 1.7 Surface Model Of Car Body Panel

A solid model is a complete description, of both geometry (the description of the geometric elements) and topology (the connectivity between the geometric elements), enabling points in space to be classified as in, on or outside an object, and cross-sections through a model to be generated, Figure 1.8. A solid model may have mass properties calculated and is frequently used as a basis for generation of representations suitable for numerical analysis methods, such as finite element (FE). Significantly, geometric reasoning tasks can be performed that are necessary in

advanced process planning [MIL94] and fixturing systems [CHI96]. Solid modelling is acknowledged as an essential element of the technological solution to automating and integrating design and manufacturing functions [ZEI91].

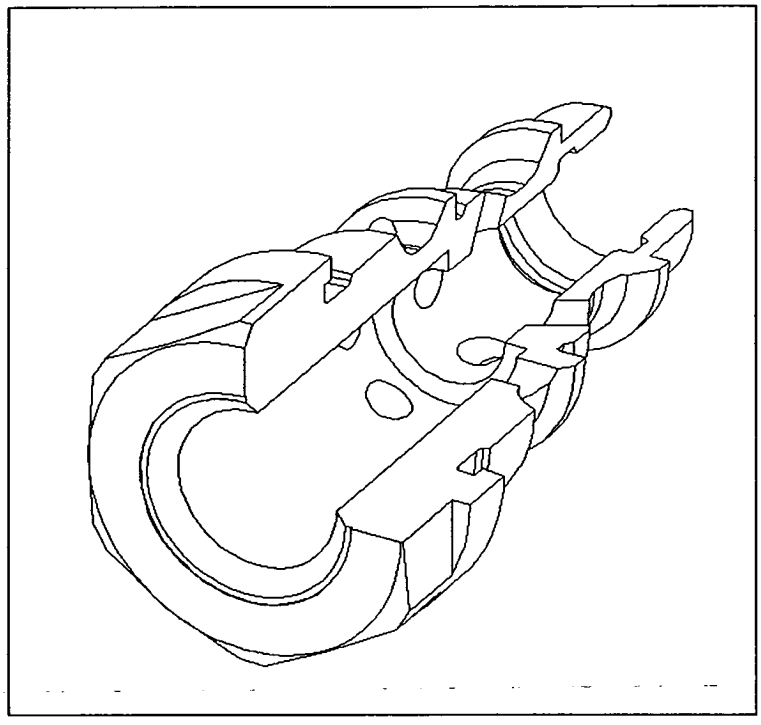


Figure 1.8 Cross-Section Through Solid Model

The two most popular forms of model representation used by solid modellers are *Boundary Representation* (BREP) and *Constructive Solid Geometry* (CSG). The former, pioneered by Braid with the BUILD system [BRA73] in the early 70's, is a complete topological description of the surfaces, edges and vertices that constitute the body, and is said to be an *evaluated model.* The latter, pioneered by Requicha [VOE77] is a tree structure of primitive volumes related by Boolean operators (union, difference, intersection), and is said to be an *unevaluated model.* Both methods produce the same geometric and visual result.

The concept of form features has been introduced as an intuitive and informative way of describing geometry. For instance, we talk about a chamfer, a hole, a slot, a key way, etc. Most standards for dimensioning and tolerancing use the term *feature* to indicate a portion of the component geometry to which a dimension or tolerance is

applied. However, the main motivation for the development of feature technology has been to integrate CAD and NC machining via Computer Aided Process Planning (CAPP), Figure 1.9.

Figure 1.9 CAD-CAPP-CAM Process Chain

At Cambridge University in 1980 pioneering work by Kyprianou [KYP80] proposed the characterisation of a component into a set of form features by recognising topological and geometric patterns in a boundary representation model. Such techniques are known as *feature recognition*. Interest in features was furthered in the mid eighties by Computer Aided Manufacturing - International (CAM-I) in the USA, with their publication of a catalogue of part features for process planning [BUT86]. Also in the mid eighties, again under the auspices of CAM-I, *design by features* was first proposed by Pratt and Wilson [PRA85]. The development of prototype Feature Based Design Systems followed from a number of researchers: Cunningham and Dixon [CUN88], Cutkosky [CUT88] and Shah [SHA88a] in the US and Krause [KRA90] and Rudolf [RUD92] in Europe. Form features used in design provide a level of abstraction from the geometric entities or primitives of solid modelling systems. Solid modelling systems can only store low level information about faces, edges, curves and points or primitive objects such as blocks and cylinders. However, designers think in high level terms of the functionality and aesthetics of shapes which are usually a combination of these primitives. Clearly, when it is known that a particular combination of faces and edges actually represent a hole, it is a simpler task to decide how to produce those faces and edges in the model, because there is significant knowledge about the manufacturing of holes which can be directly associated with part of the geometry of the model. A form feature contains not only the geometry and topology of the entity, but the semantics (meaning) as well, for example a bearing seat, Figure 1.10.

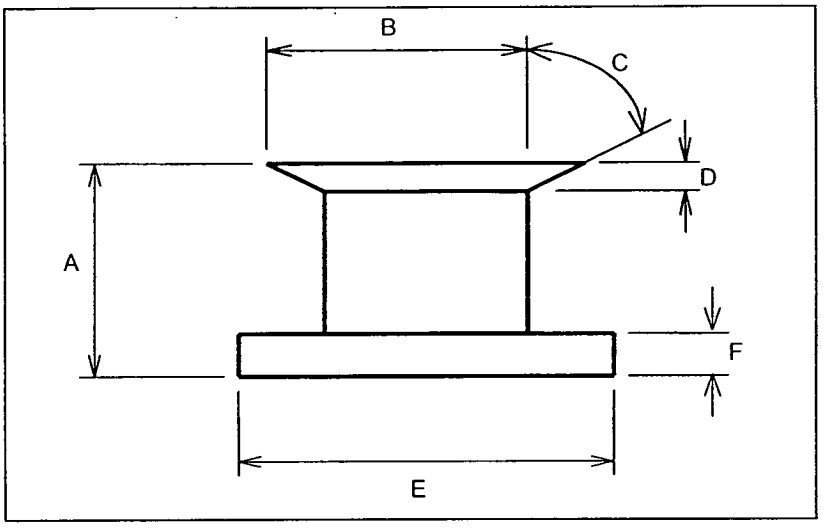


Figure 1.10 Bearing Seat Feature

This application specific knowledge makes it easier for the designer to construct a model of a product. It also means that if the form feature information can be passed to the manufacturing system then it should provide additional data (more than pure geometry) to the planning system as an aid to decision making on how to produce it. Hence form features provide for enhanced integration of CAD and CAM systems. A form feature modeller is usually a front end to a solid modeller. The geometric and topologic model is maintained by the solid modeller; the form feature model is maintained separately by the system and is evaluated by making calls to the solid modeller to produce a geometric result. Due to the application specific nature of form features it is necessary for widespread acceptance that they can be easily tailored to represent any design/manufacturing environment, hence the need for *user defined feature* capabilities [PED96a]. Such capabilities allow the user to define new features and use them in the system in combination with a pre-defined library.

It is impossible to manufacture to a perfect size and so an allowable variation, called a *tolerance*, is specified in the design depending on intended use. A manufactured part that falls within this allowable variation is said to be in tolerance. The size of this tolerance dramatically affects the selection of manufacturing process, method, and hence cost. Tolerances are typically added as the last part of detail design when nominal shape is defined. Tolerances are traditionally indicated on 2D drawings,

typically being applied to the dimensions as dimensional tolerances. More recently geometrical tolerances have been used to control the form of the model independently of its size. Dimensioning and tolerancing for design drawings are well established via the standards: ISO [ISO], ANSI [ANSI], BSI [BSI], for example.

1.4 Problem Definition

Presently, the support of tolerancing in product modellers is an academic issue as most current solid modelling systems do not yet support tolerancing [SAL93]. Clearly, if form features provide for enhanced integration between design and manufacturing by way of aiding process selection and tolerances are a core factor in development of manufacturing strategy, then it would be desirable to provide tolerance information related to the form features. This is not possible when tolerances are added to 2D engineering drawings. However if the tolerances could not only be related to the form features and the solid model data structure but could be visualised in three dimensions, then it would be possible to generate 2D engineering drawings automatically without the possibility for error. Equally integration between design and manufacture would be further advanced because the accuracy information would be available in the same model to CAPP applications. If the dimensions and dimensional tolerances that are related to both the form features and the geometric model could also be made to represent the size parameters of form features, it would be possible to control the size of the model graphically. Such a representational structure would also enable future investigation of variational models by making the size of the features change to the upper or lower limits of the dimensional tolerance. Another significant factor is that if user defined form features are necessary to fulfil the application specific nature of form features then any dimension and tolerancing system must be equally applicable to user defined features and not be restricted to system developments.

Current international standards (STEP [STEP], IGES [IGES]) govern only the exchange of geometric and to some extent tolerance data, but there is no method of

exchanging feature models or of combining tolerance data with the features. The current standards do not meet the functional needs of feature based techniques for the transfer of feature model descriptions and hence restrict the integration of design and manufacturing functions.

Feature based solid modelling systems should have highly developed facilities for the user to define new features in addition to those provided by the system to model application specific environments, especially free form type geometry. Free form geometry is widely used in the casting and plastic injection moulding industries. Tool making for these and sheet metal press tools form a very significant proportion of modelling, analysis and manufacturing needs. The feature modeller should be a general purpose modeller capable not only of supporting geometric descriptions of form but other information that a designer uses to define a product that enables the manufacturing engineer to realise the component meeting the specifications. An integral and homogeneous part of the system must be able to define and present in 3D the dimensions and associated tolerances that the designer uses to constrain the component. This representation should be equally applicable to user defined features. It should be possible to generate automatically 2D engineering drawings without the need to use a separate draughting system. It must also be noted that legacy data in terms of pure solid models must also be considered from an industrial point of view.

It is now widely recognised that the foundation for all software in highly automated processes in manufacture has to be a geometrically complete model of the product [SAB87]. Geometrical completeness is provided by a solid model. Form features are a higher level method of describing such a model. Feature modelling techniques provide a more intuitive and efficient way for a designer to work, and are accepted as an aid to process integration and simultaneous engineering strategies [CUT91] [TOE94a]. However, a number of problems restricting the impact of feature modelling technology in Computer Aided Design and Manufacturing have been identified:

1. There is a considerable legacy of purely volume based models.

2. Libraries of predefined features are too restrictive.

3. User defined feature functionality, particularly definition, has been neglected.

4. Features are assumed to have implicitly defined topology and geometry, little support being given to features of extrinsically defined form.

5. Only nominal, or perfect size geometry is supported.

6. Dimensions and tolerances are detailed in engineering drawings which are 2D representations separate from the feature model.

7. Tolerances and dimensions that are not homogeneous with the shape model preclude the development of variational models and enhanced design and manufacturing analysis.

8. There are no standards for the exchange of user defined feature models.

9. The effectiveness of advanced computer aided process planning, computer aided quality and part programming systems based on feature models derived from a separate design system is reduced because only nominal shape data is supported; other information necessary being input after interpretation of 2D engineering drawings.

These general deficiencies in current feature modelling systems lead to multiple models of the same design and redundancy. There is the possibility of differences between models and in interpretation of incomplete 2D data causing ambiguity. The effects of errors at the design stage not detected until later in product development are much more costly and time consuming to correct. There are clearly inefficiencies due to duality of effort: creating multiple models of the same design, information added to models further along the design path must be verified and corrected if modifications are made to more fundamental models.

In order to more closely integrate design, process planning and part programming functions a method is needed of representing and combining extra design information (dimensional tolerances, geometric tolerances, surface conditions, surface coatings, etc.) with both the resultant geometry and the form features of the 3D design model, so that it is available automatically to other applications. The capability of a feature modelling system to support user defined features in addition to a system library is necessary for acceptance of feature based design systems and enhancement of process integration [PED96a]. Any data model or tools developed must be able to support user defined features. Furthermore, it would be highly desirable if 2D engineering drawings could be generated directly from the 3D environment without the need for a separate 2D drafting module. This is necessary because 2D engineering drawings still form the standard and accepted method of product definition. Therefore it would be necessary to support the layout of dimensions and tolerances in 3D.

1.5 Aims And Objectives

The fundamental concept investigated in this thesis is to integrate in a single computer software representation the information required by designers to define a component for analysis and manufacturing, thereby creating a partial product model and a graphical user interface allowing the model to be created, viewed and changed. The concept necessitates combining into a single integrated and fully associative model the requirements of 2D drafting for the production of detailed engineering drawings, representing 3D solid geometry containing both analytic and free form surfaces, and, modelling dimensional and geometric tolerances, and surface conditions. Feature models are accepted as capable of supporting data rich representations which can form integrated partial product models and provide an intuitive working environment. However, feature modelling techniques have had little industrial impact to date. This thesis proposes that by investigating and advancing feature modelling techniques, it will be shown that the fundamental concept of providing an integrated partial product model can be achieved.

Therefore the aim of the investigations is to advance the understanding and functionality of feature modelling technologies in order to provide better computer aided support for design and manufacturing. Industrial acceptance of feature modelling systems will therefore be widened, and consequently process integration will be enhanced, product development cycles shortened, quality increased and costs lowered, thereby raising competitiveness.

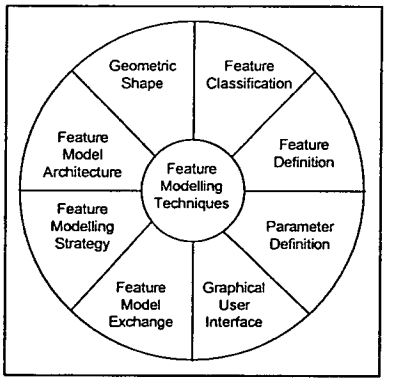


Figure 1.11 Areas Of Investigation

By examining the feature modelling techniques shown in Figure 1-11 four specific objectives have been identified for investigation and software implementation in order to address the fundamental concept and aim (stated above), and the problems described in section 1.4. The four specific objectives are:

1. Development of methods and techniques to model features of extrinsically defined form, typically swept geometry.

2. Implementation of a 3D dimension and tolerance modelling module related to both the feature and solid models.

3. Establishment of techniques to control a feature's size through the use of 3D dimensions.

4. Investigation of structures for the exchange of user defined feature models.

This focus of the work will be on the development of a dimension and tolerance model that is a homogeneous part of a full feature-based solid modelling design

system that may be applied to user defined features of implicitly and extrinsically defined topology and geometry.

The investigations make original contributions to the knowledge and understanding of feature modelling technology by proposing the following concepts:

I. The original concept that certain features cannot be completely described by their make methods has led to the collective grouping of these features and definition of an important new class of features called *extrinsic form features*.

II. The new concept of a *multi-dimensional* or *hyper* feature model has been examined in order to increase the scope and architectural complexity of feature model data structures, enabling a wider group of feature classes to be represented including form, dimension, and tolerance features.

III. The original concept that feature parameters, like features themselves, have characteristics that allow them to be grouped together in classes has enabled a novel combination of procedural and declarative approaches to be applied to user defined feature definition languages, which allows more intelligent mechanisms to be built as a part of the feature modeller kernel and graphical user interface.

IV. The new concept that combines the ideas presented in concepts II and III into a new, more sophisticated feature definition language, which describes both the dimensions and tolerances of a feature in a 3D manner in the template definition.

V. The original concept that provides "handle" like functionality which is applicable to all extrinsically defined features and utilises the body coordinate system of the objects that form the unknown aspects of the features.

VI. The new concept for the exchange of feature models that increases the information content that can be transferred by reducing the flexibility of systems to subsequently change a non-native feature model.

This work contributes to the advancement of feature modelling technology in the following areas:

- Extending procedural methods for user defined form feature definition.
- Considering the needs of the user interaction in user defined feature definition.
- Applying a feature modelling system architecture to the representation of dimensions and tolerances.
- Suggestion of a method for the exchange of feature models containing user defined features.

1.6 Baseline

Feature Modelling has been the focus of much academic research in the past ten years. A general result of this work has been the development of feature modelling system architectures where the feature and geometric data structures are separate. Earlier feature based design systems whilst offering the ability for the user to define the type of feature and its parameters suffered from the fact that the definition of the geometry had to be produced by the system developer rather than the user [KRA91][RUD92]. Two solutions have been established:

1. to develop sophisticated interpreted languages used for procedural feature definition that drive both the feature modeller and the geometric engine, and
2. to develop declarative feature definition languages which is the subject of current research.

The development of sophisticated procedural languages is a commercial task rather than a research issue. One of the first commercial Feature Modelling Systems appeared in 1992 and was developed by Strässle Informationssysteme AG [SIS], Switzerland, and is today known as FeatureM. Features may be fully user defined in a proprietary interpreted language called MCL+.

The work reported in this thesis has been undertaken in collaboration with Strässle Informationssysteme AG. The collaborative nature of the work has meant that the developments have been restricted to the use of the feature modelling system, FeatureM, together with the MCL+ language and other system components such as the geometric kernel, ACIS [SPA]. Major aspects of the system have already been defined and the developments reported in this thesis use these concepts to extend the architecture and functionality of the system. The use of a relatively well defined system has influenced the developments by providing a well defined domain in which to work, which is not the case when developing a new system.

The feature modelling system, FeatureM, will be described in chapter 3. It is necessary to provide a full description because feature modelling systems are complex software applications, and no one part can be developed in isolation or without consideration for the other parts. The author has only contributed in minor areas to the system as described in chapter 3 which is predominantly the work of others and forms the starting point for the work developed and reported in chapters 4 and 5.

1.7 Structure Of Thesis

This chapter (chapter 1) has introduced the topic of the thesis, advancing feature modelling technology, and the related subjects of geometric, dimension and tolerance modelling. Current deficiencies in feature modelling techniques have been identified, and the aims and objectives of the work reported in this thesis have been stated. The baseline for the work has been defined.

Chapter 2 will present a general review of the subjects involved in the developments of the work. An overview of the design process (including simultaneous engineering, agile manufacturing and product modelling) and the applicability of feature modelling techniques will be given. Geometric modelling techniques will be described because of the need for feature modellers to use a geometric kernel to represent shape. Parametric and variational modelling techniques will be discussed.

Feature modelling technology will be presented, including feature recognition and feature based design methods. Dimension and tolerancing practices will be introduced and the application of software support for dimension and tolerance representations associated with solid and feature models will be reviewed. Standards in the areas of product, geometric and feature modelling and the representation of dimensions and tolerances will also be reported.

Chapter 3 will present the architecture of the of the feature modelling system, FeatureM, which forms the baseline for developments reported in chapters 4 and 5. The main software components will be described and consist of the graphical user interface, MCL+, the geometric kernel, the feature modelling kernel and application modules. Feature modelling systems are complex and no one aspect can be developed independently and without consideration of the others.

In chapter 4 the existing methods of describing and controlling user defined features in FeatureM will be presented in detail. The architecture and definition methods will be extended to provide for parameter validation and increased user interface support. The modelling of features with extrinsically defined form will be described because of the importance of integrating geometry defined externally to the feature definition. This will focus on constraint based 2D sketchers, free form surfaces and swept geometry.

In chapter 5 the needs of the dimension and tolerance module will be defined, the software architecture and functionality developed will be presented. The methods developed to associate dimensions and tolerances with both solid and feature models, and represent 3D feature size parameters and 2D sketcher constraints with dimensions, will be described. Representing 3D relational constraints will be investigated by implementation of a parallel geometric tolerance. Functionality developed to enable editing of the layout of the dimensions and tolerances will be described.

Chapter 6 will explicitly describe a number of aspects of the work concerning the original concepts investigated, industrial collaboration, analysis and testing. The

chapter will present the fundamental concept of the thesis that has been investigated through the aims and objectives of the work. The importance and extent of industrial collaboration involved in the research will be described, together with references to the information and feedback provided. Chapter 6 will present the research techniques used to acquire and analyse information in order to define the software architecture which resulted in the implementation reported in chapters 4 and 5. The software testing methodologies employed for the evaluation of the experimental software and the results of tests will be given. The original contributions to knowledge made by the investigations will be stated.

The results of the work will be discussed in chapter 7. The following aspects will be considered for the feature (form, accuracy, etc.) developments: classification, system architecture, and, representation. Possibilities for the exchange of user defined form feature models and the potential of feature modelling systems to act as process integrators will also be discussed.

The conclusions and recommendations will be made in chapter 8. The references are stated in chapter 9.

Appendix 1 lists seven published papers of the author relating to the work reported in this thesis. Four papers form part of the general background to the work reported:

1. [PED89] introduces feature modelling concepts as a means to process integration.

2. [HUS90] describes a feature based manufacturing model used for automated process plan generation.
3. [HUS91] presents a component formed from features which provide specific difficulties for process planning systems.
4. [MIL93] discusses problems of representing form features focusing on the interactions between features, which may be defined as implicit or explicit.

A further three papers report work which is drawn directly from this thesis:

5. [PED95] presents the modelling methods for dimensions and tolerances as a means of representing explicit feature interactions.

6. [PED96a] reports on the needs for user defined features and the techniques developed to represent them.

7. [PED96b] discusses the problems of exchange of feature models.

1.8 Chapter Summary

This chapter (chapter 1) has introduced the topic of the thesis, advancing feature modelling technology, and the associated subjects of geometric, dimension and tolerance modelling. Current deficiencies in feature modelling techniques have been identified and the aims and objectives of the work reported in this thesis have been stated. The baseline for the work has been defined and an outline of the thesis described.

Chapter 2 will present a review of the subjects involved in the developments of the work. These subjects include the design process and simultaneous engineering, geometric, parametric and variational modelling, feature modelling, dimension and tolerance modelling and the standards associated with them.

Chapter 2

Computer Aided Design And Manufacture

2.1 Introduction

2.1.1 Chapter Overview

Chapter 1 presented the subject of this thesis, feature modelling, and the problems associated with representing dimensions and tolerances. In this chapter representation methods and standards will be reported.

Chapter 1 introduced a range of computer modelling techniques used to represent the nominal geometry of engineering components - wire frame, surface and solid modelling. In this chapter these techniques will be discussed in more detail. The underlying representations and usage will be described because the concepts are fundamental to the development of feature modellers, and in particular, the description of the system used as a baseline for this work to be presented in chapter 3.

Chapter 1 also presented the concept of tolerances which are typically added to engineering drawings to indicate an acceptable variation from nominal to allow for the inherent inaccuracy of manufacturing methods, yet maintaining functional needs. The representation of dimensions and tolerances in engineering drawings is governed by national and international standards which will be detailed in this chapter. Research in the area of dimension and tolerance modelling with computers has focused on data structures, analysis and modelling techniques that can be applied to 3D geometric models. This work will also be reviewed in this chapter because it provides a baseline and context for the developments described in chapter 5 to integrate dimensions and tolerances with both geometric and feature models.

Other computer techniques were also introduced in chapter 1 that have been developed to aid the planning of manufacturing processes (Computer Aided Process Planning) and the writing of NC code (NC Part Programming) required to control NC machine tools that produce the modelled component. These techniques mimic industrial practice of separating function and using different abstract representations to model each application specific view of the component. Much work has focused on integrating this disjoint functionality, both for the engineering structure of the workplace and the software support tools. Simultaneous, or Concurrent Engineering [BED91] is an approach that is seen to address these issues and will be described in this chapter.

The demands on software tools to support Simultaneous Engineering strategies are greater than for a single function because of the much higher degree of integration required and the more automated functionality to be delivered. Product data modelling has emerged under the auspices of the Standard for the Exchange of Product Data [STEP] as the structure for supporting all information relevant to a component throughout its complete life cycle. Product modelling and more specifically the standards relating to the exchange of product data will be discussed in this chapter.

Feature modelling techniques have been the subject of much research and commercial systems are now finding acceptance in the market place. Approaches to feature modelling will be reviewed in this chapter to provide a context for the system described in chapter 3 and a baseline for the developments reported in chapter 4.

2.1.2 Chapter Structure

The introduction forms section 2.1 of this chapter (chapter 2). Section 2.2 gives an overview of the design and manufacturing process and recent trends. Section 2.3 describes geometric modelling techniques. Section 2.4 presents approaches to feature modelling. Dimension and tolerance modelling techniques are described in section 2.5. The standards relevant to the objectives of the work are detailed in section 2.6. A summary of the chapter is presented in section 2.7.

2.2 Design And Manufacture

2.2.1 Background

Ever since computers were applied to aid engineering processes in order to improve design and production they have been applied to support particular applications. Typical design applications are drafting, 3D modelling, finite element analysis, kinematic analysis, and visualisation. The result of the design phase was passed from the design department to the planning department for preparation (process planning and NC part programming) and finally to manufacturing. More recently rapid prototyping techniques have been developed to create physical models of a component directly from its geometric solid model as an aid to design verification [CHI93]. Concurrent engineering strategies have been proposed which aim to integrate these functions into teams [OGR96]. Software support of concurrent engineering has focused on integration via the development of product models.

Feature modelling is seen as a method of realising a partial product model. A significant factor in design and manufacturing integration is the availability of tolerances associated with the features and geometry of a model. Tolerances are required for correct selection of manufacturing methods, processes and tooling.

2.2.2 The Design Process

Cross [CRO89] suggests that problems are either well defined, like a game of chess, or ill-defined. Ill-defined problems have vague, but numerous, and often conflicting constraints, are solution dependent but have no definitive solution. Design problems are widely recognised to be ill defined and consequently problem solving strategies are termed solution focused.

Attempts to model the design process have produced two approaches:

- descriptive, and
- prescriptive.

Descriptive models simply describe the sequence of activities that occur in designing. Prescriptive models attempt to provide a rigorous, systematic set of activities that lead to a solution of the "real" problem.

The simplest descriptive model consists of three processes: generation of solutions, evaluation of them, and communication of the chosen solution. French [FRE85] developed a more detailed descriptive model based on the following activities: analysis of the problem, conceptual design, embodiment design, detailing.

Prescriptive models are said to have a particular *design methodology*. A number of prescriptive design models have been proposed [ARC84][MAR84][VDI2221], however the most widely reported is that of Pahl & Beitz [PAH84]. In Figure 2.1 there are four phases: clarification of the task, conceptual design, embodiment design, detail design.

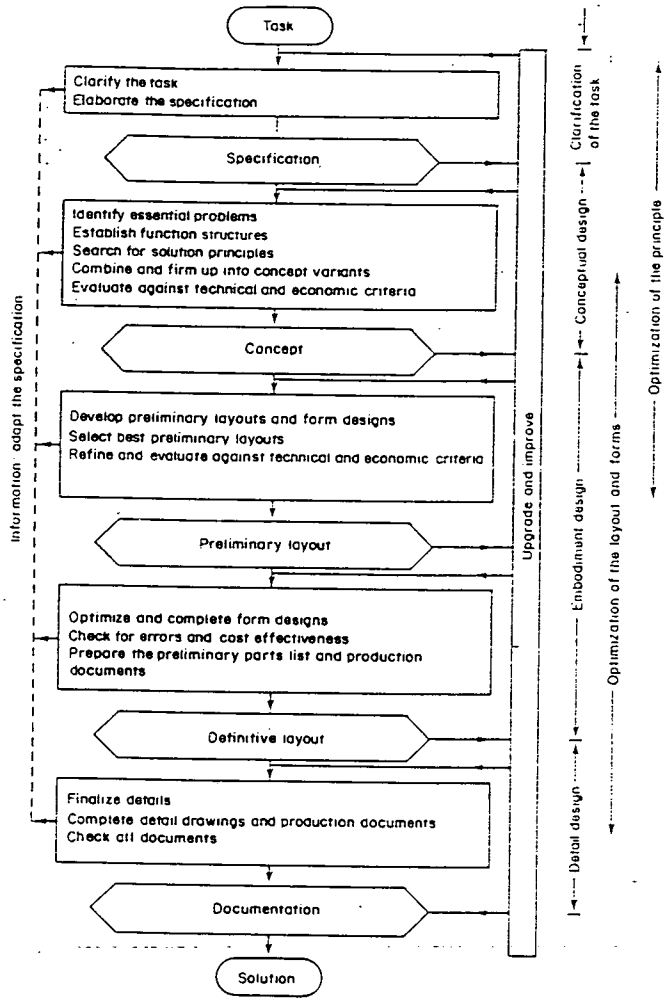


Figure 2.1 Pahl & Beitz Model Of The Design Process

To date, 3D and feature modelling techniques have been applied in the latter stages of embodiment and for detailed design. Some of these approaches will be reported in sections 2.3 and 2.4.

2.2.2 Concurrent / Simultaneous Engineering

The division of labour, specialisation and compartmentalisation has led to a cascade of processes being used to develop a product:

- design
- production planning
- manufacture

The part would be designed (more or less completely) and passed to production engineering. Once in production engineering it would be too late to have design modifications made to ease manufacture. Similarly the component would be passed to the shop floor for production, again too late to effect any changes to improve quality because the design would have been frozen. The only method of improvement was to go round the whole loop once again drastically lengthening the time to full production. Today, competitiveness is seen to be greatly effected by the time it takes to bring a product to market. The speed with which a manufacturing enterprise can respond to marketing demands enhances the probability of commercial success.

By contrast Concurrent Engineering aims to:

- shorten lead times
- increase quality
- lower costs
- consider the total life cycle of the product

Not only is it necessary to engineer the pre-sales phase, servicing of the product during its useful life must be considered, as should environmental issues such as recycling and decommissioning.

The above goals can be partly achieved by using teams of multi-disciplinary people. Success of concurrent engineering efforts however, depend on coordination, cooperation, communication and a sharing of knowledge and experience between all functions involved in the development of a product: marketing, engineering, purchasing, sales, etc. Computer tools can clearly be an aid to such strategies. The Concurrent Engineering approach is compared to a traditional strategy in Figure 2.2.

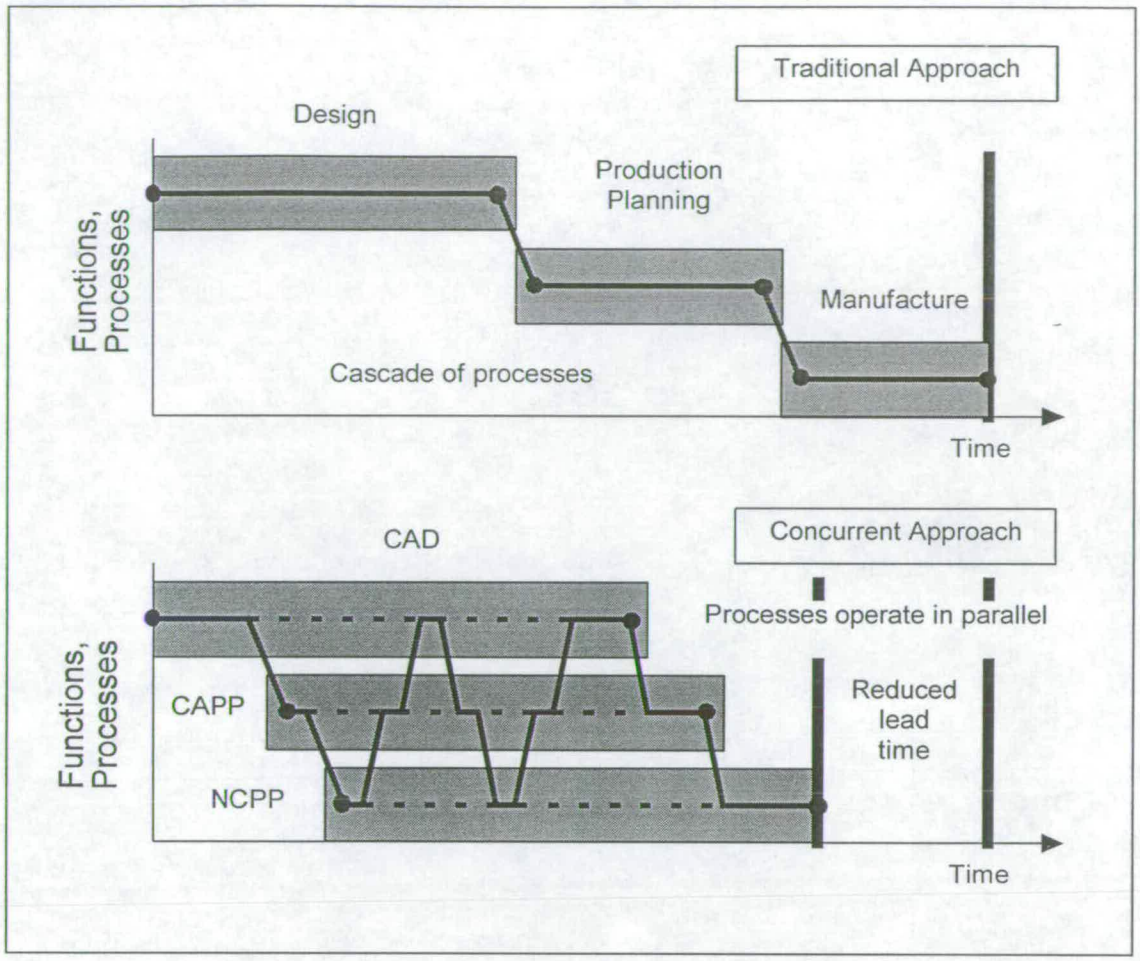


Figure 2.2 Comparison Of Traditional And Concurrent Engineering Approaches

Concurrent Engineering has been defined [IDA88] as:

> "The systematic approach to the integrated, concurrent design of products and related processes including manufacture and support. This approach is to cause the developers from the outset, to consider all the elements of product life-cycle from conception through disposal including quality, cost, schedule and user requirement"

Simultaneous Engineering is a widely used synonym for Concurrent Engineering. In 1991 Eversheim described Simultaneous Engineering as:

"Process oriented, function integrated, product and production design realised by parallel task processing in order to shorten time to market"

Results of projects such as SESAME [SES] have shown that idealised parallel processing is achieved in practice by a series of short sessions with a set of highly integrated software tools. It can be seen from Figure 2.3 that the time span for a single process to be completed increases with simultaneous engineering techniques, however, the overall development lead time for both product and production process is shortened.

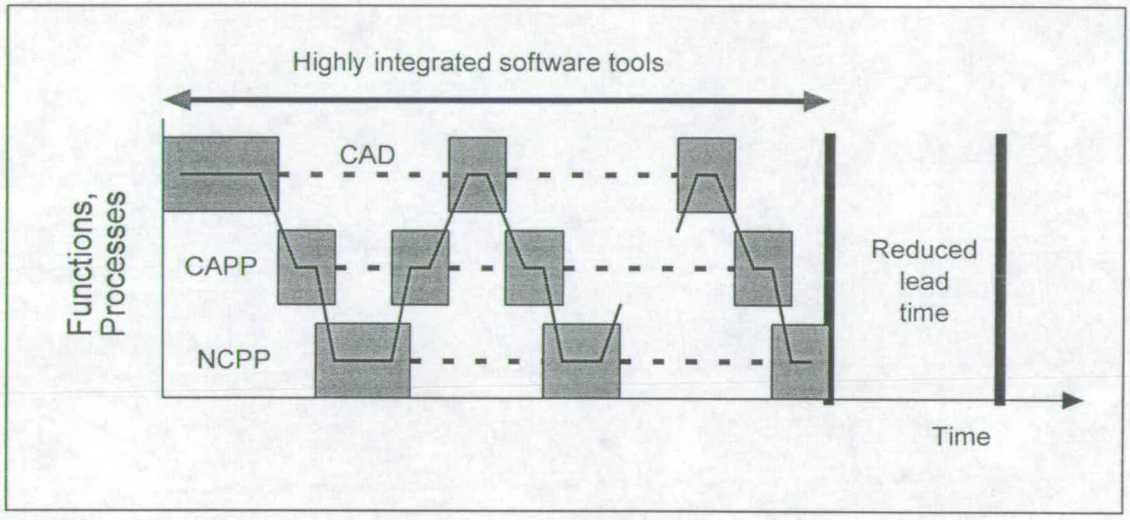


Figure 2.3 SESAME View Of Simultaneous Engineering

Fundamental to the success of Concurrent Engineering approaches is the ability to provide a workpiece description that supports the information required at all stages in the process chain. Geometry alone is not sufficient. Feature modelling is seen as a method of providing information rich models consisting of geometry, the tolerances associated with that geometry and a higher level description that captures the meaning of the geometry that allow manufacturing decision making to be performed by computer. Such rich data models can be referred to as partial product models.

The influence of all aspects of the engineering development are considered from the beginning. This integration of product and process design leads to higher quality being designed in. Customer requirements may be met without compromise, and

because engineering changes are minimised late in the design phase so costs are reduced. An added advantage of highly integrated software tools is that they provide for greater flexibility which is regarded as an essential element for success in today's markets.

2.2.3 Agile Manufacturing

Social and economic changes in industrialised society over the last decade and the globalisation of business activities have significantly changed the way in which design and manufacturing technologies are used. No longer can factories be strictly functionally organised and operate in near isolation of the market. Manufacturing systems have become customer and product oriented, aiming to reduce lead times, minimise work in progress, utilise just in time flow of materials, and maximise efficiency and flexibility of manufacturing capacity. There are complex relationships between customers, conceivers of products, manufacturers, sub-contractors and suppliers. Agile manufacturing encompasses these characteristics. Implementation of agile manufacturing methodology requires that product design, production engineering and manufacturing are closely integrated. Concurrent engineering methodologies address part of this concept. Close integration of manufacturing system functions demands that sufficiently complete and accurate information of the shape, technology, processes and operations is available. A computer representation of the shape, technology, processes and operations required to realise a product is called a *product model.*

2.2.4 Product Models

The variety of manufactured articles is vast and no one product model representation is likely to cater for all possible applications. The most general product model structure to date developed is that of the Standard for the Exchange of Product Data - STEP [STEP], which is described in section 2.6. STEP makes restrictions on the

application of the model to such areas as mechanical engineering (which is also very broad), electrotechnical engineering and ship building.

Within the area of mechanical engineering it has been found necessary to reduce still further the scope of product models for actual integrated systems. One such example is that provided by the Brite/Euram project SESAME [SES], which provided a partial product model for prismatic parts consisting of a set of company specific 2½D features. The partial product model did however extend to include not only design information, but that of process planning and NC part programming.

It is likely for the foreseeable future that commercial development will provide partial product models particularly focused on design, process planning and NC part programming. Feature modelling techniques are seen as a method of realising sufficient high level partial product models that are particularly suited to integration of process planning and NC part programming software applications.

2.3 Geometric Modelling Techniques

2.3.1 General

A geometric model is an occurrence of a mathematical system employed in shape representation [STEP48]. Wire frame, boundary representation (BREP) and constructive solid geometry (CSG) models are examples. Geometric modelling systems are relatively abstract and are applicable to broad classes of shapes. Modelling complex geometry in 3D space has developed along two relatively independent paths: free form surface modelling and volume modelling.

To date however, 2D engineering drawings still form the accepted design definition of a product. Although 3D geometric information, particularly in the case of free form surface data, is used to aid communication of the definition of a component between engineering functions, 2D engineering drawings are always supplied. This is

because geometric models do not support the range of design information required of a product model for use in downstream processes, namely manufacturing. Computer drafting systems are still an essential element of computer aided design tools.

2.3.2 Drafting

2D computer based drafting systems have developed rapidly to a high level of sophistication from the humble beginnings of SketchPad [SUT63]. 2D systems initially replicated the actions of the draftsman's pencil and paper producing a computer based representation. 2D systems have become ever more complex. The latest relational systems (ObjectD [SIS]) go some way to offering associativity between engineering views. This certainly is an aid to developing complex drawings and rapidly speeds up the implementation of changes.

However, the generic model is still 2D. It consists only of lines, arcs and textual information, Figure 2.4. These are frequently unrelated. Dimensions and tolerances are added to engineering drawings to indicate allowable variation of the manufactured part. A first interpretation of the 3D world is initially prepared by the designer. This is reinterpreted by the planning engineer who prepares the process plan. It is reinterpreted a second time by the manufacturing engineer who prepares the NC part programmes. This chain of processes is shown in Figure 2.5. Both planning and manufacturing engineers use the dimension and tolerance information which they have to relate to their interpretation of the 3D object in order to reason about manufacturing strategies. Clearly there are opportunities for misinterpretation. Direct computer aid to the planning task from the 2D data model is limited to group technology and latterly interactive planning particularly for sheet metal parts. NC part programming has benefited considerably in the 2 1/2 D area with 2D drafting systems providing contour data.

Statement of ownership of drawing and that drawing may not be copied or passed to third party without written consent

Bored

Sharp edges

Scale

Spindle Body

Un-toleranced dimensions

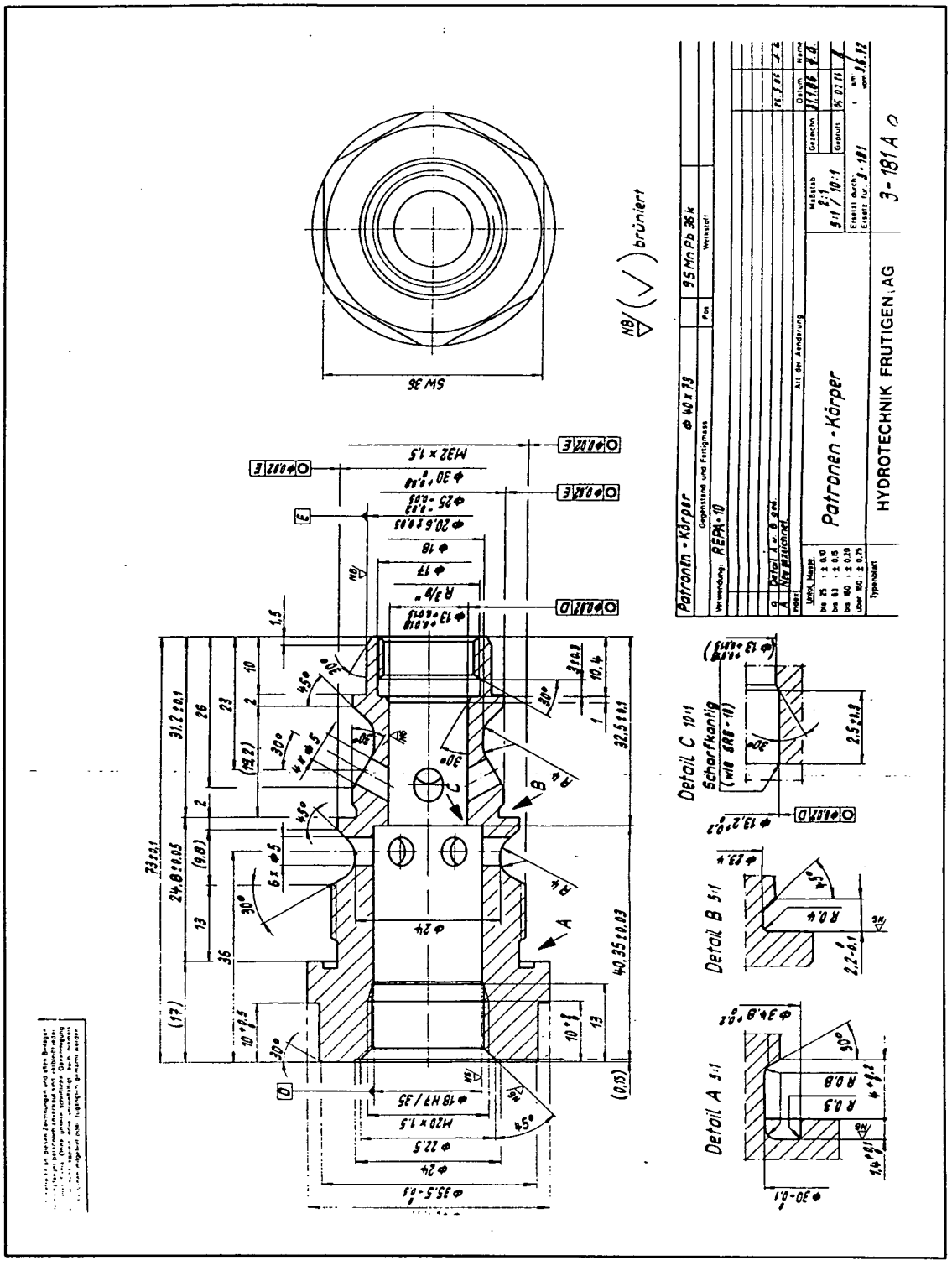


Figure 2.4 Traditional 2D Engineering Drawing

However, virtually all reasoning about the geometry is done by the engineer with little support form the geometric model. This is because of its two dimensional nature. Computer representations of engineering drawings do not provide a

sufficiently complete or accurate model of the part for integrated software tools, yet they are still required to be produced. One aspect of the work reported in chapter 5 is not only the implicit, but the explicit definition of engineering drawings directly from the 3D solid feature model without the need for a separate 2D representation.

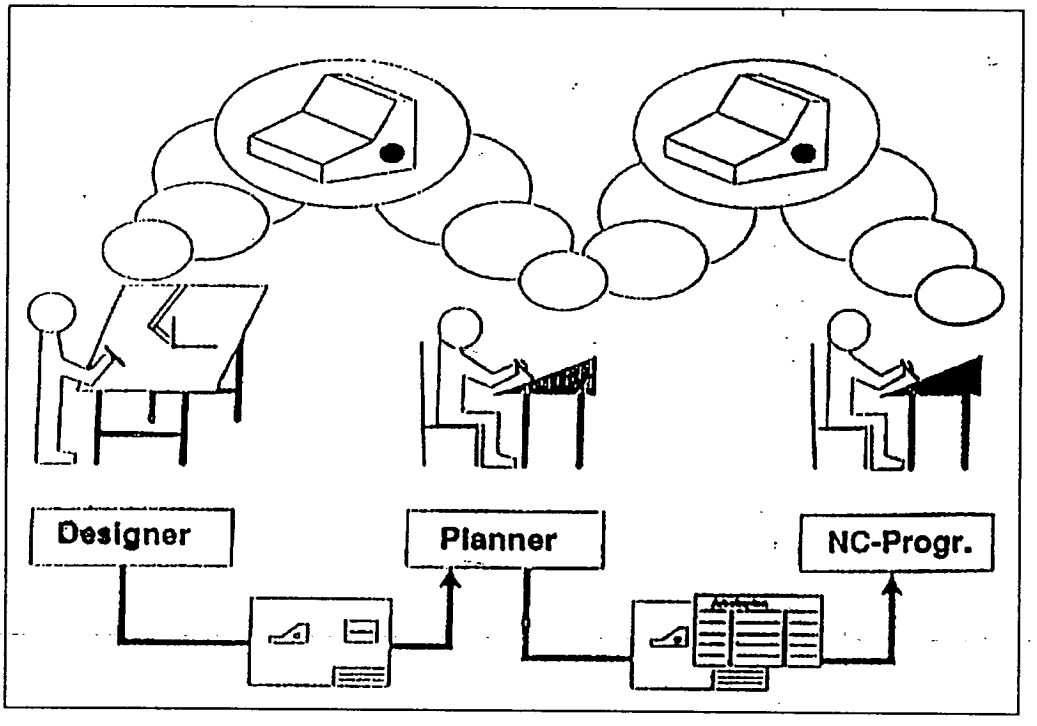


Figure 2.5 Engineering Drawings Are Insufficient For Process Integration

2.3.3 Wire Frame Modelling

The simplest 3D representation is that of the wire frame model, Figure 2.6 (a). Whilst engineering views generated from the model are associative, no information about the surfaces of the component are provided. Surfaces that cannot be explicitly defined limit greatly the type of objects that can be completely and accurately modelled. Surfaces are important for manufacturing reasoning and again must be interpreted form the data model. A wire frame model is not sufficiently complete for the requirements of product development. However, a wire frame model can represent the contours and paths that define swept objects. Despite this, the result requires a solid model representation for completeness. Objects modelled by more

complex surface or solid methods are frequently viewed as wire frames. This aids picking of geometry and display response. Wire frames of solid models are generally viewed with the addition of silhouette lines generated from the surfaces, Figure 2.6 (b). Silhouette lines are particularly important for the visualisation of non-planar surfaces.

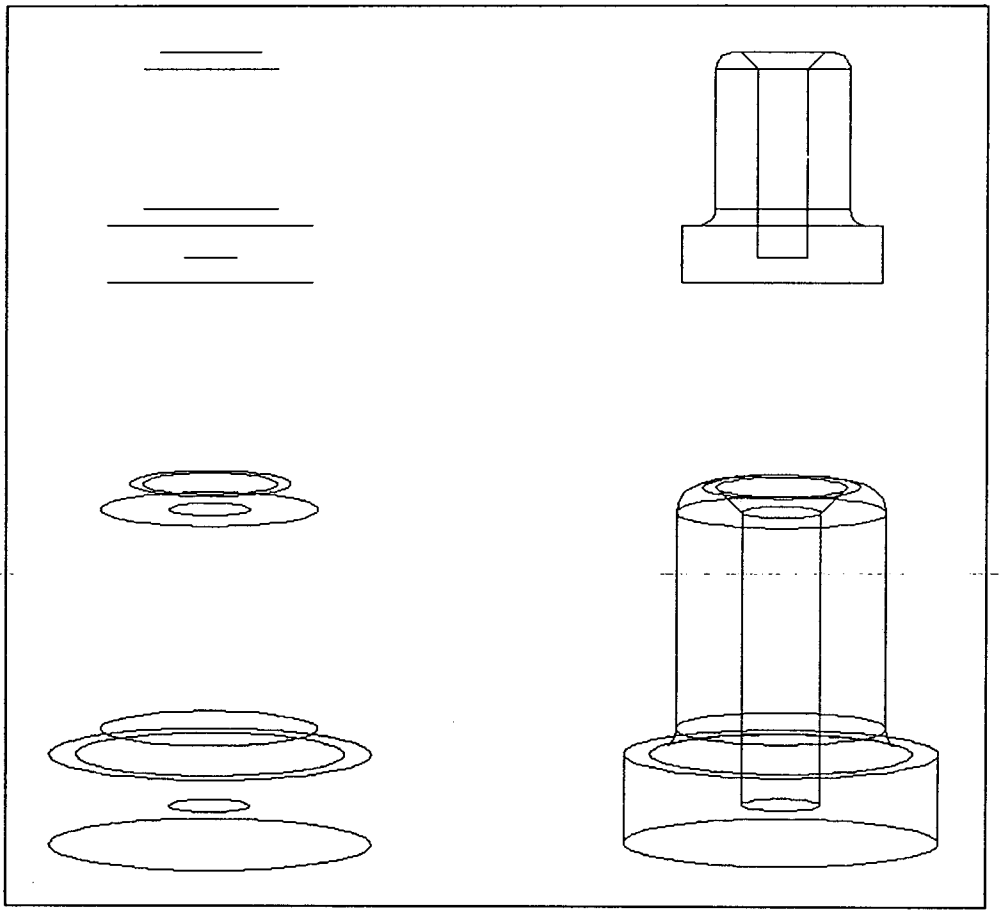


Figure 2.6 (a) Wire Frame View (b) Wire Frame View With Additional Silhouette Lines

2.3.4 Surface Modelling

Free form surface modellers were developed to meet the needs of modelling smoothly blended geometry such as created by a designer's clay model. A typical surface model of a car body panel is shown Figure 2.7.

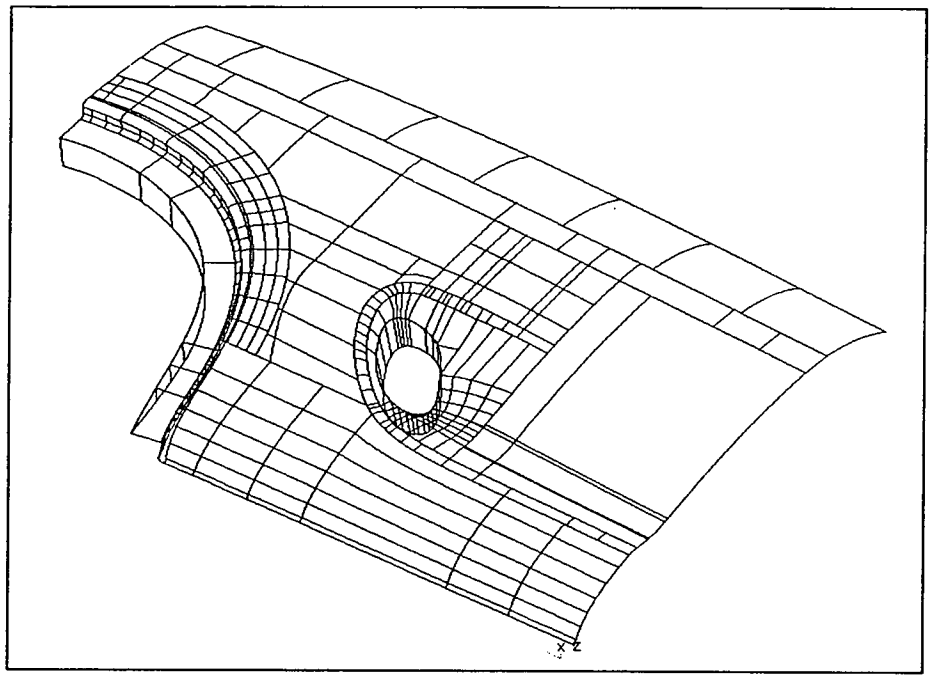


Figure 2.7 Surface Model Of Car Body Panel

Surface modelling techniques were pioneered in the automotive industry for body design by Bezier with the system UNISURF [BEZ74], and in the aircraft industry by Coons, Ferguson and Forrest [FOR72, COO74] for wing and fuselage design. B-Splines were developed as an additional representation method [GOR74]. Latterly NURBS (Non-Uniform Rational B-Spline) [DIN87] surfaces have been presented as the most general, if not widely used, modelling representation. This is because they can represent analytical geometry exactly and the representations of Bezier et al can also be supported. The mathematical representations of curves and surfaces may be found in [FAU87].

Free form curves are constructed from curve segments using parametric cubic polynomials. The geometry of a curve segment is defined by a number of controlling parameters. The curve representation developed by Ferguson uses the two end points and the tangent vectors at the end points to define the curve. The bi-cubic patches developed by Coons use the corner points, the tangent vectors and the surface twist vector at the corners to define the surface, Figure 2.8.

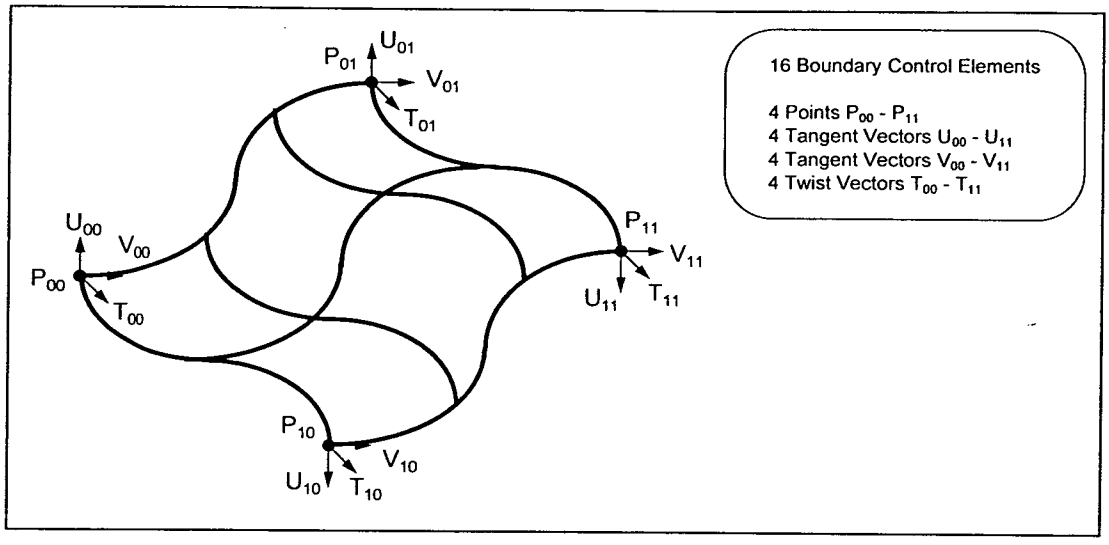


Figure 2.8 Coons Surface Patch Definition

Bezier curves are defined by four points, the two end points and two points which do not lie on the curve but are used to define the tangent vectors at the start and end points. Bezier patches are defined by sixteen points: four corner points and twelve points which do not lie on the surface but control its shape, Figure 2.9.

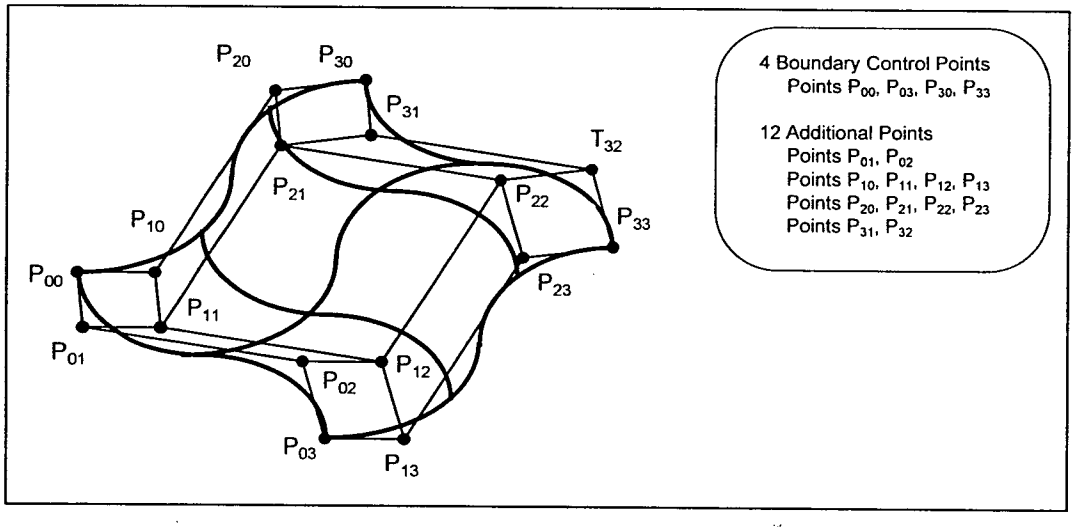


Figure 2.9 Bezier Surface Patch Definition

Free form curves and surfaces are defined over some finite region. Often boundary curves are defined within a surface patch to indicate the required shape. Coons and Bezier patches can be joined together with tangent continuity. Spline surfaces are required if curvature continuity is to be maintained. To date the Bezier form of

manipulating curves and surfaces is the most popular because of the intuitivity and representational simplicity of defining surfaces using only points.

The generation of surface models is generally very tedious. Surface modellers generally have no topological structure, no knowledge of the cross sectional surfaces of a component, and are not sufficiently complete. The methods of describing non-analytic curves and surfaces, particularly the NURBS representation, are used by solid modellers to define complex geometry. Feature modelling with free form curves and surfaces is a current research issue [AUR95].

2.3.5 Solid Modelling

Solid models maintain a description not only of the surfaces that constitute a solid, the regions bounded by the surfaces as solid or void are also defined. This enables cross-sectional models to be correctly generated, Figure 2.10. Surface areas and mass properties can be calculated. Interference checking between models may be performed. To date solid models form the most complete and accurate geometric description of a component. Although this description is limited to the nominal or perfect size of the geometry, solid models do form the definition of the geometric aspect of product models.

Solid modelling techniques have become well established since the first developments in the early 70's. The BREP approach pioneered by Braid [BRA73] and the CSG approach developed by Requicha [VOE77] have formed the basic approaches for developments since. The two methods have been widely reported in most texts on computer aided design and manufacturing, of which [ZEI91] and [FOL90] are typical. Today the BREP approach forms the basis for most commercial systems. However, both techniques will be briefly described in order to aid the reporting of feature modelling technology in section 2.4, the system description in chapter 3, the enhancements relating to extrinsically defined features in chapter 4 and the development of dimensions and tolerances in chapter 5.

Currently, solid modelling techniques are limited to *rigid, homogeneous* bodies.

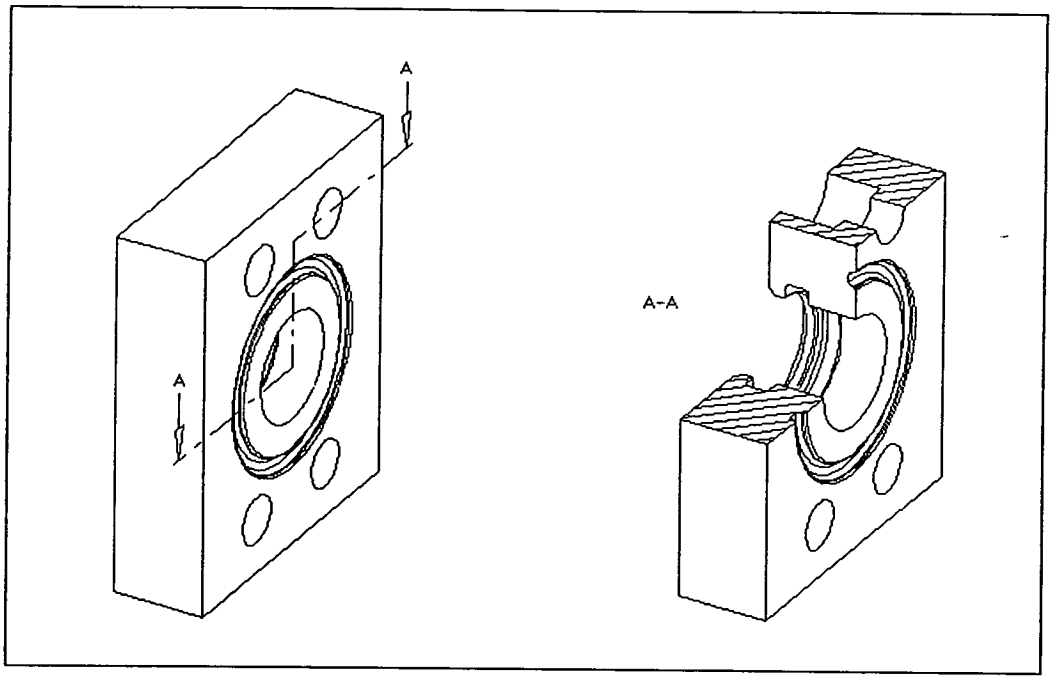


Figure 2.10 Cross-Section Through Solid Model

2.3.5.1 CSG

In constructive solid geometry a model is represented as a combination of simpler solid objects. These objects are known as *primitives* which themselves are combinations of even simpler entities called *half-spaces.*

Half-spaces are analytic geometric surfaces which divide space into two halves: solid and void. The boundary between the two regions of a planar half-space extends to infinity. Half-spaces can be semi-bounded as is the case of an infinitely long cylinder, or fully bounded as is a sphere or torus.

Fully bounded half spaces, such as a sphere, can be mapped directly to a realisable physical object. Unbounded and semi-bounded half spaces require combination with other half-spaces to produce a result which is mappable to a physical object. The manipulation of half-spaces by the user in a geometric modeller is generally very

much abstracted from the bounded objects of the real world and makes the construction of solid models tedious. For this reason half-spaces are typically combined into *primitive* entities which are directly mappable to physical objects. For example, the combination of six planar half-spaces define a cube, and the combination of an infinitely long cylinder capped by two planar half-spaces produce a cylinder, Figure 2.11. The use and manipulation of primitives, such as cubes and cylinders, is much more intuitive.

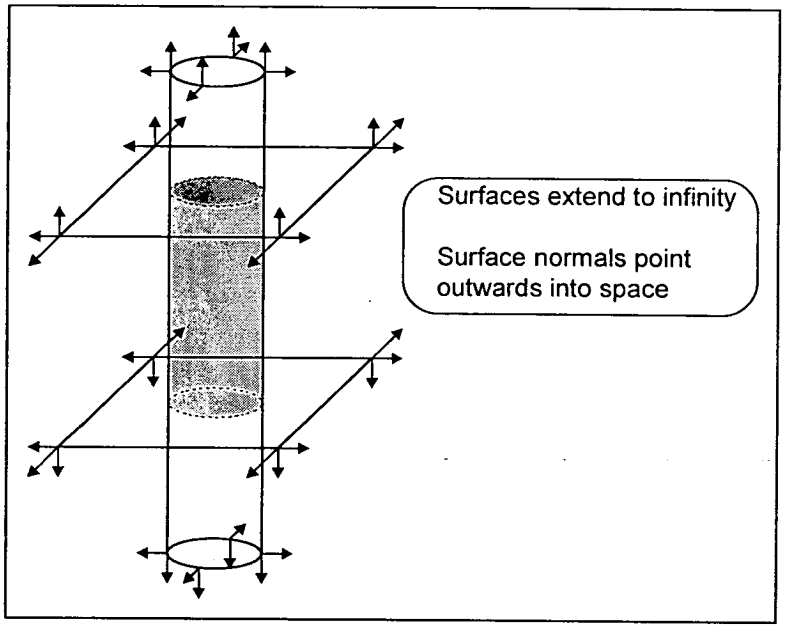


Figure 2.11 Half-Spaces Combined To Define A Cylinder

Any point may be classified as "inside", "on" or "outside" the boundary of the model, which corresponds to the point being in the solid, on its surface, or in space. The mathematical representation denotes the void region by defining that the normal vectors of surfaces point into space. The mathematical representation cannot distinguish between physical reality and the abstract model. Therefore, it is possible by reversing the surface normals to reverse the sense of the half spaces, creating what is termed a negative model. For the case of a cube, everything contained within the six half spaces is void, everything outside is solid. Such a model has a negative volume.

Primitives are combined using Boolean set operations of union, difference and intersection, Figure 2.12. The results of Boolean operations in CSG modellers are regularised to remove any dangling faces, edges or vertices. The data structure representing the complete object consists of a binary tree in which the leaf nodes represent the primitives and the internal nodes represent the Boolean operators.

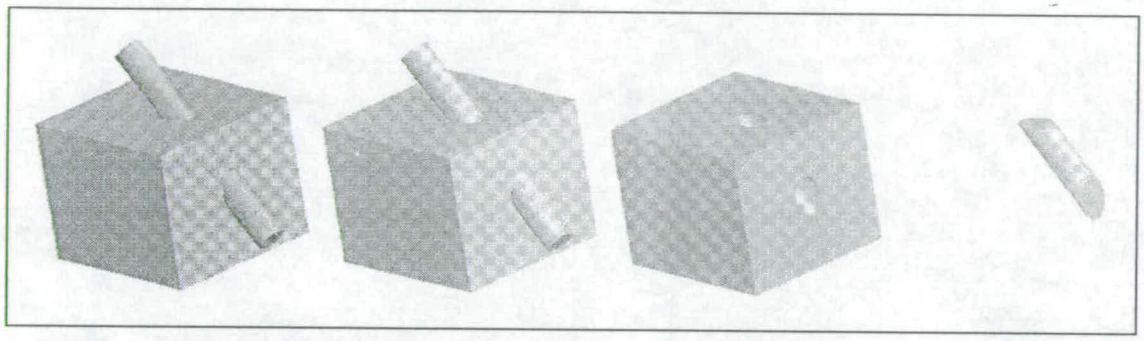


Figure 2.12 Boolean Operators (a)Bodies (b)Union (c)Difference (d)Intersection

The ordering of primitives in the tree, and hence the ordering of combination, determines the resultant geometry. It is generally the case in models of even relative simplicity that evaluating the primitives in a different order will produce a different result, Figure 2.13. A CSG model is an *unevaluated* model because the data structure does not store a representation of the resultant model. Visualisation of the faces, edges and vertices requires the process of *boundary evaluation* to be performed on the evaluated model. A significant factor affecting the performance of modelling systems is the calculation of silhouette lines, particularly if formed by non-analytic curves.

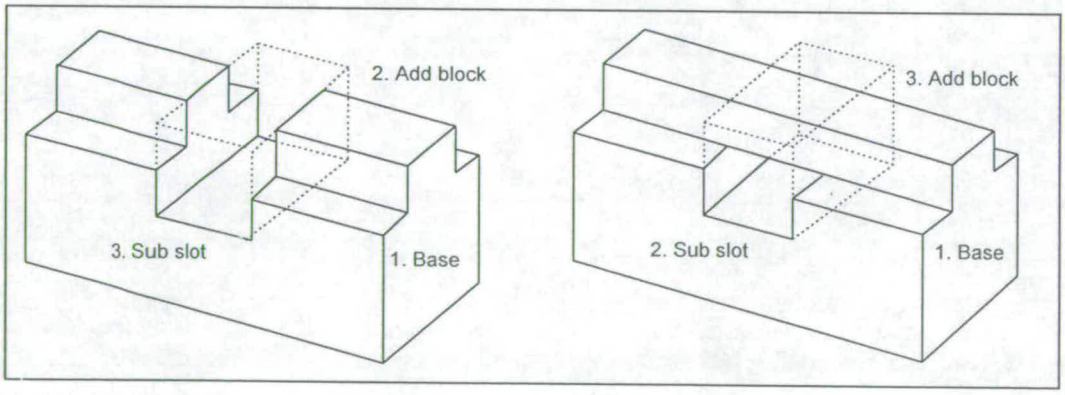


Figure 2.13 Ordering Of Bodies

The use of free form surface definitions in CSG modelling is limited because they are not infinite or periodic semi-bounded half-spaces. The geometric equivalents of natural manufacturing operations such as filleting, rounding and chamfering are tedious to achieve because the volume of material to be removed or added must be created using primitives. Much of the geometric reasoning performed by computer aided process planning and NC part programming systems uses the topological connectivity between faces and edges in the model. These are not readily available in a CSG modeller nor are they supported in a useful data structure. Proponents of CSG modellers build a separate data structure representing the boundaries of the model [REQ86]. However, the concept of the CSG tree is used extensively in Feature Modelling techniques.

