TITLE FEATURE BASED COMPUTER AIDED PROCESS PLANNING

AUTHOR Eur Ing Dilip PATIL

DEGREE Ph.D

AWARDING BODY Warwick University

DATE 1995

THESIS NUMBER DX201885

p186
missing

Feature Based Computer Aided Process Planning

by

Eur Ing Dilip Patil

BE (Mech), MTech (Production Science & Technology), MIE,

MIMechE, CEng.

A thesis submitted for the Degree of

Doctor of Philosophy

THE·UNIVERSITY·OF
WARWICK

Department Of Engineering

University Of Warwick

April 1995

SUMMARY

This research attempts to study, plan and develop an integrated feature based CAPP system that generates automated process plans for machining prismatic components. The CAPP system comprises a STEP compliant feature based commercial CAD system and the Smalltalk object oriented system. A library of features has been developed that is based on STEP based form feature taxonomy but modified to communicate the manufacturing intent and feature aggregation. The CAPP system has been developed to represent the product, process and resource domain knowledge with a number of object hierarchies, communication methods, and the user interface that would suit the concurrent engineering needs. In addition, suitable geometric and process reasoning methods have been developed in the CAPP system that use the feature based component design data to generate automated process plans.

The research also attempts to identify the problems in feature based process planning and discusses the possible solutions. A solution to the side feature interaction problem has been implemented in the CAPP system.

The CAPP system test results have demonstrated that the proposed approach has been successful and has a great potential for further improvements in terms of flexibility, modularity, emerging data exchange standards, and ease in customising the CAPP system.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

ACKNOWLEDGEMENTS

I would like to extend my gratitude to my supervisor Mr John Hill, for his guidance and help. Many thanks to my friends and colleagues within the Warwick Manufacturing Group, especially to Mr Neil Davis of Simulation Group for providing the Smalltalk object oriented programming software. Without his help this non funded part time research would not have been able to achieve its complete goal.

DECLARATIONS

I declare that the work presented herein is my own work and has not been submitted to another institution for another award.

1. SCOPE AND OBJECTIVES OF THE RESEARCH

1.1 Introduction

Global competition has led manufacturers to search for new ways to reduce the time-to-market, and product costs while improving quality. The search has led to the basic design, manufacturing and management concepts to be used throughout the product life-cycle. It has been realised, that these concepts should be reviewed, with manufacturers adopting a concurrent engineering strategy, as opposed to a traditional serial engineering approach. It is a strategy whereby all members of the product engineering team, from design to production and marketing, work from a common product master model. CAD/CAM has become the core technology in this strategy.

A key to the success of integrated product development is the free sharing of product data between design and manufacturing. The exchange of product model data between different CAD/CAM systems is rather tedious because of the differences in the internal geometry representation, and the ambiguous definitions of the exchange standards. In response to this need, major IT suppliers and users have collaborated to develop an industry-standard called STEP (STandard for Exchange of Product model data). The STEP standard is aimed at common specifications for communicating product information at all stages of the product life-cycle.

Process planning can be seen as an activity which integrates knowledge about products and resources. It refers to the product design, and decides how to manufacture it within the resource constraints present. Recognising its importance, considerable research efforts have been made for the past two decades to automate process planning by Computer Aided Process Planning (CAPP) methods.

The research efforts have tried to use various feature methodologies as a medium of communication of the design representation to manufacturing. A broad definition of a feature in the engineering domain is given by Pratt and Wilson [66] as: 'a region of interest on the surface of a part'. The broad definition is necessary to encompass the variety of items, that engineers consider to be features of a part, at various stages in the design and manufacturing applications.

The same feature could have a different meaning within different applications. For example, a bearing support is a functional feature in the conceptual design phase which may express the function and not the shape. In the later design phase, the bearing support could be described with a geometry of a cylindrical surface. When a geometry of a feature is considered, it is usually called a form feature [67, 84]. Finally, in the process planning phase, the bearing support form is viewed as a drilled or bored hole which may be referred to as a manufacturing feature.

The functional features associated with textual information without any geometry are of little use in the automation of downstream applications, such as process planning and NC programming. However, form features are suitable for such applications. In many cases the form feature will also be a manufacturing feature. For example, a cylindrical surface geometry of a form feature in the above discussion will also be a manufacturing feature (a hole). Thus, form features may be used as a medium of communication of features from design to manufacturing environment.

The design interpretation in CAPP research consisted of group technology principles that were used to code the components depending upon their features. Later, various feature recognition techniques were used to read the component CAD model and convert it into a manufacturing feature representation. Most recently, feature based CAD modelling tools are being used.

The feature based CAD representation is desirable as it eliminates the feature recognition step in design interpretation. The STEP committee has recognised this and formulated a geometric form feature standard. The standard gained immediate support on major CAD/CAM systems such as Computervision, ProEngineer and SDRC, etc.

This thesis presents an overview of new product design techniques in the context of concurrent engineering and STEP. It focuses on the detailed study of STEP feature based CAD modelling techniques in their current state. Based upon the study, the thesis proposes a CAPP system, formulated to demonstrate its effectiveness, along with various results.