2.3.5.2 BREP

In contrast to the CSG approach, a BREP representation is an *evaluated* data structure containing the elements which describe its boundary, Figure 2.14.

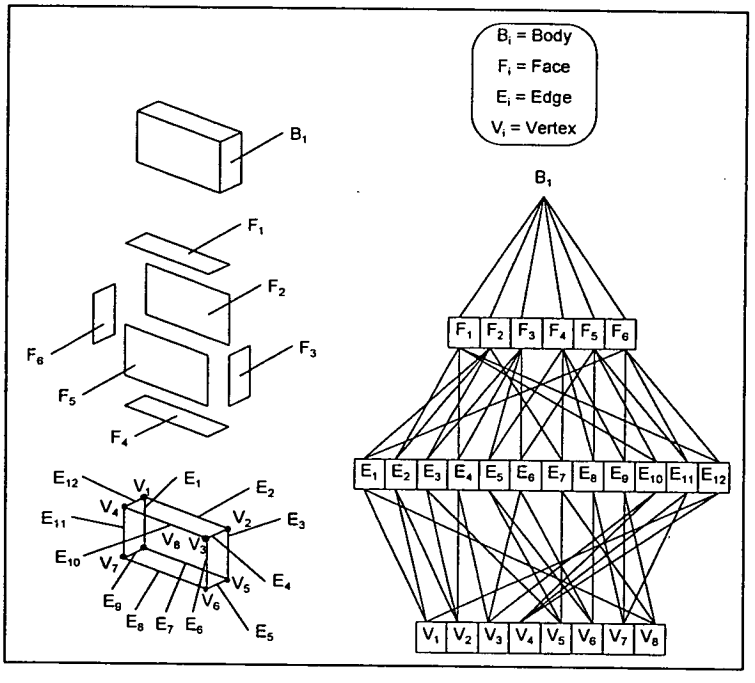


Figure 2.14 Topological Graph Structure

The elements are separated into two categories: topological and geometrical. The topological elements define the *connectivity* or relationships between the faces, edges and vertices, and are supported in a network or graph structure. One common graph structure is the winged-edge representation, Figure 2.15.

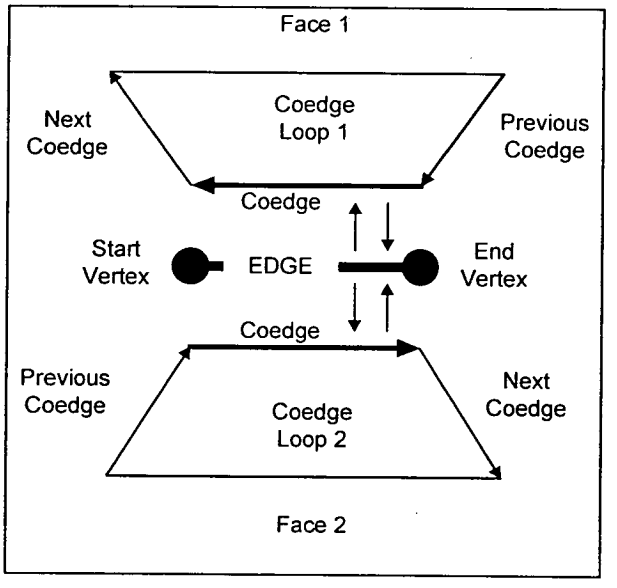


Figure 2.15 Winged Edge Data Structure

The face-edge-vertex graph contains no geometric information about the object. The geometric elements which give the topological elements form and fix them in space are represented separately with pointers from the topological elements to the corresponding geometric representations (surface equations, edge equations, positions, etc.). Such data structures contain redundant information but can be traversed easily allowing the elements of the model to be explored by geometric reasoning algorithms.

Realisable models are called *manifold* objects. A non-manifold object is joined to itself or to other objects along an edge or at a vertex. Such objects are unrealisable. A manifold body obeys the Euler-Poincaré formula [ZEI91] given by equation 2. 1.

$$V - E + F - H = 2(M - G) \qquad (2.\ 1)$$

Where V = number of vertices in object
E = number of edges in object
F = number of faces in object
H = number of interior face loops in object
M = multiplicity of object (number of disjoint bodies)
G = genus of object (number of through holes)

Closed periodic faces such as those that form the cylindrical face of a cylinder or the boundary of a sphere have special representations. A cylindrical face consists of two vertices and three edges: the two circular edges have one vertex each which denote the equivalent start and end points of the edges, the vertices are also connected by an edge. A spherical face consists of a single vertex. Objects such as wire bodies defined by edge chains, and sheet bodies defined by faces are classed as non-manifold. However, such bodies are very useful in the construction of BREP models and recent modellers such as ACIS [STI] allow their inclusion in the data structure and Boolean operations may be performed with them. The edges that connect tangentially connected faces may, or may not, be visualised although they always exist in the graph structure. BREP models may be created in a number of ways: using Euler operations, using Boolean operations, by sweeping, and by tweaking.

In a similar way to CSG modelling with half-spaces, Euler operations [ZEI91] provide the basic elements from which more natural operations can be built. Euler operations allow: edges and vertices to be created, open loops of edges to be closed to form faces, etc.

Further similarity between CSG and BREP modelling is provided by the use of a set of primitive shapes and Boolean operations for combining bodies. Evaluating the resulting body of a Boolean operation involves determining any face-face intersections. Such intersection curves are likely to form additional edges in the new structure. The old structures have to be split at the correct points and along the correct boundaries, the two structures are joined and unnecessary elements deleted. Euler operations are applied to achieve this.

Sweeping is a generic term applied to several kinds of operation: linear sweeps, rotational sweeps (swinging) and generalised sweeps (along a path). The validity of any sweep operation is determined by the complexity of geometry that the modeller will support, generating the topological structure is relatively easy.

In a linear sweep operation a face is swept a distance defined by a vector to produce a solid, Figure 2.16.

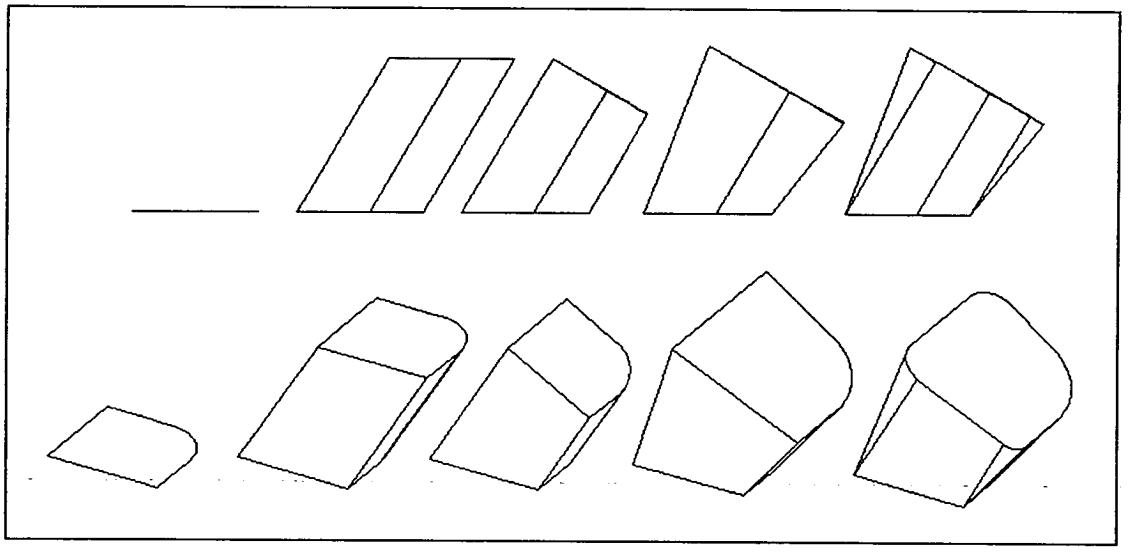


Figure 2.16 Vector Sweep Operations (a) Rigid, (b) Orthogonal, (c) Draft, (d) Draft With Rounded Edges

The way in which the face is swept may be further modified by sweeping rigidly or, for planar faces, orthogonally. In a rigid sweep the closing face is an exact copy of the swept face. However, in an orthogonal sweep the volume is closed by cutting with a planar face orthogonal to the sweep direction. The swept volume may be created with draft, and the edges created in the sweep direction may be rounded.

In a rotational sweep operation planar faces may be swung about an axis, either partially or completely creating a solid of revolution, Figure 2.17. For rotational sweeps of less than 360°, draft may also be applied and the edges modified as with linear sweeps.

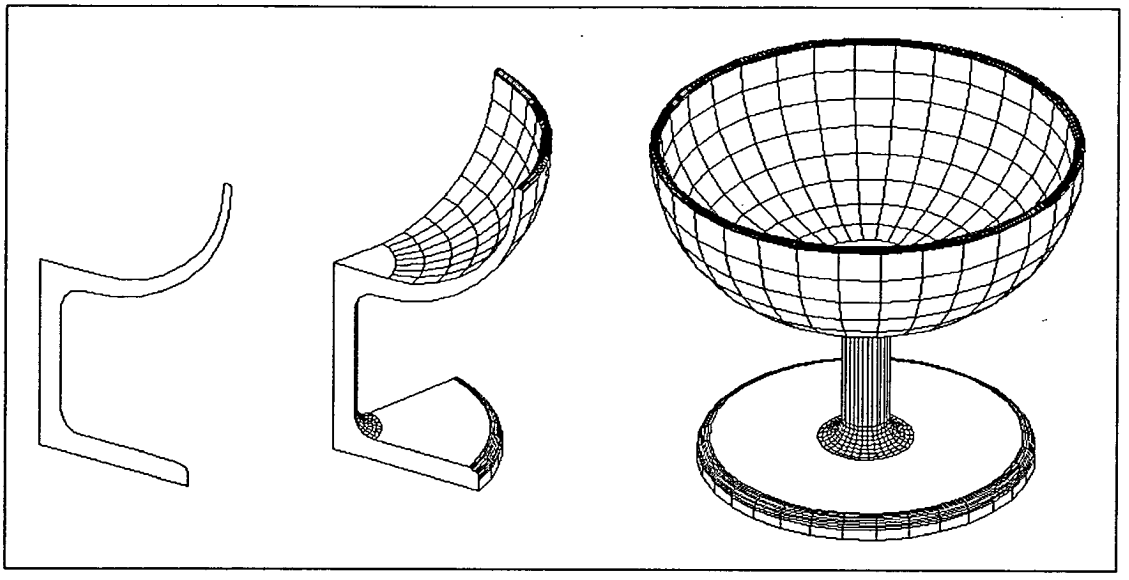


Figure 2.17 Rotational Sweep

Generalised sweeps use another object to define a path that the face is swept along. By using a straight edge a linear sweep may be achieved, Similarly, using a circular edge achieves a rotational sweep. However, very complex shapes may be achieved by using chains of edges. It is not necessary that the path be planar which is effective for modelling pipe systems in single operations, Figure 2.18. The modifiers applicable to linear sweeps may be applied. An additional modification to a generalised sweep is possible whereby the face may be twisted through a specified angle as it is swept along the path. The path must start orthogonally to the face. Complex free form faces are produced in the resulting body.

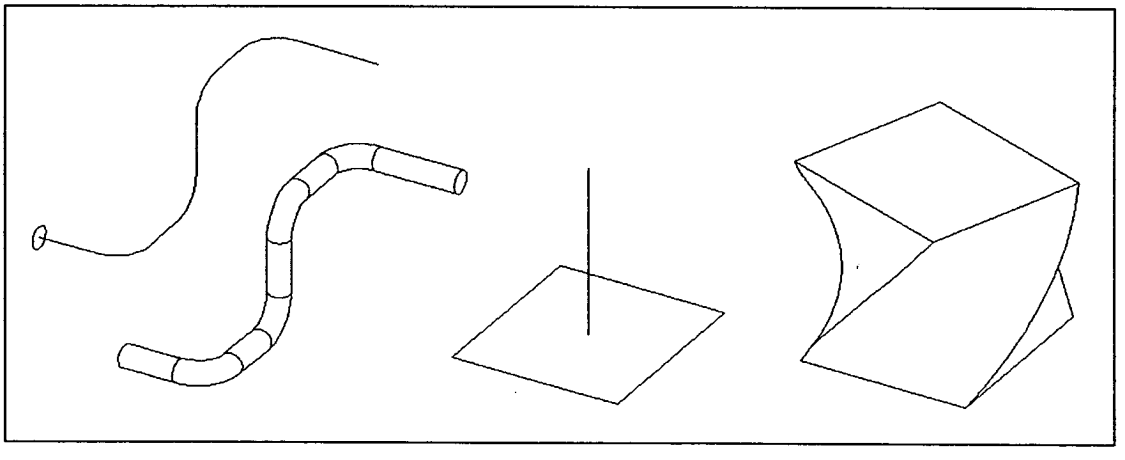


Figure 2.18 Sweep Along A Path

Tweaking is any method by which small adjustments are made to an existing shape. Hence tweaking operations are also known as local operations. Particularly filleting, rounding and chamfering are much more easily achieved in BREP models than in CSG because the elements are immediately available to be modified, Figure 2.19. Existing elements in a model may also be swept in any manner outlined above, locally modifying the model. It is very easy to create a topologically valid but nonsensical model.

In BREP models there is no need for the geometry to be infinite as with the half-spaces required in CSG representations. This means that synthetic geometry of limited extent that defines free form surfaces may be introduced. This gives BREP representations a significant advantage over CSG.

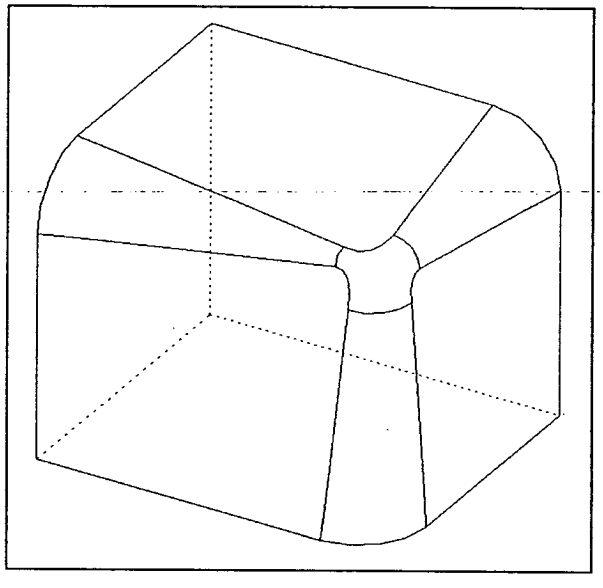


Figure 2.19 Tweaking A BREP Model

In theory there is no reason why free form geometry used by surface modellers should not be represented by BREP solid modellers. However, in practice, the precision with which a solid modeller determines that one point is equal to another is of the order of ten thousand times smaller than with a surface modeller. Directly using data generated by a surface modeller is frequently unsuccessful because the generation of control points can lead to discontinuities. Surfaces defined using splines provide problems in determining the surface to surface intersections

necessary to maintain the detailed topological structure of the model. Providing a stable solid modelling environment which supports free form geometry is currently a priority area of work for developers.

2.3.5.3 Rigid Body Transformations

A *rigid body transformation* allows a new point to be defined by translating, rotating, scaling or reflecting the original point. The rigid body transformation of a point denoted by the position vector, **P**, to a new point denoted by the position vector, **P***, is given in equation 2.2.

$$\mathbf{P}^* = [\mathbf{T}]\,\mathbf{P} \tag{2.2}$$

where [**T**] is the transformation matrix.

Equation 2.2 represents the transformations in homogeneous coordinates. This is so that translation, which is a vector addition in Cartesian coordinates, may be represented as a matrix multiplication [ZEI91].

Therefore in homogeneous coordinates, equation 2.2 may be rewritten as shown in equation 2.3.

$$[x\ y\ z\ 1]^T = \begin{bmatrix} \mathbf{T}_1 & \mathbf{T}_2 \\ \mathbf{T}_3 & 1 \end{bmatrix} \mathbf{P} \tag{2.3}$$

where the 3 × 3 sub-matrix [$\mathbf{T}_1$] produces rotation. The 3 × 1 column matrix [$\mathbf{T}_2$] generates translation. The 3 × 1 row matrix [$\mathbf{T}_3$] generates perspective projection but is set to [0 0 0] in feature modelling systems. The element "1" is the homogeneous coordinate scale factor. Scaling and reflection may also be described by [$\mathbf{T}_1$], but for feature modelling applications such functionality is not necessary. Throughout this thesis the transformation matrix in homogeneous coordinates, [**T**], will be called *transform* for convenience.

Each body within a solid model has its own coordinate system and a transform which locates and orients it with respect to the World Coordinate System. The entities that represent the body are described with respect to the Body Coordinate System. This is so that only the transform need be changed to move all entities and hence the body. Transforms and their inverses are used extensively within the feature modelling system described in chapter 3 to maintain and manipulate workpiece models.

2.3.6 Parametric And Variational Modelling

Solid models of complex objects may take weeks or months of effort to develop. Re-use of existing data in such an environment is of great importance because many new designs are based on modifications to existing ones. Solid models are modified by using the same techniques that were applied to create them. CSG models have primitives either added or subtracted. BREP models have their topology and/or geometry locally modified. Any modifications that necessitate even small changes to the model are generally extremely tedious.

Parametric modellers have been developed as a means to more productive creation and editing of geometric models. A parametrically defined shape is controlled as the name implies by a number of parameters, which express the constraints between elements of the shape, which when solved, determine the geometry. Parametric techniques are particularly suited to the control of contours which are subsequently swept to generate a solid. Such techniques are typically used for the generation of BREP models. Parametric elements have been defined for CSG models. The approach is procedural leading to constrained shapes defined by macros.

In BREP based systems a 2D sketcher is generally used to create some topology and geometry in terms of lines and arcs for example. The fact that there is some topology implicitly defines certain constraints such as coincidence between start and end points of elements, and combined with the geometry present defines other constraints like tangency conditions on the meeting of curved and straight elements.

Dimensional constraints such as distance, radius and angle are added to the structure. The set of constraints is then solved to produce a solution which is visualised as the desired geometric shape. In addition to deleting and adding geometric elements to the parametrically defined shape, the geometry may be changed through the constraints by deleting, creating or assigning new values.

Parametric and variational modelling are words that are frequently used interchangeably, particularly in commercial contexts, and such systems appear outwardly similar. The main difference is in the underlying method used to solve the set of constraints. Parametric models use explicit sequential constraint satisfaction whereas variational models use implicit simultaneous constraint satisfaction. Parametric models can be solved faster but variational models can deal with coupled constraints. For instance, in a parametric modeller suppose a straight line, *b*, is constrained to be parallel to *a*. Subsequently, *b* may only be moved if the parallel constraint is maintained to *a*. However, the orientation of *a* may be changed freely, and the system will update the orientation of *b* to maintain the parallel constraint with *a*. In a variational modeller, a parallel constraint defines that *a* and *b* should be parallel. Therefore, either the orientation of *a* or *b* may be changed, and the system will maintain the parallel constraint of the other element.

Parametric techniques have been applied to 2D drafting systems but today their main application is in commercial BREP solid modelling systems. ProEngineer from PTC [PTC] uses parametric geometry, whereas IDEAS from SDRC [SDRC] uses variational geometry. Feature modelling can be regarded as the next stage of (partial) product model development which utilises the previously described parametric and geometric modelling techniques.

2.4 Feature Modelling Technology

In similarity with the first support of manufacturing by computers for the control of machine tools it is manufacturing's need for more automated generation of NC part programs, rather than an improved method of designing, that has provided the motivation for the development of feature technology. Features provide a means of associating manufacturing processes and strategy with geometry. Feature modelling is a technology for realising product models.

2.4.1 Definition Of A Feature

Most workers in the field of feature modelling technology have attempted to define what is meant by the term *feature*. The meanings that have been presented have reflected strongly the application domains that have been under investigation.

Early definitions focused on the manufacturing domain that features were first used to describe and can be said to define *manufacturing form features* as:

> "specific geometric configuration formed on the surface, edge, or corner of a work piece" [CAMI81]

or,

> "distinctive or characteristic part of a work piece, defining a geometrical shape, which is either specific for a machining process or can be used for fixturing" [ERV88]

As the concept of features spread to design, definitions of *design form features* were proposed by those who practised *design by features*:

> "a geometric form or entity whose availability to a designer as a primitive facilitates the design process" [LUB86]

and those who attempted to define an already existing solid model by recognising the features which may have formed the model considered features as:

> "a set of faces or other geometric entities which together form a pattern useful in part analysis" [HEN91]

The concept of a form feature has been generalised to provide a domain independent description that is applicable to any stage of the life cycle of a product. However, descriptions still concern form:

> "a generic shape which carries some engineering meaning" [WIN91]

and

> "a region of interest" [CAM90]

Feature technology has been widened to include other information that is used in design and manufacturing which is related to, but does not necessarily produce, a geometric shape. Such features are accuracy features for the definition of tolerances, surface conditions and the like. Features may therefore be broadly described as:

> "any entity used in reasoning of design, engineering or manufacturing" [CAM90]

Weber [WEB96] has discussed these further and under the auspices of the FEMEX working group has produced a generalised formal definition:

> "A feature is an information unit (element) representing a region of interest within a product"

Where "region of interest" does not necessarily mean geometry, and a product is something which is realisable.

Features are high level entities which imply engineering meaning and are said to be:

> "a semantic grouping used to describe a part and its assembly" [GIA90]

Even within a single company to have accepted meaning features must be used frequently and therefore can be described as:

> "recurring patterns of information related to a part description" [SHA90]

STEP Part 48 [STEP48] proposes that a form feature is:

> "described by an implicit representation and typically has a number of constituent elements: surfaces, edges, vertices, etc. Implicit representations, by their nature, do not have referenceable *sub-representations* that correspond to the constituent elements of the shape aspects that they represent. This means that one cannot refer to a portion, as distinguished from the whole, of the representation, and hence cannot associate a representational entity with a constituent element of a shape aspect. For example, surface condition on the "base" of a pocket."

This contrasts with recent research which purposefully describes features as consisting of sub-elements which can be referred to [BRU96]. This contradiction is perhaps symptomatic of the stagnation of ISO standardisation efforts for form features. However a form feature is a special representation applicable to a limited class of shape, which usually uses lower level elements from other abstract methods such as geometric modelling.

2.4.2 Feature Recognition

Process planning is the activity in which a representation of the design of a component is assessed with the manufacturing capabilities of an enterprise and mapped into a set of instructions describing how the part will be made. For a component that is able to be made by conventional machining these instructions describe a series of operations, each of which typically has a volume of material to be removed, a metal removal process and a tool are associated. Operations are grouped by machine and by set-up. The operations define a sequence of volume removals

from the stock material to the finished part. Clearly if a model of a component could be analysed and compared to a model of the blank then a sequence of volumes could be defined. If the volumes belonged to a generic type such as a hole, they could be related to manufacturing processes and tool selection. The volumes to be removed are typically defined as *features*, the process of identifying them is known as *feature recognition*, and the process of isolating them is called *feature extraction.*

Kyprianu [KYP80] first applied *syntactic pattern recognition* to a BREP solid model to identify depression features such as slots, holes and pockets, and protrusion features such as bosses and bridges. The sets of features produced were applied to group technology classification (an early approach to CAPP). There have been many workers addressing feature recognition problems using a number of different approaches: *volume decomposition* [WOO82], *rule based* [HEN84], *graph based* [JOS87]. Feature recognition algorithms are generally applied to BREP models. Separation of disconnected machining regions in a CSG model has been reported [SHP94], although the task of dividing the regions into feature sets was not addressed. The approach to feature recognition is often divided into two stages: hint generation, and completion [HAN96].

Feature recognition is accepted as difficult and no completely successful solution has yet emerged [JAR89]. Feature recognition has a number of problems:

- A definition must exist (in whatever form) of the feature that is to be recognised.
- It is impossible to recognise information that is not supported in a solid model, tolerances and relationships for instance.
- Feature recognition does not aid the designer or the design process

2.4.3 Feature Based Design

Interest in features was furthered in the mid eighties by Computer Aided Manufacturing - International (CAM-I) in the USA with their publication of a catalogue of part features for process planning [BUT86]. Also in the mid eighties under the auspices of CAM-I *design by features* was first proposed by Pratt and Wilson [PRA85]. The development of prototype Feature Based Design Systems followed from a number of researchers: Cunningham and Dixon [CUN88], Cutkosky [CUT88] and Shah [SHA88a] in the US and Krause [KRA90] and Rudolf [RUD92] in Europe. The first commercial feature based design system was presented in 1992 by Strässle Informationssysteme AG [STI]. This system has been developed into the system, "FeatureM", which forms the baseline for the developments reported in this thesis. Commercial systems such as ProEngineer [PTC] and IDEAS (SDRC) are architecturally parameterised BREP modellers which offer feature characteristics.

Design by features, frequently referred to as *Feature Based Design*, or, *Feature Oriented Design*, offers the user a library of predefined features from which examples are instantiated in order to build up a model. A feature is defined by a set of parameters which the user can vary to change the size or shape of the feature and consequently (though not necessarily) the model.

Early feature based design systems suffered from limited sets of features which designers felt were too constraining and did not provide sufficient geometry definition functionality. However the following truism was proposed [HUS91]: in order to recognise a feature, a definition must exist, and if a definition does exist, there is no reason why it should not be made available to the designer.

Designing with features offers other advantages over conventional modelling techniques:

- form features are generic shapes with characteristics defined by attributes that define the geometry of a model and have knowledge associated with them that can be used for reasoning.

- features can represent the engineering significance of geometry not only in design but for process planning and manufacturing.

- feature modelling can be used in more stages of the design process because features can represent the function and behaviour of objects used in design.

- features ease design changes because they capture design intent and can be associated with each other.

- features can represent information that is not geometric but is related to the geometry.

- features aid more efficient and effective construction.

- features consist of shape information associated with characteristics, attributes and knowledge which aids downstream applications because reasoning is not performed at a geometric level.

The types of features that it is desirable to represent in a model have been suggested by [SHA88] and [CHU88] as:

- form features

- tolerance features

- assembly features

- functional features

- material features

The features that are used to describe a component are specific to a particular view of that model. For instance both the design and manufacturing form feature models

evaluate to the same result. However, the actual features contained in the model and their ordering will be different.

Consequently feature classification has been application specific. Large catalogues such as the John Deer Catalogue of process planning features [BUT] for machining have been produced. The IMPACCT project [IMP] focused on sheet metal parts for instance. The STEP form feature classification [STEP48] is broadly outlined in Figure 2.20.

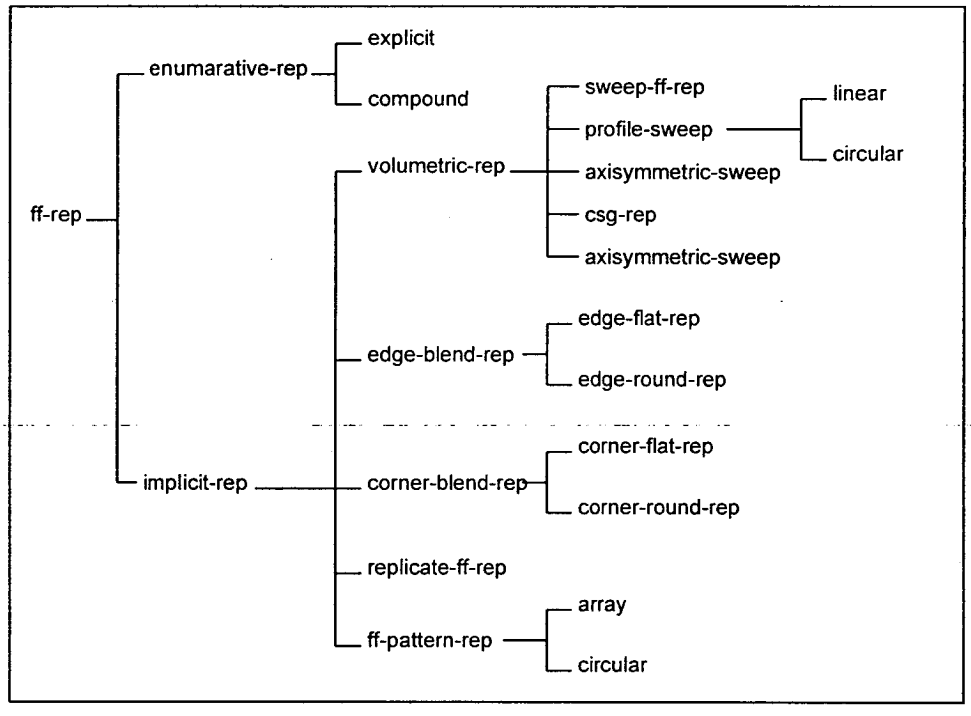


Figure 2.20 STEP Form Feature Classification

A feature consists of a number of possible properties [SHA95] of which the following are fundamentally important:

- generic shape
- dimension parameters
- default values
- location / orientation method and parameters
- relationships to other features
- tolerances

- inheritance rules
- validation rules
- non-geometric attributes

It is immediate to see that whilst the concept of feature modelling is relatively simple the practicalities of implementation are not.

So called hybrid feature modelling architectures proposed by Pratt [PRA88] have become the accepted structure for development of feature modelling systems. The architecture is called hybrid because the feature model data structure is separated from the geometric model which acts as a slave to the feature modelling system, Figure 2.21.

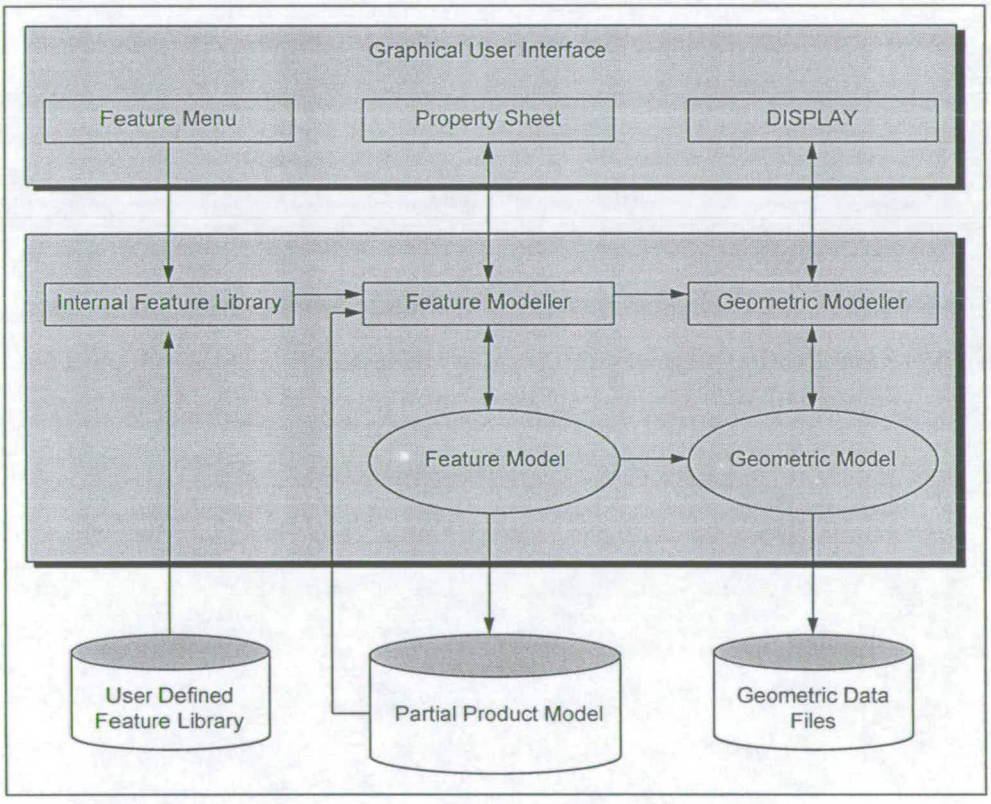


Figure 2.21 Hybrid Feature Model Architecture

Explicit instances of features are created from an implicit definition stored in an internal library. The user explicitly allocates values to the parameter descriptions. The feature model is built up as an ordered list of features instantiated from the

library. If the features represent form then the geometric model will produce a body representing the form of the feature on request from the feature modeller according to the parameters. To produce a resultant model the list of features is combined by the geometric modeller using Boolean operators which are defined for each feature. The list of features is essentially a one sided CSG tree. The features are described by a number of parameters which may be changed at any time. These parameters control the position and size of the feature. After changing one or more parameters the model may be re-evaluated by re-combining each feature in order from the tree. Changing a feature causes the feature modeller to request a new representation from the geometric modeller before combination. When the geometric modeller is a BREP modeller, features may request that local operations be performed on the current state of the model, such as filleting. This requires a mechanism to provide persistent object identity. This is particularly difficult to achieve after editing operations when the entities which constitute a feature are moved or changed. Little support for object identity maintenance is provided by academic systems.

There is no finite set of features in design [SRE91] therefore the concept of *user defined feature* definition has developed. Systems which allow the user to define features in addition to a pre-defined set enable the user to implement features which represent exactly a specific environment. A template is created in which the parameters of the feature and its methods are defined. The definition of the features exist in an external library which is parsed to produce an internal runtime version.

Approaches are described as *procedural* when a feature's methods are defined as a set of procedures, one of which controls the geometric modeller to produce the body used for combination with the rest of the model. Though a number of systems have been proposed which allow the user to define the parameters of the feature in a user defined manner, the geometric representation has been developed as a canned procedure in C++ [KRA90][AUR95]. To be fully user definable requires the development of an interpreted language that can control the feature modeller, the geometric modeller and the user interface. One such language is MCL+ [SIS].

Another approach to feature definition is *declarative*. Declarative methods [SAL95][SHA95][BRU96] use some general algorithm to construct the feature from a description of its elements and the relationships (constraints) between them. Procedural methods are simpler to implement (except the development of the interpreted language), more efficient and are completely unconstrained as to the 3D complexity of the feature. Declarative methods provide better modularity, are potentially more readily transferred between systems, less efficient due to complexity and to date have only been applied to relatively simple 2½D features. Declarative methods potentially offer greater ability than procedural methods for the user to create the new features graphically on screen. However, whichever method is used it has been said that features "know about" their instantiation and dimensioning properties [SHA95].

Furthermore, procedural methods are attractive because rather than selecting a minimum set of dimensional parameters, each dimensional parameter that might reasonably be used in industrial dimensioning practice may be represented. The user indicating which set is required by selecting a particular mode, stored as a further parameter.

The interface provided by feature based design systems to the user can dramatically effect how efficient the user can model geometry with the system. The concept of "handles" [CHA90] has been introduced, Figure 2.22. "Handles" provide characteristic (but not necessarily real) elements of the features as alternative origins for the coordinate system within which the feature is defined. Hence the user is able to efficiently position and orient the feature in the model. The implementation of "handles" will be described in detail in chapter 4.

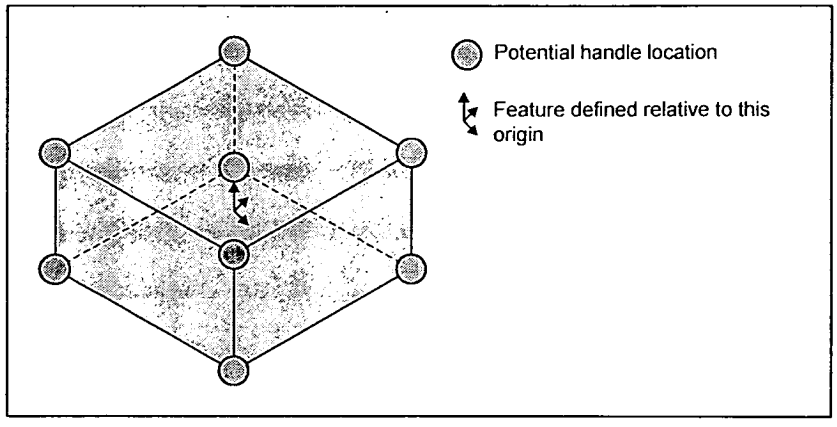


Figure 2.22 Feature "Handles"

2.4.4 Features And CAM

The problems of providing an automatic link between design and manufacturing, particularly process planning, has been a major factor in the lack of success of attempts to develop Computer Integrated Manufacturing (CIM). Features have been seen as a means of forming a link by being able to associate manufacturing processes and strategies with feature definitions. Such features are defined as manufacturing features. However, it is usual that because manufacturing features are related to manufacturing operations they are of a more primitive nature and a different form than the design features the model was efficiently constructed with. Derivation of the manufacturing feature model from the design model has been approached in two ways:

1. By applying feature recognition algorithms to the resultant model. This can be done with or without recourse to the features that were used to design the model. Knowledge of the features that were used to design the model can be used as hints for faster recognition of manufacturing features. Feature recognition techniques have already been reviewed in section 2.4.2. One such system that uses feature recognition as the method for defining a process planning model is PART [HOU91].

2. By defining algorithms that convert design features to manufacturing features. Such techniques are known as *feature mapping*. Different modes of mapping have been defined [SHA93]:

   - Direct mapping (1:1) - one design feature is exactly the same as one manufacturing feature.
   - Direct mapping with parameter changes (1:1) - one design feature generates one manufacturing feature but parameterised differently.
   - Discrete aggregation (n:1) - more than one design features are amalgamated into one manufacturing feature.
   - Discrete decomposition (1:m) - one design feature is split into several manufacturing features.
   - Disjoint mapping (n:m) - two or more design features are amalgamated and split into several manufacturing features.

Direct mapping (1:1) and discrete decomposition (1:m) methods are particularly suitable for use with 2½D volumetric features. The systems reported in [TOE94b] uses semiautomatic planning strategies, whereas [MIL94] uses a fully automatic approach with genetic algorithms as an optimisation method [HUS91].

Disjoint mapping (m:n) is typical of parts defined with free form surfaces or when positive 2½D volumetric features, such as bosses or ribs, have been used in the design process. Disjoint mapping is a non-trivial task. [GOR96] reports an approach to map a positive feature into a set of negative features but only whilst the positive feature is contained within the original form of the blank. Mapping of free from features is related to the prior selection of tools and processes based on the quality requirements defined in the design followed by an assessment of what is able to be cut and with what quality. It is partially very basic feature recognition and an iterative process.

2.5 Dimensions And Tolerances

A dimension is explicitly specified by the designer over some single geometric property of the part, defining perfect or 'nominal' size, position or orientation of the geometric elements. It is accepted that solid modellers (CSG, BREP) provide geometrically complete models of the nominal geometry. Manufacturing methods are incapable of producing parts with perfect geometry. Therefore tolerances are applied to dimensions to denote an allowable variation of the manufactured component from the nominal geometry. Traditionally these are applied to a 2D engineering drawing, as shown in Figure 2.4. The definition and representation of dimensions and tolerances on engineering drawings are governed by the standards detailed in section 2.6. The set of components that are 'in tolerance' form a variational class and a mathematical model of a variational class is known as a variational model. Clearly the tighter the constraints on the variations of a manufactured part the costlier it is to make. Tolerances have the following function:

- To ensure that parts function properly.
- To ensure that mass produced parts are interchangeable.
- To ensure that parts are manufacturable at reasonable cost.
- To ensure that the design is robust.

It has long been recognised that tolerancing information combined with complete and unambiguous representation of geometry is required for fully automatic manufacturing and assembly planning [REQ83]. Indeed, it is necessary to support both complete geometric and manufacturing data in design and to provide this automatically to process planning systems [PED89]. For advanced systems it is the modelling of the interactions between features that will be a prerequisite for progress [MIL93]. Such interactions are either explicitly stated in the model or implicitly evaluated from it [PED95]. Explicitly stated interactions are typically dimensional and geometric tolerances. Interactions implicit in the model are proximity, obstruction and the like. Geometric reasoning techniques have to be applied to

evaluate implicit interactions [MIL94], whereas a data structure and modelling format must be developed for explicit interactions.

The primary task of dimensional tolerances is to limit distance and angle deviations. The Independence Principle [ISO8015] requires that dimensional tolerances and geometric tolerances of form, position and orientation must be measured and verified independently of each other. [ISO406] defines how linear (length, diameter, radius) dimensions and angular dimensions may be toleranced and displayed in a 2D engineering drawing. Linear dimensions have the value followed by either the upper and lower deviations, or, a *limit* or *fit* descriptor. Angular dimensions have the value in degrees followed by the upper and lower deviations and the units should always be indicated (degrees, minutes, etc.).

The inherent inaccuracy of manufacturing methods, coupled with the fact that parts produced which are not "exactly" to size can function correctly, indicates that an allowable deviation, or *limit*, from nominal is acceptable. Where two parts mate to provide function they are said to *fit* together. The functionality required can be described by the type of *fit*: clearance, interference, transition. A *tolerance grade* is used to denote a level of accuracy for all basic sizes. The value of the deviations from nominal, or upper and lower *limits*, are dependent on the nominal size and tolerance grade of the shape aspect considered. A standard [ISO286] describes the system of *limits* and *fits* which provide a set of tolerances and deviations that will provide the desired mating functionality between two cylindrical surface areas or between two parallel planes. The nomenclature is presented but no indication of which fit is suitable for which function is given. In contrast to the Independence Principle the Taylor Principle, which is often applied to *limits* and *fits,* can also constrain certain geometric tolerances: straightness, flatness, roundness, cylindricity and parallelism.

Statistical tolerances [HEN95] are applied especially to tolerance chains. Worst case scenarios are unlikely to occur for all member dimensions of the chain, so they may be statistically relaxed depending on manufacturing process. A plus-minus tolerance

is defined for the boundaries of the tolerance, a central zone is specified and a percentage of parts that must fall within this zone.

Purely dimensional tolerances are insufficient to fully define the form, attitude, location and runout of the geometric elements of a part. This is partly due to the ambiguity (that the data may be interpreted differently) of 2D drawings and partly due to the limited nature of the dimensional tolerances. By defining a different class of tolerances - geometric tolerances - of form, attitude, location and runout, the relationship between size and form (attitude or location) may be decoupled, yet the desired shape is more accurately specified. Geometric tolerances according to [ISO1101] are given in Figure 2.23.

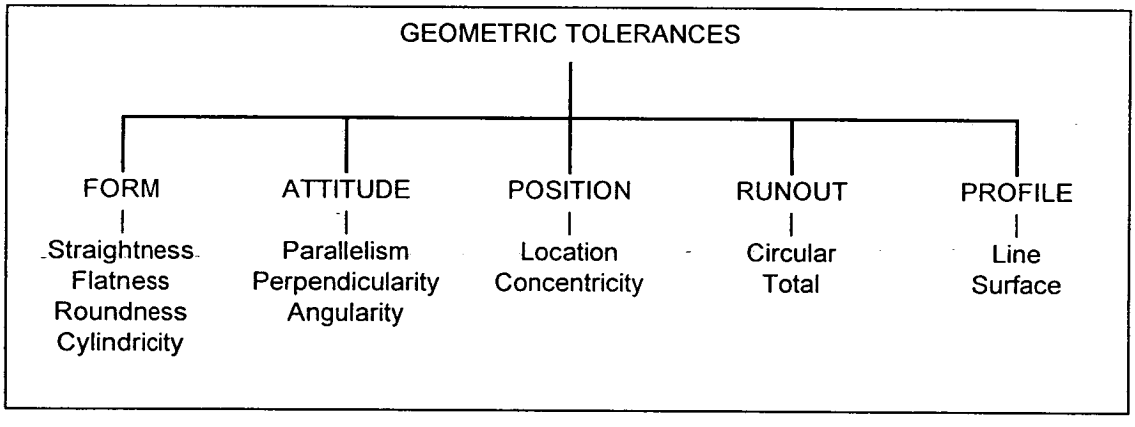


Figure 2.23 ISO Geometric Tolerance Classification

Vectorial dimensioning and tolerancing differs from conventional tolerancing because the dimensions and tolerances are specified on or between *substitute elements* rather than the actual elements. *Substitute elements* are imaginary geometrically ideal shape aspects (line, circle, plane, cylinder, sphere, etc.) calculated from the assessed points on a component's surfaces. In general, substitute elements are calculated by coordinate measuring machines. Substitute elements are defined by vectors (location, orientation) and size. To date there is no standard though the system is described in [HEN95]. The system would appear more suitable for representation in 3D geometric modelling systems because it consists of idealised elements, vectors and sizes.

Tolerance analysis techniques have been discussed that do not use a geometric modeller. In [ZHA91] linear tolerance chains are analysed for optimum cutting sequences in CAPP. In [WEI88] chains of designed dimensions are compared to the proposed measured dimensions. The effects of manufacturing tolerances such as fixturing and tool positioning are included. In [FLE85] relationships between parts were discussed and in [FLE86] tolerances were represented as networks of relationships. These representations do not meet the needs of a highly integrated and automated environment because they are not associated with the geometric model of the component.

Modelling dimensions and tolerances with 3D solid and feature modellers has been widely reported [JUS92] [TAI] and follows a number of approaches:

1. Simple textual attributes attached to a solid model.

2. Incorporated in the solid model.

3. Incorporated in form feature definitions.

4. Supported in an independent model associated with a feature and/or solid model.

Tolerances supported as textual attributes may only successfully represent the default tolerances applied to the whole component. This is clearly not sufficient for a complete description.

Techniques for incorporating tolerances into a solid model have used several approaches:

1. Variational Dimensional Model

   Direct parameterisation leads to the development of part families. Indirect parameterisation is based on the variational or parametric modelling architectures (depending on how the implicit dimensions are determined). They are generally used for 2D sketches and the tolerances are applied to the constraints. The constraints manager solving the set of tolerances. The approach is suited to

sensitivity analysis but to date has not been widely applied to 3D structures, although its application to polyhedrons has been reported [HIL78]. Tolerances of form (because the representation deals with points), position and orientation (because the there is no concept of datum and target in the constraints) cannot be supported.

2. Offset Zone Model.

The Offset Zone Model is reported in [REQ83]. The tolerance zone is created by the Boolean set difference between the maximal and minimal object volumes, which are created by offsetting the boundaries of the object. The objects boundaries can lie anywhere in the zone for a valid shape which makes the approach applicable to non-perfect geometry unlike the Variational Dimension Model. The Offset Zone Model combines the effects of all tolerances (size, attitude, form, etc.) into one tolerance which is in conflict with current practice. Topological changes caused during offsetting of the boundaries can cause problems.

3. Variational surface model

This approach extends the concept of the variational model to the 3D domain by controlling surfaces by their characteristics rather than the points that define their boundaries [MAR93]. This approach may be regarded as *vectorial tolerancing.* Surfaces may be non planar. For example a cylindrical surface may be defined by a point vector, a unit vector defining the direction of the axis and the size parameters of diameter and height. These are allowed to vary and the positions of vertices and edges computed from the result. Form tolerances can be supported by using higher degree surfaces. However, moving surfaces can lead to topological problems. [GUP91] replaces edges and vertices with imaginary faces which are used to trim the real face objects, sometimes leading to new topological elements. It is unclear how resolved or derived entities (axes, intersection points, planes of symmetry, etc.) would be treated. There is no form feature support or consideration for a 3D graphical representation.

Form feature descriptions appear to provide an intuitive structure for representing tolerances. Tolerances may be added as extra parameters to form features and can be associated with the size and position parameters. Geometric tolerances that are local to the form feature may also be supported such as roundness of a hole. An intelligent use of handles and reference features when building a feature model allows datum to be made implicit in the definition, however this is not always the natural way to design the model. It is inefficient when default tolerances are applied to a part and only those dimensions that are critical have to be especially toleranced. A most significant problem is that shape aspects in the resultant model that the designer wishes to tolerance may not be represented by a single form feature's parameters, nor may inter-feature relationships be sufficiently well represented. Simply adding a plus-minus tolerance attribute to the feature parameter is insufficient for visualisation in 3D.

The most widely investigated approach is that of developing an independent dimension and tolerance model. Pointers from the dimension and tolerance model structure tag elements in the geometric model structure. If the geometric elements are also related to the form features which created them, the dimensions and tolerances can be (to a lesser or greater degree) related to the features. Datum reference frames may be derived form the geometric elements, therefore, both intra feature and inter feature dimensions and tolerances may be represented. Various independent dimension and tolerance models related to geometric and feature models have been developed.

Work by CAM-I [RAN88] has centred around the development of an evaluated dimension and tolerance (EDT) model. A BREP solid modelling system is accessed via an application interface specification (AIS). A hierarchy is proposed whereby the lowest level geometric features are primitive face templates used to construct the BREP model. At the next level of extraction, or application layer, design and manufacturing features exist, having been previously constructed from the low level face feature templates. Further, higher level layers may exist such as group technology. Datum, dimensions and tolerances constructed in a separate model may

be assigned to point to these face feature primitives. There are obvious limitations in this implementation. Only planes and right-circular cylinder geometric surface types may be used. The BREP model must exist before the dimensions and tolerances are applied. It is unclear if the dimensions and tolerances can be associated with the high level design features parameters and whether these are user definable. No visual representation of the dimensions and tolerances is reported.

One of the most significant efforts is the development of the STEP Shape Variational Tolerance Model [STEP47] (described in section 2.6). This is very similar to the EDT model. Implementing [STEP47] as an ACIS Husk has been proposed [SIM95], but this assumes persistent object identity. It is unclear how the toleranced features (shape aspects) are generated and what would happen to the model after modifications to the underlying geometric model. It is certainly not integrated with the control mechanisms of a feature based design system used to generate the solid model to which it is applied.

The EDT approach is not applicable to a CSG modelling environment. This problem has been partially addressed by the construction of variational graphs[REQ86]. Face elements in the graph are associated with the faces of the primitives used to construct the model, and hence to the actual faces in the model. Tolerances must be attached to faces in the primitives before combination in the model. However, only one variational graph is used for the model and it is updated incrementally as changes are made to the model. This resolves the problem of persistent object identity. The representation is not feature based and is restricted to predefined volumetric primitives. Visualisation aspects are not considered.

Taiber [TAI1] reports a system that maps the contents of conventional engineering drawings into a rule based graph theoretic model. The tolerances are applied after definition of a solid model either by conventional or feature based techniques. The geometric elements of the model are used to define, or derive the shape aspects and datum required for geometric tolerance definition. The system performs tolerance chain analysis. What happens to the representation if the geometric model is changed

is not discussed. The tolerances are not related to the feature control mechanisms, nor are visualisation aspects discussed.

A hybrid form feature modeller consisting of a combined CSG/BREP architecture which allows the addition of dimensions and tolerances has been described [ROY88]. The form feature model structure is maintained in a CSG tree. Each form feature is represented simultaneously with a *face adjacency graph* (FAG), a form of BREP. The addition of new form features to the form feature tree generates a new FAG and the resultant *Structured FAG* (SFAG) is updated. The root of the SFAG represents the resultant model, each node a FAG. A *spatial relationship graph* (SRG) maintains the spatial relationship between the form features at each level in the tree using transformation matrices. Variations in size and spatial relationships is limited to plus-minus tolerances. Dimensions and tolerances may be attached to the individual FAGs in which case they are inherited by the resulting SFAG. Alternatively dimensions and tolerances may be attached to the SFAG. Datum reference frames may also be constructed. The system is limited to representing form features constructed from CSG primitives. No visualisation is considered.

[SAL95] describes a feature modelling system that uses a torsor method for describing assembly tolerances between rigid bodies. The tolerances are presented as graphs of related elements. No 3D engineering drawing representation was associated with the solid assembly model. [ROS96] investigates the use of NURBS surfaces to replace the nominal faces of form features to represent manufacturing uncertainties. The imperfect form feature model is then used for functional assessment of assemblies.

Simple textual attributes are clearly insufficient to meet the needs of complete and accurate product models. Combining the information purely in the solid model is too restricting and does not consider feature technology. Supporting the information as feature attributes is often ambiguous and incomplete. Independent structures allow use of both geometric and form feature data representations and appear to offer the entities necessary for complete descriptions and provide the best route to progress.

Modelling techniques are numerous and it is clear that no completely successful solution has emerged so far [JUS91]. Currently deficiencies exist with respect to 3D visualisation of dimensions and tolerances and linking them to the control structures and parameters of features, especially user defined, in order to use them to change the model. In chapter 5 a method is proposed that aims to address these deficiencies.

2.6 Relevant Standards

2.6.1 Product Modelling

STEP, the Standard for the Exchange of Product Model Data [STEP], is not a simple data exchange interface, but aims at the development of a reference model that includes all information concerning the life cycle of a product. Although STEP began as a European development, it has now been combined with parallel activities in the US [PDES] into an international standard consisting of a series of parts.

The parts of the STEP standard are divided into three categories:

1. Generic Resources

   Generic resources define the basic elements which are used to define and represent the objects which are used to build the product model, define relationships between objects in the model and manage the model. Fundamental to STEP is the data modelling language defined in Part 11, EXPRESS [STEP11], which is used to as the syntactical base for defining all other parts of the STEP standards. EXPRESS is an object oriented language. Generic resources are independent of application area. Other important generic resources of significance to the objectives of this thesis concern product description [STEP41], geometric modelling [STEP42], dimensions and tolerances [STEP47] and form features [STEP48].

2. Application Resources

Application resources provide product information which is application specific. For instance drafting is specific to mechanical engineering, electrotechnical engineering, ship building, etc.

3. Application Protocols

Application protocols include parts of the generic and application resources, defining the use of partial models for specific applications. For instance the exchange of engineering drawings is dependent on the explicit representation of geometry and dimensions.

2.6.2 Geometric Modelling

To date IGES [IGES] is the most commonly used and generally applicable standard for the exchange of geometric models. IGES can exchange 2D engineering drawings, 3D surface and 3D solid models, but suffers problems with large, complex free form models because of very large file sizes generated. SET [SET] can be seen as a development of IGES that solves the problems of large models, however its usage is limited to the European aerospace industry. VDA-FS [VDA] has been developed within the European automotive industry for the exchange of surface models only. It has gained favour because of its stability. The development of STEP was begun approximately ten years ago partly with the aim of providing a single representation for the exchange of geometric and associated data without the problems of stability, model size, speciality and localisation. The objective being to provide a single world wide standard.

STEP part 21: Clear Text Encoding Of The Exchange Structure [STEP21] defines the physical file structure for STEP compliant representations. Part 42: Geometric and Topologic Representation [STEP42] defines all necessary geometric and

topologic elements needed to describe a geometric model, whether 2D or 3D, surface or solid.

Although the necessary parts of the STEP standard have been available since 1994 in practice it is rarely used as an exchange format.

2.6.3 Feature Modelling

Part 48 of the STEP standard [STEP48] is concerned with the representation of form features. The standard is not yet normative. Within the context of this standard a form feature is defined as being a shape which conforms to some preconceived pattern or stereotype and is, for the purposes of some application, usefully dealt with as an occurrence of that stereotype. The standard only concerns form features which are

- generic shape properties of a product, that have no application connotation and no presumptions as to representation, or
- representations employed in shape modelling to represent shape properties (geometry).

Shape feature models that characterise information may exist without a geometric representation, similarly a geometric feature model may exist without being characterised. However, the two are mutually supportive.

Form features are characterised into three categories:

1. volume feature representing an added/subtracted volume applied to some pre-existing shape.
2. transition feature separates or blends the surfaces of a shape.
3. pattern feature represents a set of similar form features occurring in a recognised arrangement.

Additionally, a form feature element is defined as a positional relationship between a form feature and a shape aspect. For example, a planar face (shape aspect) can be said to be the "bottom of" a subtractive volume feature typically described as a pocket.

The form features classified by STEP have been presented in Figure 2.20 The representation does allow new features by combining existing features. Previously defined geometry may be integrated with the feature model. The scope of the standard is relatively narrow. Association of non-shape information with shape information is not supported. Shape tolerance information or representations are not supported. User defined form features are not supported. General capability to define inter-feature relationships is not supported. Furthermore, shape characterisation based on functionality, product class, manufacturing methods, and size is not considered. Also, general geometric representations such as boundary representations and b-spline surfaces are outwith the scope of the standard. Sweeps must be of planar profiles.

The representation methods presented and terminology are in contrast to the process planning features defined by Part 244 [STEP244]. Part 244 is an Application Protocol developed for mechanical product definition for CAPP using form features, whereas Part 48 is a Generic Resource and should therefore provide a domain independent definition. Part 48 does not prescribe any unique parameterisation in contrast to common features such as holes, slots and pockets which are uniquely defined in Part 224. Features with the same name are sometimes defined differently in Parts 48 and 224. For instance a *transition* feature may only be applied to an edge in Part 224, whereas it may be defined between surfaces in Part 48. The scope and limitations of the draft standard, Part 48, do not meet the needs of the objectives of this thesis.

2.6.4 Dimensioning And Tolerancing

The International Standards governing the definition and representation of dimensions and tolerances are produced and maintained by the International Organisation for Standardization [ISO]. These standards are the result of integration and further development of National Standards, such as the American ANSI Y14.5: Dimensioning and tolerancing [ANSI145], and the British BS308 Parts 1, 2 and 3 Engineering Drawing Practice [BSI308]. It is a migratory process by which national standards will become superseded by the international standards. The national standards to date have been concerned with the definition and representation of dimensions and tolerances on 2D engineering drawings. The international standards under the auspices of the STEP programme take the national standards to a higher level of complexity by introducing the definition and representation of dimensions and tolerances that may be related to 3D solid models [STEP47]. However, this definition and representation does not concern aspects of visualisation or of relating the definitions to the parameter descriptions of the feature definitions in [STEP48]. As has been stated [STEP48] does not include consideration for User Defined Features therefore the current standards do not fully meet the needs of the objectives of this thesis.

However, the requirements:

1. to represent dimensions and tolerances in 3D associated with the solid feature model and,

2. that the visual representation, when viewed from the correct direction, should produce a 2D engineering drawing view,

necessitates that, when displayed in the engineering view of the 3D model the dimension and tolerance representation complies with the standards for 2D engineering drawings. The standards which concern the definition and representation of tolerances on 2D engineering drawings are as follows:

- ISO 129:1985 Technical Drawings - Dimensioning - General principles, definitions, methods of execution and special indications [ISO129].

- ISO 8015:1985 Technical Drawings - Fundamental Tolerancing Principle [ISO8015].

- ISO 286-1 1988 ISO system of limits and fits [ISO286].

- ISO 406:1987 Technical Drawings - Tolerancing of linear and angular dimensions [ISO406].

Further standards describe in more detail geometric tolerancing: [ISO1101], [ISO5459], [ISO2692].

The only standard concerned with the definition and representation of dimensions and tolerances that may be associated with both 2D and 3D geometric models is:

- ISO 10303-47 Industrial Automation Systems - Product Data Representation And Exchange - Part 47: Integrated Generic Resources: Shape Variational Tolerances [STEP47].

ISO 10303-47 describes the definition and representation of dimensions and tolerances which may be related to a 3D solid model by providing a syntax to support the classes of dimensions and tolerances that are applied to 2D engineering drawings (as specified in the above standards). The shape of product may, but need not have a geometric representation. In this way the dimension and tolerance model is defined separately to the geometric model.

The standard provides a set resources in order to describe dimensions and tolerances which may be divided into three sections:

1. Shape aspect definition resources define a set of shapes that are required for applying dimensions and tolerances. In addition to shape aspects and shape aspect relationships which are defined in [STEP41], derived shape aspects and datum may be defined. Shape aspects representations are formed from the product shape

or boundaries of a solid model (they are extracted from a solid model, but are not used to create it). Derived shape aspects, which though derived from the model, are not required to define its shape, nor necessarily lie on the boundary of the model. Datum are shape aspects that provide the origins to which dimensions and tolerances are referenced.

2. Shape dimension resources that provide representations of size and relative location to meet the requirements of engineering design. Dimensions are implicitly defined in a 3D geometric model and the values are derived from the shape aspects, whereas dimensions are explicitly presented with a 2D engineering drawing. The standard defines methods for representing size and location dimensions. The measure of a dimension is derived from the geometric representation of the shape aspects. If no geometric representation exists the measure of a dimension may have a value assigned therefore defining the size of the shape aspect. Location dimensions define a relationship between the origin shape aspect or datum to the target shape aspect. This representation is non-directed. Three types of measurement path for applying a dimension are provided: linear, curved and angular.

3. Shape tolerance resources that provide definitions of limits within which manufactured shapes are permitted to vary. Two types of tolerance are represented: plus-minus and geometrical. Plus-minus tolerances may be defined in two ways: specification of upper and lower limits, specification of limits and fits found in [ISO286]. Statistical tolerances provide an additional method of specifying the allowable variation of a dimension and are represented by associating them with the direct specification of the upper and lower limits of the plus-minus tolerance. Geometric tolerances provide the structures for applying tolerance zones to shape aspects. Tolerances of form, orientation, location, profile and runout of a shape aspect are supported.

The dimension and tolerance data model is further described in [FEN], where a simple 2D example is given.

The object of the standard is to provide a computer interpretable neutral representation that is independent of any particular system but allows exchange of data between differing systems. Whilst not constraining any internal system implementation of dimensions and tolerances the concepts and definitions stated in the standard must be considered from the aspect of information content.

The STEP tolerance model does not consider the validity or completeness of tolerance data; nor does it concern itself with the practical problem of relating the tolerance information to the underlying geometric model (if one exists). Furthermore the STEP model does not propose any methods or rules for graphical visualisation in a 3D modelling space. A further and most significant deficiency is that there is no method inherent in the tolerance model to relate the dimensions to feature parameters which enables the dimensioning scheme to control the shape of the model. Consideration for surface conditions such as roughness, heat treatments, and coatings are included in other application resources. The STEP tolerance model is not sufficiently developed particularly for user defined feature developments.

2.7 Chapter Summary

Chapter 2 has summarised the needs for computer support relating to design, production planning and manufacturing due to recent economic and social trends and strategies. Geometric modelling techniques have been presented as fundamental to providing complete and accurate descriptions of parts. Feature modelling has been described as a technology that directly aids strategies such as concurrent engineering. Dimensions and tolerances are used by designers to communicate to planners and manufacturing engineers the scope of manufacturing deviations allowable from nominal geometry. Their use is governed by international standards which have been detailed. Approaches by other workers taken to integrate dimensions and tolerances in 3D geometric and feature models have been reported.

The next chapter, chapter 3, presents the architecture of the feature modelling system, FeatureM, which is used as the baseline for developments to provide less

constrained geometric feature representations (chapter 4) and the integration of dimension and tolerances (chapter 5).

Chapter 3

Feature Modelling System

3.1 Introduction

3.1.1 Chapter Overview

Chapter 1 introduced feature modelling as a promising software technology particularly focused at modelling the form of mechanical components. The baseline for the work reported in this thesis was also introduced in chapter 1, and consists of the Feature Based Mechanical Design System, FeatureM, from Strässle Informationssysteme AG [SIS]. Chapter 2 went on to describe feature modelling methods in general and the solid modelling techniques that are used in feature modelling systems to represent the form of objects. In this chapter (chapter 3) the architecture of FeatureM will be presented. The architecture, design and functionality of FeatureM are described in order to help the understanding of the developments which form the work reported in chapters 4 and 5 of this thesis related to feature definition and the representation of dimensions and tolerances. The author has made only minor contributions to the system as is reported in this chapter.

3.1.2 Chapter Structure

Section 3.1 provides an overview of this chapter (chapter 3) relating the contents to the other chapters of the thesis, and describes the chapter structure. Section 3.2 presents the architecture of the feature modelling system, FeatureM, and details the sub-components. The sub-components are described in further sections of this chapter. Section 3.3 presents the geometric modelling kernel ACIS. Section 3.4 briefly describes the interpreted language MCL+. The graphical user interface is presented in section 3.5. Application modules are briefly described in section 3.6. In section 3.7 the feature modelling module is detailed because it is fundamental to the developments relating to feature definition and dimension and tolerance representation reported in chapters 4 and 5.

3.2 System Architecture

The feature modelling system uses a number of standard software components: ACIS is used as the geometry engine with X Windows, OSF Motif and HOOPS generating the graphical user interface. The feature modelling system is currently supported on UNIX workstations.

The modeller is a combined CSG/BREP or hybrid feature modeller (as described in chapter 2) producing an ACIS BREP solid model as the result. It is a general purpose feature modeller which may represent information other than geometry. The user interface to the model is via buttons and dialogue boxes. In addition, all functionality available graphically may be programmed by the user. Visualisation of the model can be achieved in wire frame, faceted or fully shaded modes, with or without display of hidden lines. Multiple views of a model may be seen simultaneously. Each view is a projection (axionometric, single/double point perspective) of the 3D model which may be rotated interactively. A property sheet is used to present and allow modification of all feature parameters. A view of the FeatureM screen showing multiple views and the property sheet is shown in Figure 3.1. The architecture of the

modeller has been established to aid the development of User Defined Features. The way in which User Defined Features are defined will be described in chapter 4.

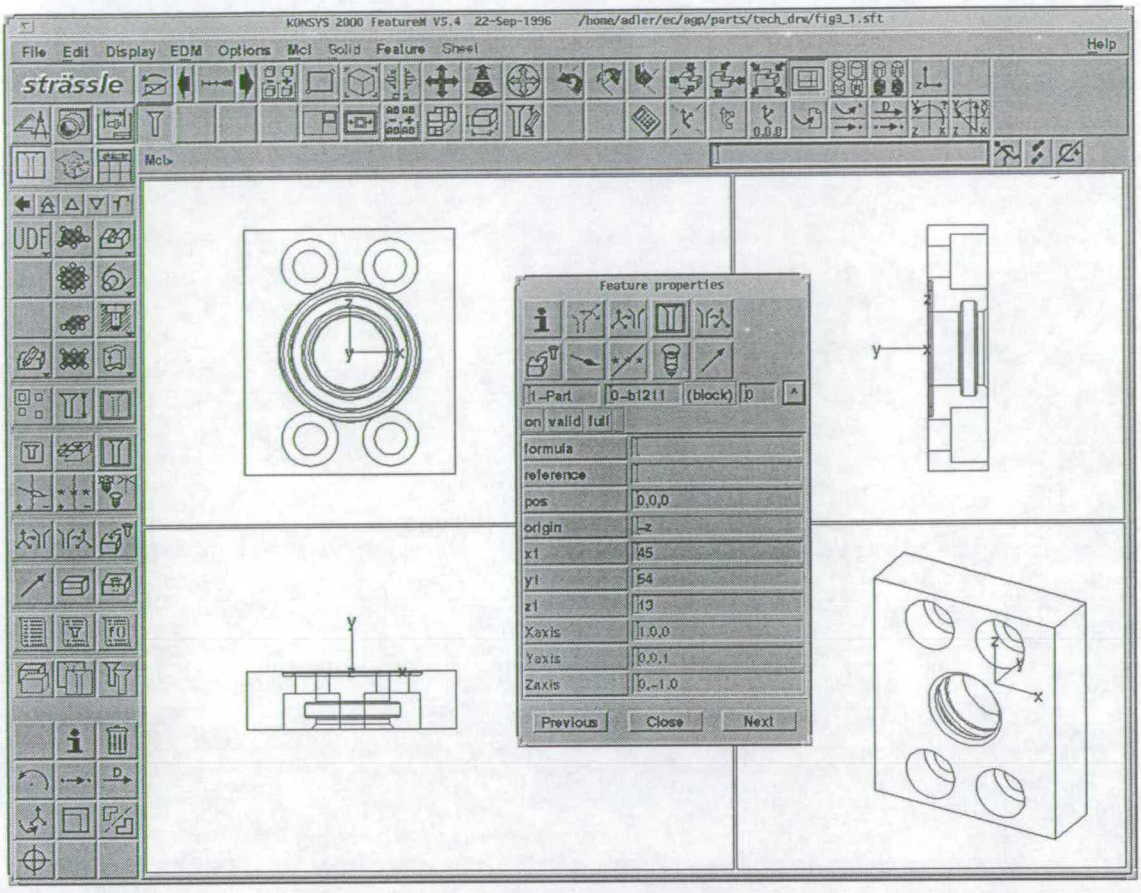


Figure 3.1 FeatureM Screen

The system architecture is shown in Figure 3.2. It consists of five main software components which communicate as indicated. The system reads externally defined feature descriptions from a library and is able to save and restore complete feature models to and from files.

The five major software components are:

1. ACIS

   ACIS is a geometry engine marketed by Spatial Technology Incorporated [STI] and is used by the feature modelling kernel to produce the geometry of individual form features and to combine them to produce the resultant solid model. ACIS is

an Object Oriented Open Architecture BREP solids modeller accessed through a well defined Application Procedural Interface.

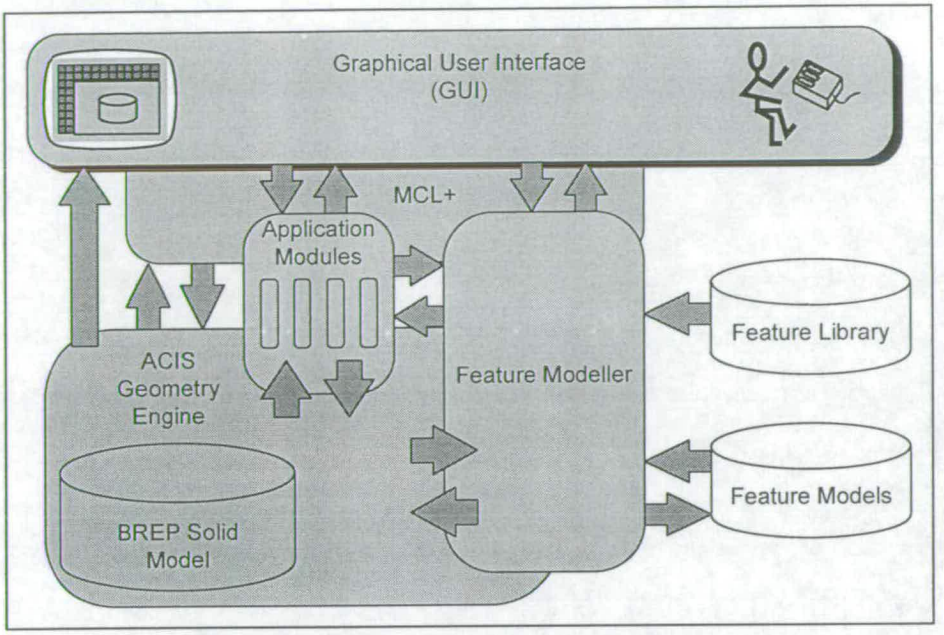


Figure 3.2 FeatureM System Architecture

2. MCL+

   This is an interpreted language developed by Strässle [SIS95a] that allows the user to control the complete system by programming. It forms the syntactical base for the definition of the feature libraries. It provides a link between the high level description of the GUI and the low level commands that form the system functions. The user can make direct calls to ACIS through MCL+.

3. Graphical User Interface (GUI)

   The GUI provides the interface between the user and the feature modelling system. The Graphical User Interface (GUI) is concerned with how the user sees and works with the system. This GUI controls screen design and layout, the management of user interaction, and, the display and visualisation of the model.

4. Application Modules

   These are modules that can:

   - Extend the functionality of the feature modeller: User Defined Commands.

- Provide functionality to aid the feature modeller: 2D Contour Sketching.
- Provide basic stand alone functionality that is also used by the feature modeller: Volume Modelling.
- Provide functionality that is independent of the feature modeller: 3D Sheet Metal Modelling.

5. Feature Modeller

The feature modeller consists of kernel functionality and template administration. The kernel provides the data model and management structures. Feature template administration is concerned with making an internal representation of the features defined in the feature libraries and linking the calls defined in the libraries to the procedures defined externally to the system. The kernel has a built in interface to the geometry engine, ACIS, through its API. The library of features is defined externally in a file. This is necessary for user definition.

The system level programming language is C++ which is functionally compatible with C (which provides many of the pre-compiled libraries).

The five software components will be described in sections 3.3 to 3.7 of this chapter.

3.3 ACIS

ACIS is an object oriented geometric modelling toolkit designed for use as a geometry engine within 3D modelling applications. Written in C++, ACIS provides an open architecture framework for wireframe, surface and solid modelling with a common unified data structure. Linear and quadratic geometry is represented analytically whilst free form synthetic geometry is represented by non-uniform rational B-splines (NURBS). ACIS supports manifold and non-manifold topology, as well as bounded, semi-bounded, and unbounded geometry. ACIS represents solids using a BREP data structure, providing CSG type Boolean operations on solid, sheet and wire bodies, and local operations directly on the BREP model. ACIS

communicates with application programmes through a well defined *Application Procedural Interface (API)*, described in [STI].

ACIS uses a boundary representation as the basis for the topologic and geometric classes used to build ACIS models. In addition, ACIS entities may have a list of attributes attached. The entities are arranged in a hierarchy of classes which allow for extension with additions made by application developers, as shown in Figure 3.3.

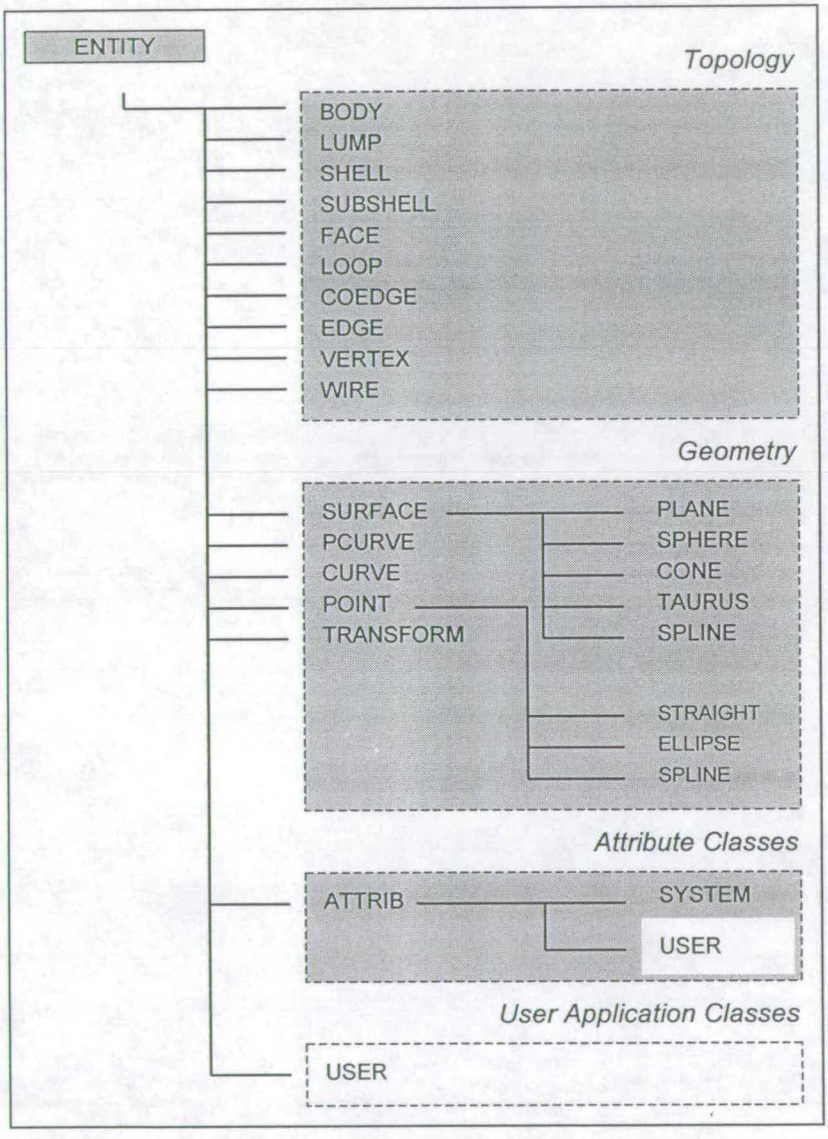


Figure 3.3 ACIS Class Structure (Source: [STI])

This is a very powerful concept because it allows the application developer to create new, specialised objects in the data model and to attach any information to any entity

that exists. The ACIS model then becomes a much more general data model particularly suited to development of feature modelling systems and enabling better integration between design and manufacturing. Such models are often referred to as product models.

ACIS entities are of four general types: topologic, geometric, attribute, and user defined. All application classes and attributes are children of ACIS entities. Hence, the model management mechanisms that perform the save, restore and roll-back / roll-forward functionality will work for application classes as well. Roll-back and roll-forward is a mechanism provided by ACIS whereby all calls the modeller and its data structure are stored in a tree like representation. The modeller may step back through each state to a desired point. Further modelling may then proceed creating a new branch. The history remains and may be returned to. This is a useful mechanism not only for the "undo" procedure, but it could be imagined that different design scenarios may be developed in the same model and explored. Such possibilities are an aid to simultaneous engineering goals.

ACIS allows wire frame entities to coexist with solid and sheet bodies, sharing edges, coedges, and vertices. This coexistence enables mixed dimensionality models to be constructed. This is useful in the development of sweep features where the contour and path exist but have not yet generated a volume. Non closed models can also be created, such as a plane with three bounding edges and one unbounded, infinite direction. Non closed sweep features provide powerful ways of creating complex free form geometry.

It is clear to see that the geometry is separated from the topology. Topology represents the spatial relationships between the various geometric elements. Each body has a transform. The transform contains the complete object space transformations for the entire body: rotation, translation, scaling, reflection and shear. In feature modelling systems it is the rotation and translation components that are of interest. It the use of the transform and its inverse that allows easy conversion from

one coordinate system to another; from feature coordinates to world coordinates for example.

3.4 MCL+

MCL+ has been developed by Strässle Informationssysteme AG [SIS95a]. MCL+ stands for Macro Command Language (MCL) with Object Oriented characteristics (+) that are derived from the C++ language with which it has been developed. It is an interpreted language which requires no compiling or linking to produce executable code. This means that code that the user develops may be immediately executed.

A great advantage of the language is portability. From the user's perspective it is platform independent. It is not untypical, often for historical reasons, that companies have more than one hardware type. If user developed code has to be compiled and linked before running it is often necessary to make small, but significant, changes in the code to be valid with that particular system's compiler. MCL+ is identical on any platform, therefore the user merely needs to make his programs available on a central server and need not concern himself with providing several versions.

MCL+ is more sophisticated than normal concepts of an interpreted language. Procedures may be loaded into system memory and are subsequently available internally on demand. ACIS entities exist in MCL+ and bindings to ACIS API functions exist. Data types and functions developed for other modules such as the GUI and volume modeller are also available through MCL+. This is very powerful because it allows the functionality for user defined features to be highly developed. MCL+ forms the syntactical base for defining user defined features and programming the control routines.

MCL+ does not allow strictly defined classes to be created as in C++ because there is no concept of scope. However, classes are simulated with *type* definitions and other object oriented functionality such as function parameter and operator overloading are provided. MCL+ not only provides typical functionality found in C++ or other high

level languages, but is considerably extended. Special data types have been provided in MCL+ that represent frequently used structures such as lists and strings. Direct pointer manipulation with MCL+ is not possible; it is provided automatically by MCL+ in the way that variables are declared and instantiated. This is an advantage to the user because it removes the problem of correct addressing. Bindings to ACIS API functions, the ACIS data structure and classes enable direct interrogation and manipulation of the model. Bindings are also provided to application classes and functions developed for the feature modeller and other application modules.

MCL+ removes much of the detailed problems of sophisticated programming languages, such as memory management, whilst providing a powerful toolbox suitable to the application of geometric modelling, geometric reasoning and the development of user defined features. MCL+ has been predominantly used for the developments reported in this thesis.

3.5 Graphical User Interface (GUI)

3.5.1 High-Level Design Methodology

The purpose of the GUI is to provide a high level interface to the feature modeller removing the need for the user to interact with the modules at programming system level. This creates a visually integrated and highly graphical working environment, which is a requirement for efficient and productive use of the system. The architectural methodology of the GUI is presented because it is structured to aid the development of user defined functionality. The detailed design and coding of the GUI will not be discussed.

The GUI is a very important part of a feature modelling system. Feature modellers whilst providing significant advances over pure volume modelling approaches for creation, manipulation and editing, can appear much more complex. The GUI to a feature modeller can make much of this complexity transparent to the user. In an

environment where user defined features are seen as a necessity, the ability of the user to be able to control the GUI to reflect the way in which the features are to be used is also essential. Another important function of the GUI is that of language support. Nowadays software is rarely both designed and used in a single language community. Industrial pressures have meant that sub-contractors (manufacturer) are often in a different country to the contractor (designer). The ability to switch between different languages in a system is a commercial need.

A schematic of the GUI of Figure 3.1 is shown in Figure 3.4.

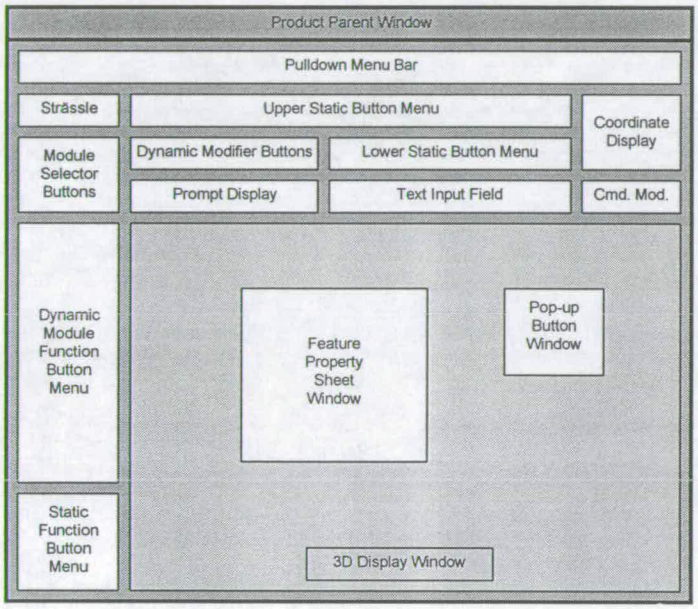


Figure 3.4 Schematic Of GUI Window Layout

It consists of:

- a parent window which controls the GUI, all other GUI objects are its children.
- a sub-window which displays the graphical representation of the model, highlights entities for identification, provides rubber banding for graphical input and allows the user to pick entities in the model.
- a pull down menu bar.

- static and dynamically controllable areas for the citing of buttons for function selection.
- pop-up windows, dialogue boxes, menus, button selectors, and slider controls.
- Command prompt display, default presentation and keyboard echo.
- Coordinate and measurement display.

The functionality of the GUI can be split into two aspects:

1. The windows and their management, buttons, menus, dialogue boxes, keyboard input, slides, and the like.
2. Graphical representation of the model and graphical input functions: highlighting, rubber banding, picking.

3.5.2 GUI Architecture

The GUI consists of a four layer architecture, which is shown in Figure 3.5.

The GUI presents all system manipulation possibilities using keyboard, mouse, windows, menus, buttons, dialogue boxes, icons, graphics, etc. All user interaction is transferred via X, Motif and SGUI to the MCL+ level. MCL+ allows the user to define his own functions; both for screen layout and modelling functionality. Special direct interfaces are provided from X to the input controller so that interactive rubber banding is as fast as possible. The HOOPS graphics model has a pre-processor that builds or changes parts of the model with respect to commands that are posted to the Feature Modeller bulletin board.

This graphical environment has been developed using a number of different software tools. The low level functionality is written in C or C++ to communicate with the graphical environment, geometric modelling engine (ACIS), and the feature modelling sub-system. The graphical display in the main window is driven by a

library of C functions provided by HOOPS (Hierarchical Object Oriented Programming System) [ITH]. Widget sets and window management are provided by OSF Motif. The windowing system is the X Window System. To provide a highly sophisticated user definable development environment, MCL+, which is able to manipulate GUI objects is used. Such a flexible environment is needed for the effective development of User Defined Feature Functionality.

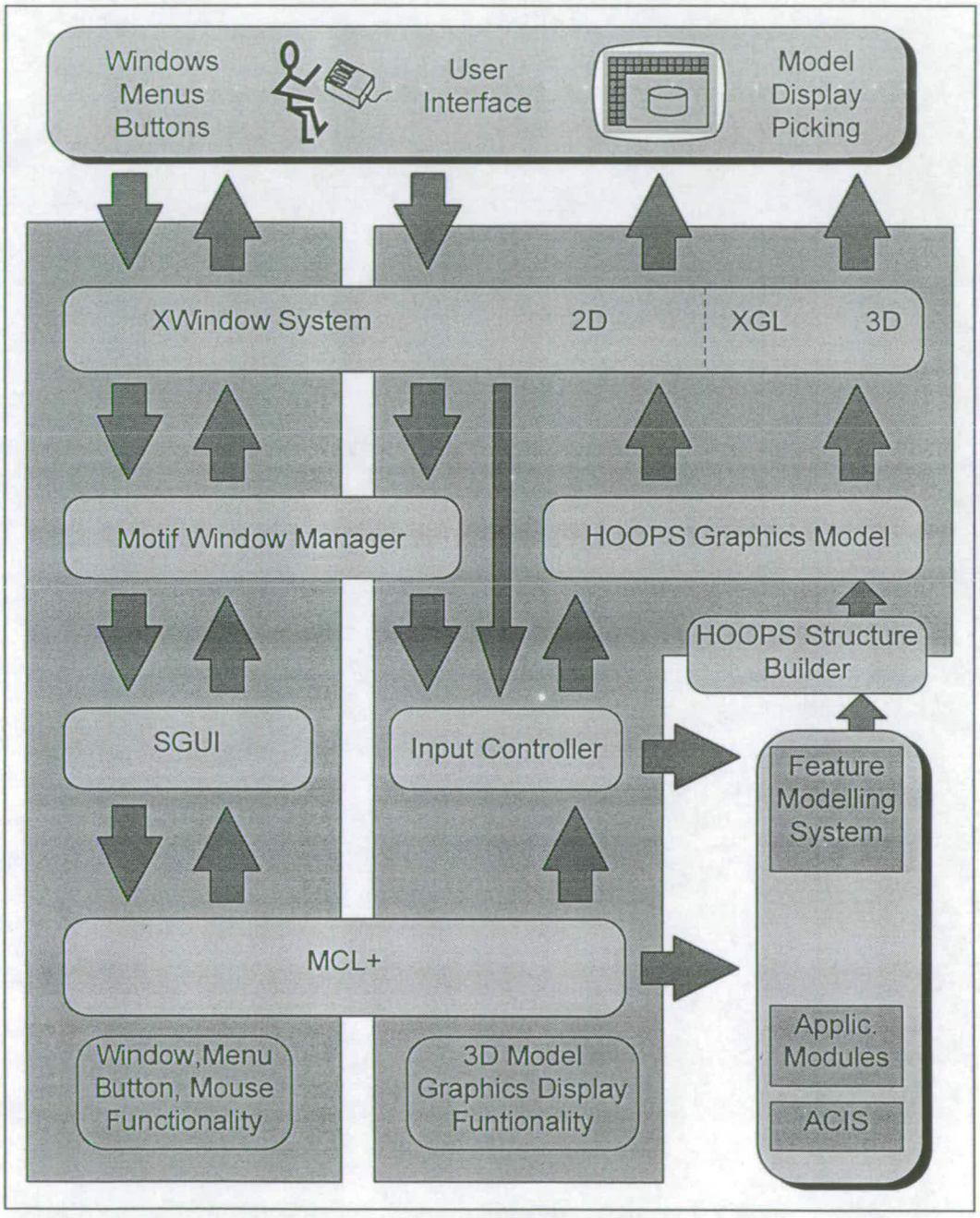


Figure 3.5 GUI Main Software Components

The window, menu, button and input control commands are transferred to the SGUI (Strässle Graphical User Interface) level, which is a high level abstracted language for the OSF Motif / X Windows standard. It provides a set of predefined objects to build the user interface with that will provide a product wide standard look and feel.

The main graphics display commands are processed directly by HOOPS. The model manipulation commands are processed by the feature modeller and ACIS followed by rebuilding of the HOOPS model which then updates the graphics display.

3.5.3 X Windows / Motif

3.5.3.1 Overview

The X Window System and OSF Motif are now accepted as de facto industry standards for window based UNIX graphical interfaces. They are described here together because in reality they are very closely related in terms of system development and functionality. Detailed descriptions may be found in [XWI] and [BER91] respectively. Due to the need for user defined functionality it is necessary to provide the user with a high level tool box. Direct programming by the end user at the X and Motif level are not possible or desirable therefore simplified and structured control functions have been provided by SGUI in MCL. These functions are user definable and closely related to the application environment of geometric and feature modelling and so provide a sophisticated toolbox.

3.5.3.2 X

The X Window system, developed at MIT in the mid eighties, provides for a (semi) hardware independent window based graphical interface. It makes applications think that the window it is using is the only display device. The application need not be

aware of the others because X takes care of this. Applications running on different hardware platforms may be simultaneously displayed on the same screen. It provides functionality to create, realise, destroy, clip, map, etc. X also provides for cursor functionality with a pointing device such as a mouse. Importantly, detection of events such as movement of the cursor, button presses and key presses are noted and passed to the window manager. The ability is also provided to return the position of the cursor in the window when a button press occurs. This is important for input functionality, such as picking.

X itself provides certain graphics functionality such as line drawing, flat polygon shading and fonts. However this is very simple pixel based and is related to the ability of the display device. Hardware vendors frequently provide extensions to X in order to provide better support for displaying 3D objects such as defined by solid models. SUN [SUN] have developed XGL and Hewlet Packard [HP] have developed Starbase for example. These give much superior support for polygon shading such as Gouraud and Phong, and rendering of curved surfaces.

In X, each window has some functionality associated with it. A window is divided into sub areas or sub windows (buttons) in hierarchical manner. Callbacks are attached to the windows (usually looking like buttons) which call application programme functions. Communication between X and the application programme is asynchronous. X simply responds to events caused by the user. The application programme may redefine what is displayed by X in response to a callback but it does not necessarily wait for a reply.

X has a tool box called X Toolkit Intrinsics. These are used by all window manager developers as the building blocks to build the Widget sets that allow the look and feel of the windowing system to be developed.

3.5.3.3 OSF Motif

Motif gives the window system its look and feel. It controls the 3D borders to the windows and buttons. Functionality such as pull-down menus, pop-up menus, slider controls, multistate buttons, toggle buttons, browsers, copying, cutting and pasting of textual entries and highlighting is provided.

The windows or objects that are displayed to provide the above functions are known as *widgets*. The widgets are built using the X Toolkit Intrinsics. They respond autonomously to events from the X server, triggering some internal actions. Actions may produce output to the window, changes in the internal state and callbacks to the application programme. The application may intervene by setting resource fields or executing functions.

The interaction between the X Window system (the X server), the X Toolkit Intrinsics, Motif Widgets and the application programme is shown in Figure 3.6

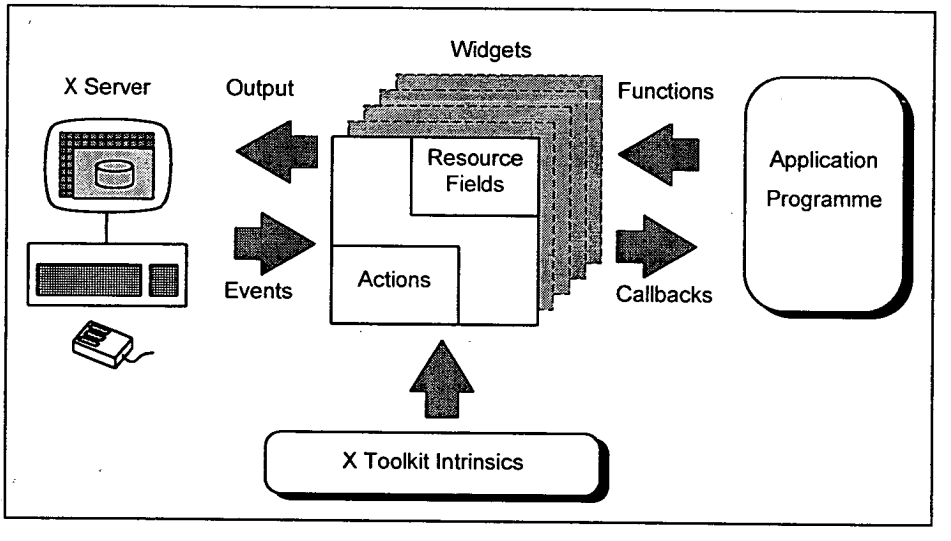


Figure 3.6 Schematic Of X/Motif Interface (Source [BER91])

3.5.4 SGUI

Strässle Graphical User Interface (SGUI) [EGG95] forms the interface between the feature modelling system and OSF Motif. SGUI is based directly on OSF Motif and consists of the following components:

- C functions for connection to OSF Motif.
- MCL+ functions.
- MCL+ parser file.
- UIL (User Interface Language), which is part of OSF Motif. Compiled into UID (User Interface Definition) files.

SGUI provides the development engineer with a toolbox of MCL+ and C objects required to define and build a user interface. It is a higher level language than OSF Motif simplifying its use, though knowledge of OSF Motif is required to use it correctly. User definability is provided through the MCL+ objects.

NLS (National Language Support) is an important software function. SGUI NLS enables all textual messages, field definitions, and the like to have different text strings associated with the MCL+ call depending on the language that is required. The messages are defined in an NLS file with a unique module and message number. There as many lines of text per message as different languages are required. The definition number remains the same but the language descriptor is different, as of course, is the text. The system works well with different alphabets, such as Cyrillic. SGUI NLS support is fully described in [PRO].

Such functionality is very important because it is rare that CAD/CAM software systems aimed at SME's will be sold into a single national market. Similarly when larger OEM's are applying simultaneous engineering strategies it is very likely that the suppliers are in a different country.

3.5.5 HOOPS

HOOPS (Hierarchical Object Oriented Programming System) [ITH] is a system for creating interactive graphics applications. The display and manipulation of graphics is a data base problem [FOL84]. HOOPS is best described as a database which stores information about which objects to draw, where they should be displayed, and how they should be rendered. HOOPS provides other systems with tools for modifying, querying, searching and displaying the database. HOOPS comprises a library of C routines that form a toolbox for the application developer.

Data consists of geometrical primitives, cameras, lights, rendering and modelling attributes, and application specific information. Related elements are grouped together in segments which are the units of organisation in the database. Segments may consist of segments and hence correspond to the nodes of the tree structure. The hierarchical grouping is an efficient way to organise data, since it allows manipulation of just the components of the objects, or groups of objects as a whole. The object oriented nature of HOOPS is provided by the way it treats segments as objects in a hierarchical manner, rather than the programming language it is written in.

All information stored in the database can be changed: geometry can be edited, attributes can be modified, and the hierarchy reshaped. After a series of changes is specified, the system updates the display to reflect the current contents of the database. The geometric definitions in the database are provided by an ACIS model. When the ACIS data model is modified, information is posted to a bulletin board. This notifies the HOOPS pre-processor which rebuilds the graphics database and displays the modified model.

In FeatureM the HOOPS implementation is shown schematically in Figure 3.7. The root of FeatureM's data structure is the display window defined in X. The display consists of one *layout* made up of one or more *views*. Each view comprises many segments defining the bodies that should be displayed, textual information such as

labelling, and of course coordinate systems. Clearly there is likely to be more than one body and almost certainly more than one dimension. Each body or dimension is treated as an individual segment. Hence, the nodes displayed in Figure 3.7 are the roots for further lists of segments.

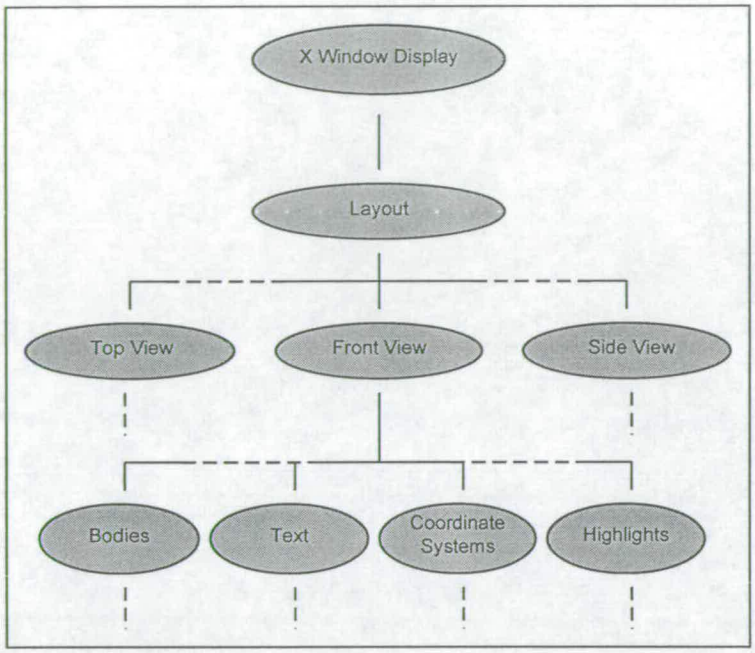


Figure 3.7 HOOPS Implementation In FeatureM

The HOOPS pre-processor consists of a series of functions that edit the HOOPS segment structure by removing branches (or leaves, or the whole tree except for the root window segment) and adding new ones. The functions are called depending on what statements exist in the ACIS bulletin board. Each entity that may be posted on the bulletin board has associated with it a corresponding procedure which describes how the HOOPS database should be built to display it. In other words, each object that can be displayed knows how to display itself. The pre-processor simply removes the old description (if one existed) and adds in the new one. A good example where partial update occurs is that of camera rotation. The faces and edges of a body remain the same but the silhouette lines are different. Thus the function asks ACIS to calculate the new silhouette lines, replacing the old ones with the new ones. From the users perspective, the X Window system, Motif and HOOPS appear as an homogeneous system.

3.5.6 Input Controller

The Input Controller controls the following functionality:

- Highlighting

Highlighting is a graphics function that changes the colour of an object or provides some graphic indication of an object. This is used to show that the object is currently the focus of attention of the cursor, or is the selected object that will be the subject of the next function call.

- Rubber banding

Rubber banding (or echoing) is the technique whereby as the cursor is dragged across the screen a graphical representation changes size correspondingly. Typically this has been used for simple geometric entities, such as straight lines, circles and arcs in 2D draughting. More recently complex 3D rubber bands have been used to fully represent the solid objects such as blocks, cylinders, prisms, etc.

- Pick filtering

Pick filtering is necessary to return an entity of the type desired even though it is impossible to pick that entity in the graphics mode used. For example, in shaded mode only faces are known by the graphics model shown in X. Therefore to pick an edge, it is necessary to find the edge nearest the pick point of the face returned.

The input controller provides MCL+ with a series of functions that are used to define input strategies through the GUI by means of an input *decoder* The Input Decoder is the method by which interactive input using textual prompts, rubber band graphics, pop-up menus, keyboard and mouse is controlled. Although implemented in C++ the functionality is provided for the user in MCL+ making the construction of input command sequences completely user definable. A necessity for successful implementation of user defined feature functionality. The decoder is described in [ALT96].

3.6 Application Modules

Application Modules are software components that are external to the feature modeller but add functionality to the system as a whole. Modules can be stand alone utilising ACIS functionality to produce solid models, or they can be interfaced to the feature modeller to provide support functionality. One module is completely user definable. The following modules are part of FeatureM: Volume Modelling, 2D Contour Modelling (Sketcher), 3D Sheet Metal Modelling, User Defined. The combination of the 2D Contour Modelling Module and some of the pure Volume Modelling functions associated with generation of sheet and wire bodies are useful in the implementation of swept geometry features.

3.7 Feature Modeller

3.7.1 Architecture

FeatureM is a full open architecture general purpose feature modeller. Features are not restrained to merely representing form but may represent any piece of information (limited only by the allowable parameter types provided for features) that the user would wish to support in a workpiece oriented data structure. This capability helps the development of product modelling techniques and the subsequent integration benefits that are a natural byproduct of supporting a more complete information set than is defined by form alone.

All feature template definitions that form the library from which users create instances to build a model are defined externally in a file. The procedures that represent the semantics of the feature are also defined externally in a file. MCL+ forms the language used for both template and procedure definition. The system is very strongly oriented to the development of user defined features through this mechanism.

A single FeatureM model may consist of many workpiece models all of which may or may not produce geometry. A form feature generally produces a volume which is integrated with the rest of the model by a Boolean operation. Certain from features do not have a volume associated with themselves but produce a volumetric result by acting locally on the BREP ACIS model. Such features are fillets / rounds, and chamfers. Previously defined ACIS bodies may be integrated with features in the model. Such bodies may form the base geometry, be added or subtracted, but their geometry cannot be changed parametrically. Features that are defined parametrically may have any parameter value changed at any time. Unusually when compared to parametric BREP models, this is equally applicable to features deep in the tree where topological changes can cause difficulties. Features are positioned using coordinate system reference frames. Features may be referenced to any other that exists before it in the feature tree, and features may be grouped. Special positioning features are used to locate one feature relative to topological entities rather than the workpiece coordinate system. This enables features to be located with respect to faces and edges. This provides more advanced functionality than general parametric modellers because not only position but attitude may be controlled together or independently.

The Feature Modeller communicates with ACIS through an API, Figure 3.8.

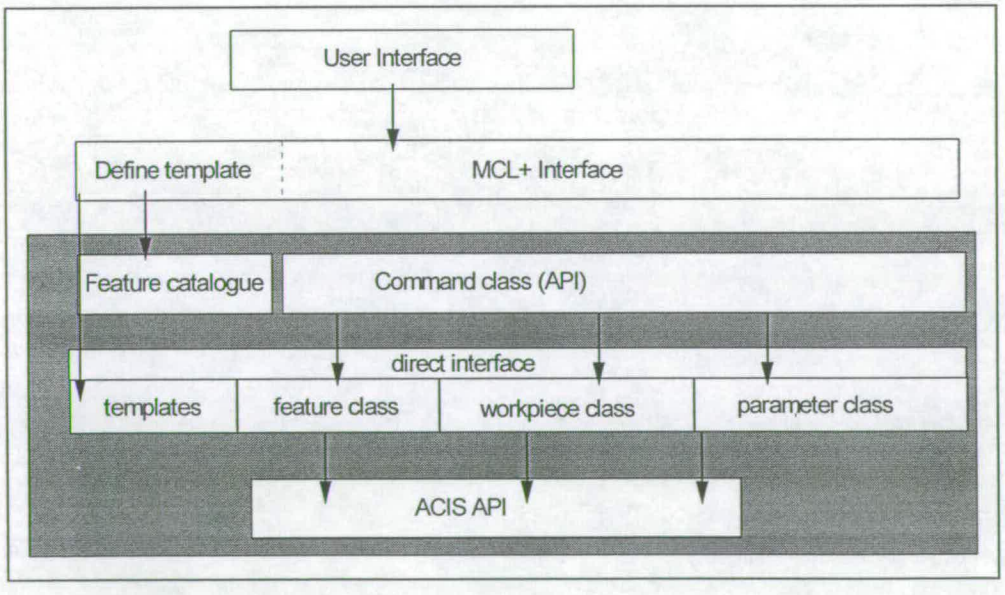


Figure 3.8 Feature Modeller - ACIS Communication

There are external parts corresponding to the definition of the feature templates and the control and build procedures which are implemented in MCL+. The internal part consists of the internal representation of the feature catalogue (parsed from the external library), the callback procedures for build and control, and the workpiece, feature and entity classes which act through the ACIS API. The Feature Modeller is divided in two main components built using the MCL+ and C++ languages and utilising ACIS through its API: the Kernel, and the Feature Template Administrator, Figure 3.9.

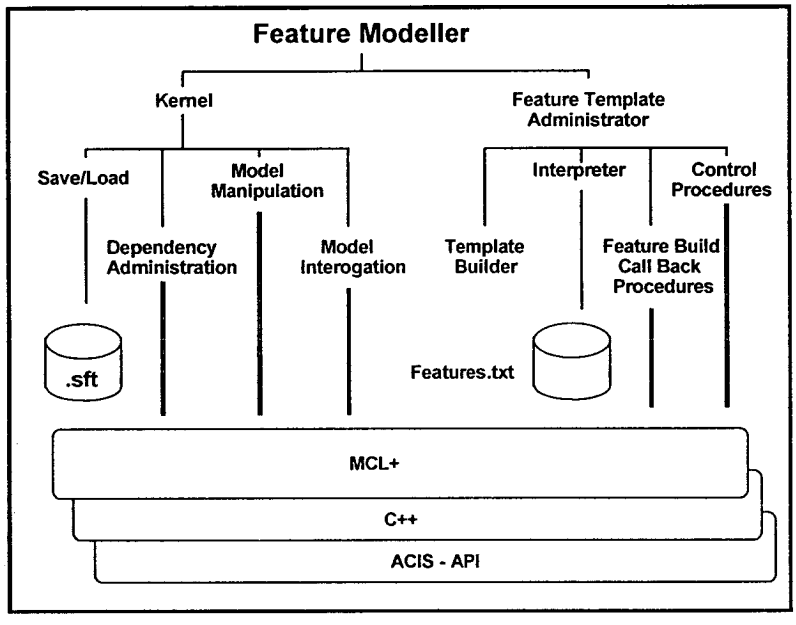


Figure 3.9 Feature Modeller Software Components

Kernel functionality is associated with model management, the Feature Template Administrator is concerned with providing the internal representation of the feature library and associating the build and control procedures defined externally with the internal call backs.

The kernel does not concern itself with the definition of features merely the instantiation of an implicit representation to create an explicit entity which can be added to, and manipulated within, the model data structure. The kernel functions build up the basic ability to generate 3D models by combining explicit instances of features.

The Feature Template Administrator interprets the external MCL+ feature template definitions building an internal data structure. Any procedures for build and control are loaded into the MCL+ data structure. The MCL+ procedures are associated with the calls in the internal implicit feature definitions.

ACIS the geometric solid modelling engine has been used through its API (Application Procedure Interface). The connection of MCL+ and the ACIS API is made by C++ function calls.

The internal representation of the workpieces, features and parameters is made by a class structure, Figure 3.10. This utilises the ACIS User Class so that all additions for part of the ACIS data structure and are therefore included in the save, restore, roll back and roll forward mechanisms. A simple link class, EntityKon, is used between the ACIS User Class and the feature modeller classes. This provides for differentiation of classes within the ACIS world.

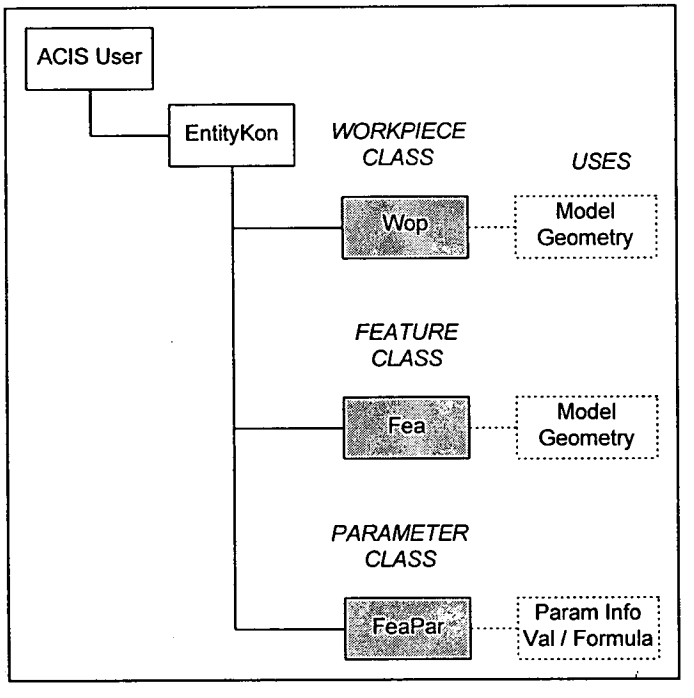


Figure 3.10 Feature Modeller Internal Class Structure

The data structure employed to support the feature model structure is one of linked lists. A model must consist of at least one workpiece producing a geometric result.

Each workpiece will consist of more than one feature, and each feature will have more than one parameter. The Wop and Fea classes provide the management functionality such as creation, deletion, determining status, listing, unique identifiers, next pointer, feature pointer, etc. The parameter class is used to support the parameter values and control methods. The class structure has been developed in order to allow extension and efficient use.

FeatureM models consists of one or more workpieces. Each workpiece forms an element of a list, with a pointer to the next. All features are contained in lists, one for each workpiece. Similarly, each feature has a list of parameters. Features may be grouped creating lists with sub-lists. This structure is shown in Figure 3.11.

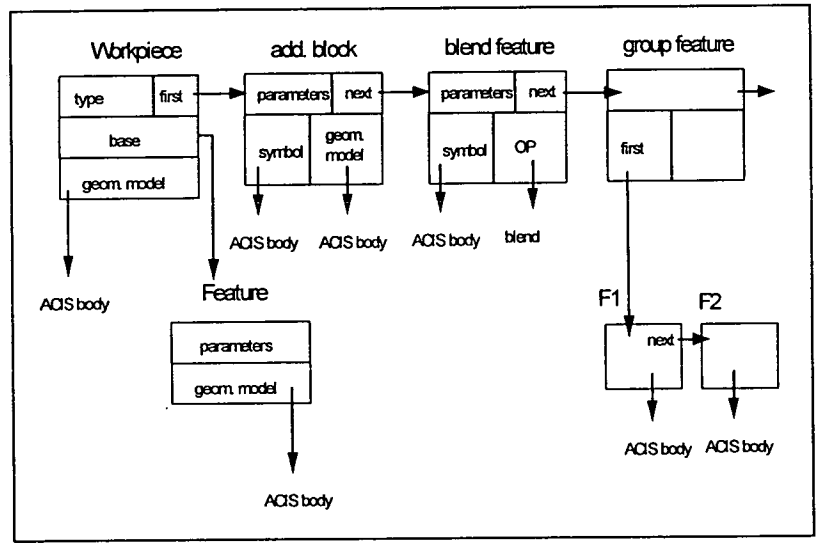


Figure 3.11 Feature Model Data Structure

This structure may be represented as a tree. The tree structure of workpieces defining form feature models capture the design history of the component, Figure 3.12. When a feature returns a volume on evaluation this is combined with the rest of the model generated from the previous part of the tree by one of the Boolean operators of union or difference. The parts of the tree representing a group of features is traversed in the order that each feature was put into the group. Group features offer a major advantage to the user because the whole group of features can be moved, copied positioned or oriented together, rather than individually. The first feature in a

workpiece representing a physical component describes the base geometry, or root, to which all other features are added, subtracted or locally manipulate the current state of the model.

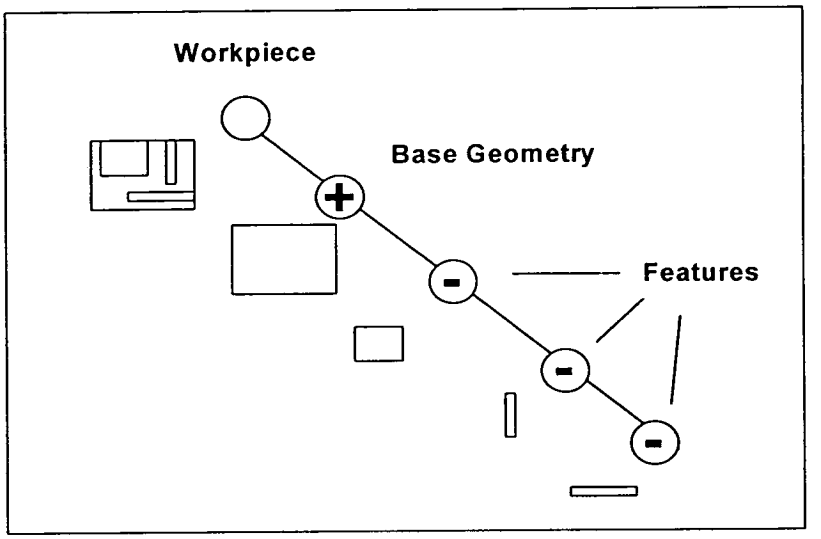


Figure 3.12 Workpiece Feature Tree Structure

3.7.2 Kernel

The kernel functionality is associated with the creation and maintenance of the model data structure. The kernel handles all interactions with the feature model creating explicit instances of the implicit library definitions. All coordinate systems are maintained by the kernel and therefore positioning of features. The important functions of saving and restoring feature models are also performed by the kernel.

3.7.2.1 Save / Restore

FeatureM models consist not only of workpiece data structures but of other elements that the user uses to define the working environment: layouts, views, coordinate systems. It is necessary to be able to Save / Restore the combined tree structure, shown in Figure 3.13.

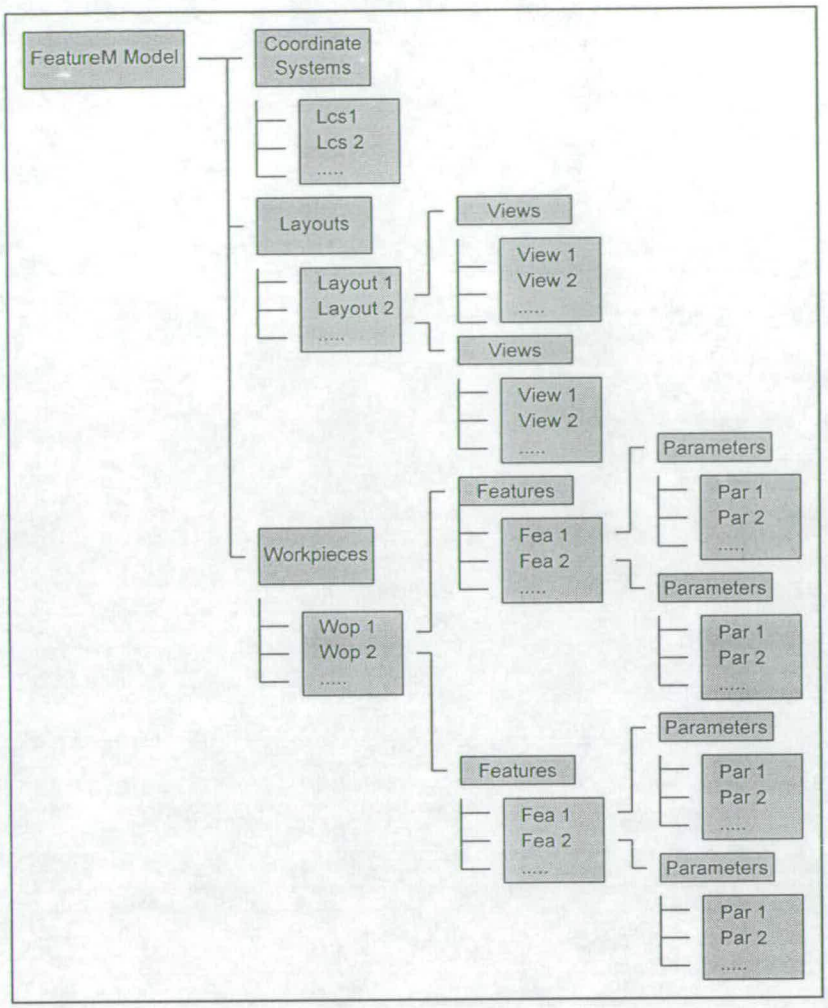


Figure 3.13 Combined Model Structure

The Save / Restore function uses a textual method to describe the workpiece, feature, and parameter tree structure. A more problematic part of the save / restore mechanism is to maintain object identities to topological elements on restoration. Such topological elements are edges which may be rounded / filleted, chamfered or swept faces. The mechanism assumes that when ACIS rebuilds a model that it does so in exactly the same manner and that the topological identifiers are the same. To date experience has shown this to be the case. As a safeguard the ACIS BREP model of the resultant geometry is also stored inside the save file. This would allow exact matching of the new topology with the old, and hence derivation of new pointers from old. The feature library is not stored in the save file.

3.7.2.2 Dependency Administration

The Dependency Administration module provides the management functionality of the kernel in order to maintain a consistent model. Dependency Administration is required in the following areas: model tree data structure, coordinate system maintenance, parameter setting, use of formulae in parameters, persistent object identity and error handling.

3.7.2.2.1 Model Tree Maintenance

The tree data structure described in section 3.7.1 requires managing to enable the user to edit it without causing inconsistencies.

Naming conventions require that all names are unique for workpieces and for features within a workpiece. A feature is not only identified by its name but that of the workpiece to which it belongs. Therefore the tuple is unique even if features in different workpieces have the same names. Management of renaming of workpieces and features is governed by the system.

The order of evaluation of features effects both the geometry created due to the Boolean operations, and entity existence in the model. It makes no sense to define parameters which point to entities that do not exist at that point in the model evaluation. Similarly, it is a nonsense to define a positioning feature after the feature that it is supposed to position. Reordering of the features is possible but only by moving them to the end of the feature list. This may be done many times to produce the result that is desired. Maintenance of the feature lists is also performed when the base geometry is exchanged.

When a feature is instanced from the library it is added to the feature tree in its implicit form. Only when the user instantiates all parameters or accepts all default values does the feature become explicit.

Optimisation parameters may be set so that only part of the model is evaluated at each regeneration. This is sensible for large models where modifications occur in localised areas. Effectively part of the tree is saved as a body, the remainder of the feature tree being evaluated with the new, pseudo base feature.

3.7.2.2.2 Coordinate System Maintenance

The Feature Modelling System uses a number of different coordinate systems:

1. World Coordinate System

   World Coordinates define three dimensional space into which all objects are positioned. It is the base coordinate system to which all other coordinate systems are referenced.

2. Body Coordinate System

   Each body that ACIS produces has its own coordinate system. A body is positioned and oriented with respect to World Coordinates by applying a transform relative to the World Coordinate System. This transform is an attribute of the body. A vector or position that is derived in Body Coordinates will maintain its direction or position relative to the body under transformation of the body. Positions and vectors may be easily converted from one coordinate system space to another. For example,

   $$World\ Position = Body\ Position * Body\ Transform \quad (3.1)$$

   $$Body\ Vector = World\ Vector * Inverse\ (Body\ Transform) \quad (3.2)$$

3. Workpiece Coordinate System

   A workpiece gets its coordinate system from its base feature. Although ACIS produces the body of the base feature it is usual and desirable that the workpiece coordinate system is different from the body coordinate system. This provides the user with the ability to position the body of the workpiece relative to its

coordinate system origin. A workpiece, via its base feature has an attribute transform which enables easy conversion from one coordinate space to another:

*World Position = Workpiece Position * Workpiece Transform* (3.3)

*Workpiece Position = World Position * Inverse (Workpiece Transform)* (3.4)

Note that the position and orientation parameters of the workpiece are defined in World Coordinates.

4. Feature Coordinate System

Each feature has a coordinate system origin. The size parameters of the feature are developed in this coordinate system. The position and orientation of the feature coordinate system origin is with respect to the workpiece coordinate system; the position and orientation of the feature will remain invariant under transformation of the workpiece, which is desirable. Each feature has a transform attribute. Conversion between feature and world coordinates is as follows:

World Position = *Feature Position * Feature Transform * Workpiece Transform* (3.5)

Feature Position = *World Position * Inverse (Workpiece Transform) * Inverse (Feature Transform)* (3.6)

Note that the position and orientation parameters of the feature are defined in Workpiece Coordinates.

5. Local Coordinate System

The Local Coordinate System (LCS) or Working Coordinate System is a movable frame of reference. The user is provided, through the GUI, with many interactive methods for positioning and orienting the LCS. Further functions are provided to enable the user to snap a Feature Coordinate System origin to the LCS (either

position or orientation only, or both). Similarly the LCS may be snapped to the Feature Coordinate System origin. This provides the user with a very intuitive and simple method of locating features in space and visualising how they are defined.

These coordinate systems are shown in Figure 3.14.

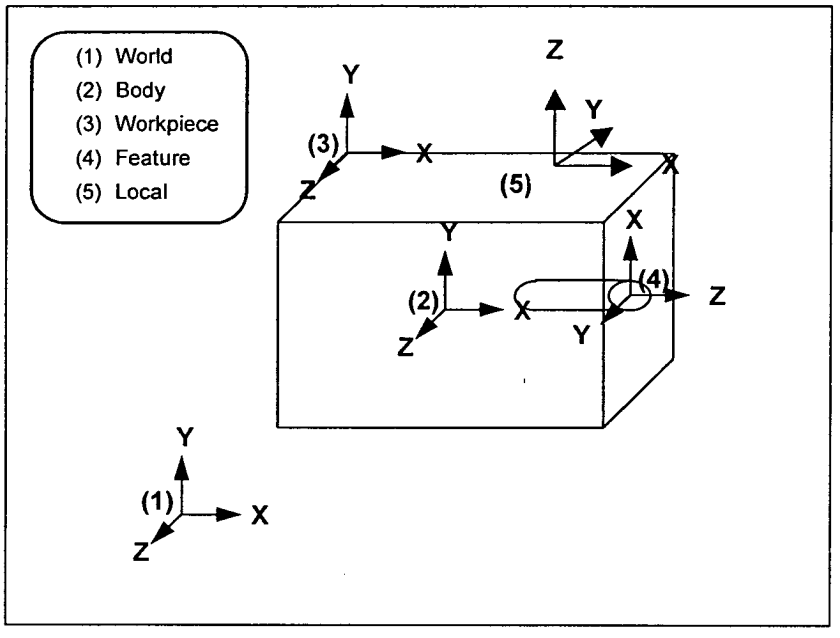


Figure 3.14 Different Types Of Coordinate Systems

Using this simple method of conversion from one coordinate space to World Coordinates it is then possible to convert to any other coordinate space. ACIS provides built in functionality for transform manipulation.

3.7.2.2.3 Attribute Maintenance

The public parameters of the class definitions of workpieces and features are available in MCL+ as attributes of the corresponding type definitions. The consistency of these attributes is maintained by the kernel.

Workpiece attributes are:

Name, Body, IsValid, Transform, Type, Kind, FullName, Ident, LinAttrib, hasColour, Colour.

Feature attributes are:

Name, Template, Body, Parameters, Workpiece, Feature Type, Faces, Faces_Exclusive, Info List, Info Text, Transform, kind, FullName, Ident.

3.7.2.2.4 Parameter Maintenance

A most important function of the Kernel is to allow setting of parameters to new values. It is important that values of the correct type are allocated. The kernel checks this automatically. Parameters may have access rights set to read only. In such circumstances editing of the parameter value is not possible. It is also possible to "freeze" a feature, which disables any editing function for that feature. Features that represent standard items such as threaded holes have geometric sizes governed by one or two parameters, M8 for screw type and Long for depth of pilot hole. The physical dimensions required to generate the body are calculated and stored as parameters of the feature. The kernel enables a feature's own control and build procedures to update its own parameters. This can cause a problem of recursion because setting a parameter's value invalidates the parameter and feature. The kernel ensures that those parameters that are set from within the same feature's control procedures do not invalidate it. Hence, recursion cannot occur. If features are working in pairs and recursion does take place this is limited to a certain number of iterations. The kernel knows if a parameter has been evaluated or not in a model regeneration since the feature was set invalid.

Another important task is that of maintaining object pointers. Pointers to topologic entities in the geometric model will be discussed in section 3.7.2.2.5. because they pose special problems; object identity is not inherently persistent. Pointers to other features in the tree, within the same workpiece, or in another, are maintained. These pointers are used by features to indicate a reference to another feature other than its

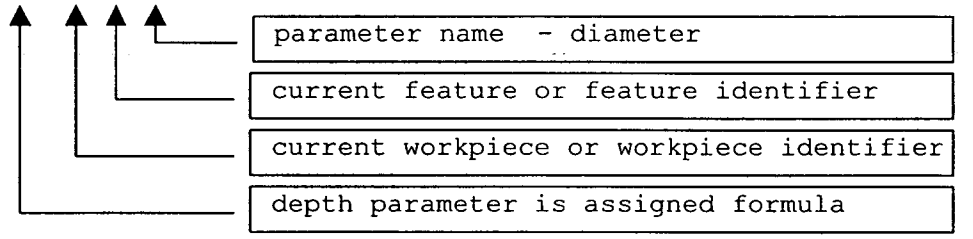
own workpiece, to point the next feature in the list, to create group features and to create related pairs. If features are deleted then the pointers must be reset. The pointers must be carefully treated upon reordering of features and copying.

3.7.2.2.5 Formulae

It is possible to define a parameter with a formula and assign this directly in the property sheet. This is possible because the value of the parameter class is always stored as a string. The string is parsed to define whether it is a value (real, vector, position) or a statement that can be evaluated.

For example, equation 3.7 calculates the depth of a hole using its diameter.

$$h_1 = \#/\#/d_1 * 2 \tag{3.7}$$



The language and syntax of a formula is that of MCL+. There is one exception: the # sign. This is an extension to MCL+ and provides the user with a shortcut to define the current workpiece and the current feature. If the # sign is not used then the full name of the workpiece or feature must be given. Using position features is one method of constraining features, formulae provide another.

If a formula cannot be parsed correctly, or a workpiece or feature that was referenced no longer exists, the kernel, in order to maintain consistency, uses the last available value. The formula remains within the parameter but is commented out. It is, therefore, relatively easy to develop.

As with the ordering of features in the tree, ordering of parameters is also crucial when formulae are used. Parameters are said to be independent if they do not have formulae that calculate their values from other parameters in the feature. If

parameters are calculated from other parameters in the feature they are said to be dependent. The order of the parameter list governs the order of evaluation of formulae. If there is a sequence of related dependent parameters before an independent parameter occurs, a once through iteration will not produce the desired result. The system iterates until all values stabilise. If a loop is generated by the formulae, the system sets a limit for the number of iterations. The last generated values are used.

Consider the example shown in Figure 3.15.

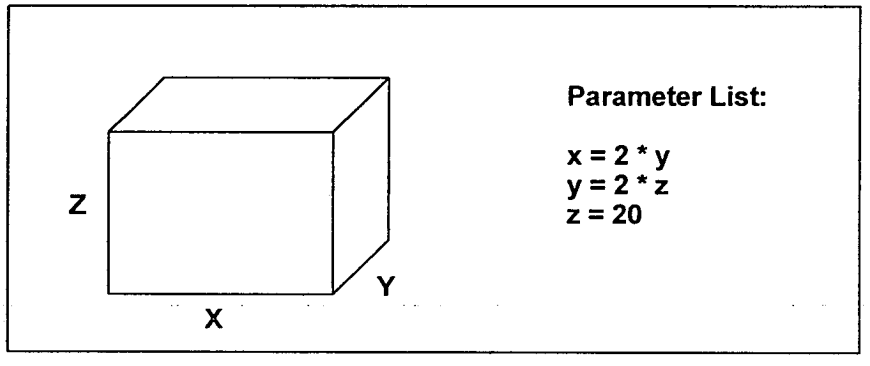


Figure 3.15 Dependent Parameters

Original values:

```
x = 30
y = 20
z = 10
```

First iteration through parameter list after z is set to 20:

```
x = 40
y = 40
z = 20
```

Second iteration through parameter list provides the desired result:

```
x = 80
y = 40
z = 20
```

3.7.2.2.6 Persistent Object Identity

A special mechanism has been developed to maintain persistent object identity for topological elements in the geometric model. This is necessary because after every generation (addition of new feature, change of parameter value, feature deletion) ACIS creates a new model deleting the old one. Hence, pointers to the entities in the old model do not point to anything and object identity is lost.

Instead of storing just a pointer to a topological entity as a parameter, a description that allows the entity to be found from the elements that it is related to is also stored. In some circumstances such as optimised partial rebuilds and the use of local operations, ACIS object identity is maintained. If the ACIS object identity cannot be found attributes such as the feature or features which were used to create the faces, the faces that created the edges, the edges that created the vertices, are used by functions to establish identity. These functions can be likened to a small Expert System. This mechanism is transparent to the user, but is subject to limitations particularly when significant changes in topology take place. Providing improved functionality is a subject of on going work.

3.7.2.2.7 Error Handling

Errors fall into two categories: trapped and untrapped.

Untrapped errors at the C++ level can be catastrophic generally causing segmentation faults and system failure. These should not be allowed to occur in system developments. However, software, like the programmers who write it, is not perfect and untrapped errors can occur. Monitoring of instructions (writing a copy of the instructions to a file) occurs when a Notestate is called in the Input Decoder. The monitor file is an ASCII file which can be edited to remove the problem command. It is in MCL+ format and can be executed by the interpreter, regenerating the model.

This provides a recovery path but does not lessen the need to develop quality robust software. User Defined Functionality is programmed in MCL+. Untrapped errors in MCL+ code do not cause system failure, the MCL+ function merely fails to produce the desired result. If this occurs in a procedure that forms part of a feature's control routines the kernel recognises that the routine has not terminated correctly, suppresses the feature and stops model regeneration.

Trapped errors in MCL+ and C++ produce controlled error messages via SGUI and NLS. The kernel handles the errors in different ways. The system can be made to rollback to the previous state of the system. In complex feature definitions where invalidity may be remedied by changing one of a choice of parameters it is not desirable to rollback. In this case the system can be forced to suppress the feature. Rollback does not occur, therefore all (even problem) parameter values are contained in the feature. The user can then adjust other parameter values to eliminate inconsistencies, unsuppress the feature and regenerate the model to a successful outcome.

3.7.2.3 Model Manipulation

The functionality provided by the kernel to enable editing the model has been discussed in Section 3.7.2.2 from the perspective of maintaining the consistency of the model. These administrative tasks are performed internally within the kernel. Interface functions must be provided that exist at the MCL+ level to enable the user to manipulate the model from outside the kernel, i.e. via the user interface. The property sheet shown in Figure 3.16 is managed dynamically and shows the administration information used by the kernel and all parameter values.

Features may be manipulated directly via MCL+ programming statements. Functions may be called to manipulate features via GUI buttons. Another way to manipulate a feature is via a property sheet which is the typical method for feature modelling systems. It is less intuitive than working graphically but is necessary for those

parameters that have no obvious graphical sign (and are not common to all features). The feature that appears in the property sheet is known as the active feature (to avoid confusion with the pre-selected feature). Note that activating a feature also activates the workpiece that it belongs to. Any commands called from the property sheet act on the active feature visibly displayed in it. For other command calls the active feature is used by default.

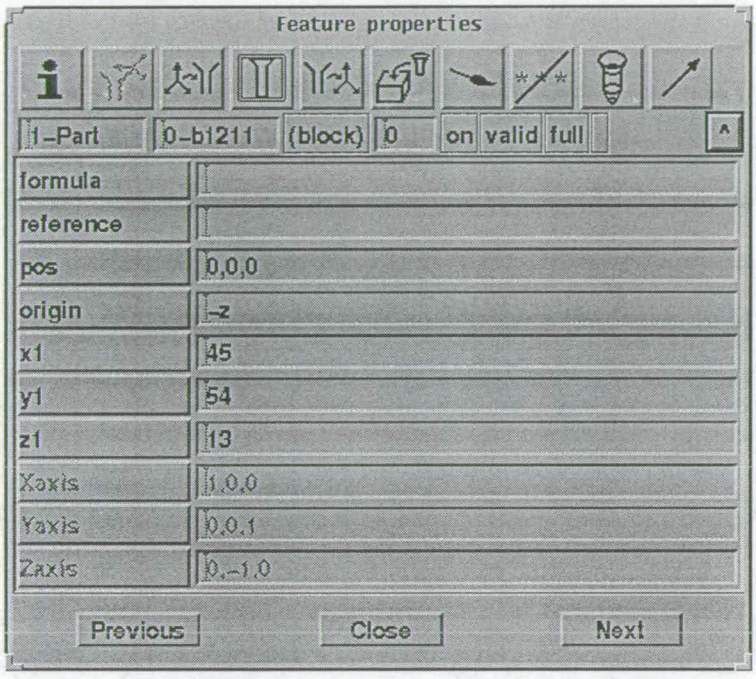


Figure 3.16 Feature Property Sheet

The following functionality is provided for workpieces:

> Create, Assign variable to, Select, Copy (as workpiece), Copy (body of workpiece), Rename, Delete, Transfer a feature from one workpiece to another, Swap feature positions in workpiece, Generate the workpiece. Save / Restore. (individual workpieces as well as complete models)

The following functionality is provided for features:

> Create, Assign variable to, Select, Copy (as feature), Copy (body of feature), Snap Lcs to feature, Snap feature to Lcs, Delete, Rename, Suppress, Fix, Mode (model, symbolic, combined), Define alias, Show Body.

The following functionality is provided for parameters:

Set value, Set rank, Define alias, Copy (as parameter value), Copy (out body of parameter)

These functions are fully discussed in [SIS95a] and [SIS95b].

Normally the attributes of a feature or workpiece are read only so that the kernel can maintain consistency of the model. The transform attribute, however, can be set by the user. This provides a very efficient way of setting position, orientation, or both, of a workpiece or feature. A transform is a defined type in MCL+ therefore it is impossible to create an invalid matrix. It has been shown that transforms provide an easy means to move between different parameter spaces and therefore of controlling features from the MCL+ programming level. The user may elect to set the parameters of a feature directly (position, wx, wy, wz - the rotations about the x, y, and z World Coordinate axes) but the rotation angles are difficult to calculate. The user has the option of viewing the direction vectors of the rotation matrix. These are read only for consistency therefore creating a valid transform and assigning it is the way to work with the direction vectors. However using the LCS and snapping the Feature Coordinate System to the LCS is a very simple and highly efficient method of setting a feature's transform. The system automatically converts the from the LCS to Workpiece or reference feature coordinates.

3.7.2.4 Model Interrogation

Interface functions must be provided at the MCL+ level to enable the user to access information contained within the model from outside the kernel, i.e. via the user interface.

The following functionality is provided for workpieces:

List all workpieces, Print workpiece.

The following functionality is provided for features:

List all features, List all invalid features, List all features that created a face, List all formulae, Get Alias, Move, Rotate, Transform

The following functionality is provided for parameters:

Get value, Get value (list of possible objects subject to identity maintenance), Get Info, Get parameter type, Get alias

These functions are fully discussed in [STR95a] and [STR95b].

The workpiece and feature class attributes that are maintained consistently by the dependency administration module can be easily accessed from MCL+. The model interrogation functions provide accessibility to all information stored in the model: a prerequisite for open systems architectures.

3.7.3 Feature Template Administration

The Feature Template Administrator contains the mechanisms to build an internal representation of the feature templates by parsing the descriptions defined in the external library. The internal descriptions can then be called from the kernel. The build and control procedures also defined externally to the system have to be correctly associated with the callbacks defined in the templates.

3.7.3.1 Interpreter

All features available to the feature modeller, whether system developed or user defined are described in a series of external files. The contents of these files will be described in chapter 4. The interpreter reads these files during the system start up, or when requested to do so. A template and its build and control procedures can be defined and loaded into the feature modeller at any time. Naturally they cannot be

used before they are loaded. Any syntactic errors in the template definition and the build and control procedures are detected at this stage. Any defective definitions or procedures will not be loaded or be accessible by the kernel.

3.7.3.2 Template Builder

The Template Builder takes the valid template definitions parsed by the interpreter and builds an internal representation. The build and control procedures are also loaded into the MCL+ table. The Template Builder associates the calls defined in the template with the relevant procedures. These procedures provide the callbacks to ACIS to create the solid bodies of the features.

MCL+ allows procedures to be overwritten. This does not cause potential problems for the build and control procedures because they do not store data. They take data stored in the parameter definitions and do something with the data or to the system. It is undesirable to overwrite template definitions because features that exist in the model may no longer have a valid definition. The Template Builder does not allow the user to overwrite templates for consistency. Problems can be caused by users changing feature template definitions and loading them at system start-up after files have been saved with features from previous definitions in them. In such cases, different algorithms must be used in the build and control procedures. Often it is better to name the modified template differently increasing the number of features in the library.

3.7.3.3 Build and Control Procedures

The build and control procedures are written in MCL+.

Only features have build procedures. The build procedures create the objects that the feature modelling kernel uses to generate the models. In the case of form features the

build procedures would return either the body of the feature to be integrated with the rest of the model, or the symbolic (independent) representation.

Both features and parameters have control procedures.

The feature control procedures are called once only: when the feature is created, and when the feature is deleted. These procedures give the user the ability to perform house keeping operations for the feature. For instance, when one feature is created it might be desirable to instance a partner feature at the same time. Similarly, when one feature is deleted that has a partner it sensible to delete the partner as well.

The parameter control procedures are used to modify the feature by allowing procedures to be called when a parameter has its value changed, or when a parameter requires a value but the user has not *set* one (and no default exists). For example, a slot feature may be straight or form a circular arc. Rather than use separate feature template definitions it is possible to define a parameter that indicates the *form*. On setting of this parameter it is desirable to change how the feature is defined: a curved slot has radius and interior angle parameters, a straight slot has a length parameter. In this case the form control procedure is used to set the visibility of parameters. By contrast, if the user has not *set* a value and no default exists, a procedure may be developed to calculate the parameter value depending on other parameters, or the wider state of the model. The parameter is said to *need* a value.

The use of build and control procedures will be described in more detail in chapter 4.

3.8 Chapter Summary

In this chapter the feature modelling system, FeatureM, which forms the baseline for the developments in chapters 4 and 5, has been described. The system consists of five major software components: the geometric kernel provided by the BREP modeller ACIS, the interpreted language MCL+ used for user defined functionality, the GUI that can be controlled by the user with MCL+ commands, application modules that provide associated functionality such as 2D sketching, and the feature modeller itself.

In the next chapter (chapter 4) the methods used to enable user definition of features will be described in detail. These techniques will be developed to enable definition features with extrinsically defined form, improved control of the GUI and better feature parameter validation.

Chapter 4

User Defined Feature Definition

4.1 Introduction

4.1.1 Chapter Overview

Chapters 1 and 2 of this thesis have introduced feature modelling technology as an aid to process integration and realisation of simultaneous engineering goals. Problems affecting the commercial impact of feature modelling systems have been stated. The feature modelling system which forms the baseline for the developments presented in this chapter and in chapter 5 has been described in chapter 3. In this chapter advances are reported which aim to address some of the problems that have been detailed in chapter 1. Namely: representing features with neither implicit or explicitly defined form (extrinsically defined form), providing enhanced GUI support, and extending the methods for describing user defined features.

This chapter presents the needs for feature modelling systems to support more complex geometry than 2½D. Feature modelling systems to date have predominantly

used catalogues of well defined features. Features where the topology, geometry and constraints are predetermined, and are implicitly described in the feature library template definition. The features are said to "know about" themselves. Such features are typically formed from geometric modelling primitives, or are 2.5D in nature. Many of the products that are desired to be produced have much less constrained shape, being 3D in nature and are said to contain free form geometry. Formed components, rather than machined are typical examples. The ability of a feature modeller to represent such objects is highly desirable.