1.2 Overview of Process Planning

As mentioned above, process planning is an activity that deals with how to make a desired object. It is usually done by a manufacturing engineer who has the necessary knowledge and experience in methods engineering. The activity involves systematic planning to produce various features on a raw state object to achieve the desired finished state object. The result of this activity is a process plan, a detailed report of the systematic procedures about how to produce the desired object.

The desired object is the finished state. It is designed to fulfil specific functional requirements. The designer uses different geometric shapes, also referred to as form features, or smaller objects to describe the shape of the object. These features are positioned, oriented and blended in relation to each other, to represent an object shape. The designer also describes the acceptable variations, called tolerances, in feature shapes, sizes and their relationships. In addition the designer specifies the information such as material type, surface quality etc., to complete the object description. The total description of an object is called the object design. The overall product design consists of a hierarchy, of one or more assemblies, sub-assemblies or objects/components. The product design produces a bill of materials that describes the component material and the total quantity required to produce a single product assembly.

The product design and the bill of materials is referred to by the manufacturing engineer to plan how to produce a product. The basic decisions on how to produce depend on the economics of manufacturing.

1.3 The Current CAPP Systems

A manual process planning approach relies on the knowledge and experience of a manufacturing engineer who can plan the process or an alternative process depending upon the availability of resources. The manual approach can produce inconsistent processes, and could be slow in response to today's core technology demands in manufacturing. The expert knowledge disappears almost instantly if the engineer leaves the organisation.

Thus, the ultimate aims of a CAPP system are alternative processes, instant and precise manufacturing cost estimates for policy decisions, quick response to changes, consistency in processes and preserving the expert knowledge within the organisation. The CAPP system should prove to be a very effective tool in testing the product design for manufacturability in a concurrent engineering environment.

Considerable research efforts have been put into the development of Computer Aided Process Planning (CAPP) systems, and over 200 systems have been reported so far. Although few have been made commercially available, a majority of them are found to be prototypes in research institutions. Even commercial systems require investments of many man years to tailor the process planning system to a specific company.

1.4 The Success of CAPP Research

The majority of research output has gone into upgrading product engineering tools and reviewing standards. Initial efforts of the research community into the usage of Group Technology (GT) and feature recognition in CAPP, has contributed to the STEP form feature standard for product design.

The difficulties in representing planning knowledge representation have been recognised, and new methods suggested such as artificial intelligence techniques.

Efficient product data management tools such as Engineering Data Management have been recommended to promote integrated product development.

1.5 Scope for Research in CAPP

In spite of some achievements, the fact remains that CAPP has not been commercially successful so far. Although the core technologies available today provide a strong basis, for example, concurrent assembly design and feature based design, it has become necessary to use these tools for the development of CAPP systems and to assess their effectiveness.

The design by feature approach has been criticised for hindering the designer's creativity. A detailed study of this approach is needed.

Conventional programming languages are not found to be suitable for knowledge representation. Knowledge based software tools such as Object Oriented Programming, are regarded as more effective means in this area. New methodologies are needed to convert massive amounts of manufacturing information into useful knowledge, and to utilise this knowledge to make better decisions.

Database management techniques have been recognised as a vital tool for effective resource management. Relational and Object Oriented databases are the current tools widely used. However their effectiveness in responding to the needs of the planning systems needs investigation.

Computerising manufacturing planning needs vast amounts of information from various domains. However, typical research projects in this field tend to be very focused, e.g. CAPP for round components, CAPP for prismatic components, etc. There is a scope for experimentation with flexible and add on software tools like Object Oriented Programming.

The CAPP research recognises and addresses the need for integration of product, process and resource technology. However, the consistent use of an integrated approach in CAPP development tends to be missing.

Today's core technologies are available from different vendors. For example, feature based modelling systems are available from Pro/ENGINEER or Computervision, a knowledge based CAD system from ICAD, an Object Oriented Programming System from Smalltalk, a relational database management system from Oracle, etc. It is obvious that a single system from any vendor may not be able to achieve total integration. Hence, CAPP research should be pursued on the basis of standard information exchange between such systems, for example, a STEP standard in Feature Based Modelling, Materials and machinability database standard, etc.

1.6 The Objectives of the Research

The objective of this research is to study and develop a feature based Computer Aided Process Planning system. A metal cutting manufacturing environment for prismatic components is used as an application of process planning.

A concurrent engineering core technology such as CADDS5 assembly design and feature based design, which supports the STEP standard, is used for product design. A detailed study of the standard feature library and use of user defined feature techniques to communicate manufacturing intent have been identified to be the key issues of study.

A feature based model interpretation needs geometric reasoning techniques to identify the manufacturing view of features that is based on feature relationships, feature attributes and the raw material condition. A study of suitable techniques for geometric reasoning remains a secondary objective.

Integration of product knowledge with process and resource knowledge needs careful study. A relational database, Oracle, C programming language and an Objected

Oriented Programming language, Smalltalk, have been available as computer based tools to develop the system. Since different systems are involved in the development, the information from various standards will be used whenever possible.