The general method of definition of user defined features, which was briefly described in chapter 3, is presented in detail in this chapter. The general method of definition[1] provides a baseline for development of features of extrinsically defined form. Developments for two types of extrinsically defined features are described. Non-parametric extrinsic features simply enable the inclusion and manipulation of previously defined ACIS bodies in the feature model. Parametric extrinsic sweep features enable ACIS objects to be swept and combined with the feature model. These features may produce different results by editing the parameters. Enhanced GUI support for features of both implicit and extrinsic form is provided by the use origins ("handles") . The feature definition method is formally extended to provide these advances for all user defined features.

4.1.2 Chapter Structure

The introduction to chapter 4 forms section 4.1, providing an overview of the chapter and details its structure. Section 4.2 presents the needs for feature modelling systems

---

[1]The extension of the general feature template description method, investigation of the domain of extrinsically defined features, and advancement of techniques using "handles" is the work of the author. The general intrinsic feature description method described in sections 4.3 and 4.4 is the work of the team.

to support features with extrinsically defined form. Section 4.3 describes the general template definition method which forms the baseline. In section 4.4 an example is presented of an implicitly defined form feature. These methods are contrasted with those of an extrinsically defined non-parametric feature described in section 4.5. Section 4.6 details the methods developed to represent extrinsically defined geometry for parametric sweep features. Support for feature origins ("handles") is presented in section 4.7. Enhanced GUI functionality has been developed which is applicable to both implicitly and extrinsically defined features. An extension to the feature template description method is presented in section 4.8 to enable the advances to be applicable to both user defined and pre-defined features. A chapter summary is provided in section 4.9.

4.2 The Need To Model Features With Extrinsically Defined Form

Features of extrinsically defined form are those features where the complete description of the topologic and geometric elements, and their constraints, is not prescribed in template definition or its methods. Such features allow previously defined ACIS bodies to be combined with a feature model. Swept features where either the base or the path are not fully described in the `make` function also have extrinsically defined form. Volumes and surfaces created by sweeping are an intuitive method of creating more complex shapes than 2½D, providing an intermediate stage between traditional feature modelling approaches and fully implementing free form surface functionality. If feature technology is to mature and become a general purpose simultaneous engineering tool, methods must be developed to support the definition and use of features of more complex form.

Customer driven demand for manufactured products has resulted in pressures to increase quality, reduce cost, provide greater diversity and reduce environmental impact. These demands have perhaps had their greatest influence in the automotive industry, though they are applicable to all branches of manufacturing. The modelling

and subsequent downstream use of free form geometry has increased dramatically in recent years. Increased use of complex one piece plastic parts for interiors, more aerodynamic and better fitting exteriors, a plethora of model derivatives and elimination of hand finishing are just some of the reasons. Components that meet these criteria are mostly manufactured by forming processes such as sheet metal pressing, plastic injection moulding, forging and casting. Although the designer thinks and designs the product, the task of designing and manufacturing the tooling to make the product is as important, if not more so. The emphasis on machining is transferred from the product to the tooling that is used to make the components.

In products manufactured by forming processes the component surfaces in the tooling will generally be different than those of the design because of factors such as shrinkage, component removal, spring back and the use of multistage tooling. The design of the tool is a complex task. Tools frequently consist of moving parts to enable the part to be removed where undercuts exist. The mechanics of these parts must be designed and modelled. The power systems (mechanical, hydraulic, pneumatic) require access as do heating and cooling passages. How such tools are designed is not addressed in this thesis. It is assumed that the designer knows what he wants to model. The feature based design system described in this thesis is to be used as a detail design tool and not as a knowledge based design engineering system (such as ICAD™) and analysis tool. However, through the use of features design intent is recorded, and formulae allow a limited amount of knowledge to be represented.

When dealing with complex 3D forms it is virtually impossible in 2D, and is still difficult in 3D surface modellers, to layout all the ancillary features (cooling passages and the like) of the tooling so that they do not conflict with the component surface functionality. A solid model is ideally suited for such applications. A feature modeller is seen as a superior tool for generating a solid model and should therefore provide support for free form features. The unconstrained nature of free form objects indicates that the topology and geometry cannot be defined implicitly in the feature template. The "how" may be defined, but not the result. For example, this form (face) is swept along this form (path) to produce this form (volume). The topology and

geometry of the resulting volume is completely dependent on the topology and geometry of the construction elements which are not implicitly defined in the template.

To date it is theoretically possible to work fully with synthetic geometries in solid modellers, but as significant performance disadvantages make interactive modelling virtually impossible, this is not a chosen solution. [FOL84] cites calculating surface / surface intersections as one reason. Stability of Boolean operations, stitching surfaces together to form solid bodies, model precision, and calculating silhouette lines are general practical problems.

Sweeping of one shape along a path (vector or curve sequence) or around an axis provides an intuitive and powerful way of generating complex geometry in a solid modeller. The geometry engine, ACIS, makes some restrictions. Splines that are used to define a sweep path must be planar. Furthermore, splines that form the base contour or surface being swept must be planar in all cases except for rigid sweeps along a path or vector.

To date FeatureM has no direct method for the user to generate spline curves or surfaces. It is relatively easy to generate synthetic curves and surfaces by performing model operations, but such methods are generally of little use to define required geometric elements. Importing such items from a surface modeller is a natural desire, particularly as the performance in generating the geometry interactively is so much better. 2D geometry is frequently manipulated in a sketcher because of the relative ease of defining constraints and more importantly, of solving them. Sweep operations make the most flexible and powerful use of geometry from any source, going some way towards full manipulation and use of free form geometry in feature based solid modellers.

Import of fixed (non-parametric) geometry into the feature modeller is of importance for two reasons:

- There are many ACIS models not generated by the feature modeller. This allows use of legacy solid models within the feature modeller and a general re-use of existing data.

- To date there are no fully implemented and accepted standards for the exchange of feature models. Any exchange with other parametric or feature modelling systems must be of pure geometry. Exchange of pure geometry is governed by a number of existing standards: VDA [VDA], IGES [IGES], STEP [STEP].

For these reasons it is highly desirable that a general method of supporting non-feature generated volumetric and geometric elements within the feature model data structure is developed. Implementation of sweep functionality is desirable because of the flexibility and power in creating complex geometry in a feature based solid modeller.

The non-implicitly defined nature of these features necessitate that significant functionality is developed to aid the user to manipulate such objects within the feature modelling environment.

4.3 General Template Definition

4.3.1 General

The MCL+ function `DefTemplate` is used to define a feature template. All features forming the feature library use this method whether provided by the system or defined by the user. `DefTemplate` has the following syntax:

```
int DefTemplate ( List feature, List parameters )
```

The argument `feature`, defines the attributes and method types that are generic to all features regardless of their parameters. The argument `parameters` defines the

different parameters that are associated with an individual feature. The value returned by the function call is 0 for unsuccessful definition and 1 for successful definition.

DefTemplate can only be called once in any session to define a particular feature. Feature templates cannot be overwritten in the feature modeller. The definition is created with a text editor and saved as a file. The file is read by the MCL+ interpreter and the internal definition of the feature is created. Instances of the feature in the model are created from the internal definition.

If any of the feature or parameter attributes are names of MCL+ procedures these must also be developed and saved in a file. It is desirable to keep the definition of the procedures separate from the definition of the template for two reasons.

Firstly, because MCL+ procedures can be over written but the template definitions cannot, it is of practical use when developing user defined features. It is relatively simple to define the template in terms of its attributes and parameters but much more difficult to develop the methods (MCL+ procedures). Loading files of procedures defining the methods is frequently done, overwriting the previous ones. This means that the system does not have to be restarted after each modification.

Secondly, it provides a cleaner interface between the declaration of the feature template and the methods which are used to generate the entity that is the feature. The methods could be much more diverse than pure MCL+ programming. MCL+ may make calls to other systems, using completely different processing logic, which is used to return the ACIS body of a form feature. This is a stage between purely procedural and purely declarative techniques that greatly increase the flexibility and hence applicability of the feature modeller.

4.3.2 Generic Feature Template Definition

This part of the feature description is generic because all features defined in the system require such a description to be prepared. This contrasts with the description of the feature parameters which will be specific to that feature.

The list `feature` consists of eight strings:

```
feature =
{ name, alias, type, parent, make, symbol, if_created, if_deleted }
```

where:

`name` the internal name of the template. The name must be unique. Note that this is the generic name of the feature not the name of an individual instance which will be created from this template and is uniquely identifiable in the model.

`alias` the internal name of the template may be given an alias. The alias is seen in any user interface, but programming must use the internal template name.

`type` the type of feature is specified as being one of:

- `BASE` The root feature of a workpiece model. All other features act with respect to this feature.
- `ADD` Additive feature. The volume produced by this feature is integrated into the rest of the model with a Boolean union operation.
- `SUB` Subtractive feature. The volume produced by this feature is integrated into the rest of the model with a Boolean difference operation.
- `LOC` Local operation directly on the BREP model.

GRP Collection of features.

parent Declares a previously defined feature from which this feature may inherit methods and parameters.

make This is the method that tells the modeller how the feature is to be evaluated. For user defined features an MCL+ function name is declared. In the case of a form feature of type BASE, ADD or SUB the ACIS body defining the geometry is returned to be integrated in to the model. Certain flags may be used instead of an MCL+ function name:

n indicates that the modeller should do nothing.

p indicates that a hard coded C++ function exists. This is of no use for user defined features, but allows system developers the option of tuning the performance.

e indicates that the feature should inherit the method of the parent.

This function is evaluated when the mode of the feature is set to model or combined. It is called on each model generation (when the feature is not suppressed), or when the graphical protocol is set to display the *ShowBody* after any change in the feature's status or that of its parameters, particularly value.

symbol This method works in exactly the same way as the make method except that any bodies returned by the procedure are not integrated into the rest of the model they are merely positioned in space.

This function is evaluated when the mode of the feature is set to symbol or combined. It is called on each model generation (when the feature is not suppressed), or when the graphical protocol is set to

display the *ShowBody* after any change in the feature's status or that of its parameters, particularly value.

`if_created` This is the method that tells the modeller what to do the first time the feature is instanced from the library. For user defined features an MCL+ function name is declared. The function is called only once per instance on creation. The codes `n`, `p` and `e` may also be used as previously described. It is used to perform complex initialisation of the feature or related elements. The initialisation is user defined and would otherwise not be known to the kernel (as setting of default values are, but not references or pointers).

`if_deleted` This is the attribute that tells the modeller what to do when the feature is deleted from the model. For user defined features an MCL+ function name is declared. The function is called only once per instance on deletion. The codes `n`, `p` and `e` may also be used as previously described. It is used to perform housekeeping operations that are user defined and otherwise unknown to the kernel.

4.3.3 Specific Feature Parameter Definition

Although the method of describing each parameter is the same, the differences in number, type, and meaning of parameters make this part of the feature template description unique to each feature in contrast to the generic part described in Section 4.3.2.

The list `parameters` consists of a series of sub-lists. Each sub-list describes one parameter:

```
parameters =        {
{name, alias, type, access, if_needed, if_set, default }, .. , .. }
```

where:

`name` Is the internal system name of the parameter. It must be used in any procedures and (to date) formulae that reference it.

`alias` The name of a parameter may be given an alias. However this is only used in the GUI to present this to the user. The alias may not be used in programming of procedures or formulae.

`type` The MCL+ data type of the parameter. The following data types are supported:

- `int` integer value.
- `real` real value.
- `String` string value.
- `Position` position in Cartesian coordinates.
- `Vector` vector or unitvector.
- `List` list containing elements of the same type. A restriction on the normal usage of a list.
- `Fea` pointer to a Feature
- `Ent` pointer to any Face, Edge or Vertex in a workpiece or ACIS Body.
- `Body` pointer to any ACIS Body but not a workpiece Body.
- `Face` pointer to any Face in a workpiece or ACIS Body.
- `Edge` pointer to any Edge in a workpiece or ACIS Body.
- `Vertex` pointer to any Vertex in a workpiece or ACIS Body.

`access` this sets the visibility of the parameter in the GUI. Where:

`w` read and write.

`r` read only.

`n` is not visible.

The `access` attribute may be set by the user via MCL+ programming.

In the property sheet the parameter names and values are shown clearly for read/write and may be set by the user.

The names and values are shown in half tone to indicate read only status; these values may not be set through the GUI or by direct programming externally to a feature's methods. The kernel is able to set read only parameters. This is desirable in such cases as the x, y and z axis rotation matrix vectors which must be maintained in a consistent state. Programming from within a feature's methods it is also possible to set read only parameters. This is valuable for representing numerical parameters in objects defined by or derived from non-numerical codes.

Non-visible parameters are not present in the GUI property sheet and cannot be set via the GUI or by programming outwith a feature's methods. The parameters still exist however and may be set by the kernel. This is useful where information should be hidden from the user. For example both the rotation matrix vectors and the Euler angles are maintained consistently by the kernel, however, it is desirable to see only one set at once. Redundancy is a frequently occurring example where a features methods maintain consistency. A cone may be defined by its base diameter and height, or by its base diameter and included angle, or by included angle and height. It is desirable to show only one pair of parameters, and only one pair at a time may be set for consistency.

Any parameter may have its `access` code changed by programming to read/write after which the parameter value may be set.

`if_needed` For User Defined Features this method defines the name of an MCL+ function. Alternatively, the `n`, `p`, `e` codes may be used as previously described. This function is used to calculate a value for the parameter if no value has been set, and no default value exists. It is a method of deriving values for parameters independently from the `make` function.

`if_set` For User Defined Features this method defines the name of an MCL+ function. Alternatively, the `n`, `p`, `e` codes may be used as previously described. This function is called immediately the user, either via the GUI, or by programming, sets the value of the parameter. It is not called if the parameter is set from within a features methods to avoid recursion. An example of its use is to set the `access` rights of other parameters in the feature after a change in `mode`.

The `if_set` procedure should not be used for parameter validation. Although the procedure appears ideal for this purpose its use as a validation check can lead to problems, particularly in features with many inter-related parameters, or where formulae are used.

`default` When a feature instance is created the values of the parameters are set to the implicit values denoted by the predefined values in the template. The user either accepts these, or sets new values, making an explicit instance. For pointer attributes, when no default can be given, the strings "`---`" or "`@UNDEF`" signify a `NULL` pointer.

4.4 Feature Template Definition With Implicitly Defined Form

Consider the block depicted in Figure 4.1. When ACIS creates such a body, the Body Coordinate System is located at the centre of the block as shown.

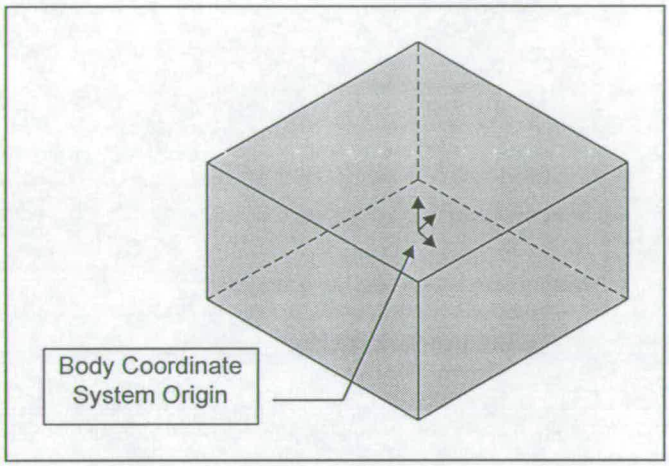


Figure 4.1 Block Feature With Form Defined Implicitly In Template

The feature template definition for the block is given in Figure 4.2.

```
outcome = DefTemplate(

// name    alias   type   parent     make     symbol  if_deleted
//                                                    if_created

  {"block", "---", "BASE", "---", "make_block", "n", "n", "n"},

// name   alias  type  access    if_set default
//                          if_needed

  {{"x1", "---", "real", "w", "n", "n", "50.0" },
   {"y1", "---", "real", "w", "n", "n", "50.0" },
   {"z1", "---", "real", "w", "n", "n", "50.0" },
}
```

Figure 4.2 User Template Definition For A Block Feature

Additionally for features of type `BASE`, `ADD`, `SUB` and `GRP` the system automatically adds the parameters shown in Figure 4.3.

```
// name      alias  type        access      if_set   default
//                                 if_needed

   {"g",      "---", "Fea",      "w", "n", "n", "@UNDEF" },
   {"pos",    "---", "Position", "w", "n", "n", "0.0,0.0,0.0" },
   {"wx",     "---", "real",     "w", "n", "n", "0.0" },
   {"wy",     "---", "real",     "w", "n", "n", "0.0" },
   {"wz",     "---", "real",     "w", "n", "n", "0.0" },
   {"Xaxis",  "---", "Vector",   "r", "n", "n", "0.0,0.0,0.0" },
   {"Yaxis",  "---", "Vector",   "r", "n", "n", "0.0,0.0,0.0" },
   {"Zaxis",  "---", "Vector",   "r", "n", "n", "0.0,0.0,0.0" },
   {"next",   "---", "Fea",      "n", "n", "n", "@UNDEF" }
```

Figure 4.3 Parameters Automatically Added To The Template Definition

The parameter `g` is aliased via NLS and is observed as reference. It holds a pointer to a feature, other than the `BASE` feature, used as a reference for positioning and orienting. The parameter `pos` contains the position of the feature. The parameters `wx`, `wy`, and `wz` contain the Euler angles defining the orientation of the feature. The orientation is also represented by the read only parameters `Xaxis`, `Yaxis` and `Zaxis`, the rotation matrix vectors. The parameters `pos`, `wx`, `wy`, `wz`, `Xaxis`, `Yaxis` and `Zaxis` are all defined with respect to the Workpiece Coordinate System or the Feature Coordinate System referred to in the parameter `g` (reference). The parameter `next` is used in group features to point to the next feature in the group.

The make function is shown in Figure 4.4. No parameter validation is performed for clarity. It returns a body the shape of a block created by ACIS.

Within the feature modeller `make` functions, the coordinate system may be regarded as being at the World Coordinate System origin. On returning from the `make` function to the kernel, the origin of the coordinate system in the `make` function is transformed with respect to the workpiece or reference feature and the parameters: `pos`, `wx`, `wy`, `wz`. As a consequence, if a body is moved from the origin in the `make` function it will appear offset in the resultant model from the location indicated by the feature parameters. In the example the Body Coordinate System of the block is not moved, therefore it is identical to the Feature Coordinate System which is located and oriented in the model with respect to the parameters: `pos`, `wx`, `wy`, `wz`. One must be aware of the interaction of coordinate systems within the `make` functions.

```
proc List make_block ( Fea this_fea_pointer, String this_fea_name )

   local real x1 = GetParVal( "x1", this_fea_pointer)
   local real y1 = GetParVal( "y1", this_fea_pointer)
   local real z1 = GetParVal( "z1", this_fea_pointer)
   // get the values of the parameters defining the block

   local Body block = Block( x1, y1, z1 )
   // create the block with the ACIS Body Coordinate System origin

   return {block}
   // give the block to feature modelling kernel

end proc // make_block
```

Figure 4.4 Block Feature make Function

Positioning the block accurately in the model is tedious because the user has always to calculated the position of the centre with respect to the dimensions of the feature and the rest of the model. This problem will be addressed with the use of *origins* ("handles") as described in section 4.7.

4.5 Feature Template Definition With Extrinsically Defined Non-Parameteric Form

The purpose of this feature is to be able to integrate any previously existing ACIS body within the feature model data structure and to combine its geometry with the rest of the geometry created by evaluating the feature tree. The form of the feature is said to be fixed, or non-parametric, because the form cannot be changed by, or within the feature modeller. However, the position and orientation may be changed using any of the techniques available to all features. The method uses the geometry of the ACIS body to define the form and size of the feature and use the feature parameters to position and orient the body instead of using its own transform.

The feature template is shown in Figure 4.5. The feature only has one user defined parameter: `Solid`. The system automatically creates the parameters pos, wx, wy, wz, etc. as detailed in Figure 4.3. This feature may be said to be a non-parametric feature

because the form and size of the feature cannot be changed by feature modelling functionality.

```
outcome = DefTemplate(
// name  alias   type  parent   make     symbol          if_deleted
//                                                 if_created
  {"np", "---", "BASE", "---", "make_np", "symbol_np", "n",  "n"},
// name     alias  type access if_needed    if_set      default
  {{"Solid","---","Body","w",   "n",  "if_set_np_solid","@UNDEF" }})
```

Figure 4.5 User Template Definition For A Non Parametric Form Feature

The feature modelling kernel treats parameters of type Body differently to other pointer type parameters to objects such as Faces, Edges, Features, etc. Pointers to objects other than Body can be regarded as true pointers because the objects remain in their previous state, fully accessible in the browsers and via MCL+, selectable and operable on. In contrast, when a feature body parameter pointer is assigned to point to a body, the body can be said to be consumed by the feature. The body is no longer in any data dictionary visible to the user, cannot be selected or operated on. The body is only accessible via the feature parameter by the special function: `CopyParSolid`. This creates a copy of the body.

An `if_set` function is used to set the feature transform to be that of the body. The `make` function returns to the kernel a copy of the body. The copy of the body has no transform applied. Its position and orientation being defined by the `pos`, `wx`, `wy` and `wz` parameters of the feature with respect to the workpiece or reference feature. The `if_set` function is shown in Figure 4.6 (without validation checking for clarity).

```
proc List if_set_np_solid (Fea this_fea_ptr, String this_fea_name)

    local SolidObj solid = CopyParSolid("Solid","",this_fea_ptr,2)
      // the body has its original transform with respect to
      // the World Coordinate System

    this_fea_ptr.Transf = solid.Transf
      // feature gets position and orientation
      // from solid relative to world coordinate system
      // internally converted relative to
      // workpiece or reference feature coordinate system

    return {}

  end proc
```

Figure 4.6 Non-Parametric Form Feature `if_set` Function

The `make` function is shown in Figure 4.7 (also without validation checking for clarity).

```
proc List make_np (Fea this_fea_ptr, String this_fea_name)

    local SolidObj solid = CopyParSolid("Solid","",this_fea_ptr,1)
      // the body has the identity transform with respect to
      // the Feature Coordinate System

    return {solid}

  end proc
```

Figure 4.7 Non-Parametric Form Feature `make` Function

In both the `if_set` and `make` functions the body that is the actual parameter (pointer to) of the feature is not changed. Instead, a copy of the body is made from the parameter. In the `if_set` function the copy is made and the transform of the original body is applied; this is with respect to world coordinates. The feature is assigned the transform of the copied body. Internally the kernel converts this transform in world coordinates to the correct position and orientation parameter values with respect to the workpiece or reference feature coordinate systems. The Feature and Body Coordinate system origins now coincide in world coordinate space. Therefore when the feature model is generated, the geometry generated by the feature appears exactly where the original body was located in space (World Coordinates). The user is now free to position and orient the feature (and consequently) body within the feature

model using any of the methods previously described. In particular the `LcsToFea` and `FeaToLcs` functions work consistently.

This method is very flexible because any valid ACIS body, regardless of complexity, can be integrated into the feature modelling structure. Thereafter the feature and its geometry may be positioned and oriented freely as with all other features. Ease of positioning and orienting is reliant on the relationship between the original Body and its Coordinate System. The Body Coordinate System origin is positioned relative to the geometry by ACIS. Where it is, is dependent upon the modelling operations and how ACIS creates the fundamental building blocks. It is not necessary for the origin to be within the body or on the surface of it, let alone at some useful point. Nor is it necessary for the origin to be aligned with any edges or face normals. This is perfectly sensible for objects such as a hemispherical shell where the body origin is located at the centre point of the imaginary sphere. This can lead to difficulties in locating and aligning the solid bodies of features because the user must locate and align the feature origin when it is not easily known how it is related to the geometry. The offset vector and angles can be calculated through the use of multiple coordinate systems but it is tedious. This problem will be addressed in section 4.7.

4.6 Feature Template Definition With Extrinsically Defined Swept Form

4.6.1 Requirements

Sweep functionality offers the designer the ability to create (semi-) free form objects relatively simply in solid modellers. The different types of sweep operation and the modifications that are possible to geometry through sweeping have been discussed in chapter 2. Implementing sweep functions as features poses a number of problems:

1. Number of objects required.

   Sweep functions require either:

   - one object which is swept using a vector, or spun about an axis, or,
   - two objects, one of which is swept along the other.

   The variation in the number of objects raises several questions: If there is more than one object, should both objects be contained within the same feature? If both objects are supported within the same feature, how is one object positioned relative to the other? If the objects are supported in different features how should the relationship be maintained? Should the position of one object be dependent on the other so as to maintain the same resultant body despite a change in position and orientation of the independent object?

2. Type of objects.

   There is a variety of objects which may be used validly with sweep functions:

   - Sheet or lamina bodies (of nil volume) consisting of two faces (identical in form and location), or single faces. Bodies consisting of single faces may be double or single sided. Sweeping of a sheet or lamina body can produce an independent volume after sweeping (generally desirable), or an open shell.
   - Faces of bodies may be swept. A face sweeping operation makes local changes to the underlying BREP model and does not produce an independent volume.
   - Wire bodies consisting solely of edge sequences may be swept and may also form the paths for other objects to be swept along. Sweeping a wire body produces a sheet or lamina body.

   The variety of object that may be swept gives rise to a number of questions: Should these different types of objects be supported in a single feature family?

What, if any validity checking is done? Should validity checking be performed on the body(ies) forming the parameters and/or on the resultant body?

3. Source of objects.

   The variety of objects discussed in point 2 may be generated from many sources:

   - Non-lamina sheet bodies will generally be created by copying an existing face and creating an independent body, or by importing from a surface modelling system.

   - Lamina bodies are typically created with the use of a sketcher. The sketcher is not internal to the kernel and generally produces contours as closed wire bodies.

   - Open wire bodies may also be created using a sketcher or by copying an existing edge, or edge sequence and creating an independent body.

   - Faces may be pointed to directly for local sweep operations.

   The following points must be considered: Must all objects be in the same workpiece? Can all these sources of object be treated in the same way? Where does the responsibility lie for covering a closed contour to create a lamina body, hence producing a volume by sweeping?

4. Definition of other parameters

   Positions and vectors are parameters of the sweep functions used to define the sweep direction or spin axis. With respect to which coordinate system should these be defined: World, Workpiece, Feature? If the swept object via its feature is relocated or aligned, should the volume or sheet generated remain the same? These three coordinate systems correspond to features that generate an independent volume. Local sweeps of faces are more difficult because the face is not contained within, or produced by, the sweep feature. It is independent,

therefore there is no coordinate system directly related to the face. Should a coordinate system be defined that is related to the face? If so, how?

5. Creation of Positive Volumetric Bodies

It is important within the feature modelling system architecture to produce positive solid volumes from sweeps with the exception of the specific functionality detailed in point 6. Single sided faces when swept produce infinitely thin semi-bounded shells like an open cardboard box. It would be extremely problematic both for the user and the feature modeller to use such objects in an effective manner. In order to create a volume either one side of a double sided face, or one face of a sheet body consisting of two identical faces, must be swept. Which side or which face that is swept determines if a positive or a negative body is created. It is undesirable to create a negative body because of problems on integration into the feature model.

How can it be ensured that a positive volume is always generated by a sweep feature?

6. Use of Non-Manifold Objects

As described in chapter 2 ACIS has the ability to perform Boolean operations with semi-bounded objects. This functionality can be easily integrated into the feature modeller. It provides a means to sculpt solid objects in a very efficient manner. Faces and open contours can be used to remove volumes without the user having to create a volume. In order to achieve this faces that are single sided must be used. Existing faces may be copied, but this does not provided very significant benefits unless the face is made from spline geometry. The biggest advantage is when an open contour is used as the swept object in a sweep operation to generate a sheet body consisting of single sided faces. The sweep produces many faces which bound part of space which can be subtracted from the model in a single operation. The side of the sheet body where all the face normals point outwards indicates space. The opposite side is defined to be solid and because of the semi-

bounded nature this extends to infinity. The semi-bounded solid creates one restriction in ACIS however. No boundary edges of the sheet object may be contained within the model being integrated, that is, the sheet object must completely cut through the model, Figure 4.8.

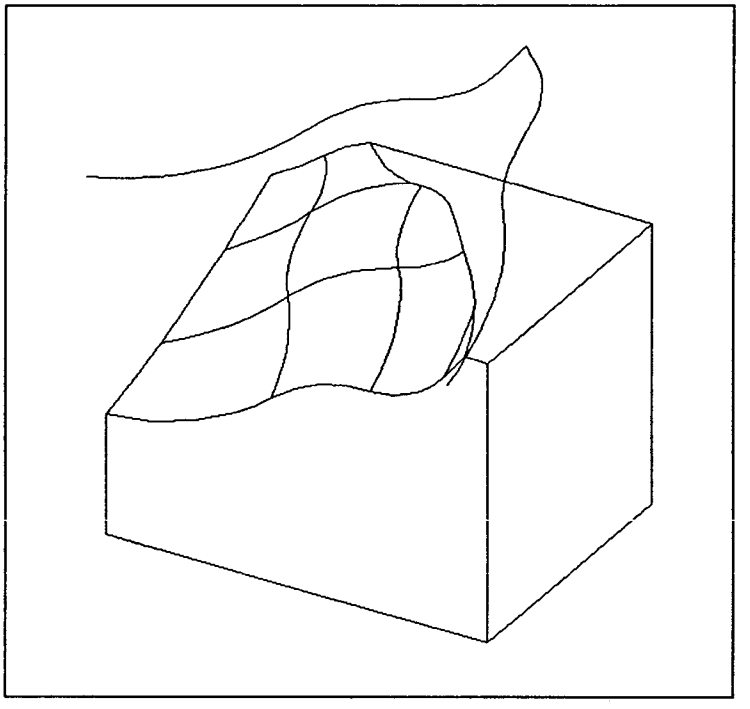


Figure 4.8 Combining Swept Semi-Bounded Sheet Bodies In A Model

How can this functionality be supported within the feature modelling architecture?

The implementational solutions to the questions raised in the above points are described in section 4.6.2.

4.6.2 Implementation

With respect to the points discussed in Section 4.6.1, sweep functionality has been implemented as follows:

1. Where more that one object is required to define a sweep feature, each object will be supported as a separate feature. This is the case with a sweep of one object

along another object, a face created from a 2D contour swept along a path formed by a wire body created from a sequence of connected edges for instance. It is not necessary that the path is planar, only that any spline segments in it are. The use of two features allows each object to be positioned independently using the feature parameters of `pos` and `wx, wy, wz`. To maintain consistency with sweep features needing only one object (the face or contour that is swept), the swept object rather that the path object is made the reference object to maintain the relationship between them. This is a bi-directional relationship. The swept feature has a parameter which points to the path. The path feature has its reference set to point to the swept feature. This means that if the position or orientation of the swept feature (the base) is changed the position and orientation of the path will remain the same relative to the base. Hence the same volumetric body (or sheet body) will be generated, only located and aligned differently in the model. Changing the position or orientation of the path, changes the shape of the resultant body.

2. The types of object that may be swept can be split into two categories:

   I. Independent bodies that form sheet (or lamina) bodies, and wire bodies. The result of sweeping produces an independent volumetric or sheet body which may be added or subtracted from the model.

   II. Topologic face elements of existing bodies. The result of sweeping produces local changes in the BREP model.

   Sweep features of the first category will be modelled as features of type `BASE`, `ADD` or `SUB`, with parameters of type `BODY` consuming the objects within the feature as described in section 4.5. Sweep functions in the second category will be modelled as features of type `LOC`, with parameters of type `FACE` denoting the face pointed to in the same workpiece model.

   Validity checking is performed to establish that for local sweep operations of existing faces, the faces belong to the same workpiece as the sweep feature.

Similarly that for sweeps that generate a fully bounded or semi-bounded volume that the generating bodies are either sheet or wire bodies.

3. The advantage of using the parameter of type BODY for the independent (not part of a workpiece model) objects is that they are all treated in the same way, regardless of their source. Although the bodies are consumed by the feature as described in section 4.5, they are treated in a more advanced manner because of the need to aid positioning, orienting and, in the case of the sweep along a path, the relative positioning of both features. This is described in section 4.7.3.

4. The feature modeller must be as consistent as possible in its treatment of features, whether they are of implicitly defined topology and geometry or not. This means that a feature's form should not be a function of its position or orientation in the model, unless that is explicitly stated by the user (by using formulae or more general programming). In order that this be the case for features of non-implicitly defined topology and geometry, any position and direction parameters must be defined in a coordinate system that is defined with respect to the object being swept.

   This is achieved relatively easily for sweeps where the object being swept is contained within the feature and the object is independent of the rest of the model. This is because the feature transform positions and orients the object being swept so that any parameters can be defined in the Feature Coordinate System by multiplying world values by the inverse transform before assigning to the feature.

   For sweeps of existing faces that generate local operations on the model, features of type LOC do not have a coordinate system associated with them. Therefore a coordinate system is generated from the properties of the face. Parameters may then be converted and stored as feature parameters in a coordinate system defined by the face. The Face Coordinate System must be maintained every time the feature is to be evaluated, parameters set, or read.

Sweep functions within ACIS may be generated with the following modifiers: rigid, orthogonal, with twist, without draft, with straight draft, with curved draft, as described in chapter 2. These are represented by string parameters indicating a mode, the angles are described by reals. Sweeps with twist can only be performed along a path which starts orthogonally to the swept object. Careful masking of the property sheet is needed to allow access to parameters only when necessary.

5. The generation of negative bodies is not desirable in the feature modelling environment because it does not coincide with the users view of the real world and with general assumptions made in the kernel and the graphics. The `make` functions control the returned bodies to ensure that they are positive. This involves checking that either two identical faces exist in the sheet body or that the face is double sided. For planar faces the face normal is easily obtainable. For non-planar faces checking of the face normals at all vertices provides the best method of determining the "sense" of the face with respect to the direction it is to be swept. The evaluation of sweeping of faces containing undercuts is performed by ACIS. Either a valid result is produced or an error generated.

6. A sweep feature by definition generates objects by sweeping. When faces (closed contours) are swept they produce a volume in the context of this feature modelling environment. Faces are created by sweeping an open contour. Sweeping an open contour creates a set of single sided faces. The use of single sided semi-bounded objects in the feature modeller makes sense only for subtraction operations. If the feature is of type BASE or ADD, the set of faces is converted to double sided which provide a body that can undergo Boolean operations and produce a valid, if non-manifold, result. If the feature is of type `SUB`, an extra parameter is exposed that indicates whether the feature should remain single sided or be double sided. The user may reverse the set of faces to control which direction the face normal points, therefore controlling which part of the model is retained.

4.6.3 Template Definition

The following general sweep functionality has been implemented:

a) Sweep of sheet or wire body along a vector path.

b) Sweep of sheet or wire body about an axis.

c) Sweep of sheet or wire body along a path defined by wire body.

d) Sweep of existing face along a vector path.

e) Sweep of existing face about an axis.

f) Sweep of existing face along a path defined by wire body.

These require the definition of 12 templates:

- 3 each for sweep functions a, b, and c of type `BASE`, `ADD` and `SUB`.
- 3 for sweep functions d, e and f of type `LOC`.

Inheritance makes the definition of the `BASE`, `ADD` and `SUB` templates simple. As an example, the template definition for sweep function "c" is shown in Figure 4.9.

```
DefTemplate(
  {"b1413","WireSweep","BASE","---",
   "mk_sweep_wire","mk_base_symb","cre_wire","del_wire"},
  {
    {"base"   ,"", "Body"  , "w", "n", "if_set_sweep_base", "---"},
    {"wire"   ,"", "Fea"   , "r", "n", "n",                  "---"},
    {"wc"     ,"", "real"  , "r", "n", "n",                  "0.0"},
    {"form"   ,"", "String", "r", "n", "n",                  "e"  },
    {"kind"   ,"", "String", "w", "n", "if_set_sweep_kind",  "r"  },
    {"reverse","", "String", "n", "n", "n",                  "fw" }
  })

DefTemplate(
  {"a1413", "WireSweep","ADD","b1413","e","e","e","e"},
  { })

DefTemplate(
  {"s1413", "WireSweep","SUB","b1413","e","e","e","e"},
  { })
```

Figure 4.9 Template Definition For Sweep Of Sheet Or Wire Body Along A Path Defined By A Wire Body

An example of a sweep operating locally on a face is shown in Figure 4.10 for sweep function "f".

```
DefTemplate(
  {"11413","WireSweep","LOC","---",
   "mk_sweep_wirefa","n","cre_wire","del_wire"},
  {
     {"f",       "","Face"   ,   "w","n","if_set_sweep_face","---"  },
     {"wire",    "","Fea"    ,   "r","n","n",               "---"  },
     {"wc",      "","real"   ,   "r","n","n",               "0.0"  },
     {"form",    "","String",    "r","n","n",               "e"    },
     {"kind",    "","String",    "w","n","if_set_sweep_kind","r"    },
     {"Csorg",   "","Position","n","n","n",                 "0,0,0"},
     {"CSxaxis","","Vector",   "n","n","n",                 "1,0,0"},
     {"CSyaxis","","Vector",   "n","n","n",                 "0,1,0"}
  })
```

Figure 4.10 Template Definition For Sweep Of Existing Face Along A Path Defined By A Wire Body

The parameters `Csorg, CSxaxis, CSyaxis` are required to provide a definition of the Face Coordinate System that is used to define any parameters that should be invariant to re-posisitioning of the face.

Additionally a thirteenth template is required to contain the path wire body, Figure 4.11.

```
DefTemplate(
 {"a162", "","ADD", "---",
  "mk_wire_symb","mk_wire_symb", "set_symb",    "n"},
    {"wire",     "","Body",     "s","n","if_set_a162",      "---"  },
    {"wirepos",  "","Position","n","n","if_set_wirepos",   "0,0,0"},
    {"wirexaxis","","Vector",  "n","n","if_set_wirexaxis","1,0,0"},
    {"wireyaxis","","Vector",  "n","n","if_set_wireyaxis","0,1,0"},
    {"wirezaxis","","Vector",  "n","n","if_set_wirezaxis","0,0,1"}
})
```

Figure 4.11 Template Definition For Feature Defining A Sweep Path

The parameters `wirepos, wirexaxis, wireyaxis, wirezaxis` are required to provide parameters that the user may set to control the position and orientation of the path feature partnering a local sweep of an existing face.

4.6.4 Methods

The methods described in this section will focus on the template definitions presented in section 4.6.3. Sweeps along a path form the most complicated type of sweep and offer the most powerful geometry creation methods.

4.6.4.1 Sweep Sheet Or Wire Along Path Defined By Wire Body

The template definition shown in Figure 4.9 requires the definition of six methods:

```
"mk_sweep_wire", "mk_base_symb", "cre_wire", "del_wire"},
"if_set_sweep_base", "if_set_sweep_kind"
```

When the feature is first instanced, its parameters are set by default except for the parameter `base` that indicates the sheet or wire body to be swept. During the instantiation process the `cre_wire` (`if_created`) method is called. This creates the partner feature that is to contain the wire body forming the sweep path. It sets the reference parameter in the path feature (template definition is shown in Figure 4.11) to point to the swept feature and sets the access to read only (not necessary if the swept feature is of type `BASE`). This prevents corruption of the model through use of the property sheet. Similarly the `wire` parameter in the swept feature is set to point to the path feature and its access is also set to read only.

When a feature has been instanced the kernel calls the `mk_sweep_wire (make)` method to attempt to generate its *ShowBody*. The `make` method returns a `NULL` pointer because the `base` parameter is also `NULL`. The first operation whether by the user acting directly through the property sheet or via the input decoder is to set this parameter.

Setting the base parameter by clicking on the button corresponding to the parameter name in the property sheet (or by picking in a decoder input step) the `if_set_sweep_base` method is called. This method is more complex than that

detailed in Figure 4.6 because the transform of both the `base` body and the feature are changed in order to position the swept object and the path object accurately to define precisely the geometry of the resulting feature As before the body is copied from the feature with its transform relative to the World Coordinate System. If its transform is exactly the same as that of the feature (set to the transform of the Local Coordinate System on feature instancing) then the method is exited to prevent recursion.

In order to aid positioning and orienting it is desirable to have the Feature Coordinate System located at a position on the boundary of the body. For a face body, one of the coedges of the closed boundary is chosen. For an open contour it is necessary to search for the starting coedge rather that the first edge in the list of edges pointed at from the body. The list of edges may be in any order whereas the coedge chain is directed from the start to the end. Hence the correct start may be found. This is very important for the wire bodies that form the path along which to sweep. If the coedge chain start point is not found it is extremely difficult to position the path relative to the base within the `make` method.

The feature coordinate system is located at the start point of the start coedge and the x axis is aligned with the start tangent. For a face the z axis is aligned with the normal to the face at that point. For a contour (open or closed) a plane is generated using the first three non-linear points (start, middle or end) taken from the coedge chain. The normal to this plane is used. If no plane can be generated then the system makes a consistent transform with the x axis direction as stated. The alignment is achieved by generating a transform using the start point, x and z axes as described. This transform relative to World Coordinates is assigned to the feature. Internally the system will set its position and orientation parameters to be in Workpiece Coordinates or Reference Coordinates. The transform of the `base` body is also changed to that assigned to the feature. This has the effect of changing the internal representation of the elements of the body so that they remain in the same place relative to World Coordinates despite having a different transform.

It is important to maintain the consistency of any other features in the model that reference the transform of the feature. This is because the geometry of the feature stays in the same place in World Coordinates but has a different transform. Any features referencing this must also have their transforms changed to maintain their positions and orientations relative to the sweep feature. This is obviously the case of the path feature which references the swept feature and where the swept feature forms the root feature of the workpiece, in which case all features must have their transforms changed. As will be described in chapter 5 this is also true of all dimensions associated with the resultant body of the workpiece.

The `mk_sweep_wire (make)` method is called after any action that sets a feature invalid. Assuming that the base parameter has been correctly set (the make routine validates that it is a sheet or wire body) it is copied out but untransformed. If the path feature does not have a valid path the `mk_base_symb (symbol)` method is called to return only the body of the swept object. It must be remembered to delete the bodies created within the feature methods if they are not returned to the kernel for integration with the model.

The `mk_base_symb` method copies the `base` parameter. If it is a closed wire body it is covered to create a single sided face. The face is then converted to a double sided face which may be integrated by Boolean operations into the model generating a non manifold result. If the `base` parameter is an open wire body the wire body is returned.

A minimal amount of validation is performed in the `mk_sweep_wire` method because ACIS provides much better checking not only of the elements generating the sweep but of the resultant body. However, there is no NLS support for ACIS error messages and some checks may be performed relatively easily. It is bad practice to duplicate tests that ACIS performs in the `make` methods because the `make` method is called after every invalidation of the feature and during every model generation. Functionality that has the effect of prolonging the run time of the `make` method should be avoided. A simple validation test is that only planar faces may be swept orthogonally or with twist.

A more problematic test is to define which side of a double sided non-planar face is to be swept to generate a positive volume. By assessing the sense of the all the face normals (at each vertex on the boundary) with respect to the start tangent of the sweep path (or sweep vector) the face may be reversed to ensure generation of a positive body. This only works for faces without "undercuts". The sweeping of undercut faces generates a trappable error in ACIS.

When an open wire body is swept along a wire to produce a face, if the type of feature is `BASE` or `ADD` the single sided face is converted to a double sided and returned. If the feature is of type `SUB`, a user controlled parameter dictates how the set of single sided faces should be treated: left as they are, reversed, or double sided.

The `if_set_sweep_kind` method controls the visibility of associated parameters in the property sheet. If a rigid sweep is selected then the form and draft angle are not applicable and are set read only. If twist is selected only the twist angle is set to read/write. If orthogonal is selected, both the draft angle and form (rounded or straight) are set to read/write.

The `del_wire` method performs house keeping operations when a feature is deleted. The related path feature is also deleted. It is rarely necessary to delete a sweep feature even if different sweep objects are required. If it is not possible to edit the contour (returning to the contour/sketcher modules whilst a sweep feature is active regenerates automatically the 2D boundary, which on returning to the feature module is rebuilt in the feature) a new body may be set.

4.6.4.2 Sweep Face In Existing Body Along A Path Defined By A Wire Body

The template definition shown in Figure 4.10 requires the definition of five methods:

```
"mk_sweep_wirefa", "cre_wire", "del_wire",
"if_set_sweep_face", "if_set_sweep_kind"
```

The `cre_wire`, `del_wire` and `if_set_sweep_kind` methods are the same as those described in section 4.6.4.2. They are only declared once in the program because they are MCL+ procedures and are globally available.

The `if_set_sweep_face` method is used to define the coordinate system that will remain invariant with respect to the face. The Face Coordinate System is constructed using the three additional parameters "`CSorg`", "`CSxaxis`", "`Csyaxis`" shown in Figure 4.10. Using a position and two directions it is possible to define a transform with respect to the World Coordinate System. The feature is of type LOC and therefore does not have a coordinate system by default or a reference feature. Faces are formed with closed (for manifold models) boundary edges. Searching for the start coedge is therefore dependent on which edge is chosen from the list as the start point, unlike an open loop. A trait of ACIS is that it orders the model consistently and identically each time it builds a model exactly the same way using exactly the same elements. This means that the list of edges attributed to the face appears in the same order. For a closed loop the first edge in the list is chosen and its corresponding coedge used to define "`CSorg`", "`CSxaxis`", "`Csyaxis`" with the coedge start point, start tangent and cross product of start tangent and start normal. The user does not see this representation of the Face Coordinate System nor is the user able to set any of its values due to careful setting of the access attribute of the parameters.

The `mk_sweep_wirefa` method is concerned with two actions. Firstly, maintaining the coordinate system with respect to the face. This is done using the method described above. Secondly, performing the sweep operation in World Coordinates because it is a direct operation on ACIS. This means positioning and orienting the wire body of the path correctly. The `wirepos`, `wirexaxis`, `wireyaxis`, and `wirezaxis` parameters in the path feature define the transform (position and

orientation) of the wire body with respect to the Face Coordinate System. Therefore to determine the location of the body in World Coordinates the wire body transform is multiplied by the face transform defined by the parameters "CSorg", "CSxaxis", "Csyaxis". The path feature has four parameters to define its transform in order that the user can set these to position the path body with respect to the face. This is in contrast to the definition of the transform of the face. The sweep proceeds as defined by the parameters: kind, form and wc.

4.6.4.3 Sweep Path Defined By Wire Body

The wire body that forms the sweep path is somewhat more constrained than the bodies forming the swept object. If it is open there are only two possibilities for suitable origin points: the start and the end point. This is because the paths must touch the base for the sweep to be acceptable to ACIS. Clearly it is easier to position the path when it is known where the start point is. The transform of the wire body is set in a similar manner to the swept object in the parent but using the if_set_a162 method.

The path feature does not itself add or subtract any body(ies) from the model (the parent sweep feature does that) but it is useful to visibly see in the model where the path is positioned to enable the user to change its position and hence the model. This is achieved by using the body of the path as a symbol. This mode of the feature is set to symbol on feature creation using the set_symb (if_created) method. The mk_wire_symb method positions the wire body in either workpiece coordinates or reference coordinates (if the reference parameter is set). The kernel then positions objects automatically in these coordinate systems. Where the wire body defines the path of a local sweep of a face the parameters wirepos, wirexaxis, wireyaxis, and wirezaxis define the transform (position and orientation) of the wire body with respect to the Face Coordinate System. Four parameters are used where only three (one position and two orthogonal directions) are required to define a transform. This is because the button displaying the parameter name in the property sheet is

used by the user as a command to set that parameter via picking and pop-up modifiers. This method should always be used because, although direct input is possible, the user does not know how the face coordinate system is defined. Input from the pick functions sets the values in Workpiece (or Reference) Coordinates. On setting, `if_set` procedures of the parameters convert the values into the Face Coordinate System Structure.

4.7 Feature Origins

4.7.1 Requirements

Feature origins may be attributed to the work of Chang [CHA90] though the term "Handle" was used. The concept is used in most academic feature modellers to a lesser or greater extent. Any predefined shape usually has a number of convenient locations on its body. If the body represents that of a particular form feature the user may wish to use these origins to locate the feature in the model and to define this as the feature coordinate system origin for defining the feature. A simple example of a block and its origins is shown in Figure 4.12.

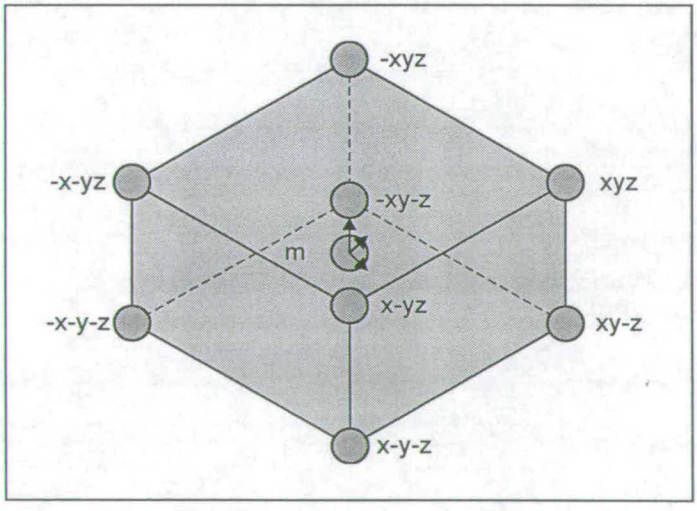


Figure 4.12 Simple Block Feature With Origins

In practice it is possible to define many more origins at positions such as the midpoints of edges and faces. Intuitive points that do not lie on the body may also be used such as the centre of radius, Figures 4.13.

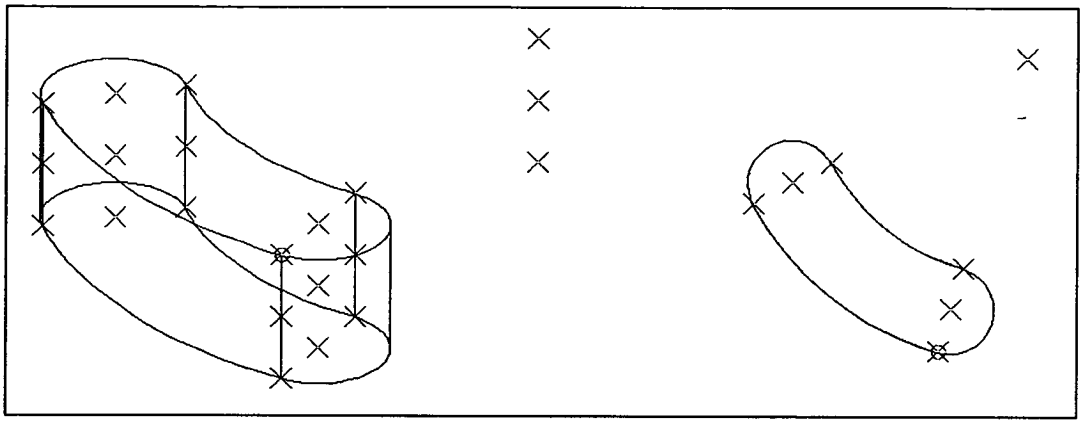


Figure 4.13 Origins Located At The Centre Of Radius

When ACIS creates a body, the Body Coordinate System origin is generally not at the desired Feature Coordinate System origin. Therefore, in the `make` function of the feature a compensation must be made by moving the body by an offset vector.

Feature origins are of major significance because of the ability of the system to snap the LCS to the Feature Coordinate System origin and to snap the Feature Coordinate System origin (and hence feature) to the LCS. This is used as the preferred method of locating and orienting features. It should also be remembered that the use of special positioning features and referencing of one feature to another is specifically related to Feature Coordinate System origins. The importance of the feature origins demands that GUI support is also strong.

GUI support for feature origins has the following requirements:

- To be applicable to user defined as well as system developed features;

  With user defined features the system does not know if the feature has origins, what the valid descriptors of the origins are, what the descriptors mean, how to calculate the position of each origin in World Coordinates, and what to do with related features.

- to be able to indicate graphically the possible choice of origins;

  Origin positions may not necessarily coincide with vertices, or midpoints of edges, centres of arcs, or even lie on the body. Purely graphical indicators of all valid origins should be generated.

- to be able to pick the graphically displayed options;

  Picking of purely graphical entities is problematic and requires relevant pick filtering.

The above assumes that the features are of a known topology and geometry. Hence, the creator of the feature template (implicit definition) can define the origins with respect to this known topology and geometry (shape but not size). Where free form geometries are concerned, either imported or generated from a sketcher this is not the case. The extrinsically defined random nature of the features increase the importance of being able to choose an origin because of the complexities of positioning and orienting. Origin definition must, therefore, function perfectly with the `LcsToFea` and `FeaToLcs` methods. It is, therefore, necessary:

- to provide a mechanism that provides the user with a good graphical method of locating and orienting non-uniform, extrinsically defined features.

- to ensure compliance with the kernel functions relating to the LCS and to the feature.

4.7.2 Features With Implicitly Defined Form

It is assumed that the user when defining a feature knows the topology and geometric elements, if not their sizes (usually, hence the parametric nature) that are to be described. An example of the template definition for the simple block shown in Figure 4.12 is presented in Figure 4.14. The parameter `Ursprung` defines the origin.

```
outcome = DefTemplate(
{"block", "---", "BASE", "---", "make_block", "n", "n", "n"},
{{"x1", "---", "real", "n", "n", "50.0" },
 {"y1", "---", "real", "n", "n", "50.0" },
 {"z1", "---", "real", "n", "n", "50.0" },
 {"Ursprung", "Origin", "String", "n", "n", "-x-y-z" }
}
```

Figure 4.14 Template Definition Of Block Feature With Origins

The system automatically creates the parameters `pos`, `wx`, `wy`, `wz`, etc. as detailed in Figure 4.3.

It is also assumed that the user knows where the origins should be located with respect to the definition of the feature's topologic and geometric elements. The user builds this information into the `make` function as a matter of course. A simple example of a make function for the block and origins shown in Figure 4.12, is presented in Figure 4.15.

The MCL+ function `make_block` creates a block with origins at the eight vertices and at the centre. The centre also coincides with the Body Coordinate System origin. For simplicity no parameter validation is performed. The function returns a List containing the body which forms the volume of the feature which the kernel combines into the model.

In this function the user has told the system what the feature is, and how it behaves with respect to its parameters. The position and orientation of the Feature Coordinate System does not change, merely the local position and orientation of the body of the feature relative to its origin.

A feature stores the body generated by the `make` function as an attribute. It is known as the *ShowBody* and is used in circumstances when it is impossible to pick a feature because all of its edges form edges of the workpiece or other features. Displaying the *ShowBody* allows the feature to be picked from its own body directly. Using the LCS and the *ShowBody* it is possible to obtain automatically the positions of all the origins without the system having any prior knowledge of them.

```
proc List make_block ( Fea this_fea_pointer, String this_fea_name )

  local real x1 = GetParVal( "x1", this_fea_pointer)
  local real y1 = GetParVal( "y1", this_fea_pointer)
  local real z1 = GetParVal( "z1", this_fea_pointer)
  // get the values of the parameters defining the block
  local Body block = Block( x1, y1, z1 )
  // create the block with the ACIS Body Coordinate System origin
  local String origin = GetParVal( "Ursprung", this_fea_pointer)

  x1 = 0.5 * x1
  y1 = 0.5 * y1
  z1 = 0.5 * z1
  // compute distances from the Body Coordinate System origin

  local Vector offset = Vector(0,0,0)
  // move vector for body

  if (origin == "xyz") then
    offset = Vector(x1,y1,z1)
  else if (origin == "-xyz") then
    offset = Vector(-x1,y1,z1)
  else if (origin == "-x-yz") then
    offset = Vector(-x1,-y1,z1)
  else if (origin == "x-yz") then
    offset = Vector(x1,-y1,z1)
  else if (origin == "xy-z") then
    offset = Vector(x1,y1,-z1)
  else if (origin == "-xy-z") then
    offset = Vector(-x1,y1,-z1)
  else if (origin == "-x-y-z") then
    offset = Vector(-x1,-y1,-z1)
  else if (origin == "x-y-z") then
    offset = Vector(x1,-y1,-z1)
  else
    // default => origin == "m" => do not move
    return {block} // give the block to feature modelling kernel
  end if

  Move(block, -offset)
  // move the body in opposite direction to offset vector
  return {block}
  // give the offset block to feature modelling kernel

end proc // make_block
```

Figure 4.15 `make` Function For Block Feature With Origins

Consider Figure 4.16. The technique is to move the LCS to the current origin of the feature. This position in World Coordinates is known, denoted by **a**. The translation component of the *ShowBody* transform describes the position of the Body Coordinate System origin, **b**. The origin parameter is set to a different valid origin, moving the ShowBody. The new *ShowBody* transform translation gives the location in World Coordinates of the new Body Coordinate System origin, **c**. The position, **d**, in World Coordinates of the new origin (but when the feature was in its original position) is given by equation 4.1:

$$\mathbf{d} = \mathbf{a} + \mathbf{b} - \mathbf{c} \tag{4.1}$$

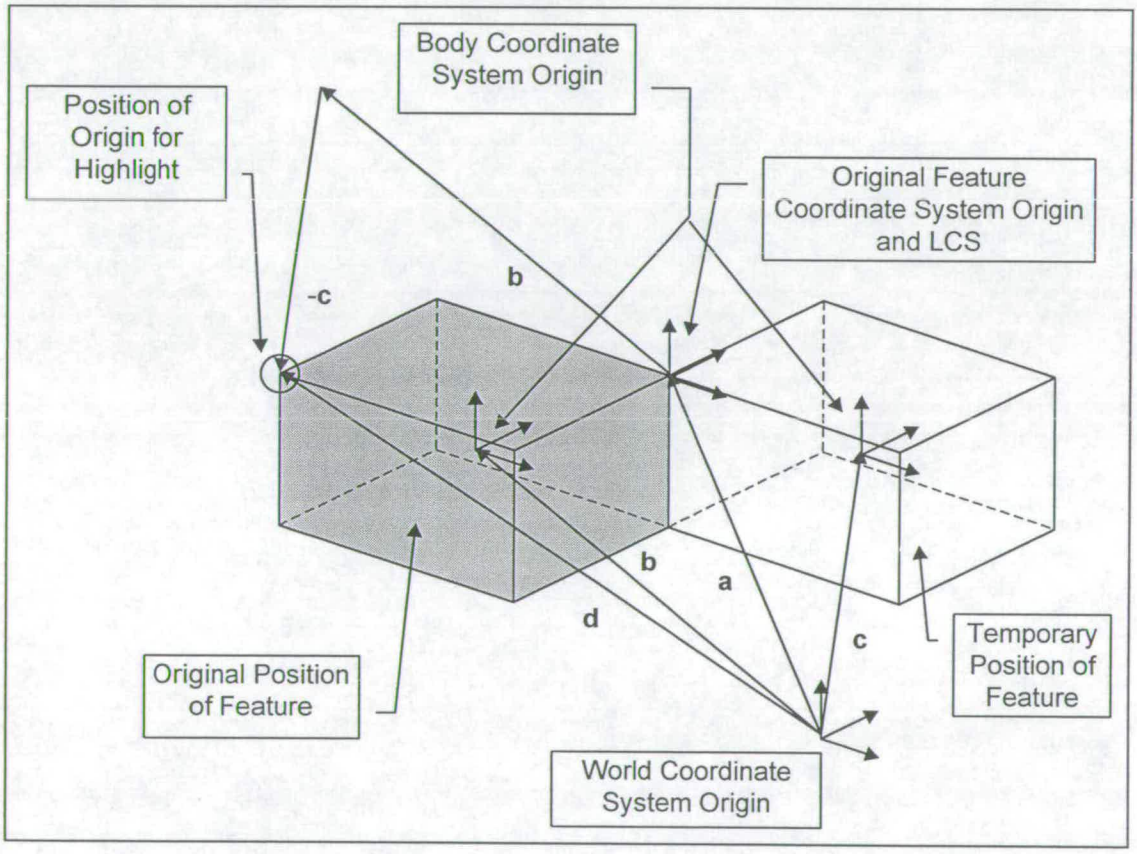


Figure 4.16 Generation Of Position Of Feature Origins

In MCL+ operator overloading defines that the addition of a vector to a position returns a position. By cycling through all possible valid values of the origin parameter the positions of all origins can be calculated by the system without

knowing specifically about the origin definitions. The method relies on the fact that the `make` function is always used and this builds the feature body in the same way each time. When ACIS builds a body consistently in the same manner the Body Coordinate System origin is always in the same place relative to the body. The `make` function naturally takes into account that the positions will be in different locations with variations in the values of the size parameters. Once the positions are known, which need not be at vertices or on other elements of the body, highlights can be displayed and picked via the Input Decoder.

Hence for features of known topology and geometric elements (but variable in size) the system can automatically calculate all origin positions provided that the following procedure is adhered to:

1. the user denotes the origin parameter as `Ursprung`.

2. the user positions the body of the feature in the `make` routine relative to the Body Coordinates System origin with respect to the origins that are defined.

3. the user tells the system what all the valid names of the origins for that particular feature are.

Currently this is limited to position movements. However, relatively easy extension would be possible for orientations by comparing the vector elements of the rotation matrix part of the transform.

The use of modifiers (multistate buttons) in the input sequence defined by the Input Decoder for feature instance creation allows the user to select the origin of choice interactively whilst sizing the feature with rubber bands. This is very intuitive because the user sees, selects and places the origin and feature graphically as the feature instance is being created. The user need not be aware of the origin descriptors. This leads to a right-first-time approach when building a model. Being the last feature in the tree, because it has just been created, it cannot have dependent features; that is features that reference it for the origin of their position and

orientation coordinates. Any changes in the chosen origin at this stage have no knock on effects to other features in the tree.

When a feature has a different origin selected, and that feature is established deep in the feature tree, undesirable effects can occur. The feature body moves relative to the feature's current position (and orientation); these parameters remaining fixed. Consequently, any features which reference it will remain static and not interact with the feature body as before. Certainly the relative positioning will be different. It may be the case, however, that the features referencing the feature (of new origin) do not interact with the feature (of new origin) at all. The difficulty is that it is not known if that is intended by the user or not.

A solution may be to provide a modifier for the origin selection function. This would:

1. Move the feature origin only.

2. Move the feature origin and by the same offset all features which reference it.

This allows the user to tell the system his intentions and for the system to support that thinking by automating the simple but tedious task of repositioning all child features. This can be non-trivial when the user does not know, or cannot easily calculate the necessary offset vector used by the system to change the origin. If it is the base feature that has its origin changed, then all features in the model that do not reference any other feature may have to be repositioned. This is functionality that is GUI oriented and, as such, is not a part of the kernel.

One obvious disadvantage of this approach is that it is not instant. There is noticeable pause of one or two seconds before the positions of the origins are highlighted. This is because there are many calls to the modeller to create, move and delete bodies. The more complex the body, the greater the number of origins defined, the slower the response. Despite this in a GUI based system where interactive working is the aim it is not seen as a problem. The pause is short enough not to cause a loss of

concentration in the user of the task in hand. It only occurs after a specific user action and not during every generation of the model which is time critical.

This disadvantage is considerably outweighed by the generality, flexibility and robustness of the approach for user defined features. System delivered features that are hard coded do not suffer from lack of response because bespoke functions may be called to generate the points without the need for creating, moving and deleting bodies. There remains one implementation task: that of the user telling the system in a consistent and reliable manner, what the valid origin descriptors are. This will be addressed in Section 4.8.

4.7.3 Features With Extrinsically Defined Form

This class of feature in contrast to those that have their topology and geometry fully described procedurally in the `make` function cannot be defined and manipulated in the same manner. The interface between the feature modeller and the rest of the system are the ACIS bodies that define (completely, or partially) the form of the feature. However these bodies are fixed and cannot be changed by the feature modeller. It may be that the body is a wire in the form of a contour which has been generated by a sketching module. The wire body is fixed with respect to the feature modeller but may be changed in the sketching module. All that the feature modeller knows about is the resultant ACIS body. The topology and geometry of the ACIS body are completely defined (for a valid body), therefore they are known, but they are unknown in a procedural manner by the feature modeller; there is no `make` function to capture to some extent the semantics of the feature. Hence the concept of feature origins developed in section 4.7.2 is not applicable.

All ACIS bodies have a Body Coordinate System origin. This is located and oriented with respect to the World Coordinate System origin to position and align the body at the desired point in space. Any (previously defined) ACIS body can be positioned, oriented and combined in the feature tree by providing that body to the feature

modelling kernel through the `make` method. The templates and methods used to instance and manipulate such extrinsically defined features have been described in section 4.5. An integral part of the sweep feature definitions was the need to position the Feature Coordinate System origin at a useful location on the body of the object. This concept can be extended to provide user functionality that enables the origin to be set to any position and/or orientation for all extrinsically defined features.

Certain points must be noted. It is not sensible to orient the origin of a planar body so that the z axis is not parallel to the normal. Doing so would preclude editing the contour within the contour/sketcher module. For open wire bodies representing the path of a sweep there are only two sensible origins; the start and end points of the loop. Closed wire bodies may sensibly have any vertex on the loop as an origin. One additional problem for the path feature is that if the origin is changed to another vertex then the order of the edge list and cooedge loop must also be changed so that the new origin is accepted by ACIS as the start point of the loop. For an open wire body the ends may be toggled between as origins. The direction of the chain of cooedges is easily "reversed", it is much more problematic for a closed loop.

The functionality described in section 4.6.4.2 that adjusts the feature coordinate system and the body transform on assignment of the wire body to the feature parameter is used. The Local Coordinate System is positioned at the desired point and with the desired orientation on the feature `base` parameter. The user via the property sheet calls a command to set the feature origin and the body transform to be that of the Local Coordinate System. Modifiers enable only the position, only the orientation or both position and orientation to be set. As with the assignment of the parameter for the first time any features that reference that being changed must also have their transforms modified.

The advantage of this approach is that it is applicable to any feature, whether system or user defined, that has extrinsically defined form. The method is completely compatible with the `FeaToLcs` and `LcsToFea` functions.

4.8 Template Definition Extension

4.8.1 Requirements

The current template definition method described so far in section 4.3 does not allow the user to define limits or ranges for values for parameters; only single default values may be defined. The user must perform any validation by controlling the parameters in the `make` function. Whilst this could be regarded as acceptable, if not desirable, it would be much more effective if the simple validation checks could be defined for each parameter in the template and the kernel would perform these automatically. This would simplify the construction of the `make` method and go some way to ensuring that invalid features do not become part of the model. In the case of more complex features where many parameters are dependent on each other it is not possible to define a test for one parameter that is independent of all others. In this instance a procedure would be required to be written with the `"@Procedure_Name"` syntax. The fact that parameter values may be set that create an invalid model means that an invalid model may be saved. In determining the validity of a feature, all parameters must be instantiated to their desired values, as well as any other features that the feature being validated is dependent on through the use of formulae or programming.

A further advantage of being able to define the valid values of parameters in the template definition is that the kernel of the feature modeller knows about them. A particularly relevant example is that of origin definition for features of implicitly defined form. It is necessary for the GUI to know what the valid origins are called and where they are located so that the user may select from the set allowable points.

Furthermore, much of the support for the definition of parameters by picking, selection of rubber band functions and modifiers set from pop-up menus is dependent on the type of parameter. For example, length, angle, diameter, radius, mode, and entity.

Therefore it is desirable that the template definition be extended to incorporate the above requirements.

4.8.2 Implementation

Each sub-list describing a parameter in the list `parameters` defined in section 4.2.2 is extended with the addition of two parameters: `kind` and `decode`. Therefore the sub-list `parameters` has the folowing definition:

```
parameters =        {
{name,alias,type,access,if_needed,if_set,default,kind,decode}
....
                    }
```

The parameter `kind` describes the class of parameter. It is defined as a string of one of the following values:

`""` Undefined (empty string)

`"l"` Length

`"a"` Angle

`"d"` Diameter

`"r"` Radius

`"m"` Mode. A set of finite values.

`"e"` Entity. A list of entity pointer types (Face, Edge, Vertex)

The parameter `decode` describes the valid set of values, ranges and types. `decode` is a list of the following general form:

```
{ data1, data2, data3 }
```

where `data1, data2, data3` take on different meanings depending on the definition and validity checking functionality required for the parameter being defined:

Undefined { }

The parameter has no validation values.

Range { `data1, data2` } or { `data1` }

The parameter must lie within a range where:

`data1` <= parameter value <= `data2`, or, `data1` <= parameter value

DeltaRange { `data1, data2, data3` }

The parameter must lie within a range and may only have a specific value within that range:

`data1` <= parameter value <= `data2`, and

`parameter value` = `data1 + n * data3`, for integer `n >= 0`

List { `data1, data2, data3` } or { `data1, data2` }

The parameter may only have a value that is stated in the list defined:

`data1`: default value.

`data2`: list defining the valid values.

`data3`: if `data2` is defined by a procedure ("`@procedure_name()`") then `data3` = "`List`".

EntityList { `data1, data2, data3` } or { `data1, data2` }

The parameter is a list of entities of specific types.

`data1`: minimum number of entities required. Eg. 1.

`data2`: list of valid entity types. Eg. {`Face, Edge`}.

`data3`: if `data2` is defined by a procedure ("`@procedure_name()`") then `data3` = "`List`".

The ability to make the validity checking dynamic is of great benefit particularly in the definition of origins for features which change their topology depending on the value of their parameters. For instance a slot feature may be straight and curved. When it is curved it has more origins than when it is straight because the axis defining the centre of radius also provides useful definition points, Figure 4.13.

To date these developments have not been fully built into the kernel. The class of parameter may be stated but the `decode` parameter is always set to undefined ({ }). The functionality is provided for the user interface during actions that use the property sheet. The template definitions are made separately using the `DefFeaPar` command which has exactly the same syntax for the validation parameters as described above. However, each list defines the parameter name, its class (`kind`) and then the validation parameters (`data1, data2, data3`). The example for slot shown in Figure 4.13 is given in Figure 4.17.

```
proc List ursprung_12214(Fea fea = Fea())
  static List origins = {
   {"y","yz","xyz","xy","xy-z","y-z","-xy-z","-xy","-xyz","-y",
    "-yz","x-yz","x-y","x-y-z","-y-z","-x-y-z","-x-y","-x-yz"},
    {"y2","y2z","y2-z","-y2","-y2z","-y2-z"},
    {"C","Cz","C-z"}}

  local List selected = { }
  if (fea.IsValid) then
    local String form = GetParVal("Form", fea)
    selected = Copy(origins[0])            // form == "A"
    if      (form == "B") then
      selected = selected + origins[1]
    else if (form == "C") then
      selected = selected + origins[2]
    end if
  end if
  return  selected
end proc

DefFeaPar({"s12214"},
  { { "Form", "", "B", {"A", "B", "C"} },
    { "Ursprung", "", "M", "@ursprung_12214())", List },
    { "r2", "r", 0.0 },
    { "x1", "l", RESABSFEA },
    { "y1", "l", RESABSFEA },
    { "z1", "l", RESABSFEA },
    { "r1", "r", RESABSFEA },
    { "wa", "w", RESABSFEA, 360.0-RESABSFEA } })
```

Figure 4.17 Parameter Validation Definition

4.9 Chapter Summary

In this chapter the general toolkit provided by FeatureM for developing user defined implicit features has been described. A simple example of the definition of a block feature has been developed. The integration of ACIS bodies in the feature model structure, regardless of their origin, has been shown to be extremely useful. Legacy solid data may be re-used and ACIS bodies form the only implemeneted standard of exchange. Such elements may be constructed in 2D sketchers, or be imported synthetic surfaces and curves, or existing ACIS bodies. Extrinsically defined non-parametric features have been developed to support such objects. Modelling of less constrained geometry is needed in order for feature modelling techniques to be applicable to wider product groups such as manufactured by forming processes. Methods have been presented to model sweep functionality. This is seen as a logical

next step from typical 2½D features towards fully implementing free form feature based surface modelling functionality. The accepted concept of "handles" has been applied to implicitly defined form features. A different method of achieving the outwardly similar results has been developed for extrinsically defined form features. Both these approaches have methods that use and interact with the kernel in a transparent manner to provide the same functionality for both user defined and pre-defined features. A further result of the work presented in chapter 4 has been to develop an extension of the template definition for each parameter. Not only is the extended description useful for improving the user interface to the feature by enabling picking of feature origins but allows validation rules for non-interrelated parameters to be built into the feature definition.

In chapter 5 a data structure and representation methods will be presented that allow dimensions and tolerances to be associated with solid and feature models. In addition, the template definition will be further extended to enable dimensions to represent the size parameters of features. Parameter dimensions will allow direct setting of feature parameters without the need for a property sheet.

Chapter 5

Feature Based Dimension And Tolerance Modelling

5.1 Introduction

5.1.1 Chapter Overview

The feature modelling system described in chapter 3 and the developments reported in chapter 4 allow the user to define features of implicitly and extrinsically defined form, and to create and manipulate models containing them. The models produced contain only nominal geometry which, as explained in chapter 2, is not sufficient to meet the needs of integrating advanced manufacturing software tools. In this chapter a data structure and representation methods will be developed to allow the generation of dimensions and tolerances that are associated with both solid and feature models. The feature template definition will be further enhanced to support the description of dimensions that represent the size parameters of features. These advances will provide a richer partial product model suitable for the integration of design and manufacturing software tools to meet the goals of Concurrent Engineering strategies described in chapter 2. Furthermore, the representation of dimensions and tolerances

will be developed such that they can be manipulated in 3D space so that specific views of the model will provide an engineering drawing without the need for a separate 2D representation. The developments will include advanced support for user defined features.

5.1.2 Chapter Structure

Section 5.1 forms this introduction. Section 5.2 will consider the requirements of dimensions and dimensional tolerances in a feature modelling environment. Section 5.3 will present a classification of the dimension features developed. Section 5.4 will present the modelling architecture used. Section 5.5 provides details of the implementation. Section 5.6 explains the editing functionality developed. Section 5.7 provides a chapter summary.

5.2 Requirements For The Dimension And Tolerance Model

5.2.1 Background

The use by design engineers of dimensions and tolerances has been explained in section 2.5. Although nominal dimensions are implicitly defined by the resultant geometric model they are represented explicitly on an engineering drawing and combined with tolerances indicate allowable variations in the manufactured part from the basic size.

Many of the problems restricting the impact of feature modelling systems highlighted in section 1.4 can be attributed to the lack of support for dimensions and tolerances associated with 3D feature and solid models. Namely:

- Only nominal geometry is supported.

- Dimensions and tolerances are detailed on engineering drawings which are 2D representations separate from the feature and geometric models.

- Dimensions and tolerances that are not associated with the feature and geometric models preclude the development of variational models needed for advanced design and manufacturing analysis

- The partial product model defined by a feature model lacking dimensions and tolerances is not sufficient for manufacturing reasoning and hence the most efficient integration of CAPP and CAM software.

- The goals of Concurrent Engineering cannot be met.

Furthermore, user interaction with instances of the features forming the model is predominantly with the use of a property sheet. The property sheet whilst providing an effective method of displaying parameter values and allowing the user to change them does not provide an aesthetically good working environment. Therefore, the lack of a graphical representation of a dimension that is associated with a feature size parameter leads to:

- a poor user interface.

The ability to measure a model in 3D would mean that any dimension desired on a 2D engineering drawing could be generated in the 3D modelling environment. Further enhancement of the definition of a feature to represent its size parameters with dimensions that the user can pick in order to set the value of the parameter and change the model would greatly reduce the need for continuous display of the property sheet improving visible display area. User interaction with the feature model is improved because of the intuitive, direct setting in the display of the parameter to be changed.

Despite the definition of geometrical tolerancing standards (Section 2.6), dimensional tolerances, used in conjunction with the Taylor Principle, still provide the widest and most frequently used method for specifying a design, particularly of

single components or tooling to be manufactured on numerically controlled machining centres. The work in this chapter is consequently focused in the area of dimension and dimensional tolerances although some geometric tolerance functionality will be investigated.

The stability and validity of the associativity of the dimensions and tolerances to changes in the form feature model and geometric model will be investigated. Most approaches to date assume persistent object identity [SIM95]; providing persistent object identity is a significant problem in hybrid feature modelling architectures.

In order to remove the need for a separate 2D engineering drawing representation the dimensions applied to the 3D model must be able to be laid out in a similar manner to 2D. This in turn means that the display must provide a set of views that correspond to the engineering views defined by a first or third angle projection. Views containing cross sections, or details, should also be automatically produced. These facilities will go some way towards removing the need to use 2D draughting systems.

Dimension and tolerance representation schemes are required in a number of areas during the development of a product, particularly when taking a concurrent engineering approach:

- Design of Component
- Design of Tooling
- Component Manufacturing Planning
- Tooling Manufacturing Planning
- Tooling Manufacture
- Quality of Tooling
- Component Manufacture
- Quality of Component

Generally the tooling used to manufacture a product (sheet metal press tool, plastic injection moulding tool, jig, fixture, etc.) does not look like or even have the same geometry as the component. This is caused by the nature of the design and

manufacturing process. Often incomplete designs are passed from design to manufacturing because they cannot be completed without expert process knowledge. Forming processes have multistage tooling; castings and plastic injection mouldings shrink. It is likely that there will be differences between the design tolerances of the final component, design tolerances applied to the tooling, and manufacturing tolerances applied to manufacture of the tooling. Although tooling and manufacturing tolerances are related to the component design tolerances it is a specialist task to define what they will be. It will not be a function of the representation scheme developed to provide for migration from the component design to the tool design. The applicability will be to the design of a single object, which may be the component or tool. Mapping or transmutation between model representations, or the representation of tolerances within external modules will not be addressed directly. Similarly the use of dimensions and tolerances with coordinate measuring machines and associated analysis will not be directly addressed.

5.2.2 Kind

In compliance with the standards (Section 2.6) linear and angular dimensions will be supported. This will require four dimension *kinds* to be implemented:

1. Linear
2. Angular
3. Diameter
4. Radius

Dimensions are prerequisite to dimensional tolerances. Dimensional tolerances of plus-minus or limit-fit must be definable for the dimensions.

An investigation of a geometric parallel tolerance will be made. It is believed that such relational tolerances will have to be treated in a significantly more complex manner because they do not always reflect the status of geometry, as a dimension does, but act as a three dimensional constraint. To date 3D constraint problems have

been limited to the assembly of rigid bodies rather than controlling the form, location or attitude of elements that constitute a body [SOL94].

5.2.3 Usage

The usage of the 3D dimension and tolerance representations developed should comply as far as is possible with 2D standards (section 2.6) and as described in respected handbooks [VSM91][HEN95]. This is so that when viewed in an engineering view the dimensions and tolerances will appear as in a 2D drawing.

All dimensions and tolerances have a dimension plane associated with them. This is obviously provided by the plane of paper for 2D engineering drawings. In 3D each dimension and tolerance must have a dimension plane defined and associated with it. In 2D there are only one set of in-dimension-plane projection lines from the shape aspect being dimensioned to the dimension line. In 3D there may be projection lines that are projected form the shape aspect to the dimension plane (to-dimension-plane projection lines), and then from the to-dimension-plane projection lines to the actual dimension line (in-dimension-plane projection lines), Figure 5.1.

Dimensions and tolerances should be able to be applied to the model as the design progresses in order to allow part finished components to be assessed by manufacturing modules providing for better compliance with the goals of simultaneous engineering. This requires persistent object identity to be maintained during each generation of the feature model and between each Boolean operation between solid bodies.

The dimension and tolerance representation should be open to interrogation from external systems because it is unlikely that all manufacturing software modules will be a part of the feature modeller. Consideration for STEP (section 2.6) compliance should be given when developing a representation that most likely will be transferred to other systems. The shortcomings and restrictions of STEP (section 2.6) should not be allowed to restrict progress.

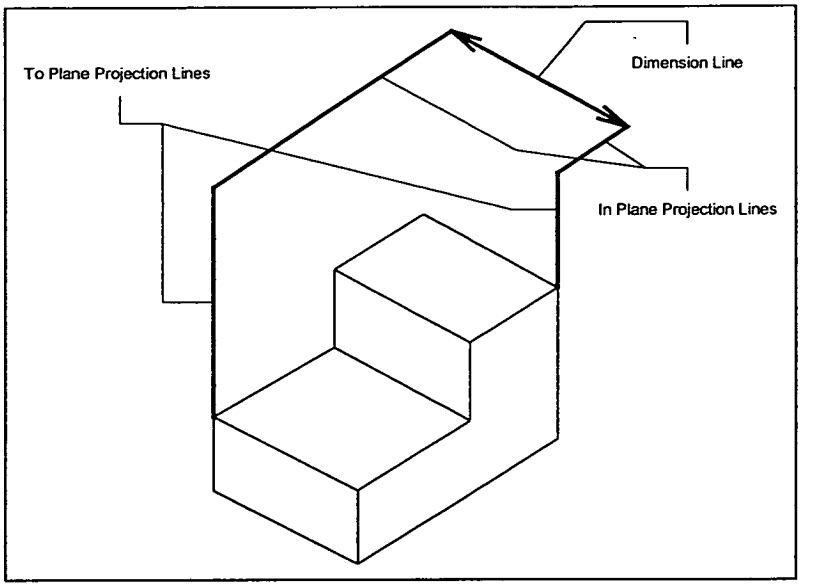


Figure 5.1 3D Projection Lines

5.2.4 Graphical Display

The graphical display of dimensions and tolerances should be fully in 3D, appearing as a homogeneous part of the feature model subject to all camera functions, such that viewing the 3D model from a set camera angle should provide a 2D engineering elevation.

The GUI for creation and editing should be highly interactive and comfortable to work with. The user should be provided with graphical rubber banding during creation and editing because of their intuitive nature; the WYSIWYG approach is very important because of the abstract nature of essentially 2D objects being applied in a 3D world. The look and feel of the GUI should be compatible with the rest of the system for consistency. Picking should be kept to a minimum.

In order to display the dimensions in a view as with a 2D engineering drawing it must be possible to manipulate:

- the location and attitude of the dimension-plane,

- the direction of projection of linear dimensions or the direction of a diameter or radius dimension line,
- the position of the dimension line in the dimension plane,
- position of the text along the dimension line,
- and the arrangement of groups of dimensions should be possible for aesthetically good layouts.

Layouts of pre-defined views corresponding to the first and third angle projections should be available. Assignment of the dimensions and tolerances to different views is necessary to provide the effect of a 2D layout. Hence, the need for a separate 2D representation is eliminated. Free cross-sectional and perspective views should also be definable

Tables of standard limits/fits should be presented that are applicable to the dimension value. The values should be able to be restricted to define a set of user preferred values.

5.2.5 Associativity

In common with the features of extrinsically defined form described in chapter 4, dimensions an tolerances have to be applied to a range of objects in a consistent manner. Consistency of treatment is important from the users perspective for the dimensions should look and feel the same despite their different origins and functionality. The dimensions and tolerances should form a homogeneous model with the geometry and features, being saveable as part of the component model.

The associativity of dimensions and tolerances to the resultant geometry and feature model can be described in terms of levels:

Level 0: Non-associative

Dimensions and tolerances are not associative with the model. They are simply a means of measuring a model in its current state providing a visual representation but contain no information about the geometric elements or features in the model. They most closely represent the after the event architectures of conventional 2D systems where dimensions and tolerances are added after the design has been completed. Changes that occur in the model are manually updated in the dimensions and tolerances by deletion and creation. This is a natural progression from 2D to 3D but does not meet the demands of feature modelling systems ability to support model changes. Non-associative dimensions do provide some useful functionality. Their static nature allows the user to visually compare the effects of design changes. Points not on the model may be dimensioned, such as references to fixtures.

Level 1: Uni-Directionally Associative

Dimensions and tolerances that are uni-directionally associative are used to measure the solid or feature model. The do however respond automatically to changes in the feature model and are maintained after Boolean operations between solid bodies. They are associated with geometric elements in the model and through these to the features that created them. They are of great benefit to represent dimensions in the model that are not represented by feature parameters.

Level 3: Bi-Directionally Associative

Bi-directionally associative dimensions and tolerances are a means of both representing feature parameters and measuring the model, and hence are a direct means of graphically changing the model. Where negative features overhang the model or features intersect they do not reflect the resultant geometry but the features that created it. Dimensions may appear to hang in space not connected with the geometry or be embedded in the solid portion of the model. Bi-directionally associative dimensions are always fully

consistent with the features used to create the model. On creation of the model much of it is pre-dimensioned easing construction; nothing is forgotten.

Level 4: Functionally Associative

Functionally associative dimensions and tolerances will be used to measure functional aspects of the model that are not described by a feature parameter. In contrast with level 1 dimensions which merely follow the resultant geometry, level 3 dimensions will be able to change the geometry. In this respect they will be more complex than the level 2 dimensions because a method of defining how the model (other feature parameters) should be changed to have the desired result. Relationships must be defined between the dimensions and other aspects of the control structure of the model.

It must be possible to associate the dimension and tolerance representations with the resultant geometry as well as to the feature instances used to create it. This is particularly necessary when features intersect creating entities in the model that did not exist in the feature templates. Hence dimensions and tolerances have to tag features, faces, edges and vertices in the model. Important points to consider are the derivation of the starting points of projection lines from objects such as faces and edges, points generated from more than one object, and how to deal with merged faces.

The functioning of the dimension and tolerance representation is dependent on persistent object identity of the entities (faces, edges, vertices) in the model. The system kernel should handle persistent object identity, it is not a function of the dimension and tolerance representation. It is impossible to always guarantee persistency of object identity, therefore a suitable method for handling lost object identities must be developed. A method for describing object identity when transferring models between systems is needed.

5.2.6 Functionality

Dimensions and tolerances may be applied directly to solid models or to workpiece (feature) models. Solid models provide the only means of importing geometry from other feature modelling or solid modelling systems to date. The function of these dimensions is to measure and respond to changes in the model.

Dimensions applied to contours created in 2D draughting systems should be maintained after importation to the 3D feature modelling system. 2D engineering drawings in many cases still provide the standard definition of components and are used for relatively complex designs. The function of these dimensions is to measure and respond to changes in the model.

Sketchers that use variationally constrained geometry have dimensions that are associated with certain constraints: radius and distance, for example. The function of these dimensions is to measure, respond to changes in the model, and be used to set the value of the constraints and hence control the model.

Dimensions should be able to represent the size parameters of features. The property sheet covers up to one third of the display area in contrast to modern user interface development philosophy which tries to maximise visualisation of the model. The majority of parameters displayed in the property sheet represent the size, position and orientation of the feature. In chapter 4 methods were developed that enabled graphical setting of feature position and orientation with the use of the LCS which did not require the property sheet. The position and orientation of features within the model may be represented graphically by level 1 dimensions. Representing the size parameters graphically as level 2 dimensions (including the special positioning feature) and being able to set the parameter values graphically will eliminate the need for these parameters to be present in the property sheet, hence reducing its displayed size, or eliminating the need for it completely. Therefore, user interaction with the feature modeller will be improved. The function of these dimensions is to measure,

respond to changes in the model, and be used to set the value of the size parameters and hence control the model.

All of these objects can be integrated in a feature model and validity of the dimensions on combination should be maintained.

The dimension and tolerance functionality should be equally applicable to both system developed and user defined features. User defined features make the system open and are a prerequisite for widespread acceptance of feature modelling systems and enhanced process integration.

Any implementation should take into account the performance for large models with features of many and complex parameters. Such models lead to a great many dimensions being present which could lead to degradation of interactive performance which they are designed to enhance.

Last but not least the dimension and tolerance scheme should be able to be applied to existing data as well as newly created models. Re-use of existing data is a major factor in the efficiency of developing designs. Significant amounts of legacy data exists in the form of solid and earlier versions of form feature models.

5.3 Classification

In section 5.2 three important requirements were established that influence dimension classification in the modelling system: *kind*, *associativity*, *functionality*. The implementational complexity of the three factors is used to determine classification.

Level of *associativity* is seen as providing the most fundamental differences in implementation strategy because of the differences between merely measuring the model statically, measuring the model and responding to changes, and both measuring and controlling the model.

Significant differences will be needed in the approach to modelling the dimensions that represent *functionally* different things. For example, simple 3D dimensions,

dimensions imported from 2D draughting systems, dimensions representing the constraints applied to a sketched contour, dimensions representing parameters of features. These considerations sub-divide the three *associativity* categories.

In each sub-division the four *kinds* of dimension provide for further division leading to the classification in Figure 5.2.

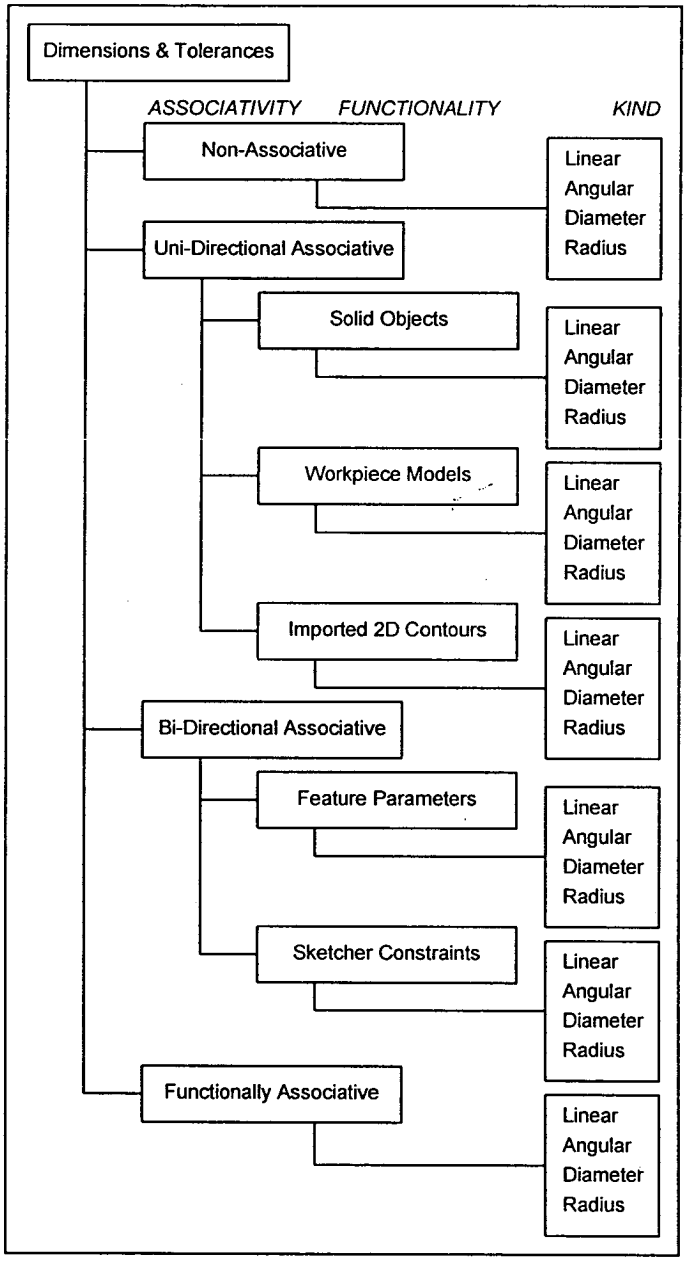


Figure 5.2 Dimension Classification

Dimensional tolerances are by definition applied to dimensions and cannot be described independently. Although there are three kinds of dimensional tolerance:

1. Plus-minus.

2. Limit/fit.

3. Statistical

They will not be independently classified, but will be included with the dimension representation.

Geometric tolerances have similarities to the dimensions; they can be non-associative, uni-directionally and (possibly) bi-directionally associative. The usefulness of non-associative geometric tolerances is extremely limited because of the lack of relationships to either the geometry or features.

Non-associative geometric tolerances will not be investigated. Of major significance are uni-directional associative geometric tolerances because they are applied to the geometry, respond to changes in the geometry and most importantly can be related to features. The possibility of geometric tolerances acting in a bi-directional manner will be discussed in chapter 6. Only uni-directional associative geometric tolerances will be considered. Of the tolerances classified in Figure 2.18, the parallel tolerance will be fully investigated because of the need to be related to two shape aspects.

5.4 Architecture

It is very tempting to use the information repositories of the form feature templates as a means of supporting dimension and tolerance data for the model. The size parameters of the features correspond one to one with size dimensions, the position of the feature and its orientation angles correspond with the location dimensions defined by the STEP Shape Variational Model [STEP47]. Geometric tolerances that are intra-feature can be supported. However, there are a number of significant problems with this approach:

- Only features of intrinsically defined form may have dimensions and tolerances represented. Features of extrinsically defined form which include swept objects, any solid bodies integrated with the feature modeller or any other objects such as imported 2D contours or sketcher contours cannot have dimensions and tolerances represented.

- Only a limited representation of inter-feature relationships can be supported. Location dimensions of position and orientation and their associated dimensional tolerances define inter-feature relationships between the origin of one feature and its reference feature or workpiece origin.

- Only resultant geometry created by independent features could be fully dimensioned. Non-intersecting features describe too simplistic geometry for industrial needs.

- Resultant geometry that is left after volume removal by a negative feature cannot be dimensioned. No feature parameters exist that represent the remaining shape aspects.

- Only geometric elements that exist in the feature templates can have dimension and tolerance attributes associated with them.

- Features that intersect creating merged faces become over dimensioned (toleranced).

- All possible tolerances have to be pre-defined in the feature templates. This is unrealistic and inefficient for intra-feature dimensions and tolerances and virtually impossible for inter-feature relationships. It is not known how many and which aspects of the feature are to be constrained, or what they are to be constrained to.

Clearly supporting the representation of dimensions and tolerances as intrinsic parameters of a form feature is too constraining and cannot meet the objectives of the work. A data structure is required that is external to the definition of the form feature workpiece but is a part of the more general feature, or product, model.

In an analogous manner to workpieces consisting of form features producing a geometric model, it is intuitive to consider a workpiece consisting of accuracy features providing the manufacturing constraints. Such a product model consisting of multiple workpieces representing different aspects of the component may be called a hyper feature model [PED96a]. There are a number of compelling reasons to use this architecture:

- The data structure is an integral part of the component model, but is separate from the form features.
- All objects in the feature modeller, geometric or otherwise are accessible.
- There are well developed and understood techniques for defining a feature template, its parameters, controlling how a features behaves via its methods, creating and deleting, saving and restoring. Defaults values for parameters may be assigned or procedures defined to calculate them.
- Persistent object identity is provided by the kernel and is needed in both form feature models for local operations and by the accuracy features for referencing.
- The order of evaluation of workpiece feature models may be set so that the accuracy feature workpiece is evaluated after any geometric workpieces.
- Features can be defined to have two methods for informing the kernel what to do with them. The `make` method and the `symbol` method. This has advantages because the procedure that generates the graphic segment can be separated from the procedure that validates the description of the accuracy after model changes.
- As can be seen from the requirements for graphical display (section 5.2.4) a dimension feature will consist of many parameter values. The user defined template definition functionality can easily define these parameters and may be readily extended if the need arises.

- The classification of dimensions and tolerances detailed in section 5.3 can be readily represented by using different workpieces to support the level of *associativity*, different feature types to support the different *functionality*, and different template names to represent the different *kinds* of features.

- Other kinds of accuracy features such as surface conditions, and bespoke developments such as flush checks and edge gap [IMP95] may be readily integrated into the environment either by developer or end user. In this way the system is not closed but is open to further development providing capacity for an ever more complete product model.

Representing dimensions and tolerances as features and utilising the user defined feature functionality of the feature modeller provides many advantages as described in the above points. The architecture can be more clearly defined by considering the classification shown in Figure 5.2. The classification has been divided into the following family types which can be clearly distinguished:

1. Type 1: Non-Directionally Associative Dimensions

   A single workpiece will be constructed that is separate from all other workpieces. Four different feature templates specifically defined for non-associative dimensions will be required to represent the four *kinds*. The workpiece will consist of instances of these four accuracy features. The parameters describing the dimension will be defined in the World Coordinate system because the dimensions are not related to any objects in the model. For the same reason the features do not require any validation of the description after changes in the model.

2. Type 2: Uni-Directional Associative Dimensions

   Dimensions applied to solid bodies or to workpiece models may be represented with the same accuracy workpiece and feature templates because in both instances the dimensions are related to geometric elements of a body; in the case of a form feature model it is the resultant body of the workpiece.

A single workpiece will be constructed that is separate from all other workpieces. Four different feature templates specifically defined for uni-directional associative dimensions will be required to represent the four *kinds*. The workpiece will consist of instances of these four accuracy features. References to the geometric elements being dimensioned providing persistent object identity must be supported in the feature in order to enable validation of the dimension description after changes in the model. The parameters describing the dimension will be defined in the Body Coordinate system of the solid, this allows the dimensions to move with any rotations or translations of the body or workpiece.

3. Type 3: Imported 2D Draughting Contour Dimensions

   These have an identical structure to the Type 2 dimensions. They require an independent workpiece and features to differentiate them from other accuracy features. This is because the dimensions are not created by the user in the interface but automatically be the import functions and should not be editable in terms of their graphical appearance to maintain consistency with the 2D system.

4. Type 4: Feature Parameter Dimensions

   Performance of the system degenerates proportionately with the number of entities in data dictionaries that are displayed by browsers. This is because of the poor response of X/Motif to building large dynamic objects. Large models consisting of many features will have many parameters. The more complex each feature, the more parameters, the more dimension representations, and the greater the problem. Any function that requires picking of objects entails filling a browser. Feature parameter dimensions will exist for each size parameter in the model and are therefore the most numerous of all dimension types. To avoid such performance problems, the workpieces and features providing the data structure for the dimensions and tolerances will not be present in the data dictionaries.

   In order to simplify management of the dimensions each form feature will have a workpiece associated with it that contains features each of which represents a

single dimension. Each dimension feature is therefore related to only one form feature parameter. References must be supported that allow a dimension to know which size parameter it represents, and vice verse. This architecture enables parameter dimensions to be managed on a feature by feature basis. One application is to display only those dimensions of the currently active feature.

The features representing parameter dimensions are defined in a different manner to the other features, i.e. they are not defined independently as user defined features with `DefTemplate`. This is because they are defined as part of the parameter description of the form feature parameter as described in section 5.5.6.4. The reason for defining them as part of the form feature description is so that they are applicable to any user defined form feature.

The parameters forming the extended template definition describe how to display the symbol of the dimension as with the earlier dimension types. All parameters that describe the display symbol of the dimension feature are defined in the coordinate system of the form feature (of which the dimension represents a size parameter). This is so that the display of the dimension symbol will always be consistent with the position and orientation of the form feature. The actual values of the parameters will have to be validated after each change in the feature so the dimensions remain consistent with the size of the feature.

5. Type 5: Sketcher Constraint Dimensions

Sketcher Constraint Dimensions are dependent on the type and implementation of the constraints. 2D Sketcher constraints typically are either single value or variable. For instance, elements may be constrained to be parallel, tangent, orthogonal, coincident, etc., or the constraints simply do not exist. By contrast distance, radius and angle constraints must have a value associated with them. It is the multi-value constraints that may be represented as dimensions. The dimension feature contains only part of the information required to represent the dimension, the other parameters are supported in the constraint. For dimensions the important constituents are the value of the constraint, the entities that the constraint is

applied to, and the aspect of the entity (start point, end point, centre point of circular arc). Constraints are applied after the contour has been sketched and so their definition is totally derived from geometry. In this respect they emulate the type 1 dimensions and therefore any defining parameters are stored in the Body coordinates of contour (a contour, though consisting of many elements only has one body). The data structure used to associate the constraints with a particular contour body is implemented using the ACIS attribute mechanism. This provides encapsulation of all information used to create that body. Encapsulation brings model management benefits. In order to maintain encapsulation a contour body will have a workpiece associated with it to which the dimension features representing the constraints will be assigned. In a similar manner to the type 4 parameter dimensions the workpieces and features will not appear in browser data dictionaries. Mechanisms must be provided to associate a dimension with a constraint and vice verse.

6. Type 6: Functional Dimensions

   To date no developments have been made. It is believed, however that such dimensions will closely reflect the development of the Complex Form Feature type. This is because in both cases the feature will have to support a set of rules defining how the constituent parts are related together. For Functional Dimensions it is necessary to define which parameter(s) of which feature(s) will be controlled from the controlling dimension, and what the relationship(s) will be.

Dimensional tolerances can only be applied to dimensions and are therefore supported as attributes of the respective dimension features forming parameters in the template definition. Perhaps the natural object oriented architecture would be to use individual features for the tolerances (only one template would have to be defined) and reference the specific dimension to be toleranced. However, there are a number of overheads which make this approach unattractive. Dimensional tolerances support simple data which can be defined in three parameters: textual indicating limit/fit or statistical distribution, and real values defining the upper and lower deviations.

Supporting the tolerances in an individual feature requires the definition of extra reference parameters and the addition of all the hidden attributes of the feature that control its existence in the workpiece. This makes a model with many dimensioned tolerances larger, particularly with the majority of dimensions being toleranced as is common practice. The graphic routines are more straight forward to implement and are therefore quicker when the data is in one location. Dimensional tolerances are associated through the geometry to features or directly with the features via parameter dimensions.

Interactions between features provide the most significant challenges for process planners and automated process planning systems [MIL93]. Geometric tolerances of form are generally related to a single shape aspect of the model. The shape aspect is often derived from a single feature, but when more than one feature has been combined to produce merged faces, any geometric tolerance applied to that face is related to both features. In contrast to tolerances of form, tolerances of location and attitude relate two shape aspects. The shape aspects can relate two elements of the same feature, two elements generated by different features, or even two elements in separate workpiece models. Unlike dimensional tolerances which have a simple definition related to a single size parameter, geometric tolerances are related to the geometric elements produced in the resultant model by the features. Supporting geometric tolerances as attributes is extremely inefficient because each tolerance must be represented as a parameter and parameters must be included for each reference entity. It is impractical in most cases and impossible for features of extrinsically defined form that each shape aspect has a set of tolerance attributes. Therefore the structure of all geometric tolerances will be implemented as with Type 2 uni-directional associative dimensions. Each tolerance will be a feature and be assigned to the Type 2 accuracy feature workpiece. Associativity with any geometry will be via entity references. Parameter definitions will be in Body Coordinates. Therefore the description will be invariant under translation and rotation. The representation is equally applicable to intra as well as inter feature tolerances. Problems with merged faces are eliminated because the tolerances are applied to the

resultant faces in the model. The tolerances are related back to the generating features via the face attribute which stores a list of the generating features.

5.5 Implementation

5.5.1 Class Structure

The class structure shown in Figure 3.10 has been extended, as shown Figure 5.3, to provide better support for the concept that a workpiece structure does not necessarily have to produce a solid model as its result. The original Wop and Fea classes have been simplified and the original functionality is now represented by the classes FormWop and FormFea which inherit from Wop and WopBase, and Fea and FeaBase respectively.

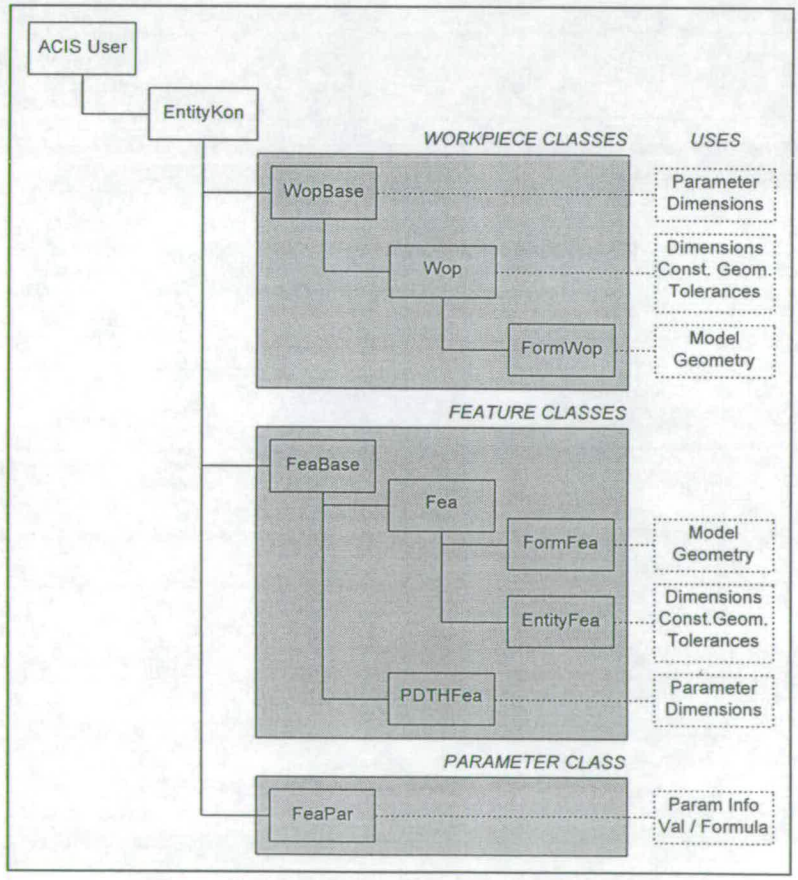


Figure 5.3 Extended Class Structure

As indicated in Figure 5.3, the WopBase class is used to define workpieces consisting purely of dimensions representing the size parameters of features. These workpieces do not produce solid geometry, nor do they appear in the browser data dictionaries. The Wop class is used to define workpieces which may or may not produce solid geometry. However, each features' geometry will not be combined to produce a resultant model, the workpieces do appear in the browser dictionaries and are used to support dimensions, tolerances and construction geometry. The FormWop class supports the original concept of a workpiece containing form features which generate geometry which is combined to produce a resultant model.

The FeaBase class is used to support common attributes and functions between the Fea and PDTHFea classes. The PDTHFea class is used to support a single symbolic graphic entity which represents the size parameter dimension of a feature. It does not produce solid geometry, nor is the feature combined to make a model, nor do the features appear in browser dictionaries. The Fea class is used to support commonality of attributes and functions for features which appear in browser dictionaries. There are two sub-types. The FormFea class represents the original concept of a form feature which produces geometry which is combined to produce a resultant model. The EntityFea class supports features which return symbolic geometry in the form of a graphic entity or solid body, which are not combined to produce a model. This class is used to represent dimensions, tolerances and construction geometry.

It is envisaged that this class structure will be further extended to provide a ComplexFea type that will be used to enable the user to create new features from combinations of existing features.

5.5.2 Workpiece Structure

The workpiece structures used to represent the dimension types are built from three workpiece/feature class systems.

All workpieces that appear in the browsers supporting non- and uni-directional dimensions are constructed as shown in Figure 5.4. Workpieces supporting parameter dimensions are structured as shown in Figure 5.5. Workpieces supporting sketcher constraint dimensions are structured as shown in Figure 5.6.

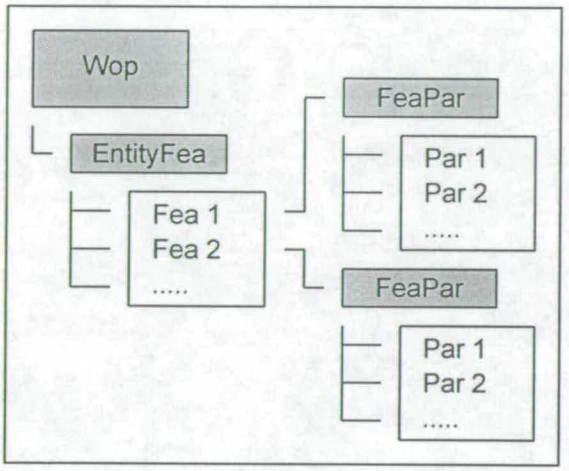


Figure 5.4 Non And Uni-Directional Workpiece Class Structure

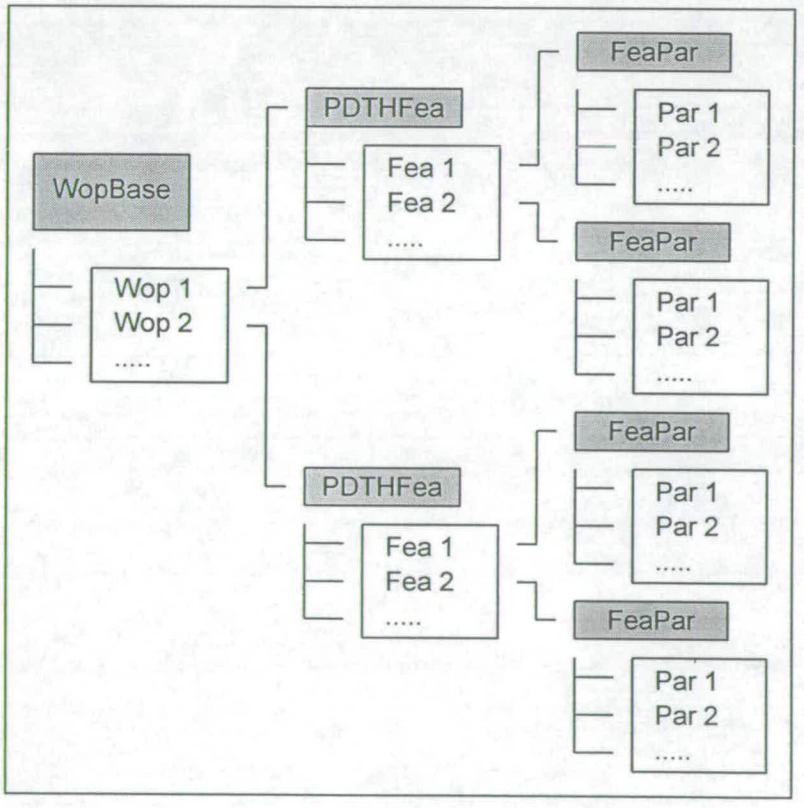


Figure 5.5 Workpiece Class Structure Supporting Parameter Dimensions

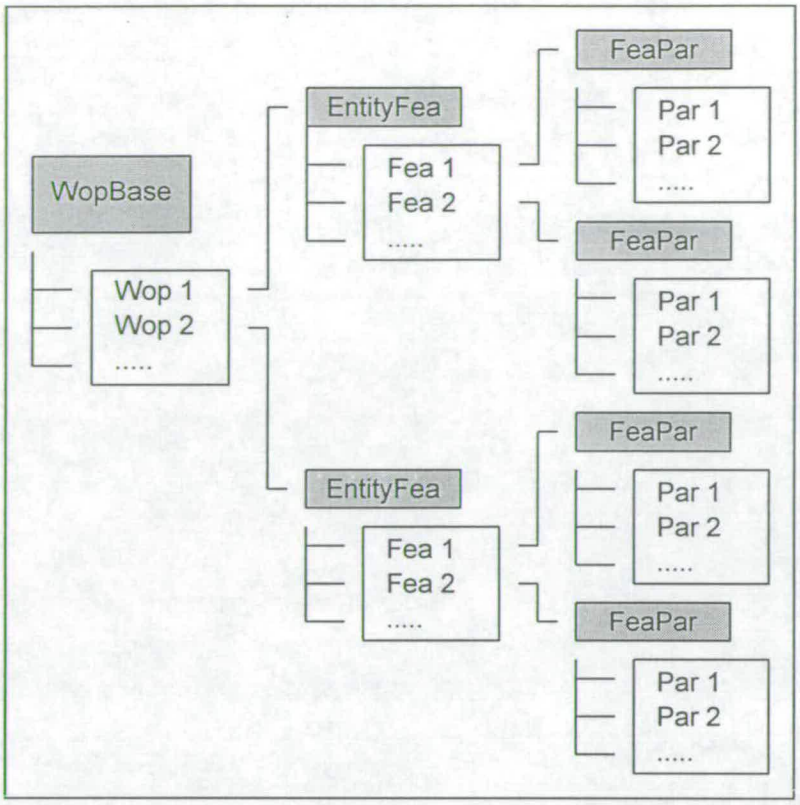


Figure 5.6 Workpiece Class Structure Supporting Sketcher Constraint Dimensions

The following workpieces have been pre-defined in the system:

":DimWop" for non-associative dimensions.

":DimWop2" for uni-directional associative dimensions created by the user in.

":DimWop3" for uni-directional associative dimensions imported from 2D draughting systems.

When the user creates a workpiece in MCL+ with a name which begins with a ":", this is automatically assigned to be of the class Wop.

The workpieces associated with each form feature to represent the parameter dimensions are created automatically by the template builder when reading the template description (section 5.5.6.4). Similarly the workpieces associated with the

sketcher constraints are automatically created on generation of a contour (section 5.5.5.5).

5.5.3 Technical Views[1]

In order to eliminate the necessity for a separate 2D modelling environment for the component it must be possible to develop hard copy 2D drawings directly from the 3D representation. A view layout has been developed, known as technical layout, in which any views defined by the user must conform to first or third angle technical engineering drawing conventions (which ever is selected). A standard layout of front, plan and right views and an isometric projection is provided. Any of the other formal views may be added but must conform in size and position with the current scheme. Views from any angle may be specified, as can cross-section, detail, and completely free views.

The use of technical views is very helpful to aid the user in creation of dimensions because the orientation of the dimension plane is defined. Each technical view has a line of sight which is taken as the dimension plane normal vector. The graphics and windowing systems of HOOPS and X/Motif (via the Input Controller) are able to determine which view a pick took place in, and the place of the pick. The combination of the view line of sight and pick position can be used to good effect to reduce the number of parameters that the user must define in constructing a dimension (section 5.6.1).

Technical views also aid the display of dimensions. A dimension may only appear in a technical view defined by the line of sight vector being parallel to the dimension plane normal. The plane of display of a dimension value is always parallel to the

---

[1]The development of technical views is predominantly not the work of the author. The use, response and behaviour of the dimension and tolerances within the technical views is the work of the author.

plane of a view, horizontal and sited on the dimension line. This is independent of camera angle. In a technical view the text is additionally maintained parallel to and above the dimension line as in a standard engineering drawing.

5.5.4 Description Of Dimension Templates

5.5.4.1 Linear

Linear dimensions are defined between two points in a model. The value of the dimension is calculated with respect to the direction of projection, and the dimension plane in which the dimension line is located. The in-dimension-plane projection lines should always have a length that helps clarity of the drawing.

This description requires the definition of the following parameters:

```
{"pos1","","Position","w","n","n",""}
{"pos2","","Position","w","n","n",""}
```

The start points of the projection lines.

```
{"norm","","Vector","w","n","n",""}
```

The normal vector of the dimension plane.

```
{"dldir","","Vector","w","n","n",""}
```

The direction of projection in the dimension plane. It is defined to be in the projected direction of `pos1` to `pos2`.

```
{"locavec","","Vector","w","n","n",""}
```

The vector points from `pos1` to the start point of the dimension line. This defines the position of the dimension plane and the position of the dimension line in the dimension plane. Hence the lengths of the to-dimension-plane projection lines and the in-dimension-plane projection lines are defined. This may produce all four projection lines of different length.

```
{"tof","","real","w","n","n","0.0"}
```

Defines the position of the value of the dimension and tolerance to be displayed on the dimension line. `tof` is zero at the centre of the dimension line, +1 at the intersection of the projection line from `pos1` with the dimension line, -1 at the other end of projection line, and proportionally positive or negative if not positioned within the projection lines.

The parameter meanings are shown in Figure 5.7.

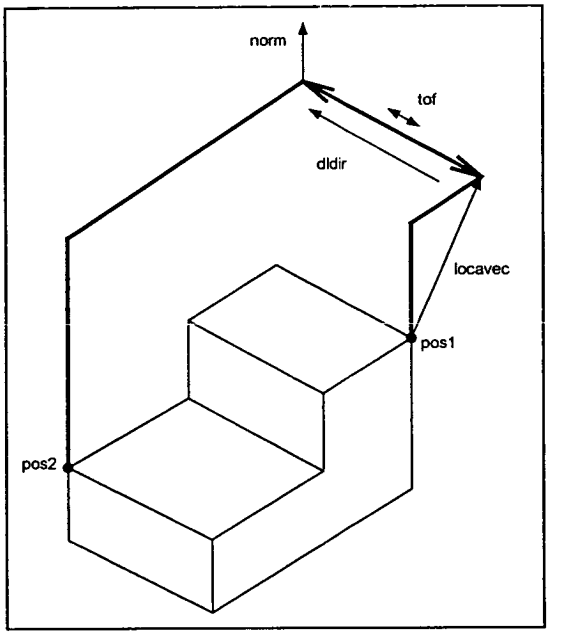


Figure 5.7 Linear Dimension Parameters

5.5.4.2 Angular

Angular dimensions are defined with respect to three points in the model. The plane defined by the three points also defines the attitude of the dimension plane. The value is calculated to be the interior or exterior angle as defined by the three points, Figure 5.8. The in-dimension-plane projection lines should always have a length that helps clarity of the drawing.

This description requires the definition of the following parameters:

```
{"pos1","","Position","w","n","n",""}
{"pos2","","Position","w","n","n",""}
```

The start points of the projection lines.

```
{"pos3","","Position","w","n","n",""}
```

The intersection point which defines the angle as the interior angle formed by two straight lines drawn through `pos1` and `pos3`, and, `pos2` and `pos3`.

```
{"norm","","Vector","w","n","n",""}
```

The normal vector of the dimension plane. Partially redundant because the normal is always calculated from the cross product of the vectors joining `pos3` to `pos1` and `pos3` to `pos2`. It is maintained for version and graphic routine compatibility.

```
{"locavec","","Vector","w","n","n",""}
```

The vector points from `pos1` to the start point of the dimension line (arc). The "x" component of the vector is defined to be in the direction from `pos3` to `pos1`. The "y" component is always zero. The "z" component is defined to be in the direction of the dimension plane normal as defined above. This is so the positions describing the dimension line will remain consistent with changes in angle. `locavec` defines the position of the dimension plane and the position of the dimension line in the dimension plane. Hence the lengths of the to-dimension-plane projection lines and the in-dimension-plane projection lines are defined. The to-dimension-plane projection lines will be of the same length, the in-dimension-plane projection lines may be of differing lengths.

```
{"tof","","real","w","n","n","0.0"}
```

Defines the position of the value of the dimension and tolerance to be displayed on the arc of the dimension line. `tof` is zero at the centre of the dimension line, +1 at the intersection of the projection line from `pos1` with the dimension line, -1 at the other end of projection line, and proportionally positive or negative if not positioned within the projection lines.

The parameter meanings are shown in Figure 5.8.

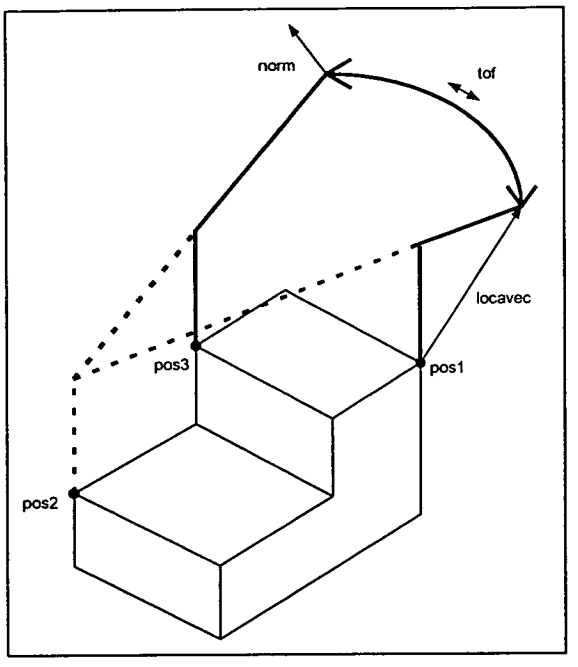


Figure 5.8 Angular Dimension Parameters

5.5.4.3 Diameter

Diameter dimensions are defined by a circular arc which may or may not be a full circle. The dimension plane is defined either by the plane of the arc or by a plane orthogonal to the plane of the arc. The value is simply twice the radius of the arc. The value may also be represented as a standard thread size such as M8. The direction of the dimension line is dictated by a vector. This vector orients the dimension plane if normal to the plane of the arc. Diameter dimensions may have full or half dimension lines. The in-dimension-plane projection lines should always have a length that helps clarity of the drawing, except if the dimension line intersects the axis of the arc, in which case the length may be zero. The dimension line never cuts the arc being dimensioned.

This description requires the definition of the following parameters:

```
{"pos1","","Position","w","n","n",""}
{"pos2","","Position","w","n","n",""}
```

The start points of the projection lines at opposite ends of a diameter for a full dimension line. For a half dimension line, `pos1` defines the start point of the single projection line, and `pos2` is the opposite end of the diameter to `pos1` but has no projection line.

```
{"norm","","Vector","w","n","n",""}
```

The normal vector of the dimension plane.

```
{"dldir","","Vector","w","n","n",""}
```

The direction of the dimension line. It is defined to be in the direction from `pos1` to `pos2`.

```
{"locavec","","Vector","w","n","n",""}
```

The vector points from `pos1` to the start point of the dimension line. This defines the position of the dimension plane and the position of the dimension line in the dimension plane. Hence the lengths of the to-dimension-plane projection lines and the in-dimension-plane projection lines are defined.

```
{"tof","","real","w","n","n","0.0"}
```

Defines the position of the value of the dimension and tolerance to be displayed on the dimension line. For a full dimension line indicating a complete diameter `tof` is zero at the centre of the dimension line, +1 at the intersection of the projection line from `pos1` with the dimension line, -1 at the other end of projection line, and proportionally positive or negative if not positioned within the projection lines. For a dimension line showing a half diameter the dimension line has a length equal to the radius. `tof` is zero at the centre of the dimension line, +1 at the intersection of the projection line with the dimension line, -1 at the other end of projection line, and proportionally positive or negative if not positioned within the length of the dimension line.

```
{"spos","","Position","w","n","n","0,0,0"}
```

Defines the start point of the arc in order to describe where any help arc may be drawn from.

```
{"epos","","Position","w","n","n","5,0,0"}
```

Defines the end point of the arc in order to describe where any help arc may be drawn from. If the arc forms a complete circle `epos` is defined to be the same as `spos`.

```
{"arcpos","","Position","w","n","n","2.5,2.5,0"}
```

Defines a position on the arc in order to define the normal to the arc and which of the two possible segments of a circle defined by `spos` and `epos` forms the arc. If the arc is open, `arcpos` is defined as the midpoint of the arc. If `spos` equals `epos`, `arcpos` is defined as a point at the end of a diameter which is perpendicular to `dldir`.

```
{"touch","","int","w","n","n","0"}
```

Defines if `pos1` and / or `pos2` lie on the arc. `touch` is needed to indicate if a help arc is to be displayed. `touch` is zero if neither `pos1` or `pos2` lie on the arc, 1 only if `pos1` lies on the arc, 2 if only `pos2` lies on the arc, and three if both `pos1` and `pos2` lie on the arc.

The parameter meanings are shown in Figure 5.9.

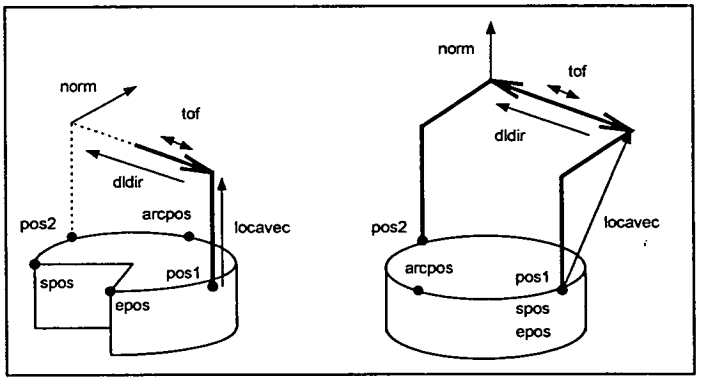


Figure 5.9 Diameter Dimension Parameters

5.5.4.4 Radius

Radius dimensions are defined by a circular arc which may or may not be a full circle. The dimension plane is always defined by the plane of the arc. The value is simply the radius of the arc. The direction of the dimension line is dictated by a vector. The dimension line direction always passes through the centre of the arc.

This description requires the definition of the following parameters:

```
{"pos1","","Position","w","n","n",""}
```

The start point of the dimension line is sited on the imaginary circle that the arc forms a part of.

```
{"pos2","","Position","w","n","n",""}
```

The end point of the dimension line. The position of `pos2` is defined by `tof`.

```
{"norm","","Vector","w","n","n",""}
```

The normal vector of the dimension plane.

```
{"dldir","","Vector","w","n","n",""}
```

The direction of the dimension line. `dldir` is defined to be from the centre of the circular arc outwards.

```
{"locavec","","Vector","w","n","n",""}
```

Is always of zero length. Kept for historical reasons.

```
{"tof","","real","w","n","n","0.0"}
```

Defines the position of the value of the dimension and tolerance to be displayed on the dimension line. `tof` is zero at `pos1` (however the value may never be zero), +1 at a distance equal to the radius from `pos1` in the dimension line direction, -1 at the centre of the circular arc, and proportionally positive or negative.

The parameter meanings are shown in Figure 5.10.

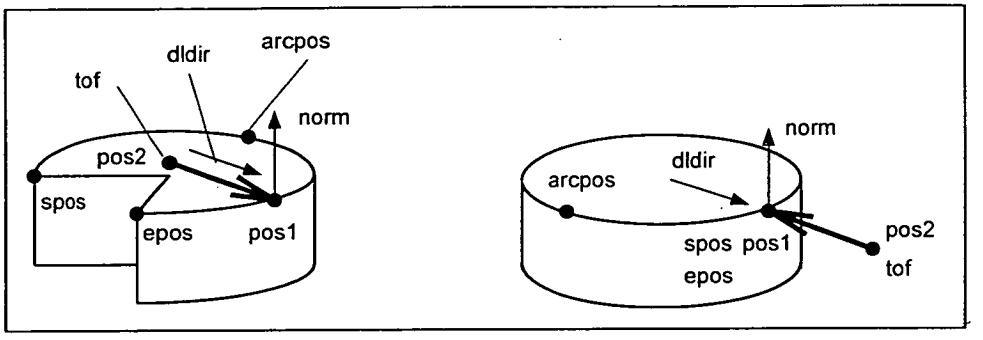


Figure 5.10 Radius Dimension Parameters

```
{"spos","","Position","w","n","n","0,0,0"}
```

Defines the start point of the arc in order to describe where any help arc may be drawn from.

```
{"epos","","Position","w","n","n","5,0,0"}
```

Defines the end point of the arc in order to describe where any help arc may be drawn from. If the arc forms a complete circle `epos` is defined to be the same as `spos`.

```
{"arcpos","","Position","w","n","n","2.5,2.5,0"}
```

Defines a position on the arc in order to define the normal to the arc and which of the two possible segments of a circle defined by `spos` and `epos` forms the arc. If the arc is open, `arcpos` is defined as the midpoint of the arc. If `spos` equals `epos`, `arcpos` is defined as a point at the end of a diameters which is perpendicular to `dldir`.

```
{"touch","","int","w","n","n","0"}
```

Defines if `pos1` lies on the arc. `touch` is needed to indicate if a help arc is to be displayed. `touch` is zero if `pos1` does not lie on the arc, 1 if `pos1` lies on the arc.

5.5.4.5 Dimensional Tolerances

Dimensional tolerances are supported by three parameter attributes as shown below.

```
{"ttype","","String","w","n","n",""}
```

Supports any textual representation of the tolerance such as limit, fit or statistical description..

```
{"tol1","","real","w","n","n","0.0"}
```

Represents the lower deviation of the tolerance.

```
{"tol2","","real","w","n","n","0.0"}
```

Represents the upper deviation of the tolerance.

5.5.4.6 Graphic Parameters

The following parameters are used by the graphics to correctly display a dimension line.

```
{"val","","real","w","n","n","10.0"}
```

Represents the numerical value of the dimension to be displayed.

```
{"dlesym1","","int","w","n","n","1"}
{"dlesym2","","int","w","n","n","1"}
```

Represents the type of arrow head to be used for the display of the dimension line. Planar arrow heads, crossed planar arrow heads, spheres and tick marks may be specified. If `dlesym2` is set to zero for a diameter dimension this indicates that it is represented as a half diameter dimension line. `dlesym2` is not necessary for a radius dimension.

```
{"adir","","int","w","n","n","0"}
```

Indicates the direction of the arrow on the dimension line. The arrow head may be sited inside the projection lines pointing outwards, or outside the projection lines pointing inwards.

```
{"locplane","","int","w","n","n","0"}
```

Indicates that a small rectangle should be displayed to show the plane of the dimension plane. A small arrow is displayed to indicate the direction of the dimension plane normal.

5.5.5 Object Decoding

It has been shown in section 5.5.4 that all dimension *kinds* require the derivation of positions to enable a graphical representation to be displayed. The positions form the start / end points for the dimension lines or the projection lines. These points are directly derived from the geometry for all dimension types apart from Type 4 (Parameter Dimensions) which are procedurally generated from the definition of the feature (section 5.5.6.4). The derivation of positions from geometry may be general or functional. Even though the derivation of points from geometry may be classed as general, the points must always be compatible with the kind of dimension being represented. It is necessary to be very specific with functionally derived points from entities in order to maintain consistency with the definition of the constraints which they represent. Geometry may be picked from the display with types 1, 2 and 5, or prescribed when importing dimensioned contours from a 2D draughting system, type 3. Object decoding is not applicable to type 4 parameter dimensions.

5.5.5.1 General Decoding

There are three types of object that can be picked in the display that can be decoded to generate dimension positions: faces, edges, and vertices. These are topological entities and the corresponding geometry is interrogated to evaluate positions. Decoding is performed in MCL+.

5.5.5.1.1 Vertices

Vertices are the least complicated. The position coordinates of the *vertex* are given by its attribute *point*. In MCL+:

```
local Position pos = this_vertex.Point
```

Where `this_vertex` is a valid vertex pointer. The coordinates must be converted from Local Coordinates to the desired (usually World) coordinate system.

5.5.5.1.2 Edges

In MCL+ all edges have an attribute `GeoType` which may be `Straight`, `Circle`, `Ellipse`, or `Spline`. To date `Spline` edge types may not be decoded or have dimensions applied. In decoding of an edge intuitive points are used. The mid point of a straight edge and the centre point of a circular edge are used to indicate that it is an edge entity that has been used, rather than the vertices which form its start / end points.

Single edges may be decoded to generate one position. In this case the position decoded is the mid-point of a straight edge, or the centre of a circular or elliptical arc.

Single edges may be decoded to generate two positions. The decoded positions are the start and end points of the edge. This is independent of edge type and may be interpreted to be the length of a straight line or the chord of an arc.

Two edges may be decoded to generate one position. The edges must be straight, in which case the intersection point of the geometric straight lines is decoded (if the edges are not parallel).

The coordinates must be converted from Local Coordinates to the desired (usually World) coordinate system.

5.5.5.1.3 Faces

In MCL+ all faces have an attribute `GeoType` which may be `Plane`, `Cylinder`, `Cone`, or `SplineSu`, etc. To date only `Plane` and `Cone` face types may be decoded to generate a point for use in dimensioning.

Conical faces are decoded to generate the position of the apex.

In decoding of planar faces there are no intuitive points that lie within the face rather than on its boundary. Utilising points on the boundary would indicate in the display that it was an edge or vertex that was used for dimensioning. A further consideration when generating a point that lies within a face is that a face may have inner loops producing holes or protrusions.

A single face is decode to produce a position that lies on the face. This is achieved by intersecting the face with a plane whose normal is defined by the dimension plane and the root point is calculated to be the mid-point of the diagonal of the box surrounding the face. This produces a series of curve segments which lie on the face. The mid-point of the segment nearest to the root point is the decoded point.

A pair of non-parallel planar faces may be decoded to produce three points: a point on each face and an intersection point. The three points are used to define an angle. The dimension plane normal vector is defined by the cross product of the face normal vectors. A point on the first face is generated as described above. A point on the second face is generated, also as described above, but the root point is defined to be the point generated for the first face. Equations of the straight lines formed by the straight line segments are intersected. The intersection point is the third point generated.

5.5.5.2 Functional Decoding

Distance constraints in the Sketcher are formed between points in the Constraint Manager [DCU]. The points are generated from edges by specifically indicating how the position should be calculated. For straight edges the points may be either the start or end points. For circular arcs, the positions may be the start or end points, or centre of radius.

Radius constraints are related to a circular arc. Further decoding is not required.

5.5.6 Dimension Types

In addition to the parameters described in section 5.4 further parameters are required depending on the dimension type. These parameters and the way in which the methods described in the feature templates are used to maintain the validity and display the dimensions and tolerances are described in this section.

In all dimension templates the symbol methods are denoted by "p", or the "p" functions are inherited with the use of "e". This means that the method is implemented as a C++ function. The symbol procedures consist of two parts: a part which creates the graphic display segments and the part that converts the parameter values from the coordinate system that they are maintained in, to the World Coordinate System that the graphic segments use. Conversion is performed by multiplying the points or directions by the necessary transform.

The templates use the following naming convention. They are of the form:

template name = d3$\alpha\beta$

where "d" indicates that the template is of a dimension,

"3" indicates that it is a 3D dimension,

"α" is an integer which indicates the dimension type (1 to 5), and

"β" is an integer which indicates the dimension kind, 1 to 4 corresponding to linear, angular, diameter and radius respectively.

Dimension types 1, 2, 3 and 5 are of feature type `DTH` which is of the EntityFea class. Dimensions of type 4 are of the feature type `PDTH` which is of the `PDTHFea` class. The feature templates representing the dimensions are described using inheritance to simplify and ease definition.

5.5.6.1 Type 1: Non-Associative Dimensions

```
DefTemplate({"ENTITY_DIM1","","DTH","---", "n", "n","n","n"},{
  {"val","","real","w","n","n","10.0"},
  {"tof","","real","w","n","n","0.0"},
  {"dlesym1","","int","w","n","n","1"},
  {"ttype","","String","w","n","n",""},
  {"tol1","","real","w","n","n","0.0"},
  {"tol2","","real","w","n","n","0.0"},
  {"adir","","int","w","n","n","0"},
  {"pos1","","Position","w","n","n",""},
  {"pos2","","Position","w","n","n",""},
  {"norm","","Vector","w","n","n",""},
  {"locavec","","Vector","w","n","n",""},
  {"locplane","","int","w","n","n","0"},
```

Figure 5.11 Type 1 Dimension Parent Template

A parent entity is used, `ENTITY_DIM1`, as shown in Figure 5.11. The parent entity is used to contain all the common parameters for the dimension kinds as described in section 5.5.4. There are no make methods associated with type 1 dimensions because there are no associated objects. Validity checking occurs once at creation, and ensures the dimension is consistent with the geometry.

The dimension kind templates are shown in Figure 5.12.

```
// Type 1 Linear Dimension
DefTemplate({"d311","","DTH","ENTITY_DIM1", "n", "p","n","n"},{
  {"dldir","","Vector","w","n","n",""},
  {"dlesym2","","int","w","n","n","1"} })

// Type 1 Angular Dimension
DefTemplate({"d312","","DTH","ENTITY_DIM1", "n", "p","n","n"},{
  {"pos3","","Position","w","n","n",""},
  {"dlesym2","","int","w","n","n","1"} })

// Type 1 Diameter Dimension
DefTemplate({"d313","","DTH","ENTITY_DIM1", "n", "p","n","n"},{
  {"dldir","","Vector","w","n","n","1,0,0"},
  {"dlesym2","","int","w","n","n","1"},
  {"touch","","int","w","n","n","0"} })
  {"spos","","Position","w","n","n","0,0,0"},
  {"epos","","Position","w","n","n","5,0,0"},
  {"arcpos","","Position","w","n","n","2.5,2.5,0"} })

// Type 1 Radius Dimension
DefTemplate({"d314","","DTH","ENTITY_DIM1", "n", "p","n","n"},{
  {"dldir","","Vector","w","n","n","1,0,0"},
  {"touch","","int","w","n","n","0"} })
  {"spos","","Position","w","n","n","0,0,0"},
  {"epos","","Position","w","n","n","5,0,0"},
  {"arcpos","","Position","w","n","n","2.5,2.5,0"},
  {"parent","","Ent","w","n","n",""} })
```

Figure 5.12 Type 1 Dimension Template Definitions

Only those parameters are detailed in the child templates that are not common to all four dimension kinds. For instance, an angular dimension has no direction of projection (`dldir`), a radius dimension does not have a second end symbol (`dlesym2`). Both diameter and radius require the parameter `touch`. In addition, diameter and radius dimensions require the parameters `spos`, `epos` and `arcpos` to be defined.

`spos`, `epos` and `arcpos` are required to define the arc of type 1 dimensions in order to allow the dimensions to be edited correctly (section 5.6.2).

5.5.6.2 Type 2: Uni-Directionally Associative Dimensions

It is a restriction of the feature modelling system kernel that feature templates may only inherit from a single parent. This makes the definition of the type 2 templates not as efficient as could be. It would be desirable that the type 2 templates could inherit from both their own type entity parent and the type 1 kind templates. This is not possible so the clearest method of developing the next branch of the dimension family is for the type 2 parent template to inherit from the type 1 parent template. The definition of the type 2 dimension kind templates is less efficient.

An interesting problem occurs when type 2 dimensions are applied to two separate solid bodies which are then combined using Boolean operations. Before the Boolean operation both bodies have different transforms and any dimensions that are applied to the bodies (or between them) are defined in one of the Body Coordinate systems. After combination there is a new body but only one Body Coordinate system. A method is required that converts any dimension parameters from their previous coordinate systems to the new one. A pre-NoteState function stores the previous transforms before the Boolean operation is performed. Then, after combination, when each feature in the dimensions workpiece is having its object identity reset to the new body a MCL+ function is called that applies the previous transform to the required parameters to convert them to World Coordinates, followed by application of the inverse of the new transform converting them into the Body Coordinates of the new body.

The type 2 definition of the parent entity, `ENTITY_DIM2`, is shown in Figure 5.13.

Type 2 dimensions are related to at least one entity (face, edge, or vertex), which is supported by parameter `obrel11`. Type 2 dimensions may be applied between entities in different bodies or workpieces. Type 2 dimension parameters are supported in the Body Coordinate system of one of the bodies. This is indicated by the parameter `obrel`, which stores the number of the entity, 11 for example. Each entity has an attribute, `Body`, which may be interrogated to get the correct transform.

Parameter, `ptypobl`, indicates how the position required by the graphics should be decoded from the entity. `delopt` is a parameter that is used to indicate how lost (non persistent) object identities should be treated. Object identities associated with solid bodies will never be refound once they are lost, therefore the dimension with lost object identities should be deleted. Object identities related to entities in a feature model may be found at a later stage after further topological changes (It is severe topological changes that tend to cause lost object identities). Therefore `delopt` is set to zero when the dimension is associated with a solid body and set to 1 when associated with a feature model. The parameters `parname` and `formfea` are used to maintain associativity between the dimension when applied to a contour and the body parameter of a sweep form feature that consumes it.

```
DefTemplate(
{"ENTITY_DIM2","","DTH","ENTITY_DIM1","n","n","n","n"},{
  {"obrel","","int","w","n","n","11"},
  {"ptypobl","","String","w","n","n","M"},
  {"obrell1","","Ent","w","n","n",""},
  {"delopt","","int","w","n","n","0"},
  {"parname","","String","w","n","n",""},
  {"formfea","","Fea","w","n","n",""} })
```

Figure 5.13 Type 2 Dimension Parent Template

The type 2 linear dimension template is shown in Figure 5.14.

```
DefTemplate(
{"d321","","DTH","ENTITY_DIM2","geo_model_d321","p","n","n"},{
  {"dldir","","Vector","w","n","n",""},
  {"dlesym2","","int","w","n","n","1"},
  {"obrell2","","Ent","w","n","n",""},
  {"ptypob2","","String","w","n","n","M"},
  {"obrel21","","Ent","w","n","n",""},
  {"obrel22","","Ent","w","n","n",""} })
```

Figure 5.14 Type 2 Linear Dimension Template

As with type 1 linear dimensions a direction of projection is required, `dldir`. This is supported in Body Coordinates of `obrell1`. The second dimension line end symbol, `dlesym2`, is also necessary. If linear dimensions are between two entities (rather than a single edge which is decoded to give the start and end points) the second entity is

supported by parameter obrel21. ptypob2 indicates how the point should be decoded from the entity. Two straight edges may be used to define an intersection point (which need not coincided with any point on or in the body), hence each point requires two defining entities. The second entities are supported by parameters obrel12 and obrel22. The parameters ptypob1 and ptypob2 are set to "I" from the default values of "M".

The make function, geo_model_d321, is used to maintain the validity of the dimension after each change in the model when the dimension workpiece is evaluated. geo_model_d321 performs the following functions:

- Tests for valid object identities. If any identities are lost the value of the dimension is set to -1, the graphic symbol routine will not generate a HOOPS segment and the dimension will not be displayed. Due to the fact that a feature may not delete itself, the feature name is added to a global list. Before the graphics are generated a housekeeping routine checks this list and the value of delopt for the features. Any features with the value zero are deleted.

- The entities are decoded to generate positions.

- The positions are tested to ensure that they are not identical. If they are, and delopt is zero, the dimension is added to the global delete list.

- The new value is calculated.

- The pos1, pos2 and val parameters of the feature are updated. The positions pos1 and pos2 are supported in World Coordinates because they are always calculated from entities that represent the current state of the model.

The type 2 angular dimension template is shown in Figure 5.15.

```
DefTemplate(
{"d322","","DTH","ENTITY_DIM2","geo_model_d322","p","n","n"},{
  {"pos3","","Position","w","n","n",""},
  {"obrel12","","Ent","w","n","n",""},
  {"ptypob2","","String","w","n","n","M"},
  {"obrel21","","Ent","w","n","n",""},
  {"obrel22","","Ent","w","n","n",""},
  {"ptypob3","","String","w","n","n","M"},
  {"obrel31","","Ent","w","n","n",""},
  {"obrel32","","Ent","w","n","n",""} })
```

Figure 5.15 Type 2 Angular Dimension Template

The angular dimension requires a third position to define its value and graphical representation, `pos3`. In exactly the same way as with the linear dimension, each point may be generated from one, or the intersection of two entities (faces, edges).

The make function, `geo_model_d322`, is used to maintain the validity of the dimension after each change in the model when the dimension workpiece is evaluated. `geo_model_d322` performs the following functions:

- Tests for valid object identities. Invalid object identities are treated as with type 2 linear dimensions.
- The entities are decoded to generate positions.
- The positions are tested to ensure that they are not identical. If the intersection point (`pos3`) is identical to either of `pos1` or `pos2`, and `delopt` is zero, the dimension is added to the global delete list.
- The new value is calculated. If the three positions are collinear (value = 0) a new normal is generated from the cross product of the old normal and the vector defined by the line of the three positions. This generates a vector that is orthogonal to the line containing the three positions. If the three positions are not collinear (value $\neq$ 0) the new normal is calculated from them.
- The positions `pos1`, `pos2`, `pos3` and `norm` are supported in World Coordinates because they are always calculated from entities that represent the current state of

the model. This is of use because the previous positions may be used to calculate the previous value which can be compared to the previous saved value. If they are different (and the stored value is not -1) this indicates that the dimension represents the exterior angle. The value calculated from the positions is always the interior angle. This is modified to be the exterior angle if necessary. The `pos1`, `pos2`, `pos3`, `norm` and `val` parameters of the feature are updated.

The type 2 diameter dimension template is shown in Figure 5.16.

```
DefTemplate(
{"d323","","DTH","ENTITY_DIM2","geo_model_d323","p","n","n"},{
  {"dldir","","Vector","w","n","n","1,0,0"},
  {"touch","","int","w","n","n","0"},
  {"spos","","Position","w","n","n","0,0,0"},
  {"epos","","Position","w","n","n","5,0,0"},
  {"arcpos","","Position","w","n","n","2.5,2.5,0"} })
```

Figure 5.16 Type 2 Diameter Dimension Template

The parameters specific to this template are as described in section 5.4.

The make function, `geo_model_d323`, is used to maintain the validity of the dimension after each change in the model when the dimension workpiece is evaluated. `geo_model_d323` performs the following functions:

- Tests for valid object identity. Invalid object identity is treated as with type 2 linear dimensions.
- The geometry of the edge entity is used to provide the centre, normal and radius. The centre and normal are converted from Local to World Coordinates.
- `dldir` is converted from body to world coordinates and used to calculated the positions of `pos1` and `pos2` from the centre.
- The positions `spos` and `epos` are provided as attributes of the entity. `arcpos` and `touch` are calculated as described in section 5.4. The parameters `pos1`, `pos2`,

spos, epos and arcpos (which are set to World Coordinates), touch and val are re-assigned to the dimension feature.

The type 2 radius dimension template is shown in Figure 5.17.

```
DefTemplate(
{"d324","","DTH","ENTITY_DIM2","geo_model_d324","p","n","n"},{-
  {"dldir","","Vector","w","n","n","1,0,0"},
  {"touch","","int","w","n","n","0"},
  {"spos","","Position","w","n","n","0,0,0"},
  {"epos","","Position","w","n","n","5,0,0"},
  {"arcpos","","Position","w","n","n","2.5,2.5,0"},
  {"parent","","Ent","w","n","n",""} })
  {"facevec","","Vector","w","n","n","0,0,0"},
```

Figure 5.17 Type 2 Radius Dimension Template

The parameters specific to this template are as described in section 5.4 with one addition, facevec. There are a number of situations when shape aspects need to be dimensioned but have no edge that correctly represents the dimension of the aspect.

Rounds or fillets applied to edges are good examples. When an edge that forms a complete circle is rounded, the resulting face forms part of a torus. In the display the silhouette lines form perfect circular arcs but they cannot be dimensioned when only edge entities may be used because no edge exists. Similarly, if straight edges have fillets or rounds applied, but the ends faces are not perpendicular to the edge then no circular edge exists. The face that the fillet or round creates is cylindrical. The facevec parameter in the case of a cylindrical face defines an offset from the point on the axis of the cylinder sited at the centre of the box containing the face. A cutting plane is sited here to generate the radius dimension, Figure 5.18.

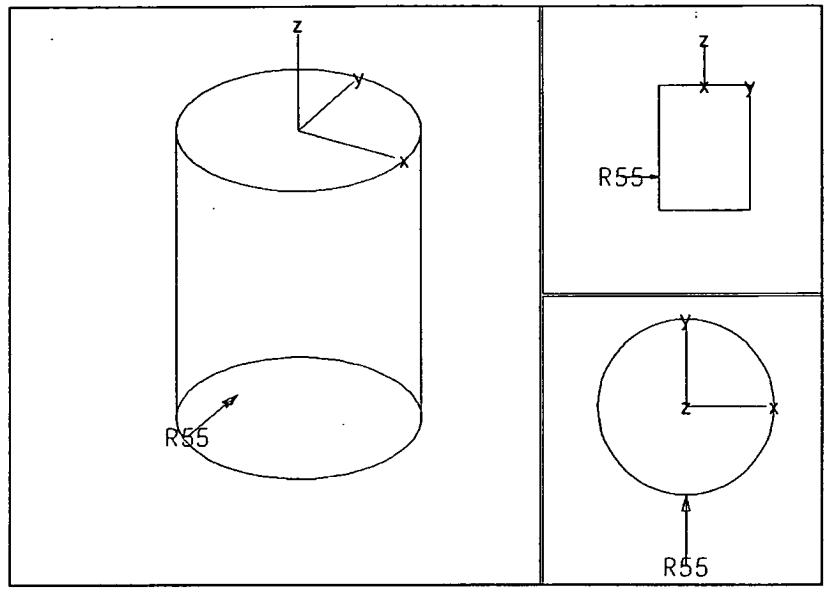


Figure 5.18 Radius Dimension Applied To Cylindrical Face

For a toroidal face, `facevec` defines a cutting plane radially from the central axis of the torus, Figure 5.19.

The make function, `geo_model_d324`, is used to maintain the validity of the dimension after each change in the model when the dimension workpiece is evaluated. `geo_model_d324` performs in a similar way to `geo_model_d323` for the diameter dimension except that `pos2` is calculated using both `dldir` and `tof` as described in section 5.4, and special checking must be undertaken to ensure that the cutting planes defined by `facevec` do actually cut the face entity.

For a cylindrical face if the cutting plane, defined by `facevec`, no longer cuts the face it is set to be located at the centre of the box bounding the face.

The treatment of a toroidal face is more complicated. The new `pos1` is calculated from the attributes of the toroidal face and `facevec` and `dldir`. If `pos1` does not lie on the face then the parameters are reflected and the opposite silhouette line is tested for, in which case `facevec` is assigned a new value. If `pos1` again does not lie on the face, the existence of some remainder of the original silhouette line is searched for, but a help arc will be needed. The possibility that a silhouette line exists on the opposite side of the torus is also checked, , in which case `facevec` is assigned a new

value and a help arc is needed. If neither test is successful a different toroidal face is searched for which has identical geometry and does produce a silhouette line that `pos1` lies on. If that fails the face ray point is used to define `facevec`.

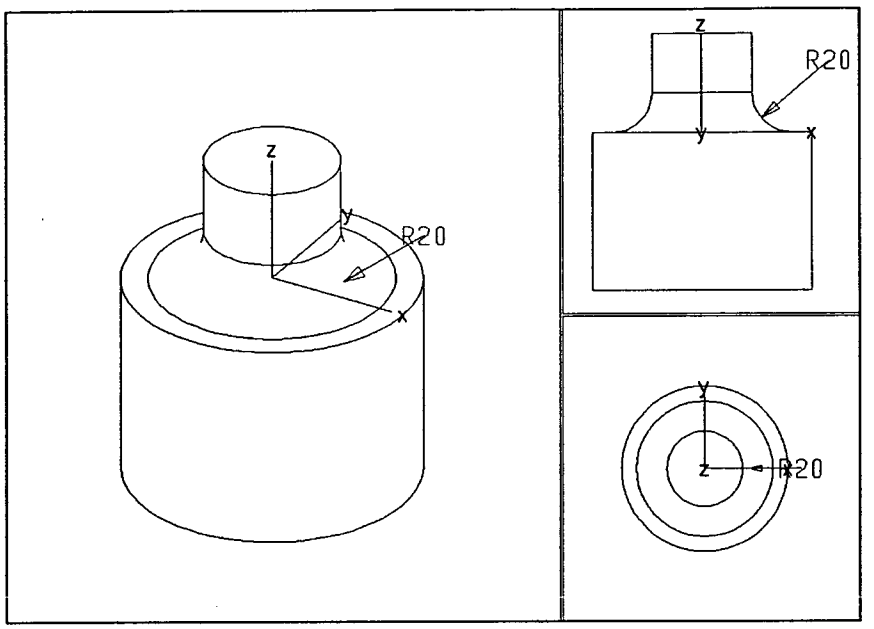


Figure 5.19 Radius Dimension Applied To Toroidal Face

The positions `spos` and `epos` are calculated from the intersection of the cutting planes with the face. `arcpos` and `touch` are calculated as described in section 5.4. The parameters `pos1`, `pos2`, `spos`, `epos` and `arcpos` (which are in World Coordinates), `touch`, `val` and any new value assigned to `facevec` are re-assigned to the dimension feature.

5.5.6.3 Type 3: 2D Drafting Contour Dimensions

Type 3 dimensions behave exactly as Type 2 dimensions. They have exactly the same parameters and methods. The only difference is in the template name. This is to be able to differentiate between the source of dimensions because this affects the functionality of editing (section 5.6.2). The templates inherit both parameters and procedures from the type 2 dimensions. The type 3 templates are defined as shown in

Figure 5.20. Their source in 2D drafting systems means that type 3 dimensions are only related to edges or vertices.

```
// Type 3 Linear Dimension Template
DefTemplate(
{"d331","","DTH","d321","e","e","n","n"},{ })

// Type 3 Angular Dimension Template
DefTemplate(
{"d322","","DTH","d322","e","e","n","n"},{})

// Type 3 Diameter Dimension Template
DefTemplate(
{"d323","","DTH","d323","e","e","n","n"},{

// Type 3 Diameter Dimension Template
DefTemplate(
{"d324","","DTH","d324","e","e","n","n"},{})
```

Figure 5.20 Type 3 Dimension Templates

The problems of performing Boolean operations with dimensions attached to the different bodies is treated in exactly the same way as described in section 5.5.6.2 for type 2 dimensions.

5.5.6.4 Type 4: Feature Parameter Dimensions

5.5.6.4.1 Method

Feature parameter dimensions are used to provide graphical representations of the size parameters that are used to define a feature. Bi-directional associativity means that the dimensions always represent the geometry of a form feature's *ShowBody*, and a dimension may be used to set the value of a parameter forcing a change to the feature, its *ShowBody,* and (though not necessarily) the resultant model.

An individual workpiece is assigned to support all parameter dimensions of a single form feature instance. The workpiece is of class WopBase because it is not required to support any geometry, nor are the features to be combined. Each feature in the

dimension workpiece is of class PDTHFea which returns an entity (in this case a graphic segment). The dimension workpiece is created when a form feature is instanced from the library and the template parameter definitions define a parameter dimension (section 5.5.6.4.) The associativity between the PDTHFea and the form feature parameter (class FeaPar) is by setting of private pointers. The link is accessible in MCL+ via the MCL+ version of the public C++ function as follows:

```
Fea dim_fea = GetParDim(Fea this_instance, String parameter_name)
```

where `this_instance` is a pointer to the form feature and `parameter_name` the name of its parameter. A pointer to the PDTHFea which supports the description of the dimension is returned, `dim_fea`.

From the user's point of view the PDTHFea is accessed exactly like the other features despite its more limited description in a form feature's template definition. More parameters are supported for a PDTHFea than are described in the form feature template dimension definition, but as read only attributes. The parameter `formfea` supports a reference to the form feature in which the dimension represents a size parameter defined by `parname`. These two parameters enable the bi-direction associativity to be supported.

The user may also switch on or off the operation of parameter dimensions. This is useful when form feature descriptions are over dimensioned, the user defining which group of dimensions to use by selecting a particular mode. The make method of the feature can turn on those parameter dimensions that are required and switch off those that are not.

The graphical representation of dimensions are described within the feature parameter description and form part of the template definition described in section 5.5.6.4.

Procedures of the form:

```
proc_name(Fea this_instance, String parname)
```

may be used instead of giving constant values in the template definition. Such a procedure is implemented by the kernel as an `if_needed` procedure for a parameter value. This is a very simple approach utilising the architecture of the feature modelling kernel but suffers two disadvantages:

1. The interface is fixed. This has the effect that procedures cannot be generalised with the use of extra data parameters.

2. More significantly, any parameter that has an `if_needed` procedure associated with it may not have the parameter set by the user. This is a problem for the parameter `locavec` of diameter parameter dimensions. It is desirable that the diameter dimension line stays a fixed distance from the shape aspect, this requires a function to program it, but it is also necessary that the user be able to set the value of the distance. This means that an extra parameter has to be used that the user may set, while the `if_needed` function reads this parameter to maintain the fixed offset from the arc. This parameter is called `ptof`.

The concept of form feature origins was introduced in chapter 4. It is clear that in many cases of feature definition these origins would make suitable attachment points for the parameter dimension representations. The parameters `pos1`, `pos2` and `pos3` may use the feature origin designator together with the "@" sign to signify that the kernel should calculate the position from the respective origin. An example is given in section 5.5.6.4.5.

5.5.6.4.2 Thread Values

Threads on objects (interior or exterior) are well defined in the standards. Threads and threaded objects have been implemented as standard features: thread, threaded hole, etc. Any thread can be created by editing the tables relating value to textual identifier. Therefore the ability to display textual thread signs is only required by type 4 parameter dimensions. Two parameters are used in the form feature: a real parameter value used to generate the geometry and the graphic representation of the

dimension lines, and a string parameter value used to support the textual description of the thread size, M8, for example. In order to support the textual parameter value in the dimension feature an extra parameter is used in the PDTHFea description: `ctext`. It has been designed to be flexible because any characters in `ctext` other than "`@UNDEF`" will be displayed instead of the numerical value. In the case of a thread the textual string M8 is assigned.

5.5.6.4.3 Template Definition Extension

Each sub-list describing a parameter in the list `parameters` defined in section 4.3.3 is extended with the addition of one parameter: `dimension`. Therefore the sub-list `parameters` now has the following definition:

```
parameters =        {
{ name, alias, type, access, if_needed, if_set, default, kind,
  decode, dimension }
....
                    }
```

The parameter `dimension` describes the graphical representation of the form feature size parameter. It is defined as a list of string values, and has the following form:

```
dimension = { pos1, pos2, pos3,
              norm, locavec, dldir,
              dlesym1, dlesym2, adir, 0, tof, ptof,
              spos, epos, arcpos, touch
            }
```

where the parameters have the meanings of the parameters already described in section 5.4, plus an additional parameter that has not already been defined:

`ptof` This parameter is used only with diameter dimensions. It is a vector that indicates the distance and direction of the midpoint of the diameter dimension line taken across the diameter from either

- the centre of the defining circle (for dimension lines which pass through the centre axis), or

- the perimeter of the circle (for dimension lines which are positioned outside an imaginary cylinder defined by the circle)

It is used because of the mapping of functions to `if_needed` procedures as explained in section 5.5.5.4.1.

The parameters `ttype`, `tol1` and `tol2` are also supported by the parameter dimension feature as described in section 5.4. A further additional parameter, `ctext`, is supported.

`ctext` This parameter is a string value that the user may wish to have displayed instead of the actual numerical value of the dimension. This is used in the case of screw threads to denote the ISO size, `M8` for example. When this parameter is set to a string other than `@UNDEF` the string is written in place of the actual value of the dimension.

All parameters may be defined by constants, MCL+ procedures, or C++ procedures (not useful for user defined features). The parameters `pos1`, `pos2` and `pos3` may additionally use the notation `"@origin"` , where `origin` is a valid `origin` definition for that feature.

5.5.6.4.4 Examples

1. Block

The block feature shown in Figure 4.12 and the template definition of Figure 4.14 may be extended to describe the size parameters with dimensions as shown in Figure 5.21. The use of the special notation to indicate that the positions `pos1` and `pos2` should be calculated from the origins is shown. The template definition of Figure 5.21 produces the feature and dimensions shown in Figure 5.22 when instanced.

```
DefTemplate(
 {"b1211","","BASE","---", "make_block", "n","n","n"},
 { {"Ursprung","","String","s","n","p","M"},
   {"x1","","real","s","n","n","120","l",{},
    {"@-x-y-z","@x-y-z","",
     "0,-1,0","0,0,-10","1,0,0",
     "1",  "1",  "0", "0",   "0.0"}},
   {"y1","","real","s","n","n","300","l",{},
    {"@x-y-z", "@xy-z", "",
     "1,0,0", "0,0,-10","0,1,0",
     "1",  "1",  "0", "0",   "0.0"}},
   {"z1","","real","s","n","n","60", "l",{},
    {"@-x-y-z","@-x-yz","",
     "0,-1,0","-10,0,0","0,0,1",
     "1",  "1",  "0", "0",   "0.0"}} })
```

Figure 5.21 Template Definition For Block With Parameter Dimensions

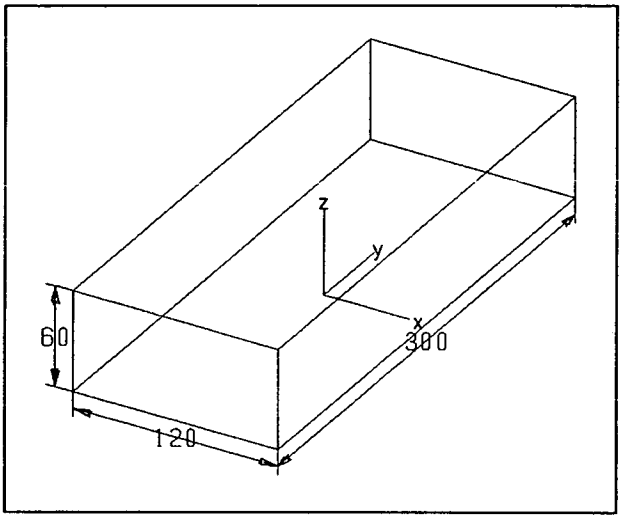


Figure 5.22 Instance Of Block Feature With Parameter Dimensions

2. Cylinder

The template definition of a cylinder feature (a subtractive feature that represents a hole) is shown in Figure 5.23. The template definition includes the extensions used to describe the size parameters with dimensions.

```
DefTemplate({"sz","Hole","SUB","---", "p", "p","n","n"},
 {{"Ursprung","","String","s","n","p","M",m,{}},
  {"d","","real","s","n","n","25","d",{},
     {"solDimDiamPos1FullNegZ","solDimDiamPos2FullNegZ","",
      "0,0,1","solDimDiamLocaVecFullZ","1,0,0",
      "1",  "1",  "0", "0",  "0.0", "0,-10.0,0",
      "solDimDiamPos1FullNegZ", "solDimDiamPos1FullNegZ",
      "solDimDiamArcPosFullNegZ", "3"}},
   {"h","","real","s","n","n","60","l",{},
     {"@-x-z","@-xz","",
      "0,1,0","-10,0,0","0,0,1",
      "1",  "1",  "0", "0",  "0.0"}} })
```

Figure 5.23 Template Definition For Cylinder With Parameter Dimensions

It will be seen by comparing the description of the length parameter, "h", representing the height of the cylinder (depth of hole) with the description of the diameter parameter, "d", representing the diameter of the cylinder (hole), that the diameter parameter requires a number of procedures to be defined. For a static dimension that could not be rotated (by changing `dldir`) origin positions could again be used. However, due to the fact that `pos1` `pos2`, `spos`, `epos`, and `arcpos` are dependent on `dldir` the parameters must have procedures associated with them. `locavec` is dependent on `ptof` to maintain a fixed offset under changes to the diameter of the cylinder, and so requires a procedure.

The procedures to calculate `pos1`, `pos2` and `arcpos` (`spos` and `epos` are the same as `pos1` and `pos2` for a complete circle) all use the face that the z origin defines the z component (in Feature Coordinates) of `pos1`, `pos2` and `arcpos`. If, as is typical, many different features use diameter dimensions to describe shape aspects, a general procedure would be useful using a particular origin as a parameter, and a switch parameter for `pos1` or `pos2`. Unfortunately the limitation of the `if_needed` function implementation (fixed parameter interface) means that generalisation is limited to features which have "z", "M", and "-z" origins, for which six separate functions exist (one each for `pos1` and `pos2`). They may however be applied to any user defined feature which has these origins with respect to its feature z axis. The procedures work with dimension planes both parallel and perpendicular to the plane of the (imaginary) arc.

The template definition of Figure 5.23 produces the feature and dimensions shown in Figure 5.24 when instanced and combined with the block to form a workpiece model.

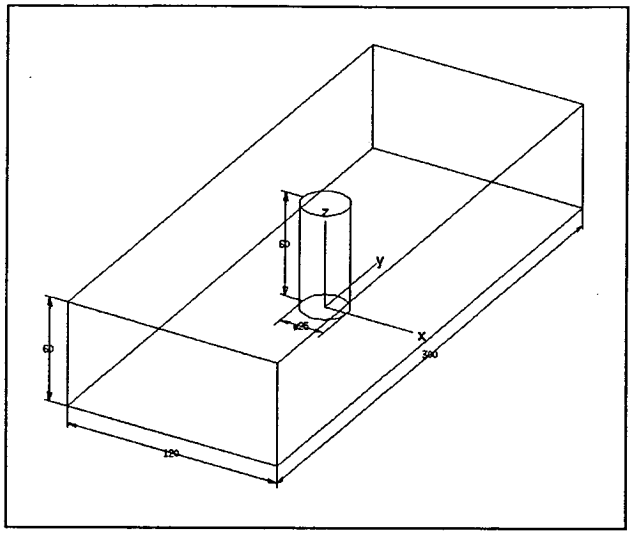


Figure 5.24 Hole And Block Features Combined In Workpiece Model

5.5.6.5 Type 5: Sketcher Constraint Dimensions

Type 5 sketcher constraint dimensions are a mixture of both type 2 and type 4 dimensions. Like type 4 dimensions information about the dimension is stored in two places, both in the dimension and in the object it relates to. In the case of a parameter dimension it is the form feature, whereas it is the constraint with type 5 dimensions. Unlike type 4 dimensions where the dimension position parameter values are programmed, type 5 values, like type 2 are directly derived from the topological and geometrical elements that are produced by the constraint manager and supported by the constraints. Unlike type 4 dimensions which must be user definable within the definition of the form feature template, sketcher dimensions are purely representative of the constraints that are supported by the constraint manager. These are provided by the system and applied by the user; the defining parameters do not need to be programmed and instances are created interactively. The elements in the sketcher are simpler than the general elements found in a 3D solid. A sketcher object consists only of edges all of which lie in a plane. For these reasons the template definition has

been able to be reduced to a minimum number of parameters without the need for redundancy that was used in the type 2 definitions. The defining positions are all supported and derived directly from the constraint entity.

In MCL+ a constraint entity has a number of attributes of which those that are important for dimensioning are: `Name`, `Genre`, `Value`, `Entity1`, `Summit1`, `Entity2`, `Summit2`. Only `Value` and `Name` are can be set directly as attributes of an instance. The other attributes are read only and are assigned via the creation function. `Name` is a unique identifier that is set to be the same as the name of the dimension that it is associated with. `Genre` is the class of dimension: Distance, Radius, etc. `Value` is the value of the constraint which is displayed by the dimension. `Entity1` and `Entity2` are the topological objects which are related by the constraint. `Summit1` and `Summit2` specify how the topological objects should be decoded to generate a point (section 5.5.5.2). Not all constraints require the definition of `Entity2`, `Summit1` and `Summit2`. A radius constraint requires only the definition of `Entity1`, for example.

A parent template definition is again used as shown in Figure 5.25.

```
DefTemplate({"ENTITY_DIM5","","DTH","---", "n", "n","n","n"},{
  {"tof","","real","w","n","n","0.2"},
  {"dlesym1","","int","w","n","n","1"},
  {"ctext","","String","w","n","n",""},
  {"ttype","","String","w","n","n",""},
  {"tol1","","real","w","n","n","0.0"},
  {"tol2","","real","w","n","n","0.0"},
  {"adir","","int","w","n","n","0"},
  {"locplane","","int","w","n","n","0"},
  {"parname","","String","w","n","n",""},
  {"formfea","","Fea","w","n","n",""}
 })
```

Figure 5.25 Type 5 Sketcher Dimension Parent Template Definition

The parameters are as previously described. `ctext` is used to display the value of a dimension in square brackets to signify that the value does not correspond to the current status of the geometry. This may occur if the value of a constraint dimension is set but the constraint manager has not re-evaluated the geometry, or if the contour is over constrained and is solved for a subset of the constraints.

To date the constraints that have been implemented as dimensions are distance and radius. The type 5 linear dimension template is shown in Figure 5.26.

```
DefTemplate({"d351","","DTH","ENTITY_DIM5", "n", "p","n","n"},{
  {"leadlen","","real","w","n","n","20"},
  {"dlesym2","","int","w","n","n","1"}
 })
```

Figure 5.26 Type 5 Linear Dimension Template Definition

A Distance constraint may be created for a single (non-closed) edge, the positions, `pos1` and `pos2`, required for the graphic segment generation are decoded as the start and end points of the edge. The decoding is implemented as part of the symbol procedure in C++. When a distance constraint is applied between two edges in the contour the points are decoded according to the respective summits. The direction of projection is defined by the implementation of the distance constraint and is always point to point. Therefore `dldir` is also calculated in the symbol procedure. The use of a vector, `locavec`, as with the other dimension types to position the dimension line in the dimension plane is not suitable for application to constraint dimensions. This is because the vector would be maintained in the Body Coordinate system, yet even though an element of a contour would change its position (either by direct editing or through evaluation by the constraint manager) the Body Coordinate system defined by the transform of the whole body would remain unchanged. Therefore `locavec` could no longer be perpendicular to the dimension line direction. Due to the fact that, `dldir`, is always the direction from `pos1` to `pos2`, the projection line direction is defined to be the unit vector of the cross product,

$$\textbf{projection\_line\_direction} = \textbf{norm} \times \textbf{dldir} \qquad (5.1)$$

where `norm` is the dimension plane normal vector. This definition is always correct with respect to the element(s) being constrained. A single real parameter, `leadlen`, is used to define the distance (positive or negative) with respect to the projection_line_direction that defines the position of the dimension line.

The type 5 radius dimension template is shown in Figure 5.28.

```
DefTemplate({"d354","","DTH","ENTITY_DIM5", "n", "p","n","n"},{
  {"dldir","","Vector","w","n","n","1,0,0"}
 })
```

Figure 5.27 Type 5 Radius Dimension Template Definition

A Radius constraint requires only one edge entity to define it. However the dimension display requires the definition of a vector, `dldir`, which cannot be derived from the edge(s) in the same way that the direction of projection can be for the Distance constraint. `dldir` will suffer the problem that, being described in Body Coordinates, it will not respond to changes to the elements of the contour, rather than the contour as a whole. However, this is does not pose any difficulties for the generation of the graphic segment because it is only the positioning of the dimension line around the arc that is affected, which may or may not require the creation of a help arc. This is easy to do given that the dimension is related to the edge and the start and end points are known as edge attributes.

A contour with type 5 constraint dimensions is shown in Figure 5.28, before and after changes to some of the contour's edge elements.

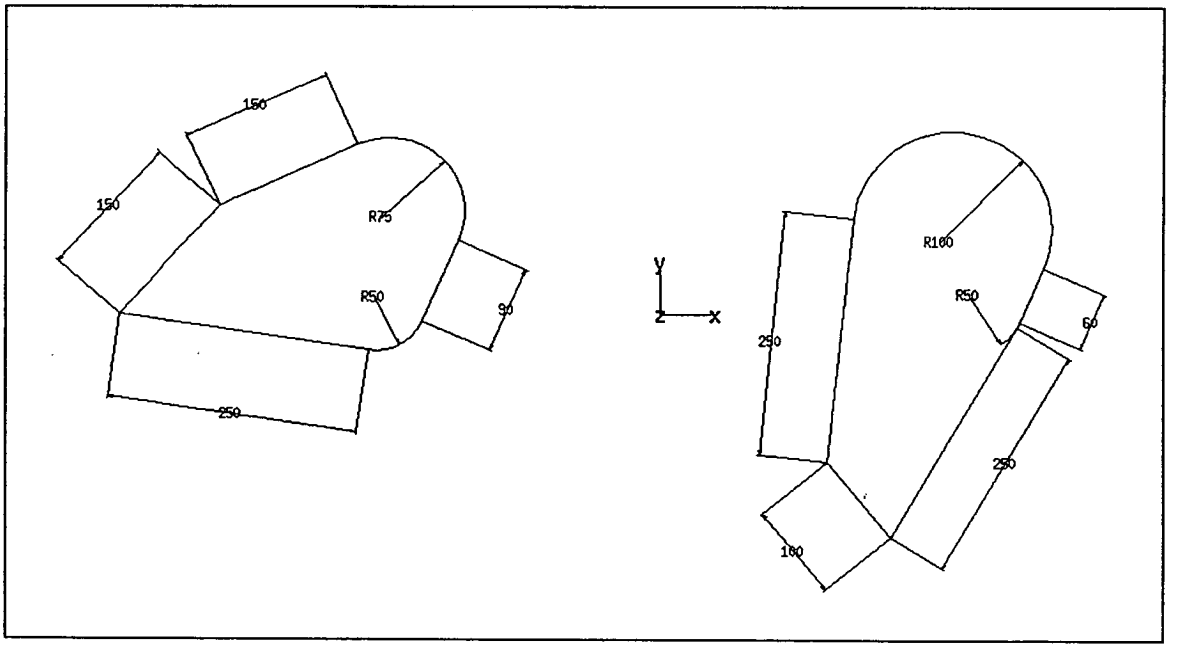


Figure 5.28 Type 5 Constraint Dimensions

5.5.7 Geometric Tolerances

Geometric Tolerance functionality has been investigated through implementation of a parallel tolerance. The template definition is shown in Figure 5.29. The graphic display of a geometric tolerance is similar to that of a linear dimension, with one important difference. The projection lines from the faces or axes being toleranced start in a perpendicular direction to each face or axis. In this implementation a parallel tolerance may be applied between planar faces and / or axes derived from cylindrical faces. The parameters defined in the template are similar to those used to define a Type 2 linear dimension which the implementation of the parallel tolerance closely follows.

```
DefTemplate(
  {"para_tol","","SUB","---","make_para","symbol_para","n","n"},

  {{"obrel11","","Ent","w","n","n",""},
   {"obrel21","","Ent","w","n","n",""},
   {"tol_val","","real","w","n","if_set_para_val","0.01"},
   {"ctext","","real","w","n","n","para"},
   {"pos1","","Position","w","n","n",""},
   {"pos2","","Position","w","n","n",""},
   {"dldir","","Vector","w","n","n",""},
   {"norm","","Vector","w","n","n",""},
   {"locavec","","Vector","w","n","n",""},
   {"obrel","","String","w","n","n",""},
   {"delopt","","int","w","n","n",""}
  })
```

Figure 5.29 Geometric Parallel Tolerance Template Definition

A parallel tolerance is defined between two entities in the model, `obrel11`, `obrel21` of which the reference body coordinate system is indicated by `obrel`. `delopt` is required because the parallel tolerance could be applied to, or between, solid rather than workpiece bodies. `pos1` and `pos2` are decoded from the entities by the general decoder as described for Type 2 dimensions. `ctext` is used to build the displayed value from the numerical value and the type of tolerance, in this case parallel.

The parallel tolerance has been prototyped by developing the symbol procedure in MCL+. The lines displayed are formed by returning a wire body from the symbol

procedure. The text is displayed by the linear dimension segment call with parameters chosen to cause no dimension lines to be displayed. Parallel tolerances are shown in Figure 5.30.

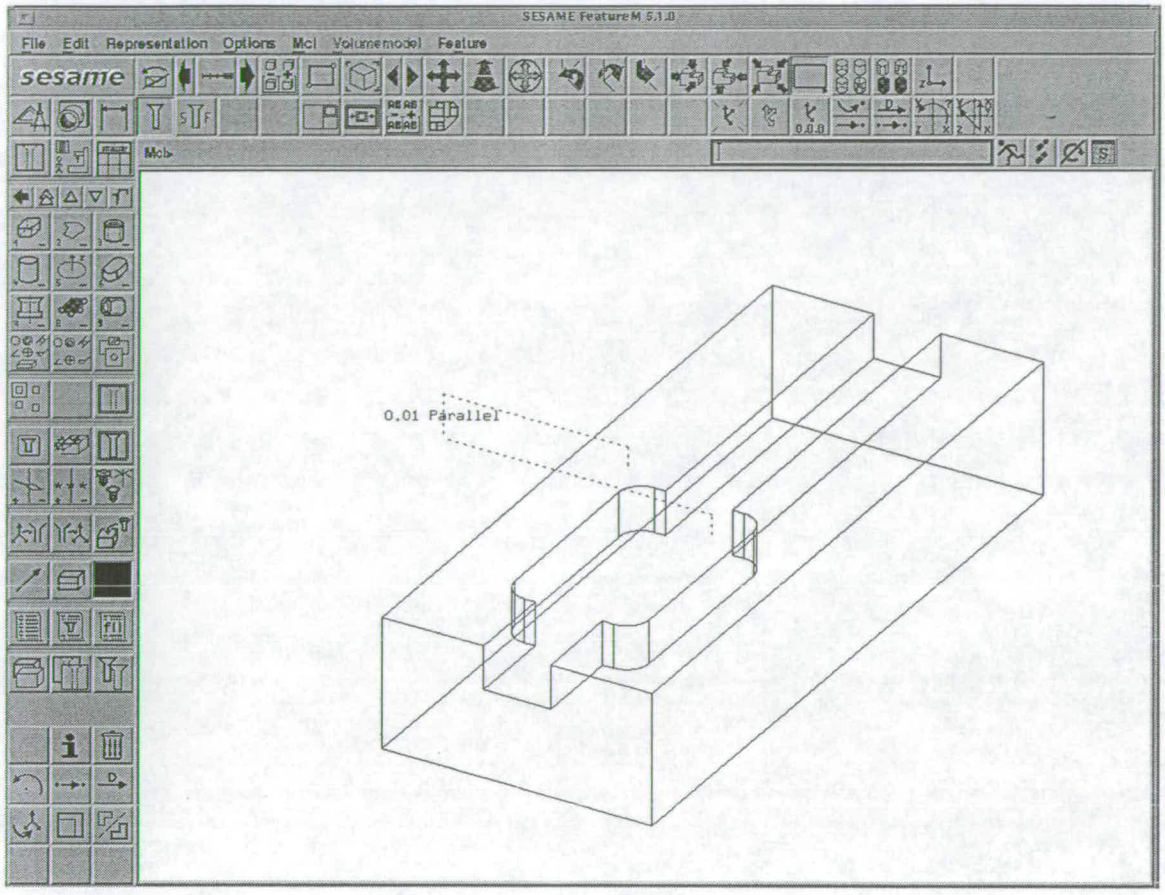


Figure 5.30 Parallel Geometric Tolerance Representation

5.6 User Interface

The main menu of buttons for the dimensioning and tolerancing module is shown in Figure 5.31. Very heavy use is made of the decoder described in chapter 3 during input. Each button may have 1, 2 or 4 commands associated with it. A button may be complete, split into two halves, or four quarters. Each element has a command associated with it which is selected depending on where the user picks. Each command is implemented as a decode list as described in chapter 3. Modifier buttons are used to direct the sequence of input steps. The aim of the input steps is to reduce

to a minimum the number of actions the user must make to create or modify a dimension or tolerance. The status of modifier buttons generally remains the same until set to another value. Modifier buttons can be toggle (two value) or multi-state (multi-value).

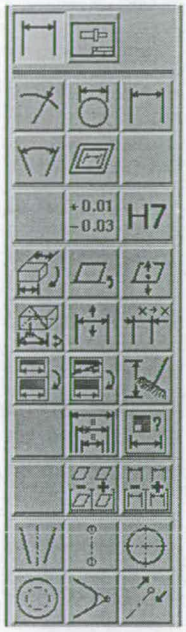


Figure 5.31 Dimension And Tolerance Module Main Menu

5.6.1 Creation

Each *kind* of dimension (linear, angular, diameter, radius) has a creation button, Figure 5.32.

Dimension types 1, 2 and 5 require input sequences; types 3 and 4 are not defined interactively. The only differences between the input sequences of types 1, 2 and 5 are the way the positions defining the dimension are generated.

Type 1 linear and angular dimensions use the built-in functionality of the system decoder to generate positions from elements or via the pop-up menu functions; the user is prompted for a position. Type 1 diameter and radius dimensions are based on the curved edge object but the object itself is not part of the dimension

representation; the user is prompted for an edge. Type 2 dimensions decode the objects into positions both during input and within the make (validation) routines. This decoding is part of the dimensioning module rather than the general decoder. The user is always prompted for objects which are maintained as part of the dimension definition. Type 5 dimensions require edge objects that are picked and decoded via the summits to provide positions from the constraints that are simultaneously created.

The type of dimension may be selected before creation by setting of a modifier button. Only types 1 and 2 may be chosen from the dimensioning module. Type 5 dimensions may only be created when the sketcher module is active.

It is important not only to know what type of dimension is to be created but with how many elements. For instance, a linear dimension may be created using a single edge (with different start and end points) or between two elements (faces, edges, vertices). Similarly an angular dimension may be created from three elements defining positions or between two intersecting planar faces or straight edges. The number of elements is selected with a modifier button.

It is clear that the object(s) must always be picked by the user to indicate what is to be dimensioned. In 3D the next most important definition is that of the dimension plane. In non-technical views the direction of the active Local Coordinate System z-axis defines the direction of the normal vector of the dimension plane. The position of the dimension plane (the attitude having been defined) can be described as *local* or *global*, and is selected by a third modifier button. The position of the *local* dimension plane is defined by the first position picked or decoded for a linear dimension, the plane defined by the three decoded points of an angular dimension, and the plane of the arcs of diameter or radius dimensions. The position of the *global* dimension plane is defined by the origin of the active Local Coordinate System. This may or may not be valid for the dimension; check procedures validate the dimension plane after picking of the positions or objects. For instance, diameter dimension planes must be parallel or orthogonal to the plane of the arc, angle dimension planes must be parallel

to the plane defined by the three defining positions. Radius and any Type 5 dimensions always use the *local* dimension plane, in which case the modifier is disabled. Picking objects in a technical view sets the dimension plane position to be *local* and the direction of the normal vector to be equal to the line of sight defined for the view (not necessarily the current line of sight as the camera in technical views can also be rotated, but may be reset). Dimensioning in technical views is effectively the same as dimensioning a 2D engineering drawing except that the dimensions are related directly to the 3D solid or feature models.

The higher level the entity that is used for dimensioning, the more likely persistent object identity will be maintained. It is better to dimension between faces, than use edges. A higher level entity can be used better to reason about how a dimension should be created. For instance, creating a linear dimension between two parallel faces allows the dimension plane and dimension line direction to be deduced. The dimension line direction is formed by the face normals. The dimension plane normal is perpendicular to the dimension line direction and in the direction of the Local Coordinate System z-axis. Single edges have the dimension line direction set to be from start point to end point. The perpendicular to an adjoining face is used for the dimension plane normal vector. These rules for choice of dimension line direction may be overridden by the user using the modifier for selecting change of direction.

The next parameter needed to define the dimension line (not for angular dimensions) is the direction of projection. This can be automatically calculated for linear dimensions or set by the user. The direction is always projected onto the dimension plane regardless of input. The automatic value is (projected) point to point. The direction must be input for diameter dimensions if the dimension plane is parallel to the plane of the arc. If the dimension plane is perpendicular to the plane of the arc (globally defined or by picking in a technical view) the direction is defined and lies in both the plane of the arc and the dimension plane. A radius dimension always requires input of the dimension line direction. The direction of the dimension line may be modified during input of the position of the dimension line in the dimension plane because the direction is best visualised by the rubber band. The final input

parameter is the position of the dimension value on the dimension line. This is required for all dimensions.

Angular dimensions have a modifier that indicates whether the interior or exterior angle should be represented. This replaces the dimension line direction modifier button which is not applicable.

5.6.2 Editing

The user is provided with the following edit functions as depicted in the menu shown in Figure 5.31:

- Change the dimension plane.

  Not applicable to radius dimensions. Only the position of the dimension plane may be changed for angular dimensions. The plane must remain parallel or perpendicular to the arc for diameter dimensions. For type 4 parameter dimensions the plane may only be changed to a plane perpendicular (but valid) to the original so as to maintain consistency (measuring direction) with the size parameter it is to represent. This is applicable to all type 4 linear dimensions and those diameter dimensions which represent the diameters of elliptical shapes that have fixed directions.

  It is useful to move the graphical representation from one technical view to another. Dimensions only appear in a technical view when the dimension plane normal and technical view line of sight (definition, rather than current) are parallel.

- Rotate the dimension plane.

  Only applicable to linear dimensions when the dimension line direction is parallel to the direction between points one and two. It is useful to visualise dimensions in a view with a non-technical line of sight.

- Move the position of the dimension plane.

  Not applicable to radius dimensions. This allows the dimensions in 3D to be positioned away from the model for clearer visualisation.

- Change the direction of the dimension line.

  Not directly applicable to angular dimensions, the function acts as a toggle between interior and exterior representation. Not applicable to linear type 4 parameter dimensions and certain diameter dimensions which represent the diameter of an ellipse. For example, it is useful to measure the "height" rather than "width" of a sloping face with a linear dimension. This allows radius and diameter dimension lines to be positioned for maximum clarity.

- Move the position of the dimension line in the dimension plane.

  Applicable to all dimensions. This allows dimension lines to be positioned for maximum clarity.

- Move the position of the text along the dimension line.

  Applicable to all dimensions. This allows the dimension text to be positioned for maximum clarity.

- Equispace dimension lines.

  It is tedious to individually position dimension lines so that together they are visually clear. This function allows dimension lines to be positioned with a constant offset (which may be zero) from a root dimension (the first selected). This allows the layout to be optimised for maximum clarity.

- Set dimensional tolerances.

  The user may set different plus-minus tolerances (the default), or by selecting from a pop-up menu, equal plus-minus tolerances, plus-zero tolerances, minus-

zero tolerances, or zero-zero tolerances (which removes the dimensional tolerance from the display).

- Set limit.

  Limit values may be set textually or selected form a pop-up menu. The pop-up menu presents the ISO 286 [ISO286] standard limits and fits from 1 to 500 mm appropriate for the value of dimension to be toleranced.

- All dimensions, other than type 4 parameter dimensions may be deleted. Deleting a type 5 sketcher dimension also deletes the associated constraint.

In addition, there are other functions which filter the display of the dimensions. A small indicator in the form of a rectangle and arrow may be switched on or off in the graphical display to show the attitude and direction of the dimension plane. Type 4 parameter dimensions (because they cannot be deleted) may be hidden from display. This may be done per workpiece, feature, dimension, or view. Other filters that assign or remove dimensions from views are implemented automatically for technical views, i.e. only those dimensions may be visualised in a technical view when the dimension plane normal vector is parallel to the definition (not necessarily current) line of sight of the technical view.

5.6.3 Auxiliary Dimension Features[1]

Auxiliary dimension features provide 3D representations of the 2D indicators for symmetry, centre lines and pitch circles, and construction geometry such as single lines or intersection points. These are implemented as in the same way as type 1, 2 and 3 dimension types, each type having its own work piece. They produce symbolic solid objects that may be used to generate positions for dimensioning.

---

[1]The implementation of auxiliary dimension features is predominantly not the work of the author.

5.6.4 Example

The parts shown in the 2D engineering drawing of Figures 1.5 and 2.4 have been supplied for evaluation and subsequently modelled with the feature modeller as implemented in chapters 3, 4 and 5. The feature model of the part shown in Figure 1.5 in presented in chapter 7, Figure 7.14, where it will be discussed. The representation shown in Figure 5.32 is of the feature model of the part shown in the drawing of Figure 2.4. The model representation is complex and is not typical of that presented by traditional a solid modeller.

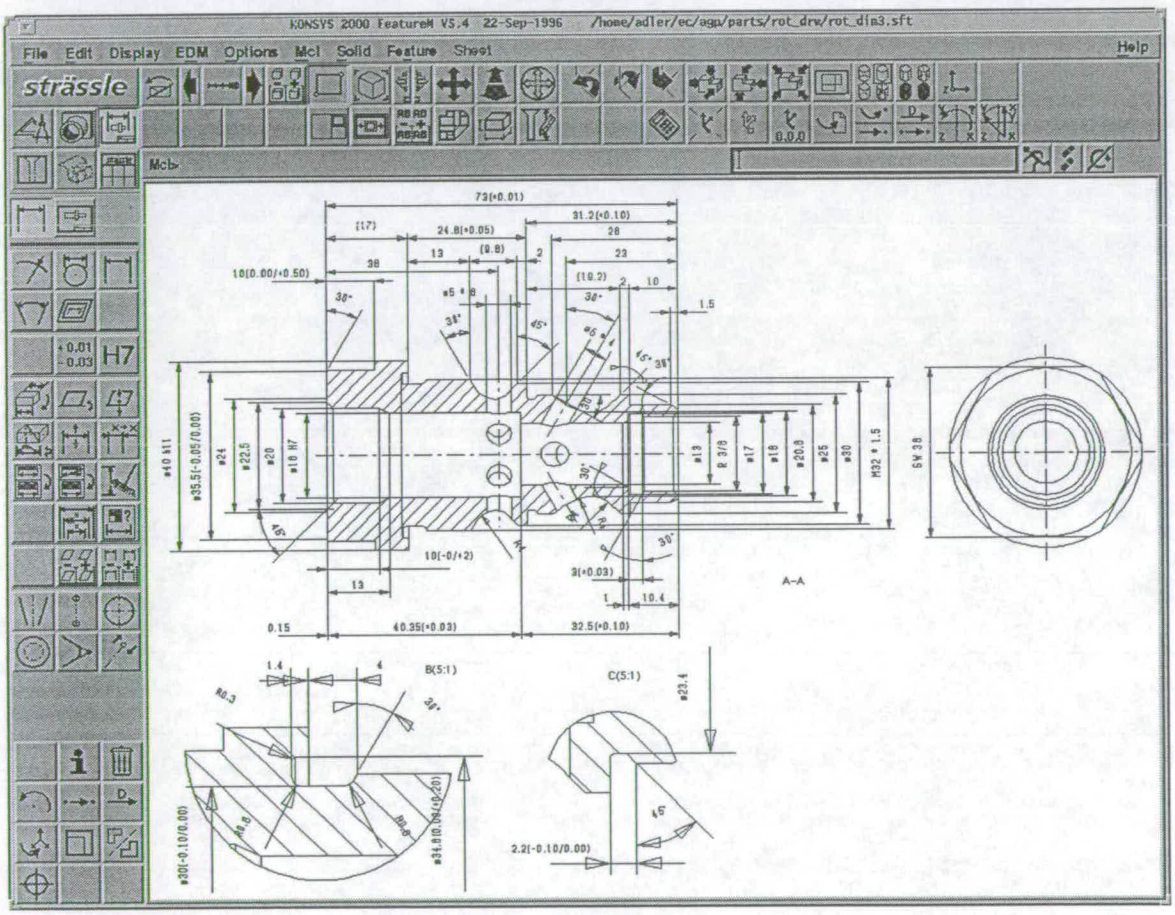


Figure 5.32 Feature Model Using Dimensions And Tolerances

The model is completely built from features. Intrinsic and extrinsic features are used; the latter forming the complex grooves by rotationally sweeping contours defined in a sketching module. The model uses the technical views, cross-section views, detail

views, auxiliary dimension features and the dimension and tolerance features described in chapter 5 to display the 3D model as if it were a 2D drawing. However all views are of the 3D model and each view may be rotated and fully shaded. The model is constructed so that the parameters of features correspond to the dimensions of the drawing. In this case parameter dimensions are used and can be used to change the geometry. Where the resultant geometry of the model is to be dimensioned uni-directional associative dimensions and tolerances are applied. Thereafter, they respond to changes in the model. A significant complication is that all dimensions (with one exception) are applied to cross-section or detailed views of the model. This poses even greater difficulties for the persistency of object identity and it was necessary to use non-associative type 1 dimensions for some of the detailing of the grooves.

The aim of removing the need for separate 2D drafting systems, even for detailing, by applying feature based techniques to the representation of dimensions and tolerances, has been shown to be possible in Figure 5.32.

5.7 Summary

In this chapter the needs of dimension and tolerance representations associated with both solid and feature model have been established. A classification of dimension types has been proposed and an architecture to support these types developed. A system has been implemented which allows both solid and feature models to be dimensioned and toleranced. The dimensions and tolerances are uni-directionally associative with the models and will follow any changes made. The template definition for user defined features presented and extended in chapter 4 has been further enhanced to enable size parameters to have dimension representations described within the feature template. Such parameter dimensions may be used to change the model and are said to be bi-directionally associative. Bi-directionally associative dimensions representing sketcher constraints have also been implemented.

The next chapter (chapter 6) will explicitly describe a number of aspects of the work concerning the original concepts investigated, industrial collaboration, analysis and testing. The next chapter will present the fundamental concept of the thesis that has been investigated through the aims and objectives of the work. The importance and extent of industrial collaboration involved in the research will be described, together with references to the information and feedback provided. Chapter 6 will present the research techniques used to acquire and analyse information in order to define the software architecture which resulted in the implementation reported in chapters 4 and 5. The software testing methodologies employed for the evaluation of the experimental software and the results of tests will be given. The original contributions to knowledge made by the investigations will be stated.

Chapter 6

Analysis

6.1 Introduction

6.1.1 Chapter Overview

The purpose of this chapter (chapter 6) is to describe explicitly a number of aspects of the work concerning the original concepts investigated, industrial collaboration, analysis and testing.

Chapter 1 introduced feature modelling technology and stated the aims and objectives of the work reported in this thesis. Chapter 2 presented the technologies associated with the concepts presented in the thesis and reviewed the literature. Chapter 3 described a form feature modeller that allows user defined feature definition. The feature modeller described in chapter 3 was selected as a test bed for the new concepts implemented in chapters 4 and 5. In chapter 6 (this chapter) a number of aspects of the work will be reported. Chapter 6 will explicitly present:

a) The fundamental concept of the thesis that has been investigated through the aims and objectives of the work.

b) The importance and extent of industrial collaboration involved in the research. References to the information and feedback provided will also be given.

c) The research techniques used to acquire and analyse information in order to define the software architecture.

d) The software testing methodologies employed for the evaluation of the experimental software and the results of tests.

e) The original contributions to knowledge made as an outcome of the investigations.

In this chapter the relationships between the areas of investigation, the sources of information provided and the sections of the thesis analysing the information are presented.

In the next chapter (chapter 7) the results of the investigations will be discussed.

6.1.2 Chapter Structure

Section 6.1 explains the purpose of this chapter, introduces its contents and provides an overview of the chapter structure. Section 6.2 presents the fundamental concept of the work represented by the aims and objectives. Section 6.3 details the extent of industrial collaboration. Section 6.4 describes the techniques used to acquire and analyse the information used in defining the software architecture. Section 6.5 presents the software testing methods used. Section 6.6 states the original contributions made to knowledge by the work. A chapter summary is provided in section 6.7.

6.2 Fundamental Concept

The fundamental concept investigated in this thesis is to integrate in a single computer software representation the information required by designers to define a component for analysis and manufacturing, thus enabling software based simultaneous engineering strategies to be implemented. In essence, the aim is to create a partial product model and a graphical user interface allowing the model to be viewed and changed. The concept necessitates combining into a single integrated and fully associative model the requirements of 2D drafting for the production of detailed engineering drawings, representing 3D solid geometry containing both analytic and free form surfaces, modelling dimensional and geometric tolerances, and surface conditions. The concept of an integrated model has been described in section 1.4 and conceptualised in Figure 6.1.

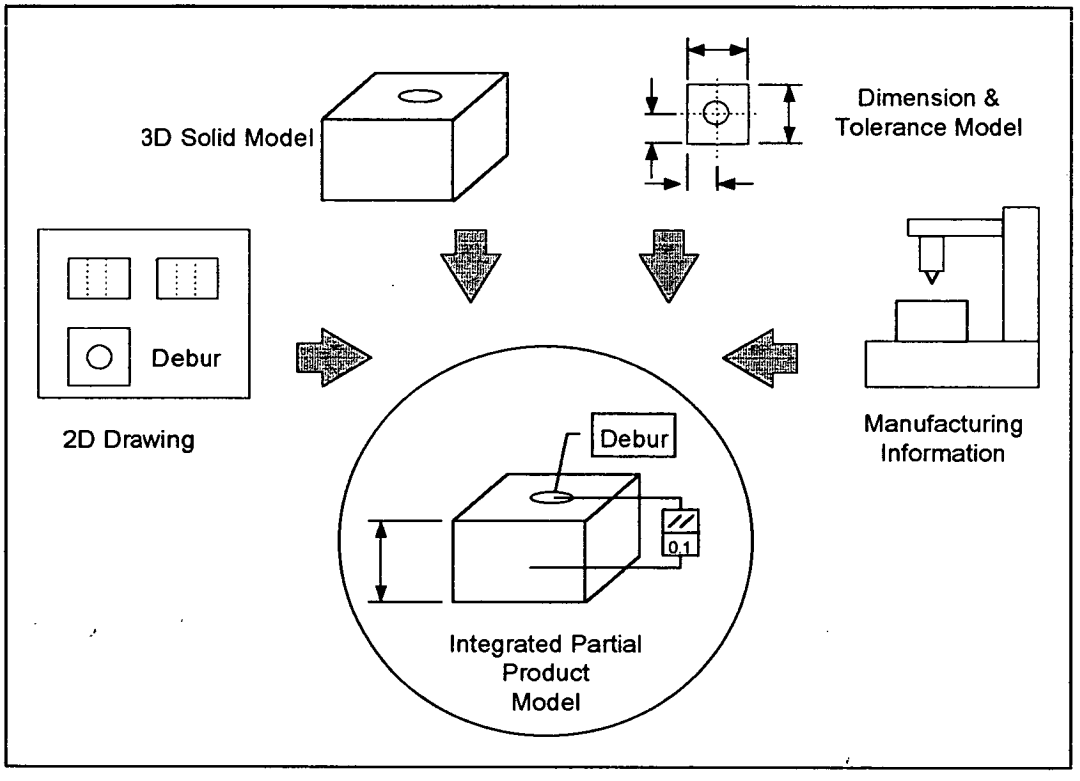


Figure 6.1 Fundamental Concept

Current product model definition standards, described in section 2.6, constitute a series of parts covering 2D drawings, BREP solid models, dimension and tolerance

models, and feature models, but do not provide the necessary integration. Product modellers have focused on the data structure, are unwieldy and do not provide the required design environment or functionality for model creation and manipulation. Feature models are accepted as capable of supporting data rich representations which can form integrated partial product models, as described in section 2.2.3 and 2.4, and provide an intuitive working environment.

A feature based representation has therefore been chosen as a basis for further investigations in order to realise the concept of an integrated partial product model.

6.3 Industrial Collaboration

Industrial collaboration has been necessary in order to conduct the applied research reported in the thesis. The collaboration has been important to the success of the investigations because it has focused on user defined functionality and the advanced nature of the intended implementation. Both end users and software developers have been helpful in a number of areas such as during problem definition, user requirements specification, prototype assessment, implementation, testing and evaluation.

6.3.1 Collaboration With Users

The companies that have collaborated specifically in the investigations as users of design and manufacturing software tools, produce a variety of differing products and have differing needs. The varied products, product information and needs provide a basis for the general applicability of the work. The companies that have specifically collaborated as end users are:

- GEC Marconi[GEC] British defence electronics designer and manufacturer. Metal housings for electronic circuits are manufactured to provide radio frequency screening.

- Mandelli[MAN], Italian machine tool designer and manufacturer. Utilise heavy steel castings into which predominantly 2½D features are machined.

- Rover Group[ROV], British motor car designer and manufacturer. Concerned with design of sculptured surfaces and the design and manufacture of sheet metal press tooling and plastic injection moulds.

- Cadam Model[CAD], Italian plastic injection mould tool designer and manufacturer.

Collaboration with users is helpful for three reasons:

1. Users provide component geometry and function focused information and needs.

2. Users provide design and manufacturing process focused know-how and needs.

3. Users test and evaluate the results of implementation work.

In a user defined feature modelling environment end users provide information on feature shapes, feature functions, relationships between features, feature constraints and associated manufacturing data, particularly tolerances. End users provide useful knowledge of the design and manufacturing process and current shortcomings, especially lack of systems integration as stated in section 1.4. This information is vital input to the analysis phase leading to software architecture definition as described in sections 4.2, 4.6.1, 4.7.1, 4.8.1, 5.2, and 6.3. A part of the process of architectural definition is that of prototyping ideas. User evaluation of the software prototypes has played an important role in the formulation of the chosen software architecture as described in section 6.4. Final end user evaluation of the software implementation has provided feedback with respect to both functionality and, "look and feel". The examples presented in the thesis (Figures 1.2, 1.3(a), 2.7, 7.4, 7.6, 7.9, 7.10) have been supplied by end users.

Numerous other companies have also contributed indirectly to the investigations as end users through their input at software user group meetings. Many more needs and

examples from these users have been assessed as part of these investigations. These have been supplied via software engineers as reported in the next section (section 6.3.2).

6.3.2 Collaboration With Software Engineers

Software engineers from Strässle Informationssysteme AG [SIS] have collaborated in the research.

Collaboration with software engineers is helpful for three reasons:

1. Product support and marketing provide contact with a wider group of customers increasing the diversity of information used in the investigations.

2. Complex and advanced graphically oriented software systems require large quantities of code to be written for results visualisation. Software engineers provide a programming resource for graphics implementation.

3. Software engineers provide evaluation of software quality in terms of the architectural stability and scalability of the experimental implementation.

User groups highlight specific needs for shape creation, manipulation and the quality of the graphical user interface. Complex software systems, such as feature modellers, necessitate considerable programming resources in order to achieve sophisticated results, especially for graphical interaction. Software developers have provided programming support to implement system designs in much the same way that technicians construct a machine to a given design in order that experiments can be performed. Software engineers have undertaken assessment of the results of the work by modelling customers' components.

The examples presented in the thesis (Figures 1.3(b), 1.5, 1.7, 1.8, 2.4, 2.7, 2.10, 3.1, 5.32, 7.11, 7.14) have been supplied by end users through user group meetings and product support.

6.4 Research Techniques

In this section the methods used to acquire and analyse information will be detailed. There are many sources of information which have enabled the new concepts reported in this thesis to be identified and investigated. The sources range from academic research teams, material reported in the standards and literature, other software engineers and the very significant group of end users for whom the results of the investigations are intended to benefit. Analysis of the information acquired led to the design of the software system architecture and implementation that has been reported in chapters 4 and 5.

6.4.1 Acquiring Information

6.4.1.1 Literature and Standards

The literature and applicable standards have been reviewed in chapter 2. The literature provides information about existing techniques, the approaches other investigators in the area have taken, and reports their achievements. This review allows the current state of knowledge to be established and enables the areas of further investigation to be determined (section 6.4.2.1) from the shortcomings identified. The standards are important in this research because they embody the currently approved and used methods for describing products in a format that can be exchanged and is understood by all users. The purpose of the research is to investigate methods of combining the information presented by a number of standards into a single integrated model suitable for the interactive creation and editing of designs. Investigation of the direction in which the standards are progressing is important because the model must not only be capable of supporting current requirements but also of being flexible in order that future needs may be addressed, particularly that of the emerging vectorial tolerancing standard (section 2.5).

6.4.1.2 Research Meetings

Meetings with other researchers are important because they allow more in depth investigation of the work reported in the literature. Approaches and their limitations can be better established. The meetings also allow the work reported in the thesis to be presented and be subject to peer group review. The published papers reporting the work of this thesis that have been peer group reviewed are listed in Appendix A. Research meetings have been of the following types:

- Conferences – in both Europe and the USA where world wide feature research is presented.
- Workshops – by invitation, to enable less formal and more investigative presentations and discussions to be held.
- Visits – to major research groups in Europe and the USA [HUS91b] provide the most detailed review of progress and trends. Teams at the following research institutions have been visited: [MPG][IFW][WMG][IPK][FHI][ETH][LKC] [USC].

6.4.1.3 Collaborating companies

The companies that have directly collaborated in the investigations have been identified in section 6.3.1. They have provided a considerable quantity of information relating to the following topics:

- Shape
- Shapes function
- Associated information and use
- Current representations

- System Requirements

This information has been presented in a number of reports:

1. The shape, shape function and associated information of hole, slot and pocket features machined into billets by Marconi are reported in [MIL91].

2. Mandelli provided information on shape, shape function, associated information, and particularly related tolerance data that has been reported in [DFL92] and [FTR95]. Mandelli provided information that enabled the user requirements documents to be prepared for both the man machine interface [MMI93] and the Feature Based Design System [FBDS93]. Requirements documents have been prepared following the guidelines of the European Space Agency Software Engineering Standards (PSS-05) [ESA]

3. CADAM Model have provided information on shape and shape function which has been reported in [DSF95] and forms the user requirements for solids based sculptured surface generation.

4. Rover has provided information on shape and shape function that has been reported in [DSF95] and forms the user requirements for surface based sculptured surface generation.

Indeed many design and manufacturing processes investigated in the collaborating companies already show feature characteristics even though the companies are unaware of this. This lends support to the claim that design by features is an intuitive method of construction.

6.4.1.4 Software engineers

Software engineers have provided information as part of the investigations because they have been exposed to the needs of a wider group of users than the specific collaborating companies. The information provided by software engineers was more

generalised than the specific requirements of the collaborating companies and concerned the following areas:

1. Shape creation – more flexible ways to create sculptured shapes was required as described in [TAN95] and section 4.6.1.

2. Look and Feel – concern was voiced about the dominance of the property sheet in a graphically interactive system as the main method for user input. This manifested itself as the requirement for "point and click" methods to change the model as stated in section 5.2.1.

3. Performance – users of large models consisting of hundreds of components suffered from the poor dynamic characteristics of Motif for the browsers. This produced the requirement to reduce to a minimum the number of objects necessary to be presented in browsers as stated in section 5.2.6.

6.4.2 Analysis Of Information

6.4.2.1 Areas Of Investigation

The review of the literature presented in chapter 2 identified feature based representations as the most promising approach for realising an integrated design system capable of supporting the needs of down stream manufacturing applications. Though feature modelling techniques have been proposed for over a decade further investigations are required to establish if there are deficiencies in the understanding of features which has resulted in little industrial impact of the technology to date and the continuing lack of design model and manufacturing application integration. None of the collaborating companies were users of feature technology.

The review of the literature reported in chapter 2 has highlighted a number of areas of feature modelling technology that would benefit from further investigation. In

section 2.4 it was reported that to date feature modellers have focused on modelling form and simply displaying the resultant solid geometry as a means of representation. Little use of free form geometry has been made. Section 2.5 reported on the modelling of dimensions and tolerances and experiments to associate dimensions and tolerances with feature models. Little graphical feature based support has been provided.

The particular characteristics of feature modelling techniques highlighted in chapter 2 that will be investigated in this thesis are shown in Figure 6.2.

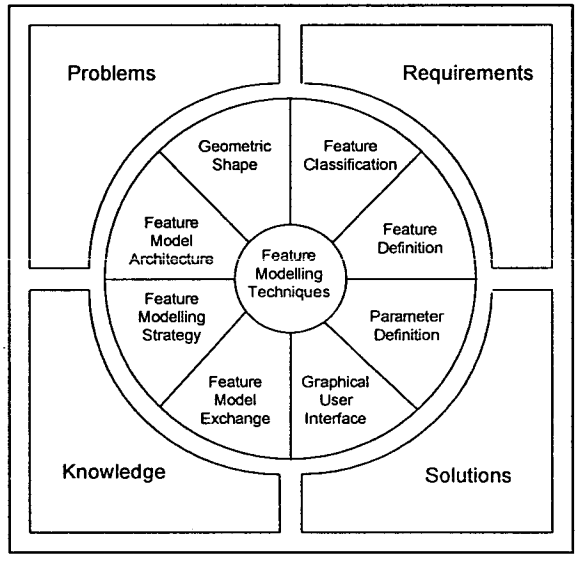


Figure 6.2 Areas Of Investigation

The areas identified in Figure 6.2 have formed the focus of investigations reported in this thesis for the following reasons:

1. Feature Classification

   Feature classification has been dominated by parallels drawn with shape classification in solid modelling. Little emphasis has been placed on user defined features and the functionality or meaning that can be represented. Investigations are needed to identify any commonalities between seemingly varied and functionally specific features that may help to improve built-in system support.

2. Geometric Shape

To date little emphasis has been placed on feature modelling with sculptured surfaces. Investigations are required to show how this may be achieved.

3. Feature Definition

It has been assumed that features "know" all about themselves. This is not necessarily the case when elements that form geometry cannot be pre-defined. Such is the case in modelling free form geometry when sweeping one object along another. The mechanism can be defined but not the result. Investigations are needed to show how such "unknown" objects can be represented and manipulated in a feature modelling system.

4. Feature Model Architectures

Feature models have predominantly consisted of single tree structures modelling a single component. Simultaneous engineering strategies necessitate that manufactured parts fit together in assemblies requiring multiple component models and the definition of dimensions and tolerances between them. Investigations are required to assess the feasibility of more complex feature model architectures and their uses.

5. Parameter Definition

The parameters that define a feature's shape are often represented by dimensions on a 2D drawing. There has been no attempt to define a graphical representation for the parameter that is consistent with 2D drawing standards and can also be used to change the model. Investigations are necessary to show how complex parameter descriptions can be achieved.

6. Modelling Strategy

Dimensions and tolerances are typically added to the finished model or drawing. Simultaneous engineering necessitates continuous review and modification of the design, which requires that dimensions and tolerances be added at any stage of the design process. Experiments are necessary to show that feature models can represent partial product models during an evolutionary design process.

7. Graphical User Interfaces

Graphical user interfaces to experimental feature modellers have been rudimentary compared to the "point and click" sophistication of parametric modellers. Experiments are needed to explore the potential to make feature definition languages more expressive in order to improve user interaction provided by the system.

8. Feature Model Exchange

Current standards are applicable only to shape and specifically exclude associated function and user defined features. There has been little attempt at the general transfer of user defined features which requires investigation.

This thesis proposes that by investigating the areas detailed (1 to 8), it will be shown that the fundamental concept of providing an integrated partial product model can be achieved by making specific advances to feature modelling techniques. Therefore, as described in section 1.5, the aim of the work to provide better computer aided support for design and manufacturing will have been addressed. The areas of investigation (1 to 8) have been used to form a set of identifiable objectives for software implementation and testing as stated in section 1.5. Hence, the industrial impact of feature modelling technologies will be increased, leading to better process integration, thereby shortening product development cycles, improving quality, lowering costs and consequently increasing competitiveness.

Figure 6.3 shows the correlation between the areas of investigation, the major sources of information provided, and the analysis of the information in the thesis. The analysis of the information has been presented in the thesis in an amalgamated format either as a statement of current practices or of user requirements.

Area of Investigation	Source Of Information					
	Literature & Standards	GEC Marconi	Mandelli	Rover	Cadam Model	Software Engineers
Dimensions & Tolerances	2.5, 2.6.4	5.2	5.2	5.2	5.2	5.4
Geometric Shape	2.3, 2.6.2		4.2	4.2, 4.6	4.2, 4.6	4.2, 4.6
Feature Classification	2.4.3	7.3	7.3	7.3	7.3	
Feature Definition	2.4.1, 2.4.3					4.3, 4.4
Feature Model Architectures	2.4.3					3.7
Parameter Definition	7.4					4.3
Modelling Strategy	2.4.3					3.7,
Graphical User Interfaces	3.5		4.7	4.7	4.7	4.7, 5.6
Feature Model Exchange	2.6.3					

Note: The numbers in the cells refer to the applicable sections of the thesis.

Figure 6.3 Sources Of Information Used And Impact On Analysis

6.4.2.2 Selection Of Feature Modelling Test Bed

Three major requirements were significant in the choice of feature modeller on which the investigations reported in this thesis would be conducted. Firstly, the diverse forms of feature identified by the end users indicated that the feature based system chosen must have the ability to define user defined features. Secondly, the feature modeller should have a true hybrid architecture (the feature model is the dominant software sub-system, rather than the geometric model) in order to explore representations of features other than form features. Thirdly, a commercially available feature modeller was desirable because of the industrially applied nature of

the investigations necessitating a highly developed user interface and an interactive form feature definition language.

Prior to 1996 the only commercially available feature modeller that could meet these requirements was produced by Strässle Informationssysteme AG [SIS] and was called FeatureM. Therefore, FeatureM was selected to be used as the test bed for the investigations. FeatureM has been described in chapter 3. The interactive form feature definition language as described in section 4.3 was used as the baseline for the investigations which led to the software advancements implemented as described in chapters 4 and 5.

Furthermore, Strässle Informationssysteme AG [SIS] became an industrial collaborator with the intention of both aiding and exploiting the research.

6.4.2.3 Prototyping

Prototyping has formed a significant part of the investigations in determining the chosen software architecture and implementation reported in chapters 4 and 5. The issues and concerns of both end users and software engineers, produced both as requirements and as feedback from prototyping and testing, have been described in the sections of the thesis denoted in Figure 6.3. Prototyping has formed the major analysis method used during the investigations in order to produce a feasible architectural concept capable of addressing users needs. The user requirements for the Man Machine Interface [MMI93] and the Feature Based Design System [FBDS93] were translated into corresponding system software requirements described in [PED93a] and [PED93b]. A number of experimental implementations were developed from these software requirements to test ideas corresponding to the major objectives of the work stated in section 1.5. These are described below:

6.4.2.3.1 Extrinsically Defined Features

An early attempt to provide a feature with form that could be fully determined only after instantiation was unsatisfactory in genuinely representing the requirements of Mandelli. Mandelli required a 2½D pocket feature that could have as many sides as designated by the designer: an "n" sided pocket [DFL92]. The prototype implementation declared a feature template with a large number (30) of possible positions for the intersection points of the sides, and radii parameters for the corresponding (30 possible) connecting corners. Mandelli agreed that 30 sides would be sufficient in normal circumstances. However, the implementation is not completely flexible and is very inefficient because of the amount of memory required. Property sheet masking was used to hide the irrelevant parameter slots. Such features are best represented by defining contours that are then swept to produce the required volume (section 4.6.3).

This was the first example of a feature where it was not desirable to have its form fully defined in its make methods. The need to support the geometry of castings within the integrated model provided the next example of an extrinsically defined feature. The requirements of Rover and Cadam Model to represent sculptured geometry provided further examples and highlighted the importance of recognising the differences between what is known about a feature and what is not (section 4.2). These differences led to the two different approaches described in chapter 4 being investigated for intrinsically and extrinsically defined features.

6.4.2.3.2 Handles

The demands on the complexity of handles has evolved as form features have become ever more elaborate. There have been five stages of evolution as shown in Figure 6.4.

Technique	Disadvantages	Advantages	Applied To
Single handle at feature coordinate system origin	• Inflexible • Does not aid designer	• Very simple to implement • Applicable to all form features	
Multiple named handles described by a single parameter of the form: oben, unten, links	• Language specific • Ambiguous interpretation • Not suited to complex 3D form features • Difficult to provide built-in support	• Locally understood • Related directly to function • Simple to implement	
Multiple handles described by two numeric parameters forming a 3D grid of the form: handle (0-9), level (0-2)	• Not suited to complex 2½D form features • More than one parameter	• Not language specific • Easier to provide built-in support	
Multiple handles described by a single parameter based on references to the coordinate system origin of the form: x, -xz, xyz	• Reliant on understanding of feature coordinate system • Not suitable for extrinsically defined form features	• Not language specific • Built-in support may be automated • Suited to complex 2½D form features	Intrinsically defined form features
Single coordinate system origin which may be repositioned and oriented	• Difficult to implement • Slow response	• Suited to complex 3D form features	Extrinsically defined form features

Figure 6.4 Handle Techniques Investigated

Feedback from end users and software engineers about the implemented approaches has led to the logical development of the techniques. However, even the most advanced technique (multiple handles referenced to the feature coordinate system) applied to intrinsically defined form features suffers from the fact that the user must always understand the position of the feature coordinate system origin. This need led to the requirement of graphical indicators for the handles that could be picked from the screen (section 4.7.1). This was the motivator for the extensions to the feature definition language reported in section 4.8.

The establishment of extrinsically defined form features immediately warranted the investigation of a new approach (section 4.7.3). A new approach was required because the procedural methods described in section 4.7.2 could not be applied due to the fact that the shape of the feature and hence location of the origins are unknown.

6.4.2.3.3 Dimensions

A major prototype was built to establish the feasibility of using the feature model architecture as a starting point for investigating representation of dimensions and tolerances.

The prototype generated linear dimensions between the coordinate system origins of features. Simple wire bodies formed the graphical representations of the dimensions. The prototype established that:

1. The information content of a dimension could be supported within a feature template.

2. The information describing the graphical representation of a dimension could be supported within a feature template.

3. The procedural nature of the feature modeller allowing features to have a symbolic representation meant that the graphical representation was not combined with the geometric model, but visualised with it.

4. It was possible to save and restore models containing both geometric and dimension features.

5. The dimensions were associative with the positions of the features. The dimensions reflected changes to the positions of the features.

6. The mechanism of the feature modelling kernel allowed one feature's methods to set the value of the parameters of another, enabling the dimensions to be used to control the model.

7. Maintaining the dimensions in the same workpiece structure as the form feature was unsatisfactory.

numbering the faces during construction of the feature. However, experiments showed that the numbering scheme was not stable when features of variable topology ("n" sided pocket) were used [PED94c], or the geometry was built in a different way (in a different system for example). Thus, the method of using ray point and ray vector was established [PED 94d].

6.5 Software Testing

Software testing has two significant purposes. Firstly, it is used to show conformance of the software implementation to the specified requirements. Secondly, it forms a part of the investigative process by which continuous assessment of the implemented system can be reviewed to provide feedback for improvement.

6.5.1 Informal Testing

Informal review of the architectural concepts and testing of the implemented software was performed by other software engineers. The proposed architectural solutions and experimental implementations were reviewed from the practical perspectives of:

- compatibility with existing software sub-systems
- module integration
- reliability
- model stability (saving/loading)
- performance (speed of response)

The results of these informal tests, which were reported verbally, have been incorporated in sections 4.6.2, 5.4 and 5.5.1.

numbering the faces during construction of the feature. However, experiments showed that the numbering scheme was not stable when features of variable topology ("n" sided pocket) were used [PED94c], or the geometry was built in a different way (in a different system for example). Thus, the method of using ray point and ray vector was established [PED 94d].

6.5 Software Testing

Software testing has two significant purposes. Firstly, it is used to show conformance of the software implementation to the specified requirements. Secondly, it forms a part of the investigative process by which continuous assessment of the implemented system can be reviewed to provide feedback for improvement.

6.5.1 Informal Testing

Informal review of the architectural concepts and testing of the implemented software was performed by other software engineers. The proposed architectural solutions and experimental implementations were reviewed from the practical perspectives of:

- compatibility with existing software sub-systems
- module integration
- reliability
- model stability (saving/loading)
- performance (speed of response)

The results of these informal tests, which were reported verbally, have been incorporated in sections 4.6.2, 5.4 and 5.5.1.

Further informal testing was performed by demonstrating the implemented software solutions to the marketing department of the collaborator, Strässle, and to other academic research groups. The feedback from the marketing specialists focused on the look and feel of the implemented software, particularly concerning:

- The number of input steps to achieve object creation.
- Clarity of understanding during creation and editing of the model.
- The potential for automatisation.

This feedback has been incorporated in the requirements specified in sections 5.6.1 and 5.6.2.

The feedback from other academic research groups has concerned the novelty of the concepts and the fact that experiments with the 3D parallel tolerance show it to act as a 3D constraint in the model. The academic feedback was discussed in section 7.5.

6.5.2 Formal Testing

Formal assessment of the implemented software has been performed in a number of ways and has provided feedback throughout the investigations. The methods used were product comparison, collaborator testing, and product assessment.

6.5.2.1 Product Comparison

In [SIO96] the enhanced feature modeller, as described in chapters 4 and 5, was compared with CADDS 5. Whilst not being so flexible in geometric shape creation, the resulting system was more intuitive, allowing easier model construction and editing.

6.5.2.2 Collaborator Testing

6.5.2.2.1 Mandelli Evaluation Report

In [FUR95], Mandelli report on their experiences with the enhanced system. Mandelli used the system to model a large casting (Carro W2200). The casting had several very complex pocketed areas machined into it. One of these areas has been modelled independently of the complete casting, and is shown in Figure 7.4 without dimensions and tolerances for clarity.

In the report [FUR95] no errors in the system compared to the requirements [MMI93] and [FBDS93] were found. A number of enhancements were recommended to the user interface of varying severity.

The highest priority was the need to provide visual representation of all hole end conditions [DFL92]. Nine were specified of which four had exact geometric representations. Mandelli regarded this as important because of the design functionality associated with end condition and the absolute need for a hole to be manufactured to design specification. All end conditions were supported by name as a text parameter but only those with flat or spherical ends, 90° or 120° tip angles were represented geometrically in the model.

Of medium priority was the desire to ease the positioning of features by combining the movement of the local coordinate system and the feature coordinate system.

The above concerns were addressed by further programming of the existing architectural design.

Of major significance was concern about the relationship between parameters in the property sheet and the physical dimensions of the model. This relates to size and handle parameters. The reason for the problem is that it is not obvious which handles exist, what they are called, and which is where on a rotated feature. A similar

problem exists relating all size parameters to the physical dimensions of the model following rotation. These concerns have formed part of the requirements stated in sections 4.7.1 and 5.2.

6.5.2.2.2 SESAME Final Assessment

The enhanced system was presented at the final assessment meeting for the SESAME project [SES]. The CEC Liaison Officer and an independent technical consultant conducted assessment. The example shown in Figure 7.4 and a further example of a gearbox casing also prepared by Mandelli were presented. Although project aims had been accomplished both assessors supported the comments made by Mandelli regarding the difficulty in establishing the correct relationship between parameters and the physical model being worked on. This was especially highlighted for complex, real life models. These comments showed the importance of the investigations to address these problems presented in chapters 4 and 5.

In [FTR95] Mandelli estimated that 90% of the features machined into their components could be designed using the small library of features presented in Figure 7.3. Mandelli estimated that using the feature based design system would lead to a 60% reduction in input time for geometry creation. Of the remaining 10% of features necessary half were geometric primitives (wedge, torus, prismatic solid – Figure 7.12) which have since been implemented as intrinsic procedural features with parameter dimensions. The remainder are extrinsic features such as a pipe (Figure 2.18), solid of revolution (Figure 2.17) and sculptured surface (Figure 4.8). These needs reported in section 4.6.1 were implemented as extrinsic sweep features reported in section 4.6.2.

6.5.2.2.3 IMPRESS Midterm Assessment

The investigations into the representation of swept geometry and the software implementation (section 4.6) were presented at the midterm assessment meeting for the IMPRESS project [IMP]. The proposed solutions as described in section 4.6 were reviewed by the CEC Liaison Officer and several designers from Rover and Cadam Model. It was noted that the geometry creation possibilities were still limited compared to free form surface modellers. However the possibility to change interactively the parameters controlling the channel feature shown in Figure 6.9 was seen as very clear. It was recommended that the freedom to position such extrinsic objects be improved. This recommendation formed the requirement for providing a mechanism to support the concept of feature origins for extrinsically defined features as described in section 4.7.1.

6.5.2.3 Product Assessment

The Product Services Manager for FeatureM at Strässle has tested the implemented software with examples sent from current and prospective customers. The Product Services Manager has modelled the examples of the connecting plate (Figures 1.3(b), 1.5, 2.10, 3.1, 7.14), the rotor (Figures 1.8, 2.4, 5.32) and soap box (Figure 7.11).

Feedback from the product services manager concerned a number of areas:

1. Development of more sophisticated rubber bands displayed during feature and dimension creation.

2. Reduction in the number of parameters required to be defined during interactive definition of dimensions. This feedback was helpful in defining the necessary object decoding described in section 5.5.6 and the user interface for creation described in section 5.6.1. The effects were to assume logical values for the dimension plane and direction of the dimension line.

3. Concern about the performance of large models with many (thousands of) dimensions. This was another motivator for the investigation of dimensions representing the dimension parameters of features as stated in section 5.2.6.

6.6 Contribution To Knowledge

This thesis has reported on the investigation of a number of new concepts concerned with expanding knowledge of feature technology:

I. Analysis of feature classification, industrial needs and modelling practices has shown (section 4.2) that there are a number of form features which cannot have their shape pre-determined within the build methods of the feature. The original concept that certain features cannot be fully described by their build methods enables these features to be collectively grouped together as an important new class of feature called *extrinsic form features*. Investigations of methods to represent, and model with, *extrinsic form features* have shown that such features have the ability to support legacy data in the form of pure solid models (section 4.5), and are particularly suited to representing swept and limited forms of free form geometry (section 4.6). Furthermore, form features have been shown to fall into one of three sub-groups (section 7.3): primitive, standard, specialised.

II. Analysis of dimension and tolerance representation techniques has shown that the dimension and tolerance model must be wholly integrated within a feature modelling architecture in order to provide the rich data models necessary for the full application of simultaneous engineering strategies. Therefore the scope and architectural complexity of feature modelling techniques have been investigated leading to the new concept of *multi-dimensional* or *hyper* feature models (section 5.4). A *hyper* feature model consists of a number of feature trees. Each tree contains only one major class of feature: form, dimensions, tolerances, etc. The trees are related by links from parameters in one feature

tree to features or resulting geometry in another. Dimensions and tolerances within a *hyper* feature model have been shown to act as 3D constraints governing the manner in which a form feature model may be subsequently manipulated (section 7.5).

III. Features and their parameters have been analysed leading to the concept that feature parameters, like features themselves, have characteristics that allow them to be grouped together in classes because they behave in a similar manner and can be treated in the same way by the kernel (section 4.8.1). For example, length, width, depth and height parameters are all of class *length*. A novel combination of procedural and declarative approaches applied to user defined feature definition languages has been investigated (section 4.8.2). This concept enables more intelligent mechanisms to be built as a part of the feature modeller kernel and GUI. These new mechanisms are able to understand the meaning or function of the parameters that a more sophisticated and extended feature definition language is able to describe. These experiments show that feature definition languages rather than their methods can be used to control how the user is to interact with a feature and that feature modellers can be built which automatically provide superior GUI functionality. A typical example is the representation of "handles" on intrinsic form features (section 4.7.2).

IV. Investigation of the methods used to manipulate a feature model has shown that interaction has been predominantly keyboard based via tabular representation of feature attributes and parameter values. The majority of these values describe the parameters of a feature that are typically represented by dimensions on an engineering drawing. By focusing at the feature level and combining the ideas presented in II and III into a new concept for a feature definition language, it is possible to describe both the dimensions and tolerances of the feature in a 3D manner in the feature template definition (section 5.5.6.4). This new and sophisticated concept for feature definition enables:

- Features to be instanced fully dimensioned.

- The dimensions to be used for changing the model because they represent the parameters and are therefore called *parameter dimensions*.

- The dimensions can be laid out so that an engineering view of the 3D model appears as a 2D drawing removing the need for a separate 2D representation.

- The use of *parameter dimensions* virtually eliminates the need for the tabular property sheet providing a more intuitive user interface and increasing available screen space for modelling.

V. Further analysis of extrinsically defined form features has shown that the idea of "handles" for ease of feature instantiation and manipulation is highly desirable but cannot be achieved using the concept proposed in III because of the extrinsic and therefore unknown nature of the feature's form (section 4.7.1). Experiments have been undertaken with a new concept utilising the body coordinate system of the objects that is applicable to all extrinsically defined features and which enables the same functionality to be provided as with intrinsic form features (section 4.7.3).

VI. Current standards for the exchange of product data (STEP) use a number of separate models for 2D representations, 3D geometric models, dimension and tolerance models, and feature models, with no clear indication of how each model can be related. Furthermore, the feature model explicitly excludes user defined features, which are seen as essential for effective product modelling. Methods for the exchange of user defined feature models have been investigated (section 7.6). A new concept has been proposed which would integrate within the feature modelling architecture the information necessary for 3D geometric, and, dimension and tolerance representations, which would also convey any information traditionally supported by 2D drawings (section 7.6). By reducing the flexibility of systems to change subsequently a non-

native feature model, much greater information content can be transferred. This is an acceptable compromise in situations where downstream applications are better supported by rich product data that enables more informed decision making automation to proceed.

6.7 Chapter Summary

A number of important aspects of the research have been explicitly described in this chapter (chapter 6). Chapter 6 has described the fundamental concept proposed by the thesis from which the aims and objectives of the investigations have been derived. The necessity for industrial collaboration to the investigation of the concepts presented in the thesis has been stated. How the collaboration impacted on the proposed software architecture through the information provided, analysis and feedback from testing has also been reported. The research techniques used to acquire and analyse information leading to definition of the software architecture and implementation described in chapters 4 and 5 have been presented. The software testing methodologies employed and the results of tests have been detailed. In chapter 6 the relationships between the areas of investigation, the sources of information, and the sections of the thesis analysing the information have been presented. The original contributions to knowledge made by the investigations have been stated.

In the next chapter (chapter 7) the software implementation and results of the investigations reported in chapters 4 and 5 will be discussed. The applicability and suitability of the approaches will be examined and compared to those of other researchers. Communication of the partial product models developed with other applications will be assessed because of the importance of data exchange to concurrent engineering strategies.

Chapter 7

Discussion

7.1 Introduction

7.1.1 Chapter Overview

The previous chapters of this thesis have presented tools, technologies and developments to feature modelling techniques that have been applied to address the deficiencies of current feature-based design systems defined in chapter 1. In this chapter, the advances made to feature modelling techniques described in chapters 4 and 5 will be discussed with respect to the objectives of improving CAD and CAM process integration, their limitations and other approaches taken.

In chapter 2 the basic technologies relevant to the work of this thesis were presented: the design and manufacturing process, geometric modelling, feature modelling, dimension and tolerance practices, and the international standards applicable to these areas. Chapter 3 described a hybrid feature-based solid modelling system that was used as the baseline for the developments advancing feature modelling technology presented in chapters 4 and 5. Chapter 4 explained how the methods of User Defined Feature definition were extended to support extrinsically defined form, interactive origin selection and parameter validation. Chapter 5 explained the techniques that have been developed for dimensioning and tolerancing of solid and feature models.

These techniques include further enhancement to the User Defined Feature definition method that enables dimensions to be associated with the size parameters of features, which consequently can be used to change the model. In addition, dimensions have also been developed that represent sketcher constraints; these may also be used to change the model. Chapter 6 described a number of important aspects of the work concerning the original concepts investigated, industrial collaboration, analysis and testing. In addition the relationships between the areas of investigation, the sources of information provided and the sections of the thesis analysing the information were presented.

In this chapter (chapter 7) the contribution of the work reported in this thesis to the advancement of feature modelling technology will be stated. The feature modelling system presented in chapter 3, the advances reported in chapter 4, the dimension and tolerance implementation described in chapter 5, and the results of tests reported in chapter 6 will be discussed from the following perspectives:

1. Classification.
2. Modelling architecture.
3. Representation.
4. Exchange of feature models.
5. Feature technology as a process integrator.
6. Applicability to other feature modelling systems.

7.1.2 Chapter Structure

Section 7.1 forms this introduction to the discussion by giving an overview of the contents of previous chapters, and presents the topics and structure of this chapter. Section 7.2 will state the contribution of the work reported in this thesis to the

knowledge and advancement of feature modelling technology. Section 7.3 will discuss feature classification, considering both form and accuracy features. In section 7.4 the architectural issues of the feature modelling techniques used will be contrasted with other approaches. In section 7.5 the representation techniques and functionality developed will be considered. This section will include consideration of the wide applicability of the dimension and tolerance system, and particularly the description of dimensions in the form feature template definition. Section 7.6 investigates possibilities for exchange of User Defined Feature Models. Section 7.7 considers the claim that feature modelling enhances process integration. The applicability of the techniques described in this thesis to other modelling systems will be discussed in section 7.8. A chapter summary is provided in section 7.9.

7.2 Contribution

This thesis has reported on the investigation of a number of new concepts concerned with expanding knowledge of feature technology:

I. The original concept that certain features cannot be completely described by their make methods has led to the collective grouping of these features and definition of an important new class of features called *extrinsic form features* (section 4.2). Investigations of methods to represent, and model with, *extrinsic form features* have shown that such features have the ability to support legacy data in the form of pure solid models (section 4.5), and are particularly suited to representing free form geometry (section 4.6). Furthermore, form features have been shown to fall into one of three sub-groups (section 7.3): primitive, standard, specialised.

II. The new concept of *multi-dimensional* or *hyper* feature models has been proposed in order to increase the scope and architectural complexity of feature model data structures enabling a wider group of feature classes to be represented including form, dimension, and tolerance features (section 5.4). Dimensions and tolerances within a *hyper* feature model have been shown to

act as 3D constraints governing the manner in which a form feature model may be subsequently manipulated (section 7.5).

III. The original concept that feature parameters, like features themselves, have characteristics that allow them to be grouped together in classes (section 4.8.1) has enabled a novel combination of procedural and declarative approaches to be applied to user defined feature definition languages, which allows more intelligent mechanisms to be built as a part of the feature modeller kernel and graphical user interface (section 4.8.2). Investigations have shown that such feature definition languages rather than their methods can be used to control how the user is to interact with a feature and that feature modellers can be built which automatically provide superior graphical user interface functionality. A typical example is the representation of "handles" on intrinsic form features (section 4.7.2).

IV. The new concept that it is possible to describe both the dimensions and tolerances of a feature in a 3D manner in the template definition by combining the ideas presented in concepts II and III into a new, more sophisticated feature definition language (section 5.5.6.4). Investigations of the new language have shown that:

- Features can be instanced fully dimensioned.
- Dimensions can be used for changing the model because they represent the feature parameters and are therefore called *parameter dimensions*.
- Dimensions can be laid out so that an engineering view of the 3D model appears as a 2D drawing removing the need for a separate 2D representation.
- Use of *parameter dimensions* virtually eliminates the need for the tabular property sheet providing a clearer, more intuitive user interface.

V. A new concept applicable to all extrinsically defined features has been proposed to provide "handle" like functionality which utilises the body coordinate system of the objects that form the unknown aspect of the features (section 4.7.1). Investigations have shown that the same functionality can be provided for extrinsic form features (section 4.7.3) as that described for intrinsic form features in III.

VI. A new concept has been proposed for the exchange of feature models that contain the information necessary for 3D geometric, dimension and tolerance representations, which would also convey any information traditionally supported by 2D drawings (section 7.6). By reducing the flexibility of systems to subsequently change a non-native feature model, much greater information content can be transferred. This is an acceptable compromise in situations where downstream applications are better supported by rich product data that enables more informed decision making automation to proceed.

Implementation of the previously stated concepts (I to VI), which has been reported in chapters 4,5 and 6 and is discussed in this chapter, contributes to the advancement of feature modelling technology in the following areas:

A. Developing methods to model extrinsically defined form features.

B. Extending and applying a feature modelling system architecture to support the representation of dimensions and tolerances associative with both solid and form feature models to create hyper feature models.

C. Extending procedural methods for user defined form feature definition to include advanced GUI support and the graphical description of dimensions representing the size parameters of form features.

D. Developing techniques to provide GUI support for extrinsically defined form features.

E. Techniques for the exchange of user defined feature models.

7.3 Classification

In response to the question of: "What is a feature?", there have probably been more definitions proposed than systems developed. The author would like to suggest the following, perhaps less abstract, description developed within the context of the reported work:

> "A feature is an information unit, geometric or otherwise, that is used to define, or help the understanding of, a component model."

In this context features may produce geometry, defining the nominal form of a component, when designing for instance. Features may consist of (but do not produce) geometric elements providing further design definition, when adding dimensions and tolerances for example. Features may also provide another view of a component, as is the case with a manufacturing model. Features may contain information that is associated with a component but not directly related to the shape of the geometry, the material definition and other, particularly administrative, data.

In chapter 2 features were shown to be environment specific. In the context of design, a feature is called a design feature, in the context of process planning it is called a manufacturing feature, and in NC part programming it is called a machining feature. Even within a single view of a component (such as design) with the geometry accurately specified, it is necessary to indicate that areas of the geometry are of special significance. For example, in the design of car body parts it is important to know which parts of the geometry of a component will be visible in the end product and which parts will not. This has significant impact on manufacturing reasoning about where to position tool split surfaces, and machining strategy such as not dwelling on visible surfaces. For these reasons, features that contain information about the geometry that is created by other features is of importance in understanding the model of the component. Such features are highly environment specific.

Feature classification and development of feature taxonomies is an issue that has received great attention, much of it preceding the development of feature based

systems and now forming the basis of approaches to exchange of feature models. Thinking about features and the consequent development of system architectures has focused on the fact that a feature is presumed to have a known volumetric shape, which is described in the definition of the feature. This is not true of free form features (except if explicitly defined, but then they are of limited value). In fact, sweep features do not necessarily produce volumes. They can also produce surfaces which is a natural approach to integrating free form geometry into a solid feature based environment.

Under the auspices of the SESAME Project [SES] the author has developed a relatively small, focused taxonomy of design features [PED94], Figure 7.1

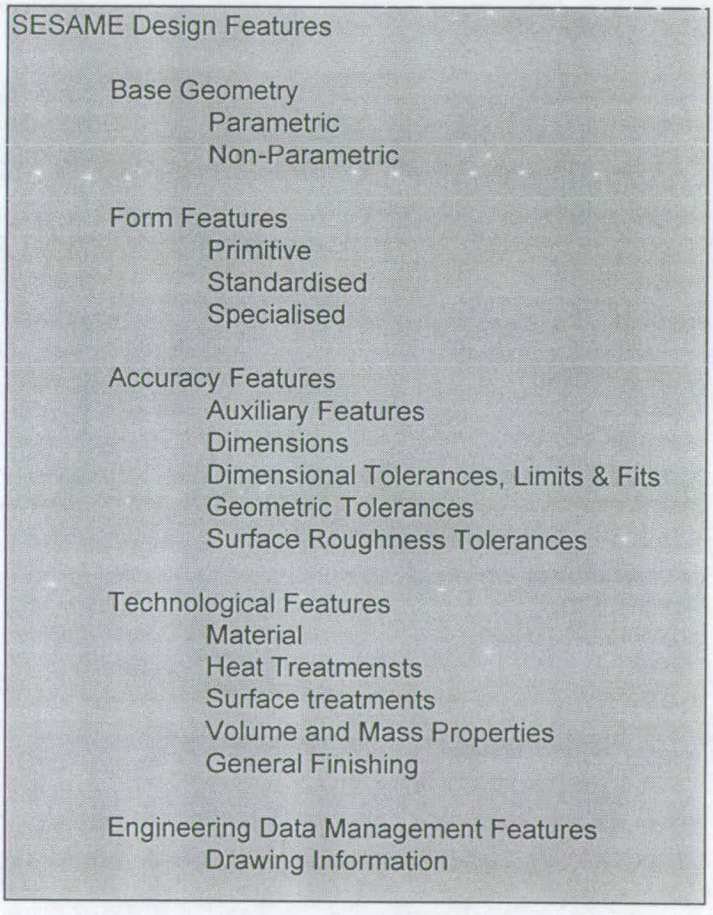


Figure 7.1 SESAME Design Feature Classification

The emphasis of the SESAME project was the design, process planning and NC part programming of parts consisting of 2½D form features machined on conventional

milling machining centres, as proposed by machine tool manufacturer, Mandelli Spa of Italy, a partner in SESAME. Base Geometry and Form Features taxonomies are presented in greater detail in Figure 7.2 and Figure 7.3 respectively.

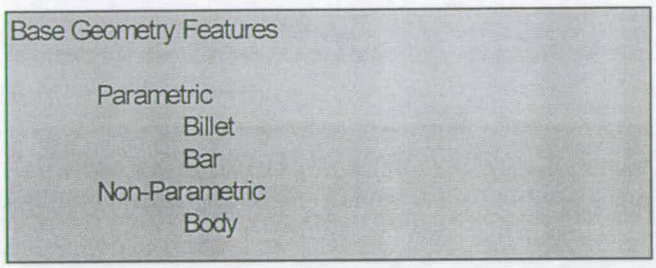


Figure 7.2 SESAME Base Geometry Feature Classification

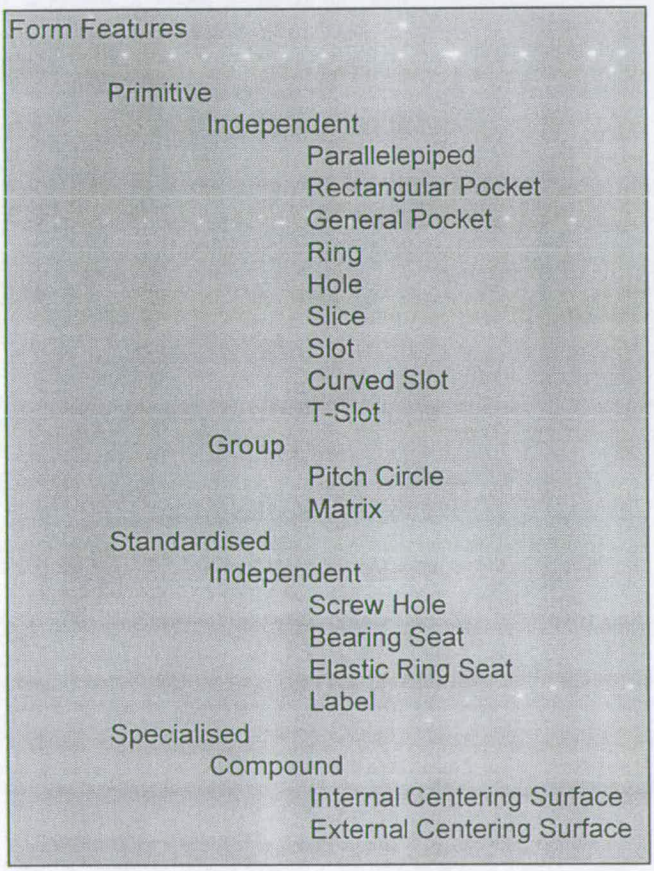


Figure 7.3 SESAME Form Feature Classification

The classification provides a rich set of objects which allow the designer to define and describe a component creating a partial product model suitable for driving downstream applications such as process planning. The feature modelling system

described in Chapters 3, 4 and 5 of this thesis has been used to model the design features detailed in the classification of Figures 7.1, 7.2 and 7.3. Figure 7.4 shows a part of a large casting which forms the table of a machine tool. The area modelled is a complex pocketed area where the guides and bearings are fitted. There are some forty features in the model.

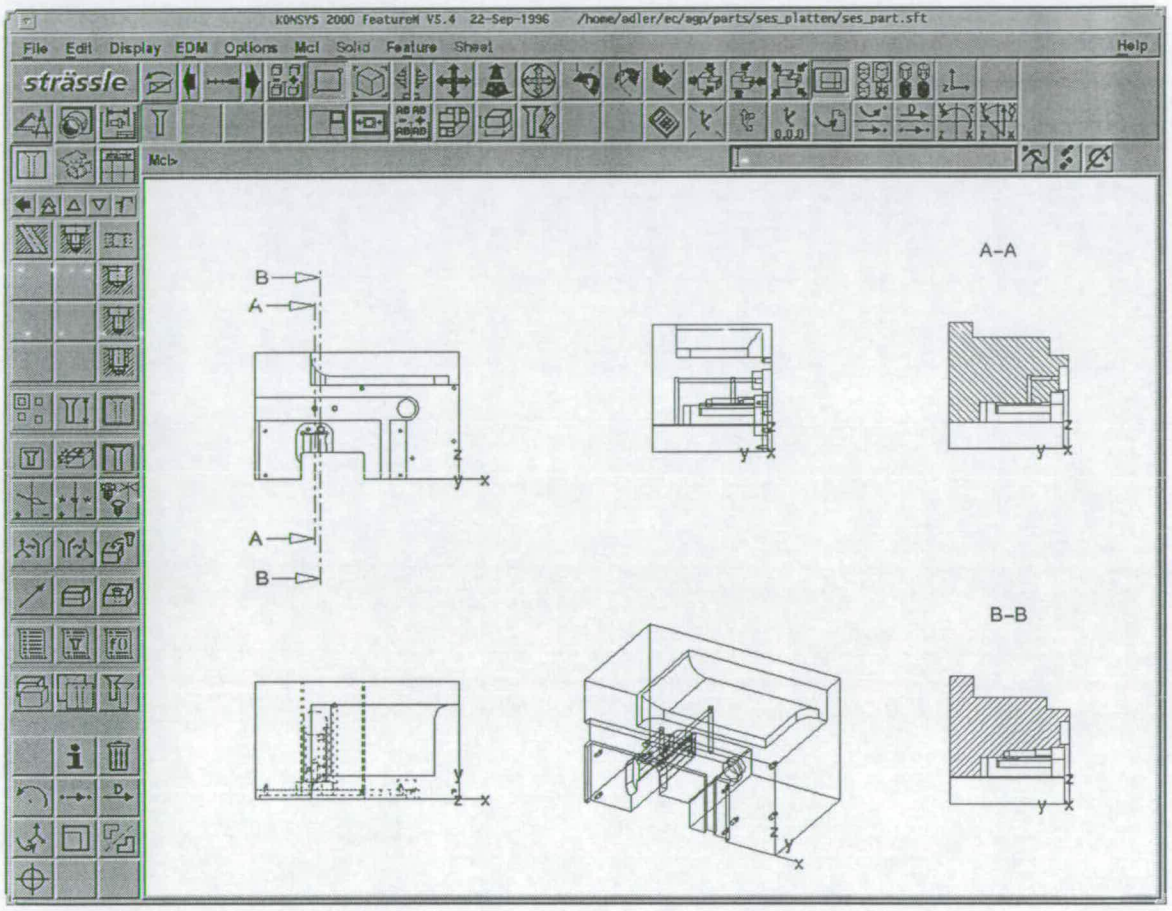


Figure 7.4 Model Of Machined Part Of Casting

The design features used to create solid geometry have been split into two categories: Base Geometry Features and Form Features. The SESAME Base Geometry Features have been presented in Figure 7.2. A distinction has been made between a Base Geometry Feature that forms the root of the feature tree which is positive (volume creating) and the Form Features which are negative (volume removing).

Only 2½D negative form features were provided in SESAME because the aim was to map the design features to manufacturing features to enable semi- and automatic

process planners to develop a manufacturing strategy for the design. 2½D negative features can generally be mapped to one or more manufacturing features. Positive and non 2½D features are problematic and will be discussed further in section 7.6. Base geometry features were either of a fully parametric implicit nature in the form of a billet or bar, or were extrinsic, non-parametric, solid features used to represent castings.

Castings are of major significance because they frequently form the stock part which is machined to realise the desired component. Castings typically contain free form surfaces which (due to the complexity both of design and manufacturing) were not the subject of feature development in the SESAME project. However, in SESAME the features that are machined out of castings are 2½D in nature.

The form feature classification presented in Figure 7.3 is divided in to three categories:

1. **Primitive**: Relatively simple features that would be provided by a system as a library of general features. These features would be widely applicable and not specifically related to a particular design environment.

2. **Standardised**: Features that are defined in ISO, national or company standards and are fully parameterised from tables depending on the selection of one (or several) parameters. For instance, a threaded hole for any given thread size, may have its length defined as short, normal, or long. Some features such as, circlip grooves, may be automatically positioned a certain distance from the end of the hole or shaft. These distances are defined in the tables of standards.

3. **Specialised**: Completely defined by the user, functionally very specific, complex, combinations of many features, with few variable parameters.

The classification of Figure 7.3 is a novel way of classifying features because previous approaches have only considered the shape of the feature, either the implicit

or explicit nature, or the geometric method of construction. This classification focuses on the semantics of the feature, the way in which a feature is to be used, and the representation methods (the way it is defined).

This classification clearly shows the need for User Defined Feature functionality to be present in systems because of the Specialised features. Such features are functionally very specific and are unlikely to be found, or used in exactly the same way in different companies.

The most promising area in which standardisation may be valid in all feature modelling applications is the use of features that represent specific items defined completely by international standards. The most common example is that of threaded objects, such as holes and shafts. User Defined functionality is also required despite the seemingly high degree of standardisation currently used. This is because companies will derive preferred sub-sets of the very broad international standards. Companies supplying stock items such as bearings, and many parts of injection mould tooling, not only require representations (actual or symbolic) of the objects but of the cavities that must be cut to locate them in the parts. There will never be a finite set of features, therefore User Defined functionality is essential.

General features such as hole, pocket, etc., would appear to not to require User Defined Functionality because of their simplicity and broad applicability. However, the naming of features should reflect the company and application specific usage, which requires User Defined functionality. The ability of a feature modelling system to offer User Defined functionality is essential if the system is to become more than a parametric volume modeller, playing a much more active role in the definition of product models which provide a high level view of the component demanded by manufacturing applications.

The modelling of free form geometry with features has received relatively little attention to date. The IMPRESS project [IMP] aims to investigate this by considering the design and manufacture of car body panels and plastic trim consisting predominantly of free form surfaces. Although some support for solid representation

is provided, the free form features are predominantly supported within a surface modelling environment. The taxonomy of form features classified by the author within the project is shown in Figure 7.5.

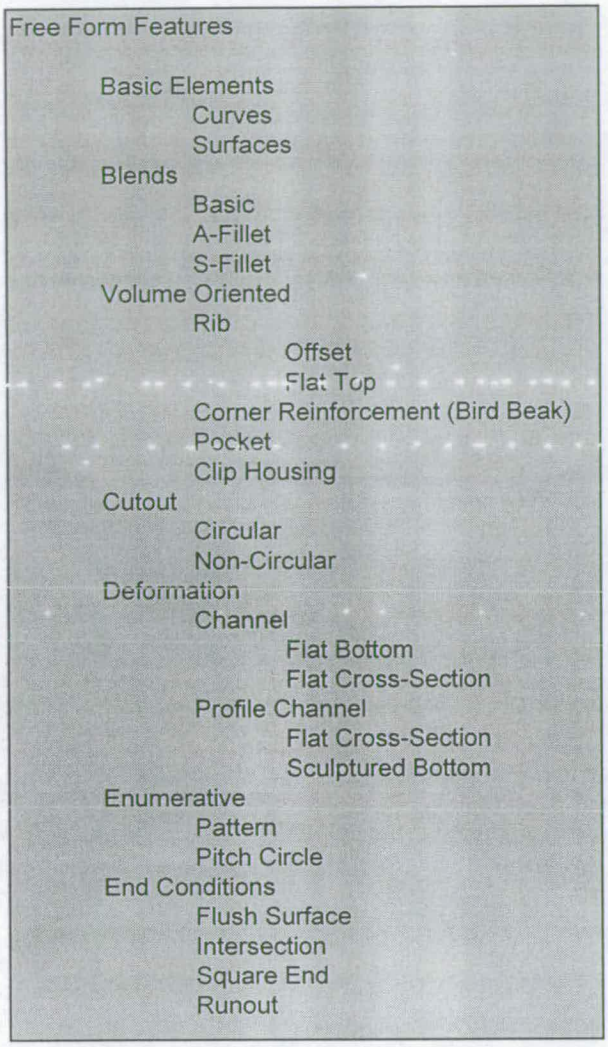


Figure 7.5 IMPRESS Free Form Feature Classification

An important aspect to consider is the naming of features because naturally this implies meaning. However, objects such as car body panels and plastic trim which contain the free form features are produced by forming processes using tools. The manufacture of the tool rather than the component itself becomes the focus of computer aided support. Tools have two halves, and what may be perceived as a channel by the designer of the product is transformed into a channel in one half of the tool and a protrusion in the other half, yet still based on the same design feature. The

usage of features with names such as channel also implies a sidedness to the design which poses problems for validation. It is interesting to note that designers classify features according to use rather than topologic or geometric content. The rib features and the channel features have exactly the same topology, geometry and parameters but different proportions. Importantly a channel is perceived as a depression, whereas a rib is perceived as a protrusion. A fact common to all of the features is that the resultant form of the feature cannot be completely described within the template definition. Certain aspects of the features may be more well defined than others, the cross-section of a flat bottom channel compared to a basic surface, for example, Figure 7.6.

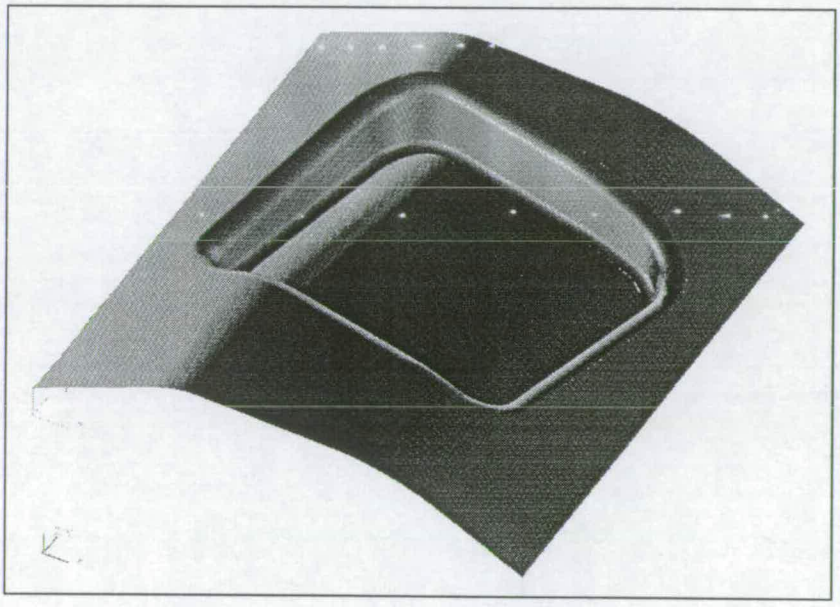


Figure 7.6 IMPRESS Flat Bottom Channel Feature

However the complete form of the feature is only known after the feature has been evaluated, because the form depends on the elements of the model with which it interacts, which are not implicitly (or explicitly) defined in the template definition. The path along which the cross-section is swept for example. Indeed, the usage of such free form features is much more complex than with the usual concept of form features because an important part of the feature definition is the way in which the feature interacts with the other surfaces it meets. These must be trimmed and blended

together in a defined manner. The concept of the complex feature for 2½D features is akin to such definitions.

The classification of Figure 7.1 which distinguished between those features which constitute the Base Geometry and the Form Features that are subtracted from this to realise a model in SESAME is not applicable to the classes of feature derived in IMPRESS. Free form models are typically built by adding surfaces to the model, and trimming or removing parts of others. There is no simple concept of additive and subtractive as with solid modelling.

The following more general form feature classification is proposed in Figure 7.7. It is logical not to classify separately the Base Geometry Features of Figure 7.2. They could be of any class in Figure 7.7. It is the semantics of the feature that indicate its suitability as base feature rather that its geometric shape. Extrinsic features, as classified in Figure 7.7, are typically primitive in nature due to their wide applicability. However, they can represent complex shapes and are certainly not simple to implement. The classes of general body and general sweep were described in chapter 4. Extrinsic features are unlikely to be formed into standardised classes as is the case with intrinsic, implicit features because of their unknown characteristics. Specialised features, and consequently user defined functionality, are likely to represent many free form features in an environment (company/product) specific basis. User defined support must therefore be provided for extrinsically defined form features.

Intrinsic features represent the bulk of features considered by developers of feature based systems to date, and are represented in primitive, specialised and standardised forms. Compound features are by their nature compounds of any other and so could either have intrinsic or extrinsic form. The general sweep implementation described in chapter 4 is completely extrinsic. This high level classification differs considerably from the STEP Form Feature Model [STEP48]. This is due to the rather narrow view of the STEP model which excludes User Defined Features, and concentrates purely

on the representation of form without considering the practical needs of manipulating features through a high level user interface.

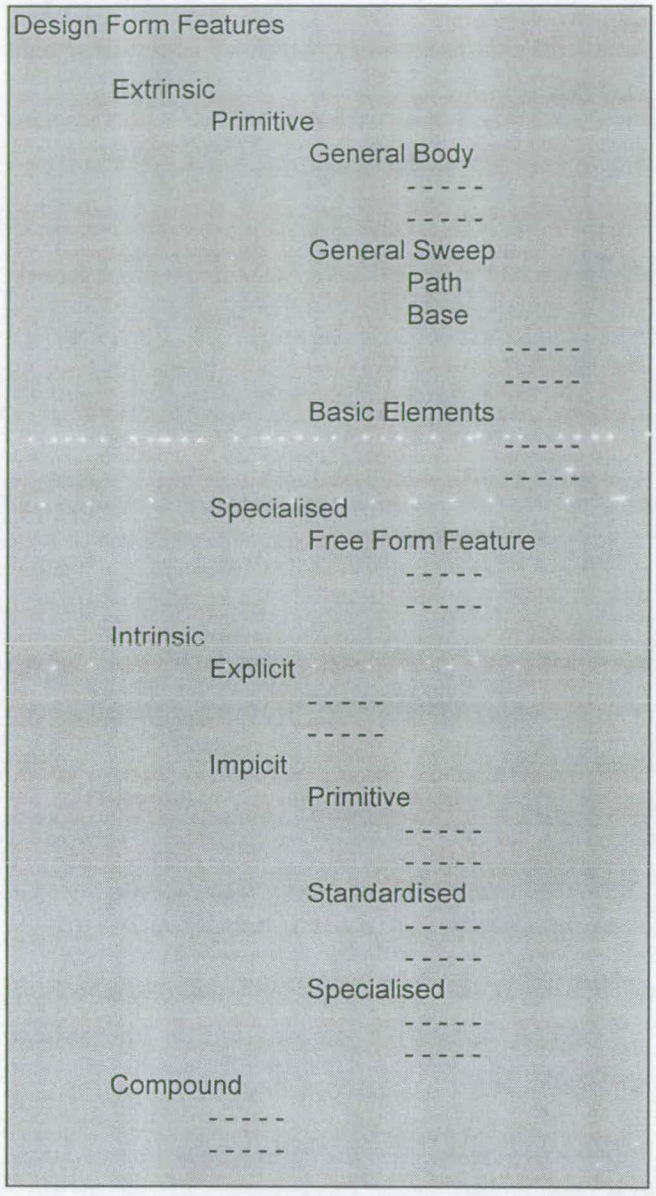


Figure 7.7 General Design Form Feature Classification

The accuracy feature classification developed in chapter 5, Figure 5.2, may be broken down into three levels, as summarised in Figure 7.8. There are two levels of classification, *Associativity* and *Function*, before the *Kind* of dimension or tolerance is detailed. This not the case with other classifications which start with the *Kind* of the dimension. This is because either the implementational details, such as object

associativity and modelling functions supported, are not considered as with the STEP Shape Variational Model, or there is no advanced graphical representation as with academic systems. These considerations greatly difference the actual implementation of the accuracy features as described in chapter 5.

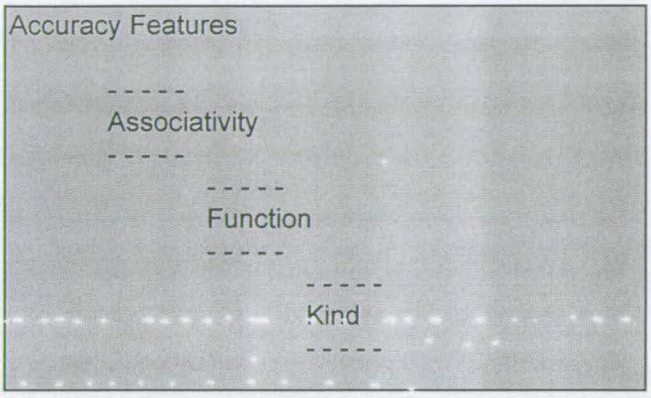


Figure 7.8 Levels of Accuracy Feature Classification

7.4 Model Architecture

In common with systems such as EMOS [AUR95] and SEW [MIL94], which have strong links to manufacturing applications, the system presented in chapters 3, uses a procedural architecture. There are certain advantages to using the hybrid CSG tree and evaluated BREP architectural approach, described in chapter 3, to purely parametric or variational BREP models, reported in chapter 2.

The hybrid models are more robust to topological changes caused by editing deep in the feature tree. This is because features are positioned relative to a reference coordinate system. When this reference is the workpiece or another feature it is completely invariant to topological changes. Only when specific features are positioned relative to topological (and geometric) entities can changes in object identity cause instabilities. The positioning and orienting of features is less constrained and can be subsequently changed at any time during construction of the model. The biggest problem with this approach is that of maintaining persistent object identity. Parametric and variational BREPs can actually maintain pointers

during addition and editing of the model. All pointers are lost during each evaluation of the model in the hybrid approach, therefore another mechanism is required to provide persistent object identity. This is a different function to that provided by all geometric modelling engines (ACIS included) which allows objects to be saved to file and pointers to be reset to their original elements after reloading. In this case the BREP model remains fixed and is simply rebuilt rather than evaluated from a sequence of operations. Providing robust persistent object identity is an area which requires further investigation.

Academic systems have predominantly focused on single workpiece models, particularly where close links to process planning and NC part programming applications have been the aim. Multiple workpiece models have only been considered for assembly planning and by commercial systems that offer assembly modelling capabilities. It is a novel approach to use the workpiece data structures as dynamic functioning structures supporting the multiple *types* and *kinds* of dimension and tolerances. This is particularly the case when the workpieces are assigned internally per form feature, to represent parameter dimensions, and per sketcher contour, to represent size constraints. The architecture is very robust because it can be applied to solid and feature models, represent size parameters of features and sketcher size constraints.

The method of defining features described in chapter 4, whether system delivered or user defined, is procedural. Currently there is interest in the development of declarative feature definition languages [BRU96]. Declarative languages appear attractive because of the potential development of graphical support tools that would aid the definition of user defined features in much the same way in 3D as a 2D sketch is developed using a constraint manager. There are currently great difficulties in developing general solvers for 3D objects consisting of constrained topologies. To date declarative languages have only been applied to relatively simple 2½D features such as slots. The advantage, and disadvantage, of procedural methods is that for effective user defined functionality an advanced interpreted language is required to drive both the feature modelling kernel and the geometric engine. It is believed that

the implementation described in chapters 3 and 4 that uses MCL+ does meet these needs. Certainly, the only restriction on the form or complexity of the feature developed, is in the bindings between MCL+ and the functionality provided by ACIS, and the ability of the user to understand and program the system. However, an excellent attribute of the declarative feature definition proposed by [BRU95] was the ability to define extrinsic constraints for a feature. That meant that if a slot was defined to contain a bottom, left and right side faces, then these, and these only, must be present in the faces generated by the feature on combination with the rest of the model. This means that the semantic meaning of the slot is maintained. This could be achieved by programming in the feature make method of the procedural system described in chapters 3 and 4 but is currently not done.

The use of both make and symbol procedures in the feature definition described in chapter 4, unusually, allows a feature to be assigned two representations. This is of practical value for form features when it is desirable that a symbolic rather that full representation is used to simplify a model. It is most convenient to represent the path used to generate a sweep feature symbolically because the path feature itself does not return the body that is combined with the model which is produced by the base feature. The use of the two procedures also allows one to control or update the parameter values in MCL+ whilst the other may use hard coded C++ procedures to drive graphics. This functionality was of particular use in development of the dimension and tolerance functionality.

The envisaged extension to the feature definition method to allow parameter constraints to be described, which has been implemented as property sheet functionality, is similar to several approaches based on the EXPRESS definition language [AUR95][KRA91][BRU96]. The advantage is that the procedural definition of features is simplified because the error handling must not be performed by the user in make method. Consistency is achieved because both system delivered and user defined features are provided with the same display and options to rectify. The extensions also facilitate advanced GUI support for feature modelling.

The implemented extension of the feature definition method to allow size parameters to be represented as dimensions in the display is a novel advance. Not only are the dimensions represented and can be use to change the model, but they may be laid out aesthetically in order to generate 2D engineering drawings with the aid of the technical views. To date graphical representations of feature definitions have concentrated on displaying the constraints applied in declarative methods as graphs [SOL94], or on representing the size constraints used by parametric or variational modellers such as ProEngineer [PTC] and IDEAS [SDRC], where features are defined as contours and swept. The approach developed in chapter 5 is applicable no matter how complex the feature so long as the user is able to procedurally define the parameters required to define the dimension.

7.5 Representation

The representation of features within the context of this work consists not only of defining the form but also of defining the dimensions and tolerances associated with the form.

The representation of features with intrinsic and implicit form enables dimensions and their associated tolerances to be described using the same feature definition method as described in chapters 4 and 5. Subsequently the dimensions can be used to change the model graphically without the need for use of a property sheet. For complex features this is an advantage because the user does not need to know which parameter controls which shape aspect of the feature. This can be tedious to check because the user must always snap the Local Coordinate System to the Feature Coordinate System.

The use of features with extrinsically defined form provides the modelling system and hence designer with much extended shape creating functionality. Such functionality is essential for products manufactured by forming processes. Such products frequently have high aesthetic content and are therefore in high demand, whilst also having high turnover to meet changing market needs. The ability to model

such objects in a feature based modelling system is of major significance to the applicability of the modeller. Features of extrinsic form, by their very definition, cannot have dimensions (and tolerances) described in the definition of the feature. The dimensions (and dimensional tolerances) must be supported in the objects that are use to create the body generated in the make method. This creates the added problem of maintaining consistency when the dimensioned objects are used to generate features which are controlled by the feature modeller.

The implicit nature of the form feature definitions allows highly developed GUIs to be implemented as described in chapter 4 in order to aid and control the positioning and orienting of features within the model. The philosophy of using a Local Coordinate System, being able to snap this to the feature coordinate system origin, or conversely, to snap a feature coordinate system origin to the Local Coordinate System, provides a comfortable method of changing the model through the GUI. The calculation of feature origins, whilst physically performed in the make method, requires definition of the available origin descriptors, in the feature template. The origin descriptors represented by positions calculated from the make method form logical attachment points for dimensions representing the size parameters of features.

Features with exstrinsically defined form, implemented as the general sweep features described in chapter 4, use the same Local Coordinate System philosophy to maintain GUI consistency. More specialised extrinsically defined form features, such as the flat bottom channel shown in Figure 7.9, may be partially defined using the parameter dimensioning technique described in chapter 5.

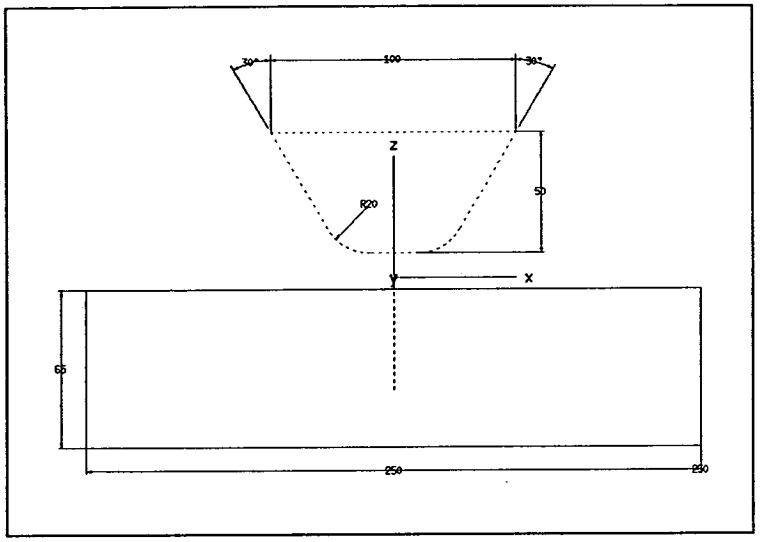


Figure 7.9 IMPRESS Flat Bottom Channel Feature With Parameter Dimensions

The representation of the feature differs between surface and solid environments. In a surface environment the feature is added to the model but in so doing part of the existing geometry is removed. In solid modelling the feature is either added or subtracted from the existing geometry. The solid environment is therefore much more suitable for creating the model of the press tooling to produce a sheet metal component with such features. In this case the cross-section of the feature is closed to produce a volume by sweeping, which is combined with the rest of the model, Figure 7.10, to produce either the male or female tool halves.

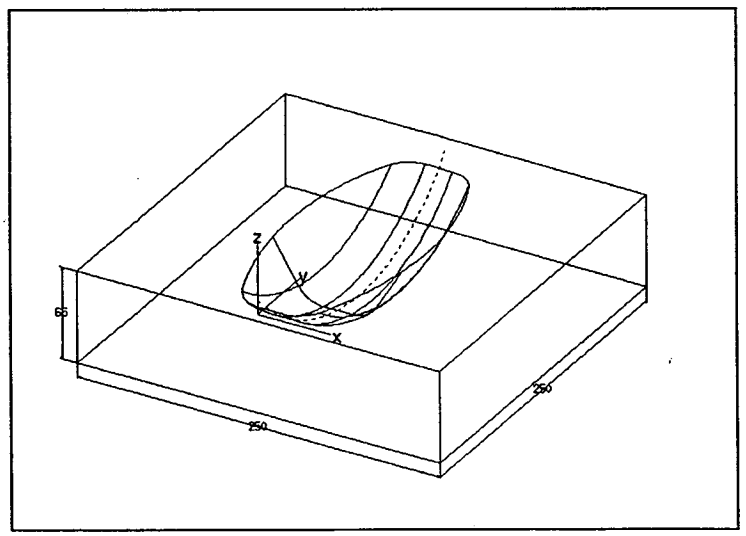


Figure 7.10 IMPRESS Flat Bottom Channel Feature Forming Female Tool

The path along which the cross-section is swept is generated from the original surface of the component between two points. A significant aspect of the use of the feature is how the end conditions are formed. These may be formed with simple blends or by more complex geometry when the form of the cross-section runs out into the main surface. There are two approaches to achieving such complex end conditions: create each end condition as a separate sweep feature, or by more careful construction of the sweep path, include the definition of the ends as well. Creating the paths for end conditions is a general contour generation problem which can be addressed with the use of a sketcher.

The use of a sketcher generates many constraints implicitly, such as coincidence and tangency, whilst the user adds further constraints explicitly such as size (length, radius) and form (parallelism, perpendicularity). Keeping the constraints out of the feature and in a dedicated software component is more efficient. This is achieved by attaching the constraints as attributes of the body generated by the sketcher. This enables feature template definitions to be more general when taking bodies as parameters, because it does not matter how they are generated. The dimension feature for a sketcher constraint supports only that data necessary to generate the graphical representation. Although the dimensions are associated with entities in the model the problems of maintaining persistent object identity do not occur because the object associativity is maintained by the constraint manager. The advantage of the sketcher constraints is that the user creates them graphically, rather than by programming, which is very fast and intuitive.

Currently the contour that is to be swept (base) forms the reference coordinate system in which the sweep path is related. When the sweep base is considered the major item, such as with the flat bottom channel, this works well. Constructing models when the sweep path may be considered the important item causes some positioning problems. Take for instance, the construction of the walls of a soap box, Figure 7.11.

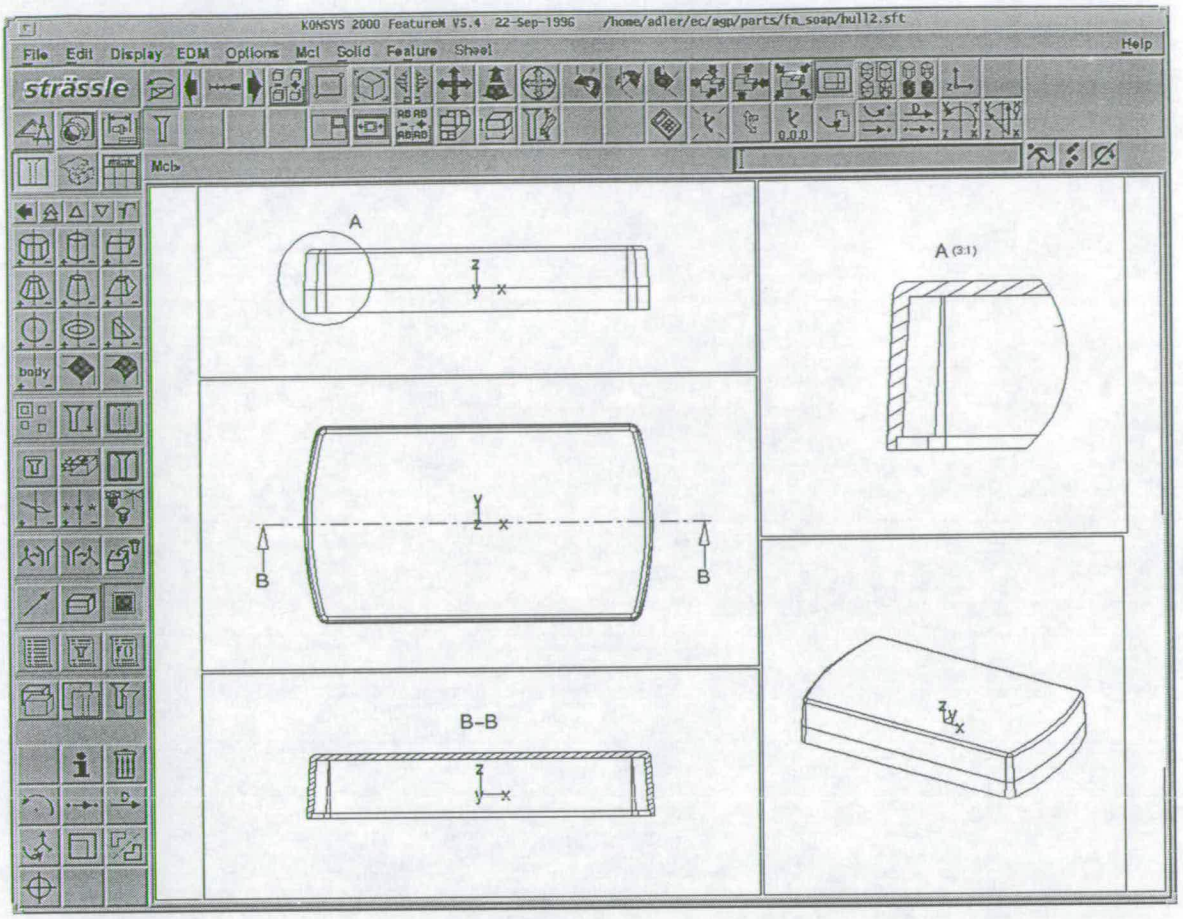


Figure 7.11 Soap Box

In this example the tendency is to locate the base contour (forms the cross-section of the wall) relative to the path (plan of the box). However, this does not actually change the relative positioning of the two features (and importantly bodies) for the sweep, because both features are moved equally due to the path being defined relative to the base coordinate system.

Feature parameter dimensions describe the feature and are completely associative with the feature *ShowBody,* they are not associative with the resultant model. How a feature is combined within the model does not change how the feature parameter dimensions are displayed. For example, if a hole feature is not positioned completely within the model but is sticking half out, then after combination with the model, one end of the parameter dimension representing the depth of the hole will appear in space. It would be relatively easy to integrate rules into the feature make method to

represent extrinsic constraints, however, such simple constraints are only likely to be valid when the hole is placed perpendicular to a planar face. It might be argued that for manufacturability, hole forming should always occur in a direction normal to a planar face. This however is only true for the entrance and not the exit of a through hole. Such restrictions placed in the make routine, implying a presumption about how the form could be manufactured before considering the resultant model seem unhelpful. Furthermore from the users perspective, spot facing of components in order to enable drilling for instance, would use the same feature in a general feature based design system. A purely manufacturing oriented system might well distinguish between blind and through holes, and spot facing features, but such a system would probably prove too restrictive to the designer. Feature validation and representation of the semantic meaning is therefore an on going topic.

Feature parameter dimensions are intuitive to describe in the template definition, particularly when the feature origins may be used as the defining points for the dimension. There are, however, several difficulties with the implementation.

In order to provide support for user defined feature definition each parameter in the feature template definition requires a dimension description. When the make method is relatively complex, as is the case with the flat bottom channel (Figure 7.9), the defining points of the dimensions are calculated in a procedure. This procedure calculates all coincidence points of the elements that make up the contour. Each parameter of the dimension description requires an if_needed method to define its value. This procedure calls the coincidence calculating procedure each time, which is very inefficient. A more efficient method is to associate a procedure with just the pos1 parameter. This relies on the fact that any function that reads parameters from the dimension must read pos1 first. In the pos1 if_needed method all other parameters for that dimension may be set. This means there is only one procedure call per parameter set against six previously. This change noticeably improves the graphical redraw time of the feature. Of course it would be possible to set all dimension parameters in the feature make method each time. The performance gains to be achieved are slight and the associativity of a particular method with a particular

function is distorted. A further problem associated with the definition of methods to calculate the parameters of the dimension description is that to date the interface is fixed which makes the generation of generally applicable methods difficult.

Certain features such as the obelisk, Figure 7.12, are defined using diameter dimensions despite the fact that they do not have any circular elements; the diameter is used to define a circle which encloses all points forming the base, for example.

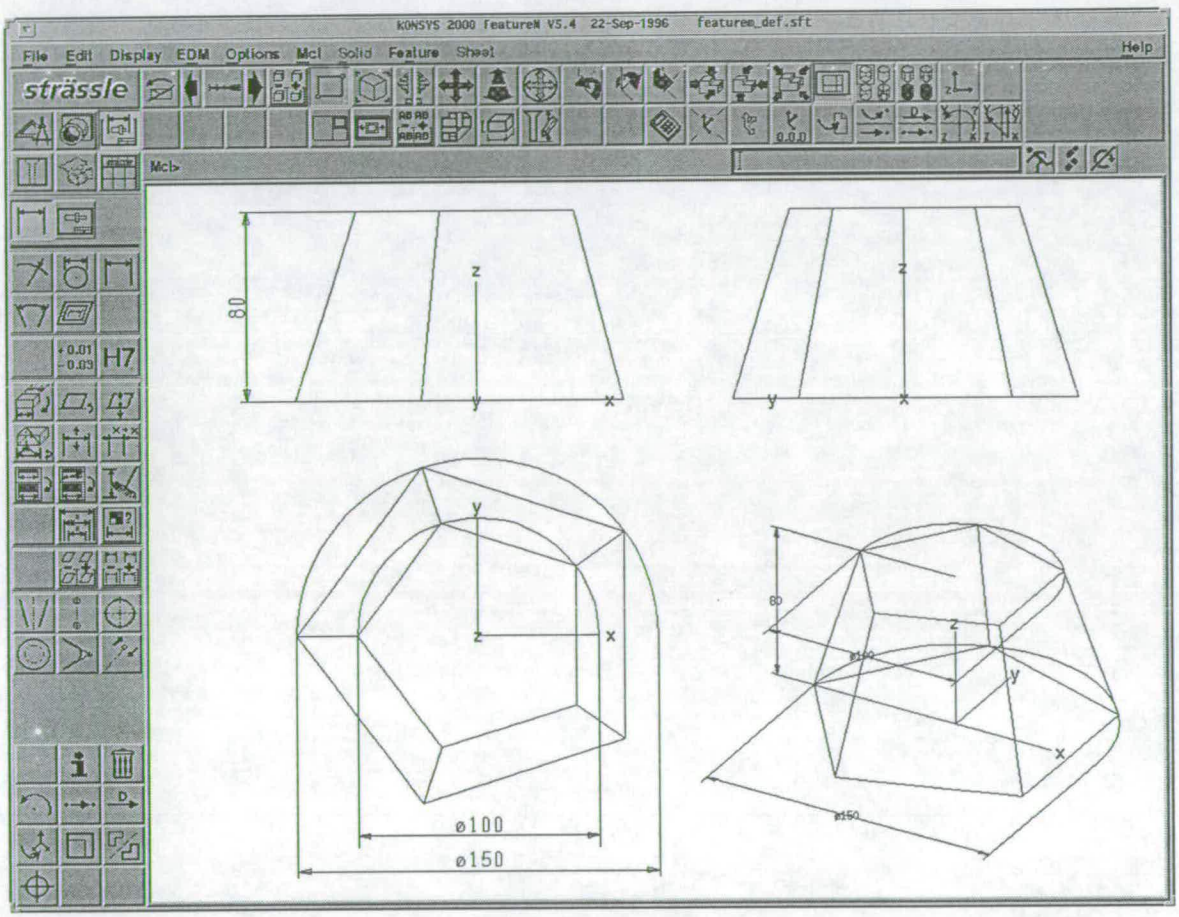


Figure 7.12 Obelisk With Help Lines

Simply defining a diameter dimension is not helpful when there are few points because the attachment points appear to sit in space. The visualisation and understanding can be improved by constructing a help circle containing all the points. Currently, a full construction circle cannot be drawn for a dimension, just the shortest arc from pos1 to either the start or end vertex of the arc being dimensioned.

This means that the largest construction arc that can be generated is just smaller than a semi-circle which is simulated by setting of spos, epos and arcpos.

Local operations such as blending (rounding, filleting) are problematic when describing parameter dimensions that represent the blend radius. This is because to describe the radius dimension correctly the definition requires the start and end positions of the arc to be described. These are not described in the feature. They are available only after the blend operation has been evaluated in the model. Currently the dimension representation is set to point to the edge to which the local operation will be applied. This provides an intuitive graphical representation before evaluation of the blend feature. However, the graphical display of the dimension remains in the same position with respect to the original edge. Therefore, unfortunately, the graphical display does not represent the engineering drawing representation of the dimension. Further work is required to re-associate the representation with the resultant geometry using the techniques similar to those described in chapter 5 for generating uni-directional radius dimensions from faces rather than edges. The problem is further complicated by the ability to generate variable radius blends, potentially generating spline surfaces.

The use of the technical view layout provides significant advantages in the automatisation of generation and layout of dimension schemes, Figure 7.13. Construction of dimensions in technical views automatically defines cutting planes and location selection simulating the ability to dimension silhouette lines, blend radii for example. The graphic display of the dimension text may be optimised to sit above and along the dimension line as is found in 2D engineering drawings.

The camera angle of the technical views may be changed as with any normal view, the difference being that the view may be instantly reset to its engineering definition.

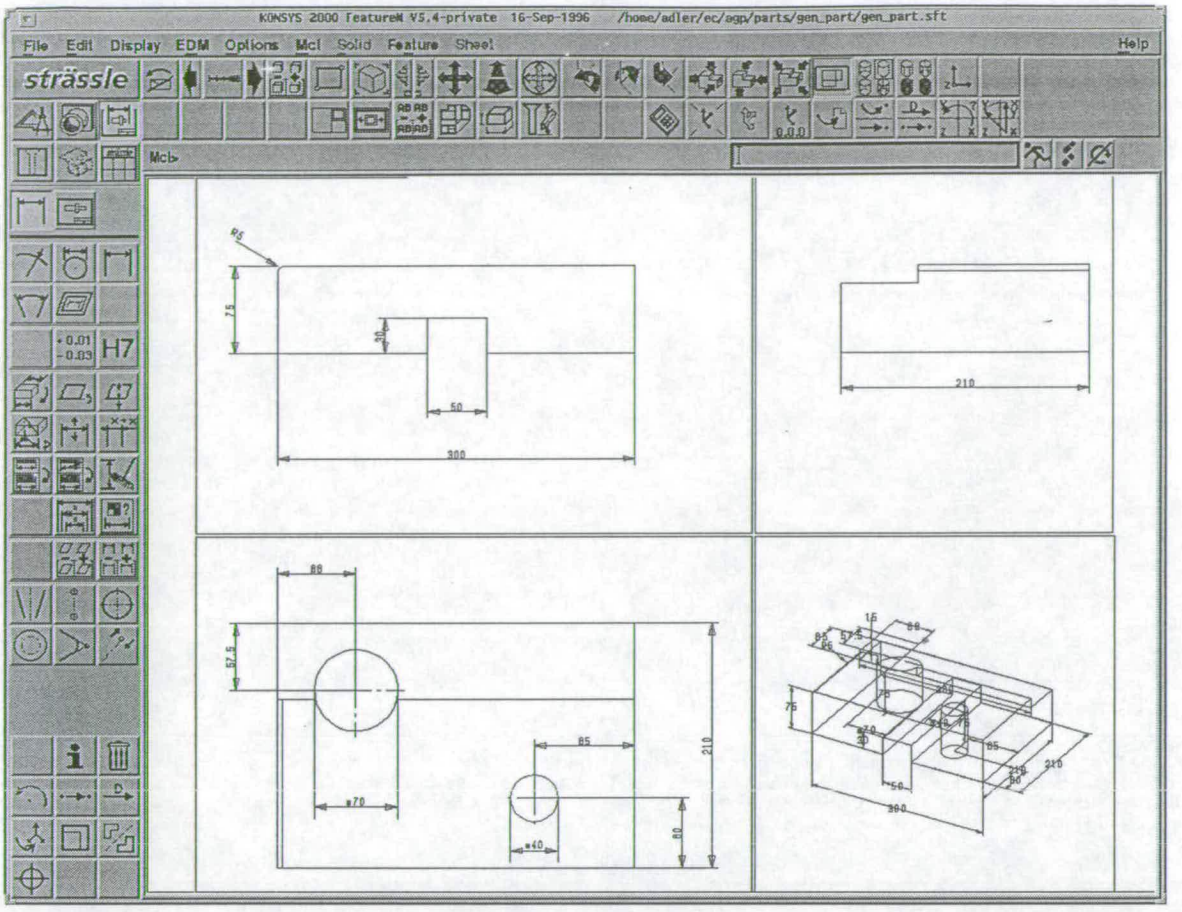


Figure 7.13 3D Model Displayed As 2D Drawing

The use of cross-section views, Figure 7.14, poses as yet unsolved problems for the maintenance of object identity. Currently the kernel is unable to provide persistent object identity for the elements of a cross-sectional view because this is a different body to that of the original model which does not exist in the browser data dictionaries. The cross-section model and dimensions associated with the original (whole) body, but viewed with the cross-section, will be automatically updated to represent changes made to the original model. The cross-section may be directly dimensioned using type 1 non-associative dimensions but these of course will not respond to changes in the model.

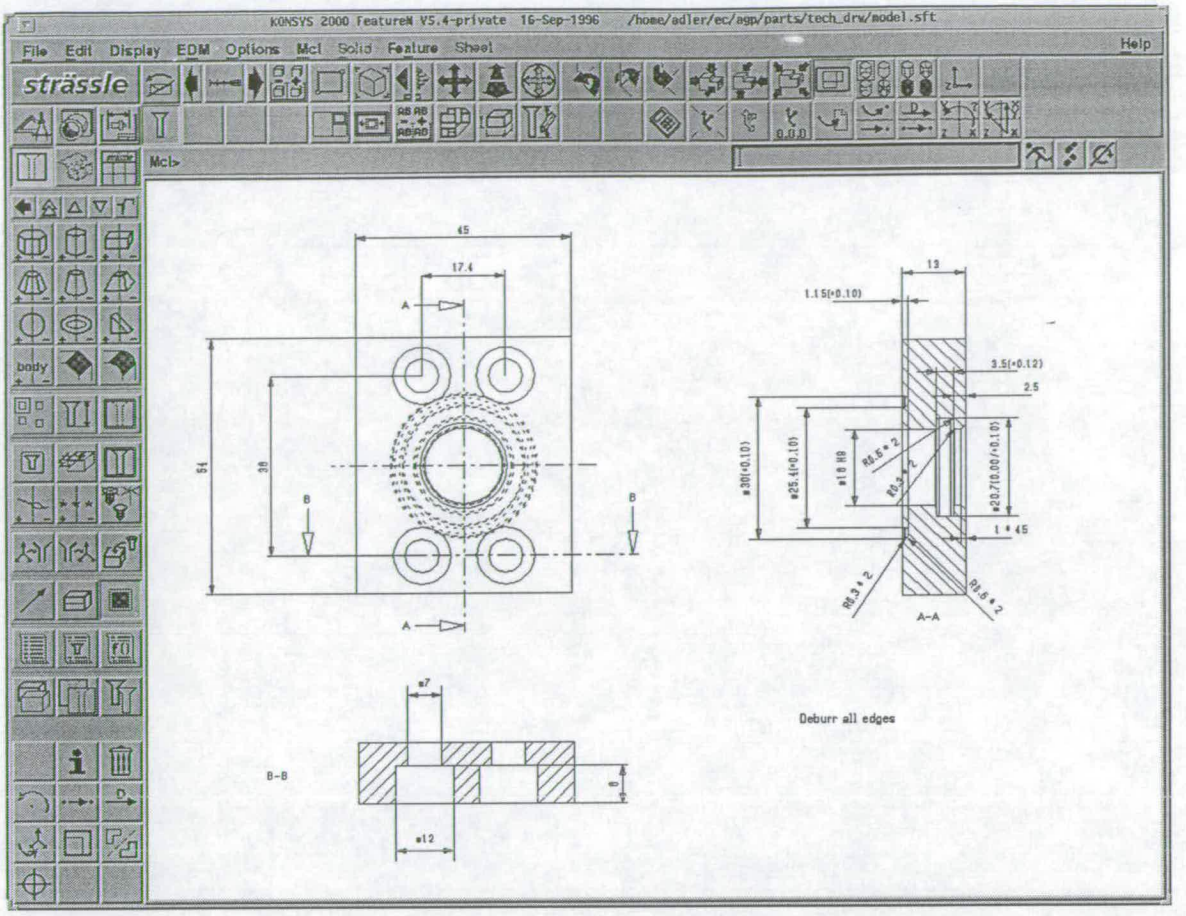


Figure 7.14 Cross-Section With Dimensions

The term accuracy feature is often used ambiguously to refer to the physical aspects of the model which are being dimensioned or toleranced, or to the dimension or tolerance applied to those physical aspects. The architecture implemented in chapter 5 amalgamates both meanings because the data structure supporting the accuracy feature representations is the feature model data structure and the physical aspects of the model are referenced in the accuracy feature.

The STEP Shape Variational Model [STEP47] does not consider the visualisation of dimensions or tolerances, nor does it concern itself with direct association of dimensions and tolerances with the underlying geometric model. The dimensions and tolerances are associated with shape aspects or derived shapes aspects such as centre lines and planes of symmetry, etc. No direct method of associating dimension and tolerance information with the Form Feature Model [STEP48] is given. The STEP

model differentiates between directed and non-directed dimensions and tolerances. The representation described in chapter 5 implicitly describes all dimensions as directed because of the graphical representation being defined from pos1 to pos2. Derived shape aspects are either represented implicitly in the objects which are either generated by parameter dimensions, provide object identity, or form the elements of the symbolic construction geometry. Direct relationships are provided between parameter dimensions and their associated features or between faces in the model and the objects that created them. Analysis of the topological model allows any vertex or edge to be related to its generating features via its connecting faces. To date no attempt has been made to generate a STEP compliant representation of the dimension and tolerance model. However, the author believes that the richness of the current representation due to the implicit nature should be able to be mapped to the STEP model.

The vectorial dimensioning and tolerancing system [HEN95] described in chapter 2 demands that all elements that are dimensioned or toleranced are represented by points, vectors and sizes. These are implicitly supported by the geometry of a solid modeller. However certain elements may have no unique representation. For example, the point describing the root for the normal vector describing a planar face could be any point in the plane. The graphical representation of the dimensions and tolerances evaluates this information as the start point of either of the projection lines. Such points may also be used to define the *effective plane* in which size dimensions are calculated. The description of points and vectors implicit in the description of the dimensions and tolerances is readily applicable to definition of vectorial tolerancing schemes and the consequent advantages for inspection process integration because a coordinate measuring machine can readily determine the measured substitute elements which can be directly and automatically compared to the dimension and tolerance representation applied to the geometric model. To date such inspection practices have not been investigated.

The uni-associative parallel tolerance investigated in chapter 5 has provided some interesting results, Figure 7.15. Its construction is virtually identical to that of a linear

dimension between two planar faces or axes. However, the linear dimension between two planar faces or axes can always be calculated between the anchor points derived from each face. The parallel tolerance requires that the planar faces or axes are parallel. The parallel tolerance follows changes in the geometry and whilst parallelism is maintained this causes no problem. This is always the case between two faces generated by the same feature.

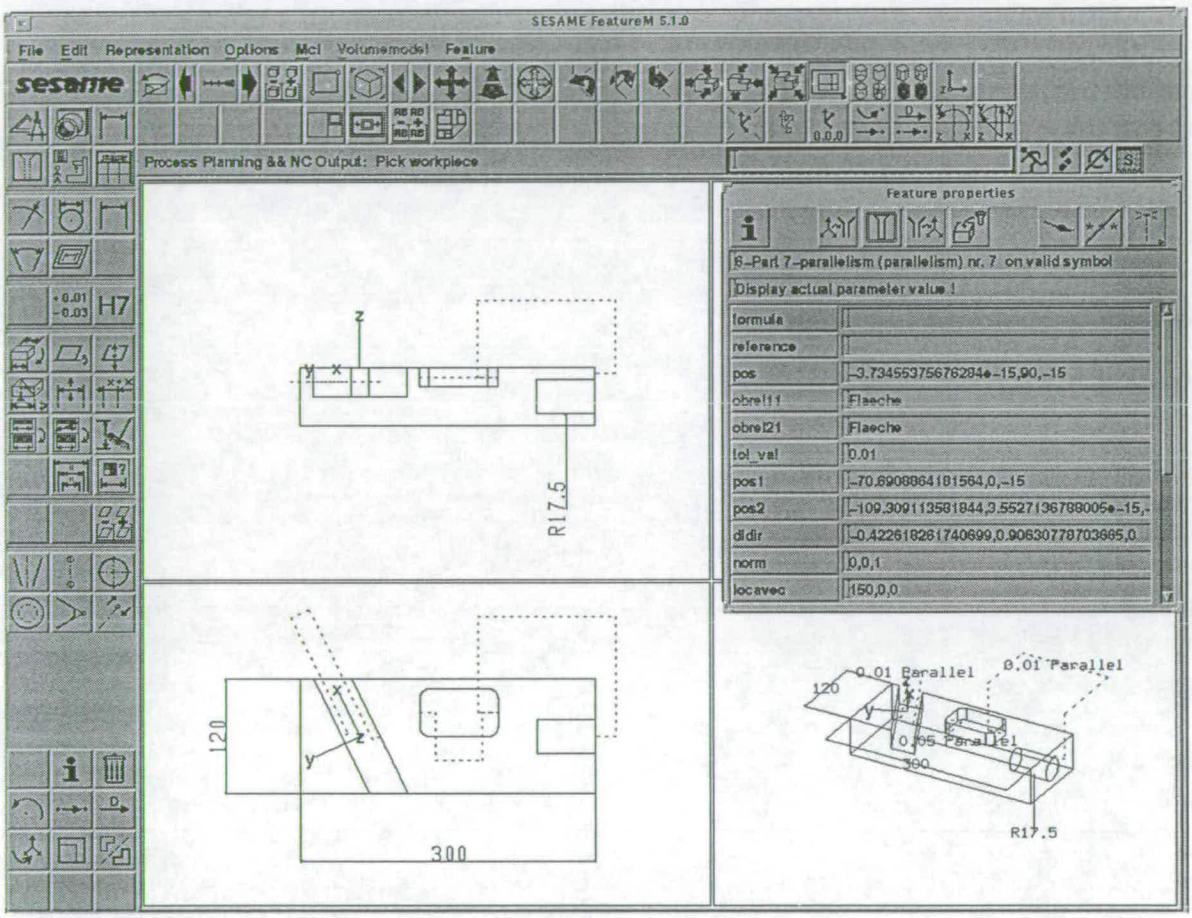


Figure 7.15 Parallel Tolerance

It is interesting to consider the problem of a parallel tolerance constructed between two parallel faces of two separate features. On constructing the tolerance the parallelism is determined before the tolerance may be created. If subsequently one of the features is rotated so that the two faces related by the tolerance are no longer parallel this is detected in the make method. The question is which has higher priority - the orientation of the second feature or the parallel tolerance. The tolerance

could be considered as a functional constraint that the designer explicitly states by using a parallel tolerance; in which case the second feature should be automatically rotated to maintain parallelism. On the other hand if the designer rotates a feature knowing that a parallel tolerance is attached, does it mean that the act of rotation implies that the original functionality indicated by use of the tolerance is no longer valid and the tolerance should automatically be deleted. If it is decided that the second feature should be automatically rotated to maintain parallelism between the two faces a further significant problem is determining in which manner the second feature should be rotated.

The use of relational geometric tolerances in 3D is analogous to the use of such constraints in 2D. The solution of constraints in 2D is relatively well understood, however to date, there is no general solution to the 3D problem particularly when considerations for feature validation and merged faces are included.

In a similar analogy how should a dimension or tolerance between two entities in the model be treated when one of the entities no longer exists because of movement of the feature to outside the model or interaction with another feature which obliterates it. Clearly the tolerance is no longer valid and cannot be displayed, but should it be completely removed from the data structure. The approach taken is that it is not removed from the data structure merely switched off. Thus in future, if the object identity is re-found by further changes in the model, the tolerance will be redisplayed. In the case of two faces that are no longer parallel, when parallelism is regained by realignment of one of the features, the tolerance is redisplayed.

Although consideration has been given to dimensions that can be used to control the model, but do not represent size or location parameters of features, currently no implementation work exists. It is envisaged that an approach similar to that being developed for the generation of new features out of a combination of existing features will be applied. In both circumstances rules will have to be represented to describe how and which location and/or size parameters have to be controlled.

The complete feature catalogue supplied with the feature modeller described in chapter 3 uses the parameter dimension definition methods described in chapter 4.

7.6 Exchange Of Feature Models

To date attempts to formalise exchange mechanisms for feature models have centred on attempts to define sets of features for all users and applications that are capable of describing the geometry of a component. The STEP efforts have focused on providing a Form Feature Model (Part 48) and a Process Planning Features Model (Part 224). It is the authors understanding that development of the Form Feature Model has been suspended. The Process Planning Features have been developed from the John Deer Catalogue. To date there are no standards which address the subject of the exchange of User Defined Feature Models.

Exchange of feature models may be split into a number of distinct areas:

- the syntax of the features description
- the semantics and meaning of the features
- the geometry of the features and the resulting model
- feature ordering
- object identity
- validation of parameter values
- control of feature manipulation
- control of the user interface to the feature

In order to transfer a model containing user defined features both the implicit definition of the feature described by the template and the explicit instance of the feature in the model must be supported in a standard manner. For procedural methods this poses particular problems for the control methods that are declared in the template definition. For declarative methods the problems are passed to the ability of three dimensional constraint solvers to generate the same geometry.

Exchange of feature models is currently achieved by agreeing feature template names, parameter names and the form that is intended to be generated. Models generated with the features classified within the SESAME project have been exchanged using two formats:

1. CODL - Component Description Language [SAL94]

2. TEBES - Technische Element Beschreibung Sprache [TOE94a]

CODL has its own syntax where as TEBES is constructed using the EXPRESS language. The CODL language is specifically focused at process planning applications. TEBES is a general method for defining form features. The template definition method described in chapter 4 is based on MCL+ syntax.

For instance, in the system described in chapter 3, all functionality with respect to defining the feature local coordinate space, how entities in that coordinate space are manipulated, what geometry is made and put together is defined by the user in the make, if_created, and if_set methods. These functions enable the user to control the user interface by manipulating the access rights of parameters. Features may be programmed to have completely different topologies in different modes, and the user is able to define different representations for the features - volumetric, symbolic, combined. Implicit in the methods is an ability to control the feature modeller and system interface itself. Such wide ranging and specific functionality is unlikely to be supported in a standard manner which suggests that the definitions of features must be kept much simpler. However the ability to define different representations for the feature and define how it interacts with the model is a complexity that should be maintained. Purely declarative languages do not have any system or modeller control functionality and the definition of the feature is completely separate from the modelling system that uses it. At first this may appear attractive as an exchange format but there are problems with how the constraints are defined and solved. One need only take note of the differences between parametric and variational constraint solvers. To date such languages have considered relatively simple shapes and it is

difficult to see know how successful they will be at representing much less constrained shapes compared to the functionality described in chapter 4.

A further and important issue that has received no attention to date is the transfer of models containing descriptions of dimension that represent the parameters or constraints associated with feature parameters. To make the analogy with 2D, if a 2D drawing is exchanged, one would expect the dimensions to appear in exactly the same place on the exchanged drawing. Therefore in 3D, why should one expect any less that the dimensions that control the model and with the use of technical views, represent 2D dimensions, should appear any different between system representations? This means that a method of associating the dimensions and defining the start / end points of the dimension lines must also be exchanged. The STEP model does not begin to address such issues.

To date the models exchanged in the CODL or TEBES format have used hard coded make methods in the receiving applications. No attempt has been made to exchange the graphical representation of the dimensions or tolerances.

An important factor in exchanging models is also being able to exchange object identities such that faces, edges and vertices may be re-found on rebuilding of the model. This is a different problem to persistent object identity that is internally required on model evaluation. This is because the geometry does not change, only the pointers must be reset. If it is assumed that the workpiece models are consistent (if not manifold), as is the case with the models described in chapter 3, then in a single body no two vertices will occur at the same point, no two edges, or faces will be identical. Vertices may be re-found by searching for vertices in the correct body and checking for equality of position, edges may be defined by the start, end and middle points which may be similarly found. Faces can be defined by using a ray point and ray vector. All solid modellers have the ability to return objects hit by the ray, careful selection of the point will return the required entity. Accuracy features that point to faces may then be related back to the generating features because the description of a face includes the list of features. Merged faces are therefore not a problem.

EXPRESS is a data modelling language, MCL+ is a programming language. EXPRESS is good at modelling information but not function. Hence, it is able to easily (if verbosely) describe the feature in terms of its parameters and their ranges, but not the detailed control functionality required by powerful feature modelling systems. The ability of EXPRESS to describe validation conditions for parameters is a standard method of supporting the template definition extension described in chapter 4. This a positive aspect.

It is interesting to consider other possibilities for the exchange of feature models. User Defined Features are the most needed element for process integration.

The look, feel and functionality of a system is what differentiates it from its competitors. If this is transposed into the feature environment then it is the control and user interface mechanisms that are proprietary to systems not the description of the feature parameters, their validity, and the geometry of the features. If the control and user interface functions are discarded then it should be possible to define, using EXPRESS and the other parts of the STEP standards for the exchange of volume models (Part 42), a suitable format.

The syntax of features is clearly described by EXPRESS. Object identity at time of transfer is relatively easy using ray point and vector definitions. The volumetric geometry representing the current instance of a feature may be described by the standards [STEP42], its position and orientation in the world are known and can be easily described. The semantics or meaning of the feature are partially described by the naming conventions used and the types of parameters. However the functionality of the feature is bound to the explicitly described control algorithms which are a function of a particular system and its interpreted programming language. Such an interpreted language that can make direct calls to all modellers is unlikely to be developed. This treatment is similar to NC CL data which is mostly a one way transfer of the tool paths which have to be post processed into a set of control codes. The control codes may be locally changed but that change is not generally propagated back to the original source, certainly not automatically. In fact sometimes

it is desirable not to change the source geometry but to provide such effects as surface relief by changing the NC code generated. Feature models by definition should be able to be changed. Regulation of parameter values can be defined in EXPRESS as described or simple procedures may be described and called via attribute mechanisms. Feature ordering is very important for generation of the correct resultant geometry and can be well defined in the order that the features are written. What is not clear is how well interaction (editing) of the feature can take place in the non-original system. Certainly exchange of user defined features and their geometry should be possible which is much of what is demanded by CAPP and CAM systems.

7.7 Feature Technology As A Process Integrator

An advantage of the hybrid technique is that each form feature (except for local operations) produces an individual volume which is combined with previous features to create a model. If a volume can be broken down into smaller volumes that can be assigned a sequence of manufacturing operations, the smaller volumes are said to be manufacturing features. Mappings may be defined between design features and the corresponding manufacturing features. Therefore a manufacturing view of the component may be generated. The combination of the manufacturing view with the design constraints explicitly stated in dimensions and tolerances enables more sophisticated process planning tools to be implemented. However feature mapping techniques do suffer a number of problems.

Direct feature mapping only works when features remove material. This can be successfully applied to 2½D component models where the mapping is of a one to many nature. Feature combinations such as those forming complex pockets cannot be successfully planned using feature mapping alone. Local operations such as rounding and chamfering may be mapped to removal processes whereas filleting indicates that material must not be removed, effectively requiring changes to be made to the feature(s) producing the edge that is filleted by the local operation. A locally swept

face into a model, whilst possibly producing a machinable region, does not posses any higher level meaning helpful to a mapping process because the form produced is completely dependent on the face contour that is swept, the sweep path and the form of the face. Sweep operations in general cannot be mapped directly because they typically produce 3D or free form geometry. Machining strategies for free form geometry are different to 2½D approaches because the techniques used attempt to machine the whole component as if it were a set of continuous tangentially connected faces. The strategies are said to be surface oriented. It is a face based approach and manufacturing features are better defined as area (rather than volume) and process combinations. The generation of the manufacturing feature model is an iterative process significantly dependent on tool availability and quality requirements. Machined free form surfaces are always approximations because relatively small radius tools are used to create relatively large radius (even flat) surfaces. The quality requirements are defined by tolerances associated with the faces of the design model which may be represented by the techniques developed in chapter 5. Therefore, whether feature mapping strategies or surface based algorithms are used for 2½D or 3D components the inclusion of tolerance information associated at a face level is a requirement for process integration which the developments of chapter 5 address.

Feature recognition techniques have been applied in a constrained fashion to re-map positive features to new combinations of negative features but only so long as the positive feature is contained within the body of the base feature. More general feature recognition techniques can be helped by the generation of hints. Design by features clearly can provide a relatively robust method of hint generation.

It has been stated that the ability for systems to model the interactions between features will be a prerequisite for progress of integrated design and manufacturing software tools [MIL93]. Interactions may be defined as explicit or implicit. Explicit interactions are stated by the designer and added as information to the model. Indicating which surfaces in a model form the aesthetic surfaces, dimensional and geometric tolerances constraining an object. An implicit feature interaction is derived from the model using geometric reasoning techniques to identify proximity,

obstruction, and how two or more objects interact. Explicit feature interactions are supported by the dimension and tolerance representation. The integration of the generating features as attributes of a face would enable rapid reasoning to generate hints as to where interactions may occur. Supporting interactions between features in this way aids the integration of CAD and CAM software tools.

Current interfaces to feature modellers tend to be too like interfaces to solid modellers. Consequently usage is as a parametric solid modeller rather than enabling the inherent semantic of the features to be used by advanced planning and NC systems. The use of 3D feature modellers is not as intuitive to engineers used to 2D draughting systems as might first be thought. It appears that the concept of drawing a 2D profile followed by sweeping is natural for 2D designers. Users of solid models build models in the same way with features simply treating them as producing either positive or negative volumetric shapes. These are perhaps two reasons for the success of commercial parametric or variational BREP modellers, which also to date provide better functionality for constructing a wider range of geometry. However, the advances reported in chapters 4 and 5 of this thesis are aimed at addressing these deficiencies. Furthermore, parametric and variational systems suffer commercially from a lack of high level manufacturing interfaces, relying on feature recognition strategies as exemplified by the PART system [ICEM].

To enhance concurrent engineering strategies it is desirable that the base geometry represents the stock material, and the form features represent the machining volumes subtracted from this to create the desired component. It must be stated that this is not always an intuitive way to design, particularly when using swept free form type geometry as discussed earlier in this section.

To assume that a solid model of the stock material exists and does not require designing, whilst typical of many original equipment manufacturer - supplier relationships, is not consistent with the complete life cycle model of a product required by concurrent engineering strategies. This is because a casting requires tooling to produce the mould which also must be modelled and produced. To date the

manufacture of castings receives relatively little direct CAD support primarily due to the fact that the model required to generate the pattern is different from the model of the component. Even high value precision castings produced by die casting techniques which require accurately machined metal moulds will rarely be fully filleted because of the significant time and cost penalties. Large castings used as the starting point for machining to form a component are by nature inaccurate because of the casting process, the number of castings is usually small, and therefore the patterns are made by hand from wood or polystyrene by considering 2D drawings.

However, the architecture of the workpiece tree model does enable efficient modification to offer advantages when life cycle modelling. In product development, functional prototypes are frequently required for testing [CHI93]. If the application demands that a metal prototype is required this is typically machined from solid. As development progresses, prototypes that are not only functionally representative, but materially representative of production components may be needed for testing. These would generally have to be made by a similar manufacturing process. For example, if pressure die casting was the desired production process, a form of low pressure sand casting would be used. This necessitates making a pattern for mould making. Three approaches are typically used: hand fabrication, machining and rapid free form fabrication techniques such as stereolithography. For rapid free form fabrication a complete geometric description is essential and for machining it is desirable (it is not always necessary to model all fillets because careful tool selection can create these as a by product). A feature model of the raw casting can be created for this purpose, but a different problem is posed for the finishing of the casting compared to machining from solid; the base geometry is different, and fewer features will typically be required because less surfaces need to be cut.

The architecture described in chapter 3 meets these needs by allowing interchangeable base geometry. This is impossible to achieve in parametric systems where the positioning of the features is dependent on the topological entities that previously exist in the model. In this system the features remain in their same positions and orientations relative to the workpiece. Those features that are not

needed may be suppressed (remain in the model data structure but are not used to generate geometry), the remainder may be readily tuned, if at all necessary. This removes the need to start the new design from the beginning, provides data consistency, and shortens modelling time. Thus the methods developed in Chapter 4 to allow features to be assigned solid bodies as parameters is necessary. The base geometry for one model may itself be a copy of the resultant geometry of another feature model. The manipulation techniques that allow setting of the origin for a non-parametric (extrinsic) feature are of significant importance. Feature locations are referenced from the workpiece origin defined by the origin of the base feature. It must be possible to position the new base geometry so that it is in the correct position relative to the other features and its origin is in the same location (World Coordinates) as the previous workpiece origin. The techniques described in chapter 4 to enable the selection of origins for features of extrinsically defined form make this possible.

In keeping with the concurrent engineering concept, of designing both product and process, one form feature has been introduced that captures the fact that a casting is imprecise. The parallelepiped feature is used to indicate that flat surfaces on castings require a machining process to create them. In SESAME machining could only take place when negative features existed in the model (due to feature mapping). Despite the fact that the designer's computer model showed a perfectly flat face, by creating it with the parallelepiped, this indicated to the manufacturing engineer that it was not known exactly where the surfaces of the casting would be and would definitely require machining. Although implemented in the design system as a pocket which has sides because the system requires a valid body to be returned from the make routine for combination (subtraction) from the model, the manufacturing meaning does not restrict tool access from the sides as is the case with a true pocket.

It is important to consider if Feature Based Design and Manufacturing Systems help to produce better products. At the component level such systems are focused on details and production possibilities. They are necessarily constrained but easily adapt to development cycles and are therefore useful for the manufacturing oriented

engineer. Unconstrained aesthetic design is less well supported by feature based techniques as is the problem of design automation. Current feature based design systems are limited when used to support complex tasks such as designing a car, but are highly applicable when detailing an individual component ready for manufacture. There is clearly need for further investigation of the combination of large complex design automation systems with functional manufacturing oriented feature based systems.

A constant criticism of feature based design has been the constraints that are placed on the designer. However, one need only look to two recent examples (Lotus Elise, Porsche Boxter) from the motor industry of design concepts presented at shows being considerably altered for production. Issues such as safety, cost, quality and manufacturability influencing the design modifications. Feature based design aims to produce manufacturable designs of high quality and of reasonable cost. Therefore it can be concluded that feature based techniques do aid the production of better products, if not, the most purely aesthetic design.

7.8 Applicability To Other Feature Modelling Systems

The architectural design is applicable to development in other hybrid feature modellers. The techniques are not applicable to parametric or variational BREP modellers. Two significant aspects of the functionality that must be supported in any system attempting to apply the reported techniques is the ability to provide persistent object identity during the process of model evaluation and the flexibility to create multiple workpiece models and control the order of evaluation of them. The techniques developed for defining user defined features are suitable for procedural not declarative systems.

7.9 Chapter Summary

In this chapter the contribution made by the work reported in this thesis to the advancement of feature modelling technology has been stated. The Feature Based Design System presented in chapter 3, the techniques developed in chapter 4 to support extrinsically defined form, and the methods implemented in chapter 5 to represent dimensions and tolerances have been discussed with respect to classification, modelling architecture, and representation. Investigation of the possibility to develop general exchange techniques for user defined features has been shown to be limited. Feature technology, however, is undoubtedly an aid to design and can contribute substantially to process integration.

In the next and final chapter (chapter 8) the conclusions of this thesis will be presented and recommendations for further work will be summarised.

Chapter 8

Conclusions

8.1 Introduction

Chapters 1 and 2 of this thesis have presented feature modelling technology, associated problems and the need for further development. Chapter 3 presented the system used as the baseline for the developments reported in chapters 4 and 5. These developments allow complex geometry to be integrated in the feature model, and enable models to be dimensioned and toleranced. Chapter 6 presented the original concepts investigated, decribed the techniques used to acquire and analyse information, detailed industrial collaboration and stated the results and feedback from software testing. The techniques presented in chapters 3, 4, 5 and 6 were discussed in chapter 7 from the perspectives of classification, architecture, representation, model exchange and process integration.

In this chapter (chapter 8) the conclusions of the work will be stated in section 8.1. In section 8.2 the scope for future work will be presented.

8.2 Conclusions

1. The fundamental concept investigated in this thesis has been to integrate in a single computer software representation the information required by designers to define a component for analysis and manufacturing, thus enabling software based simultaneous engineering strategies to be implemented.

2. One of the recent and significant trends in manufacture is to link automation with flexibility in order to meet market needs, management of new technology, and criteria for financial investment. Integration and automation of design, process planning and part programming functions has long been recognised as a means of improving competitiveness. Feature modelling technology is believed to offer the possibility to meet the demands for flexibility which address the life cycles of the products. The aim of the investigations into feature modelling technology described in this thesis has been to provide better computer aided support for design and manufacturing.

3. A number of deficiencies in feature modelling techniques have been identified which have resulted in a lack of success in achieving the aims described in 1:
   - There is a considerable legacy of purely volume based models.
   - Libraries of predefined features are too restrictive.
   - User defined feature functionality, particularly definition, has been neglected.
   - Features are assumed to have implicitly defined topology and 2½D geometry, little support being given to features of complex extrinsically defined form.
   - Only nominal, or perfect size geometry is supported.
   - Dimensions and tolerances are detailed in engineering drawings which are 2D representations separate from the feature model.
   - Tolerances and dimensions that are not homogeneous with the shape model preclude the development of variational models and enhanced design and manufacturing analysis.
   - There are no standards for the exchange of user defined feature models.

- The effectiveness of advanced computer aided process planning, computer aided quality and part programming systems based on feature models derived from a separate design system is reduced because only nominal shape data is supported; other information necessary being input after interpretation of 2D engineering drawings.

4. The following objectives were defined to address the deficiencies detailed in 3:
   - Development of methods and techniques to model features of extrinsically defined form, typically swept geometry and non-parametric solid models.
   - Implementation of a 3D dimension and tolerance modelling module related to both the feature and solid models.
   - Establishment of techniques to control a feature model through the use of 3D dimensions.
   - Investigation of structures for the exchange of user defined feature models.

5. The classification of form features has been extended to include extrinsically defined features. Extrinsic features do not have a complete description of the topology and geometry described in the definition of the feature. They allow the description of more complex geometry than is typical of 2½D features.

6. Techniques have been developed to allow ACIS solid bodies to be integrated with the feature modeller. Such features are said to be non-parametric because the size and form of the feature cannot be changed. However the location and orientation of the feature within the feature model may be controlled through the feature model structure. Such features allow re-use of legacy data and form the only standard method of feature model exchange.

7. The concept of feature origins has been implemented in order to provide a more natural method for the user to locate features in the model. For implicitly defined features the method of describing the feature template has been extended. The enhancements enable valid values and / or ranges of values to be defined. For any user defined feature, parameter validation may be performed and advanced GUI support may be provided for highlighting and picking of origins.

8. Extrinsic features formed by sweeping operations have been implemented. These features enable geometry created in a number of complementary systems to be used within the feature modeller to generate more complex geometry than 2½D. Sculptured surfaces imported from a free form surface modeller, contours developed in a sketcher, or imported from a 2D drawing, may be used with extrinsic features. Careful maintenance of coordinate systems is required to maintain the shape of the feature generated on movement of the feature within the model. Further manipulation and maintenance of the coordinate systems enables advanced GUI support to provide the same functionality as the definition of origins for implicit features.

9. A multiple workpiece structure has been developed which supports not only form models but the dimensions, tolerances and construction geometry associated with a form model.

10. Classification of dimensions and tolerances requires that the level of associativity with the model and the function that the dimensions and tolerances convey, must be considered before the kind of dimension.

11. Non-associative, and uni-associative linear, angular, diameter and radius dimensions and dimensional tolerances have been implemented that may be applied to solid and feature models. They are applied to the 3D models and are visualised in 3D. Dimensions imported from 2D systems attached to contours are represented as uni-associative dimensions.

12. Linear and radius dimensions have been implemented that represent the distance and radius constraints controlling a 2D sketcher. The dimensions not only represent the geometry but may be used to change it.

13. The template description method for user defined features has been further extended to enable the definition of dimensions and tolerances that represent the size parameters of features. This includes the logical and graphical definition. The location of features may be defined through the size parameters of the

positioning feature. These dimensions are fully associative with the feature model and may be used to control it. They do not necessarily represent the resultant geometry.

14. A uni-associative parallel geometric tolerance has been implemented. It has been shown to behave as a 3D geometric constraint.

15. The use of a window layout consisting of technical views enables the 3D model to be viewed as though it were a 2D engineering drawing. The dimension and tolerance descriptions use the technical views to speed construction. Edit functions have been provided that enable the dimensions and tolerances to be laid out as though they were in 2D. This removes the need for the use of a separate 2D draughting system.

16. Feature models containing both form and accuracy features have been output in neutral formats (CODL, TEBES) that can be read by two different process planning systems. The description can be said to be a partial product model. Any associative dimensions and tolerances applied to geometric entities in the feature model may be related to the features that generated them. This is important for feature based process planners. The method used relied on hard coded procedures to define the volumetric body of the features in the process planning systems. Thus the method was only applicable to pre-defined feature sets.

17. A method for the exchange of user defined feature models has been proposed. It is necessary to transmit both the implicit definition of the feature and the explicit definition of the feature that makes up the model. By defining the current state of the feature in the model as a volumetric body in the implicit definition the form of the model should be able to be built up correctly. The feature attributes (names and parameters) may be conveyed by existing EXPRESS based methods. The only limitation is that the form and shape of the individual features in the receiving model may not be changed. However, the individual features, their parameters, and their form are conveyed, which is what is required by downstream applications.

18. The methods reported in this thesis contribute to the extension of feature modelling technology in the following areas:
    - Extending procedural methods for user defined form feature definition.
    - Considering the needs of user interaction and GUI support in user defined feature definition.
    - Applying a feature modelling system architecture to the representation of dimensions and tolerances.
    - Proposal of a method for the exchange of feature models containing user defined features.

19. The following original contributions to the knowledge and understanding of feature modelling technology have been made:

    I. The original concept that certain features cannot be completely described by their make methods has led to the collective grouping of these features and definition of an important new class of features called *extrinsic form features*.

    II. The new concept of *multi-dimensional* or *hyper* feature models has been investigated in order to increase the scope and architectural complexity of feature model data structures enabling a wider group of feature classes to be represented including form, dimension, and tolerance features.

    III. The original concept that feature parameters, like features themselves, have characteristics that allow them to be grouped together in classes has enabled a novel combination of procedural and declarative approaches to be applied to user defined feature definition languages, which allows more intelligent mechanisms to be built as a part of the feature modeller kernel and graphical user interface.

IV. The new concept that combines the ideas presented in concepts II and III into a new, more sophisticated feature definition language, which describes both the dimensions and tolerances of a feature in a 3D manner in the template definition.

V. The original concept that provides "handle" like functionality which is applicable to all extrinsically defined features and utilises the body coordinate system of the objects that form the unknown aspects of the features.

VI. The new concept for the exchange of feature models that increases the information content that can be transferred by reducing the flexibility of systems to subsequently change a non-native feature model.

20. The techniques developed are applicable to all procedural feature based design systems capable of supporting persistent object identity and multiple workpiece models.

21. The work has clearly demonstrated the feasibility of providing homogeneous form feature, dimension and tolerance models. Feature modelling techniques have been shown to produce better designs because of their support for simultaneous engineering strategies and consideration for the life-cycle of product development.

8.3 Future Work

Feature technology is a broad subject that in application is very environment specific. The scope for future work is also large. There are a number of areas that have been identified in the course of the work reported in this thesis which would benefit from further investigation:

1. Persistent object identity is fundamental to the functioning of the feature modeller and dimension and tolerance model described. Current performance is reasonable but could be improved. However it is not currently possible to maintain object identity for separate but associated bodies such as those forming cross-section or detail views.

2. General refinements to the dimension and tolerance module to enable:
   - Decoding of more entity types. For example the application of diameter dimensions to cylindrical faces.
   - More general definition of angles with respect to fixed directions rather than between two entities.
   - Better conformation of the graphical representation to the standards expected for 2D engineering drawings.

3. Improvement in the representation of parameter dimensions representing local operations. For example, the update of the dimension description to the actual geometry after the formation of a fillet.

4. Extension of the parameter dimension to workpiece level to control shape aspects in the model not directly defined by feature parameters.

5. To extend the parameter dimension to control the relative positioning of workpieces in a model forming an assembly.

6. Application of the architecture developed for the dimension and tolerance model to all geometric tolerances. Investigation of the constraint based nature of inter-feature geometric tolerances.

7. Further development of the feature parameter validation method to enable extrinsic constraints to be defined for a feature.

8. Development of techniques to define the semantic meaning of the feature that may be exchanged and understood by other feature modelling systems.

9. Development and testing of the method proposed to allow the exchange of feature models consisting only of user defined features

10. Assessment of the role of feature based design systems in context with large, typically knowledge based, design automation systems.

Chapter 9

References

[ALT96] Altermat U., "Echo.txt", Straessle Informationssysteme AG, Kanalstrasse 33, 8152 Glattbrugg, Switzerland

[ANSI] ANSI Y14.5M-1982 Dimensioning and Tolerancing, The American Society of Mechanical Engineers, New York, USA.

[ARC84] Archer L.B., "Systematic Method for Designers", in Cross N., "Developments in Design Methodology", Wiley, 1984.

[AUR95] Aurich J. PhD Thesis, Hannover University, 1995.

[BED91] Bedworth D.D., Henderson M.R., Wolfe P.M., "Computer-Integrated Design and Manufacturing", McGraw-Hill, 1991.

[BER91] Berlage T., "OSF / Motif Concepts and Programming", Addison-Wesley, 1991.

[BRA73] Braid I.C.,"Designing with Volumes", CAD Group, Cambridge University, Cambridge, UK, 1973.

[BRU96] Brubetti G., Ovtcharova J., Vieira A., "A proposal for a feature description language", Proc. ISATA 29, Mechatronics, Florence, Italy, 1996, pp117-124.

[BSI] BS 308: Part 1, 2 and 3, British Standards Institution.

[BUT86] Butterfield W.R., Green M.K., Scott D.C. Stoker W.J., "Part Features for Process Planning", CAM-I Report C-85-PPP-03, Arlington, Texas, USA, 1986.

[CAD] Cadam Model Srl, 71 Strada Giaveno, 10040 Rivalta, Italy.

[CAM81] N.N., "Glossary of Form Features", CAM-I, R-80-PPP-02.1, 1981.

[CAM90] N.N., CAM-I, P-90-PM-02, 1990.

[CHA90] Chang, T.C., "Expert Process Planning For Manufacturing", Addison Wesley, 1990.

[CHI93] Childs T.C.C., de Pennington A,. Pedley A.G., Smith A., "Computer Aided Rapid Prototyping - Market Survey", Dept. of Mechanical Engineering, University of Leeds, June 1993.

[CHU88] Chung J.C.H., Cook R.L., Patel D., Simmons M.K., "Feature based geometry construction for geometric reasoning", ASME Computers in Engineering, San Fransisco, 1988.

[CRO89] Cross N., "Engineering Design Methods", Wiley, 1989.

[CUN88] Cunningham J., Dixon J.C., "Designing With Features: The Origin Of Features", Proc. ASME Computers in Engineering Conf., San Fransisco, 1988, pp237-243.

[CUT88] Cutcosky M.R., Tenenbaum J.M., Muller D., "Features in Process Based Design", Proc. ASME Computers in Engineering Conf., San Francisco, 1988, pp557-562.

[CUT91] Cutcosky M.R., Tenenbaum J.M., Brown D.R., "Working with Multiple Representations in a Concurrent Design System", ASME J. of Mechanical Design, Feb. 1991.

[DCU] "The Dimensional Constraint Manager", D-Cubed Ltd, Cambridge, UK.

[DEN93] Denzel H., Vosniakos G., "A feature based design system and its potential to unify CAD and CAM", Proc. IFIP, Darmstadt, 1993.

[DFL92] "Design Feature List", Brite/Euram Project 4539: SESAME, Commission of the European Communities, Brussels. 1992.

[DIN87] Ding Q., Davies B.J., "Surface Engineering Geometry for Computer Aided Design and Manufacture", Ellis Horwood, 1987.

[DSF95] "Users Requirement Document for Design Surface Feature Library", Brite/Euram Project 7049: IMPRESS, Commission of the European Communities, Brussels. 1995.

[EGG95] Egger B., "Anwenderdokumentation: Straessle Graphical User Interface V2.1.1", Straessle Informationssysteme AG, Kanalstrasse 33, 8152 Glattbrugg, Switzerland, Juni 1995.

[EHR93] Ehrmann M., "Supplement of the CODL Format for an Extended Description of the Design Features", Brite/Euram Project 4539: SESAME, Commission of the European Communities, Brussels. 1993.

[ESA87] "European Space Agency Software Engineering Standards", PSS-05-0, Issue 1, 1987.

[ETH] Dr. J. Taiber, Eidgenössischen Technischen Hochschule, Zürich

[FAU87] Faux I.D., "The geometry of curves" and "The geometry of surfaces", in Rooney J., Steadman P., Computer-Aided Design, Pitman, 1987.

[FBDS93] "Feature Based Design System – User Requirements Document", Brite/Euram Project 4539: SESAME, Commission of the European Communities, Brussels. 1993.

[FEN] Feng S.C., Yang Y., "A Dimension and Tolerance Data Model for Concurrent Design and System Integration", National Institute of Standards and Technology, Gaithersburg, MD, USA.

[FOL84] Foley J.D., Van Dam A., "Fundamentals of Interactive Computer Graphics", Addison-Wesley, 1984.

[FRE85] French M.J., "Conceptual Design for Engineers", Design Council, London, 1985.

[FTR95] "Final Technical Report", Brite/Euram Project 4539: SESAME, Commission of the European Communities, Brussels. 1995.

[FUR95] Furpi F., Mazzocchi G., "SESAME System Evaluation Report", Brite/Euram Project 4539: SESAME, Commission of the European Communities, Brussels. 1995.

[GEC] GEC Marconi, Donibristle, Fife, Scotland.

[GIA90] Giacometti F., Chang T.-C., "A model for parts, assemblies and tolerance", Proc. IFIP, Enschede, 1990.

[GRO84] Groover M.P., Zimmers E.W., "CAD/CAM: Computer Aided Design and Manufacturing", Prentice-Hall, 1984.

[HAN96] Han J.-H., Requicha A.A.G., "Hint Generation And Completion For Feature Recognition", Proc. 29th Int. Symp. on Automotive Technology and Automation, Florence, 1996, pp89-96.

[HAR92] Hartley J.R., "Concurrent Engineering, shortening lead times, raising quality and lowering costs", Productivity Press, Cambridge, Mass., USA, 1992. p20.

[HEN84] Henderson M.R., Anderson D.C., Computer Recognition And Extraction Of Form Features: A CAD/CAM Link", Computers in Industry, Vol.5, 1984, pp329-339.

[HEN91] Henderson M.R, in "Bedworth D.D., Henderson M.R., Wolfe P.M., "Computer Integrated Design and Manufacturing", McGraw-Hill, 1991.

[HEN95] Henzold, G., "Handbook of Geometrical Tolerancing", Wiley, 1995.

[HIL78] Hillyard R.C., Braid I.C., "Analysis of dimensions and tolerances in Computer Aided Design", Computer Aided Design, Vol. 10, No. 3, 1978, pp161-166.

[HOU91] Houten F.J.A.M. van, "PART: A Computer Aided Process Planning System", Ph.D. Thesis, Unversity of Twente, Enschede, 1991.

[HUS90] Husbands P., Mill F.G., Pedley A.G., "HAPPI", 6th International Conference on Computer Aided Production Engineering, London, November 1990, pp1-6.

[HUS91] Husbands P., Mill F., Pedley A.G., Warrington S., "The Edinburgh Composite Component", Proc. 5th Int. Conf. On Manufacturing Science and Technology of the Future, Enschede, 1991.

[HUS91b] Husbands P., Pedley A.G., Salmon J.C., "Report on Research Visit to USA", Dept. of Mechanical Engineering, The Unversity of Edinburgh, 1991.

[IDA88] Report R-338, Institute for Defense Analysis, USA, 1988.

[IFW] Dr. J. Aurich, Insitut für Fertigungstechnik und Spanende Werkzeugmaschinen, Universität Hannover, Germany.

[IGD] Dr. J. Ovtcharova, Fraunhofer Institut für Graphische Datenverarbeitung, Darmstadt, Germany.

[IGES] The Initial Graphics Exchange Specification, Version 5.1, National Computer Graphics Association, 1991.

[IMP] IMPPACT, ESPRIT Project 2165, Commision of the European Communities.

[IPK] Prof. F.L. Krause, Fraunhofer Institut für Produktionsanlagen und Konstruktionstecknik (IPK), Berlin, Germany.

[ISO] International Organisation for Standardization, Geneva, Switzerland.

[ISO1101] ISO 1101:1983 Technical Drawings – Geometric tolerancing – Tolerancing of form, orientation, location and run-out – Generalities, definitions, sysmbols, indications on drawings. International Organisation for Standardization.

[ISO129] ISO 129:1985 Technical Drawings – Dimensioning – General principles, definitions, methods of execution and special indications. International Organisation for Standardization.

[ISO2692] ISO 2692:1988 Technical Drawings – Geometrical Tolerancing – Maximum material principle. International Organisation for Standardization.

[ISO286] ISO 286-1 1988 ISO system of limits and fits. International Organisation for Standardization.

[ISO406] ISO 406:1987 Technical Drawings – Tolerancing of linear and angular dimensions. International Organisation for Standardization.

[ISO5459] ISO 5459:1981 Technical Drawings – Geometric Tolerancing – Datums and datum-systems for geometrical tolerances. International Organisation for Standardization.

[ISO8015] ISO 8015:1985 Technical Drawings - Fundamental Tolerancing Priciple. International Organisation for Standardization.

[ITH] Ithaca Software, 1301 Marina Village Parkway, Alameda, CA 94501, USA.

[JAR89] Jared G., "The Feature Recognition Battle - Latest From The Front", New Tools For Shape Modelling, The British Computer Society, 1989.

[JOS87] Joshi S., Chang T.C., "Graph-based Heuristics For Recognition Of Machinied Features From A 3D Solid Model", CAD, March, 1987.

[KRA90] Krause F.-L., Ulbrich A., Vosgerau F.H., "Feature Based Approach For The Integration Of Design And Process Planning Systems", Proc. 23rd Int. Symp. on Automotive Technology and Automation, 1990.

[KRA91] Krause F.L., Kramer S., Rieger E., "PDGL: A Language for Efficient Feature-Based Product Gestaltung", CIRP Annals, 1991, Vol 40 No 1, pp135-138

[LKC] Prof. W. Weber, Lehrstuhl für Konstruktionstechnik/CAD, Universität des Saarlandes, Germany.

[LUB86] Luby S.C., Dixon J.R., Simmons M.K., "Designing with Features: Creating and Using a Features Data Base for Evaluation of Manufacturability", Proceedings, International Computers in Engineering Conference, ASME, Vol. 1, 1986, pp285-292.

[MAN] Mandelli SpA, 35 Via Caorsana, 29100 Piacenza, Italy.

[MAR84] March L.J., "The Logic of Design", in Cross N., "Developments in Design Methodology", Wiley, 1984.

[MAR93] Martinsen K., "Vectorial tolerancing for all types of surface", Proc. 15th ASME Design Automation Conf., Vol 2, Albuquerque, USA.

[MIL91] Mill F., Naish J., Pedley A.G., Salmon J., "Feature Oriented Design: Review Report No. 2", Dept. of Mechanical Engineering, University of Edinburgh, 1991.

[MIL93] Mill F.G., Pedley A.G., Salmon J.C., "Representation Problems in Feature Based Design and Process Planning", Int. J. Computer Integrated Manufacturing, 1993, Vol. 6, Nos. 1 & 2. pp27-33

[MIL94] Mill F.G., Naish J.C., Salmon J.C., "Design for machining with a simultaneous-engineering workstation", Computer Aided Design, Vol.26, No.7, July 1994.

[MMI93] "Man Machine Interface – User Requirements Document", Brite/Euram Project 4539: SESAME, Commission of the European Communities, Brussels. 1993.

[MPG] Mr. F. Mill, Manufacturing Planning Group, University of Edinburgh, Scotland.

[OGR96] O'Grady P., "Concurrent Engineering Systems", Keynote paper, 12th Int. Conf. on Computer Aided Production Engineering, Cookville, TN, USA, 1996.

[PAH84] Pahl G., Beitz W., "Engineering Design", Design Council, London, 1984.

[PARAM] Parametric Technology Corporation, 128 Technology Drive, Waltham, MA 02154, USA.

[PDES] Kelly J.C., "The Product Data Exchange Standard (PDES)", Federal Computer Conf., USA, 1985.

[PED89] Pedley A.G., "Feature Oriented Design - the Missing Link", 5th Int. Conf on Computer Aided Production Engineering, Edinburgh, 1989.

[PED93a] Pedley A.G., "Man Machine Interface Software Requirements Document", Brite/Euram Project 4539: SESAME, Commission of the European Communities, Brussels. 1993.

[PED93b] Pedley A.G., "Feature Based Design System Software Requirements Document", Brite/Euram Project 4539: SESAME, Commission of the European Communities, Brussels. 1993.

[PED94] Pedley A.G., "SESAME Design Feature Classification and Implementation", Technical Report, Brite/Euram Project 4539, SESAME.

[PED94b] Pedley A.G., "Describing Face, Edge And Point Related Data Within CODL", Brite/Euram Project 4539: SESAME, Commission of the European Communities, Brussels. 1994.

[PED94b] Pedley A.G., "Features And Their Face Numbers", Brite/Euram Project 4539: SESAME, Commission of the European Communities, Brussels. 1994.

[PED94b] Pedley A.G., "Minutes Of Technical Meeting To Discuss Describing Face, Edge And Point Related Data Within CODL", Brite/Euram Project 4539: SESAME, Commission of the European Communities, Brussels. 1994.

[PED95] Pedley A.G., Ehrmann M., "Explicit Feature Interaction Modelling in CAD and CAM Systems", 11th Int. Conf. Computer-Aided Production Engineering, IMECHE, London, 1995, pp269-275

[PED96a] Pedley A.G., Buchwald, Denzel H., "User Defined Features: A Means to Advanced Process Integration", Proc. of 29th International Symposium on Automotive Technology and Automation" Florence, Italy, 3-6 June 1996, pp125-133.

[PED96b] Pedley A.G., "The Potential to Exchange Feature Models With User Defined Feature Libraries", J. of Materials Processing Technology, Vol. 61, Nos. 1-2, August 1996, pp78-84.

[PIE96] Pierce S., Rosen D., "Free-Form Surface Modelling as a Tool for the Analysis and Selection of Assembly Tolerances", Proc. 29th Int. Symp. on Automotive Technology and Automation", Florence, 1996, pp 35-42.

[PRA85] Pratt M.J., Wilson P., "Requirements for Support of Form Features in a Solid Modelling System", CAM-I Report R-85-ASPP-01, Arlington Texas, USA, 1985.

[PRA88] Pratt M.J., "Synthesis of an Optimal Approach to Form Feature Modelling", Proc. Computers in Engineering Conf., 1988, pp 263-274, ASME.

[PRO] Probst T., "Entwichlerhandbuch NLS", V1.0, Straessle Informationssysteme AG, Kanalstrasse 33, 8152 Glattbrugg, Switzerland.

[PTC] Parametric Technology Corporation, 128 Technology Drive, Waltham MA 02154, USA.

[RAN88] Ranyak P.S., Fridshal R., "Features for Tolerancing a Solid Model", ASME Computers in Engineering San Francisco, 1988. pp275-280

[REQ83] Requicha A.A.G., "Representation of Tolerances in Solid Modelling: Issues and Alternative Approaches", Proc. Solid Modelling by Computers, Michigan, 1983. pp3-22

[REQ86] Requicha A.A.G., Chan S.C., "Representation of geometric featutes, tolerances and attributes in solid modellers based on constructive solids geometry", IEEE J. Robot. & Automat. Vol. RA-2 No. 3, 1986. pp156-165

[REQ89] Requicha A.A.G., Vandenbrande J.H., "Form Features for Mechanical Design and Manufacturing" ASME Computers in Engineering, 1989.

[ROV] Rover Group Ltd, Gaydon Test Centre, Warwick, England.

[ROY88] Roy U., Liu C.R., "Feature based representational scheme of a solid modeler for providing dimensioning and tolerancing information", Robot. & Comput. Integr. Manuf., Vol. 4 Nos. 3/4, 1988. pp 335-345

[RUD92] Rudolf F.N., "Konfigurierbare Technische Elemente Für Konstruktion Und Arbeitsplannung", PhD Thesis, Hanover University, 1992.

[SAB87] Sabin M.A., "A research programme in CAE", Computer Aided Engineering Journal, 1987, Vol 4 No 2, pp79-82

[SAL93] Salomands O.W., Houten F.J., Kals H.J.J., "Review of Research in Feature Based Design", Jounal of Manufacturing Systems, Vol.12, No.2, 1993, pp 113-132.

[SAL94] Salmon J.C., Rieken H.R., "Component Description Language (CODL) Implementation", Internal Report 18, Manufacturing Planning Group, Department of Mechanical Engineering, The University of Edinburgh, July, 1994.

[SAL95] Salomons O., "Constraint Specification and Satisfaction in Feature Based Design for Manufacturing", PhD Thesis, University of Twente, Holland.

[SDRC] Structural Dynamics Research Association, USA.

[SHA88a] Shah J.J., Rodgers M.T., "Feature Based Modelling Shell: Design and Implementation", Computers in Manufacturing, San Francisco, 1988.

[SHA88b] Shah J.J., Rodgers M.T., "Functional requirements and conceptual design of the feature based modelling system", ASME Computers in Engineering, San Francisco, 1988.

[SHA88c] Shah J.J., Rodgers M.T., "Expert form feture modeling shell", Computer Aided Design, 1988, Vol. 20 No. 9, pp515-524

[SHA88d] Shah, J.J., Bhatnagar A., Hsiao D., "Feature Mapping and Application Shell", Computers in Engineering, ASME, 1988.

[SHA90] Shah, J.J., "An assessment of features technology", CAM-I, P-90-PM-02, 1990.

[SHA93] Sha J.J., Hsiao D., Leonard J., "A Systematic Approach For Design Manufacturing Feature Mapping", in Wilson P.R., Pratt M.J., Wozny M.J., "Geometric Modelling For Product Realization", Elsevier, 1993.

[SHA95] Shah J.J., Mäntylä M., "Parametric and Feature Based CAD / CAM", Wiley, 1995.

[SHP94] Shpitalni M., Fischer A., "Separation Of Disconnected Machining Regions On The Basis Of A CSG Model", CAD, Vol.26, No.1, Jan 1994, pp46-58.

[SIM95] Simons B., "Manufacturing and Tolerance Features Development", ACIS Open Meeting, 1995, Chicago, USA.

[SIO96] Siow S.C., "Feature Oriented CAD/CAM", Dept. of Mechnical Engineering, The University of Edinburgh, 1996.

[SIS] Straessle Informationssysteme AG, Kanalstrasse 33, 8152 Glattbrugg, Switzerland.

[SIS95a] Straessle Informationssysteme AG, "MCL+ V5.2 Benutzerhandbuch", Kanalstrasse 33, 8152 Glattbrugg, Switzerland, 1995.

[SIS95b] Straessle Informationssysteme AG, "FeatureM V5.2 User's Manual, Kanalstrasse 33, 8152 Glattbrugg, Switzerland, 1995.

[SOL92] Sohlenius G. "Concurrent Engineering", Annals of the CIRP, Vol 41 nr 2, 1992. pp645-655

[SOL93] Solomons O.W., van Houten F.J.A.M., Kals H.J.J., "Review of Research in Feature-Based Design", J. of Manufacturing Systems, Vol. 12 No. 2, 1993, pp113-132.

[SRE91] Sreevalsan P.C., Shah J.J., "Unification of Form Feature Methods", IFIP Workshop on Intelligent CAD, Columbus, Ohio, 1991.

[STEP] ISO 10303 Industrial Automation Systems - Product Data Representation And Exchange. International Organisation for Standardization.

[STEP11] ISO 10303-11 Industrial Automation Systems - Product Data Representation And Exchange - Part 11: Description Methods: The EXPRESS Language Reference Manual. International Organisation for Standardization.

[STEP21] ISO 10303-21 Industrial Automation Systems - Product Data Representation And Exchange - Part 21: Clear Text Encoding of the Exchange Structure. International Organisation for Standardization.

[STEP224] ISO 10303-224 Industrial Automation Systems - Product Data Representation And Exchange - Part 224: Application Protocols: Mechanical Product Definition For Process Planning Using Form Features. International Organisation for Standardization.

[STEP41] ISO 10303-41 Industrial Automation Systems - Product Data Representation And Exchange - Part 41: Integrated Generic Resources: Fundamentals of Product Description and Support. International Organisation for Standardization.

[STEP42] ISO 10303-42 Industrial Automation Systems - Product Data Representation And Exchange - Part 42: Integrated Generic Resources: Geometric and Topological Representation. International Organisation for Standardization.

[STEP47] ISO 10303-47 Industrial Automation Systems - Product Data Representation And Exchange - Part 47: Integrated Generic Resources: Shape Variational Tolerances. International Organisation for Standardization.

[STEP48]. ISO 10303-48 Industrial Automation Systems - Product Data Representation And Exchange - Part 48: Integrated Generic Resources: Form Features. International Organisation for Standardization.

[STI] Spatial Technology Inc., 2425 55th Street, Building A, Boulder, Colorado, USA.

[SUT63] Sutherland I.E., "Sketchpad: A Man-Machine Graphical Communication System", Spring Joint Computer Conf., Spartan, Baltimore, USA, 1963.

[TAI93] Taiber, J.G., "Computer-Aided Rule-based Dimensioning and Tolerancing in 3D-Modelling", Institute of Machine Tools and Manufacturing, Swiss Federal Institute of Technology, ETH Zurich, Switzerland.

[TAI96] Taiber, J.G., "Entwurf eines feature- und constraint-basierten CAD/CAM-Systems zur fertigungstechnischen Optimierung", PhD Thesis, Institute of Machine Tools and Manufacturing, Swiss Federal Institute of Technology, ETH Zurich, Switzerland, 1996.

[TAN95] Tani P.-L., "Wünschen Für Komplexierte Geometrie Erzeugung In FeatureM", Strässle Informationssysteme AG, Zürich, Switzerland, 1995.

[TOE94a] Toenshoff H.K., Baum T., Ehrmann M., "SESAME - A System for Simultaneous Engineering", Proc 4th Int. FAIM Conference on Flexible Automation and Integrated Manufacturing, Blacksburg, Virginia, 1994, pp380-389

[TOE94b] Toenshoff H.K., Aurich J., Baum T. "Configurable Feature-Based CAD/CAPP System", Proc. IFIP Int. Conf. on Feature Modelling and Recognition in Advanced CAD/CAM Systems, Valenciennes, France, 1994, pp757-769

[TOE94c] Toenshoff H.K., Aurich J., Hamelmann S., "Formale Elementbeschreibung fuer Konstruction and Arbeitsplanung", VDI-Z, Bd. 135, Nr. 11/12, 1994, pp.113-116

[TOE96] Toenshoff H.K., Baum T., Ehrmann M., "SESAME: Simultaneous Engineering System for Applications in Mechanical Engineering", in Life Cycle Modelling for Innovative Products and Processes, Krause F.-L., Jansen H., Chapman Hall, 1996, ISBN 0 412 75590 4, pp515-526

[USC] Prof. A.G. Requicha, Computer Science and Engineering, University of Southern California, Los Angeles, USA

[VDA] VDAFS: "VDA Surface Data Interface", Verband der Automobilindustrie e.V. (VDA), Frankfurt am Main, Germany, Version 2, 1987.

[VDI2221] VDI 2221: "Systematic Approach to the Design of Technical Systems and Products", Verein Deutscher Ingenieure, Germany.

[VOE77] Voelcker H.B., Requicha A.A.G., "Geometric Modelling of Mechanical Parts and Processes", Computer, Vol.10, No.48, 1977.

[VSM91] "Normen Anzug Für Technische Schulen", Vereins Schweizerischer Maschinen-Industrieller, Normenbüro, Zürich, Switzerland, 1991.

[WEI88] Weill R., "Integrating dimensioning and tolerancing in Computer Aided Process Planning", Robotics & Computer Integrated Manufacturing, Vol. 4, Nos. 1/2, 1988, pp 41-48.

[WIN91] Wingard L., "Introducing form features in product models, a step towards cadcam with engineering terminology", Thesis, Royal Institute of Technology, Stockholm, Sweden.

[WMG] Mr. R. Bobrowski, Warwick Manufacturing Group, University of Warwick, England.

[WOO82] Woo T.C., "Feature Extraction By Volume Decomposition", Conf. on CAD/CAM Technology in Mechanical Engineering, MIT., 1982.

[XWI] "The X Window System Series - Vols. 0, 1, 2, 3, 4, 5", O'Reilly & Associates, Inc., Sebastopol, CA, USA.

[ZEI91] Zeid I., "CAD/CAM Theory and Practice", McGraw-Hill, 1991.

Appendix A

Published Papers

The author has published seven papers related to the investigations reported in the thesis. The papers may be found from the following references:

1. Pedley A.G., "Feature Oriented Design - the Missing Link", 5th Int. Conf. on Computer Aided Production Engineering, Edinburgh, 1989. Addendum.

2. Husbands P., Mill F.G., Pedley A.G., "HAPPI", 6th International Conference on Computer Aided Production Engineering, London, November 1990, pp1-6.

3. Husbands P., Mill F., Pedley A.G., Warrington S., "The Edinburgh Composite Component", Proc. 5th Int. Conf. on Manufacturing Science and Technology of the Future, Enschede, 1991.

4. Mill F.G., Pedley A.G., Salmon J.C., "Representation Problems in Feature Based Design and Process Planning", Int. J. Computer Integrated Manufacturing, 1993, Vol. 6, Nos. 1 & 2. pp27-33

5. Pedley A.G., Ehrmann M., "Explicit Feature Interaction Modelling in CAD and CAM Systems", 11th Int. Conf. Computer-Aided Production Engineering, IMECHE, London, 1995, pp269-275

6. Pedley A.G., Buchwald, Denzel H., "User Defined Features: A Means to Advanced Process Integration", Proc. of 29th International Symposium on Automotive Technology and Automation" Florence, Italy, 3-6 June 1996, pp125-133.

7. Pedley A.G., "The Potential to Exchange Feature Models With User Defined Feature Libraries", J. of Materials Processing Technology, Vol. 61, Nos. 1-2, August 1996, pp78-84.