This work is concentrated on the following sub-objectives:

1. A study of the STEP form feature information model, which consists of feature hierarchy for design representation.

2. Recognition of additional information to be associated with the standard features to communicate the manufacturing intent.

3. A study of various feature based modelling strategies to suggest methods so that the designer's creativity remains unaffected.

4. To study and suggest new methods in geometric reasoning techniques.

5. A study of various knowledge representation schemes for the development of CAPP system.

6. To develop communication methods to manipulate product, process and resource information to generate process plans.

1.7 Justification of the Approach

The research proposes an integrated approach for CAPP by proposing product, process and resource knowledge representation schemes and communication strategies. A product design environment has been considered on the basis of current CAD core technology such as assembly design and feature based design, supporting the STEP standard.

An Object Oriented environment is recognised as an effective tool for process knowledge representation. Its effectiveness needs experimentation in representing, manipulating, adding and modifying process knowledge.

The STEP committee has initiated a STEP part 224 application protocol which deals with mechanical product definition for process planning using machining features. It is intended to specify the requirements for the representation and exchange of information needed to define product data necessary for manufacturing single piece mechanical parts. The product data is based on the existing part designs that have their shapes represented by form features. The details about part 224 are not available as it is the topic of current research. Any research initiative in this direction will be in line with the international research efforts.

The relational databases such as Oracle have become industry standards in resource management. A detailed database design approach would facilitate comparing database entities with objects. Therefore a study of database design and its relationship with objects should still keep the emphasis of research on standards.

An automated process plan should be possible for those components which are designed by feature based approach. The 'process modelling approach' can be suggested to associate the process information manually but efficiently through a communication interface for those components which are designed by conventional approach.

Because of the free data exchange of product, process and resource information, the approach should facilitate concurrent engineering.

1.8 The Thesis Structure

This chapter has provided the preface to the work carried out over four years of research. It identifies the subject background, the availability of new core technology, the STEP developments, the limitations of techniques used in CAPP research and emphasises the needs of an integrated approach and indicates the objectives of the research.

The second chapter provides a critical review of the various tools and techniques used in the development of variant, generative, and knowledge-based computer aided process planning systems. It also discusses in brief the STEP standard development so far.

The third chapter discusses the concurrent product modelling and design by feature concept. It also identifies the manufacturing attributes to be associated with the feature definitions. Case studies provide the measure of effectiveness of design by feature approach in practical applications.

The fourth chapter discusses the initial CAPP system plan and its shortcomings and a proposed new system plan.

The fifth chapter discusses the concept of an Object Oriented Programming environment. It presents the details of CAPP system development that consist of product, process and resource domain knowledge representation; integrated user interface development; geometric and process reasoning techniques and process plan report generation.

The sixth chapter illustrates the CAPP results with case studies.

The seventh chapter identifies the problems in feature based process planning. It discusses the attempts of various researchers to solve them and also elaborates on the probable solutions.

The eighth chapter covers the implementation procedure for the solution to a side feature interaction problem and discusses the test results.

The ninth chapter reviews the work and findings of the research with recommendations for further work.

The tenth chapter concludes the report with the main investigative findings.

2. CAPP: TOOLS AND TECHNIQUES

With rapid developments in information technology over the past two decades, the tools and techniques used by researchers in computerising process planning have changed significantly. The process planning tasks have evolved from data driven types of variant approaches to knowledge driven types of generative approaches [1]. The new developments of computer software paradigms, such as artificial intelligence (AI) techniques, have changed the traditional way of using computers as data storage and information machines.

While the new tools and techniques are being tried, it is essential to review the various techniques used so that the goal of the research and implementation remains on target.

2.1 Process Planning in Manufacturing

The purpose of process planning is to select and define systematically, in detail, the processes that have to be performed in order to transform raw material into a desired shape. The primary objective is to define feasible processes. Cost and throughput are secondary objectives and available resources (e.g. cutting tools, machine tools, labour) act as constraints [2]. Process Planning includes: interpretation of part design data, determination of production tolerances, selection of machining processes, selection of machining operations and their sequence, selection of machine tools, selection of cutting tools, design of jigs and fixtures, selection of cutting conditions, selection of inspection devices, calculation of overall times, determination of tool paths and NC program generation. All the decisions made at the process planning stage are limited to a specific part only.

Process planning in manufacturing is usually conceived with the determination of processes in a metal cutting environment. Very few research efforts have been made in computerising process planning for other environments, such as foundry, injection moulding etc. It may be due to the fact that these environments need their own production process designs to produce tools. The production process will remain the same throughout the life of a tool, and therefore he process can easily be associated with the production process design as 'process layout'.

Metal cutting environments however are different. There could be different alternative processes and resources to produce the same object. Significant economical benefits could be obtained if optimum production concepts were used.

2.2 The CAPP Approaches

Two approaches to computer aided process planning are traditionally recognised, the variant approach and the generative approach. Many CAPP systems do not exactly fit into this classification and combine both approaches, that are usually referred to as a semi-generative approach.

Few papers have been published on a general survey of CAPP Systems. The surveys were based on questionnaires [3], the state-of-the-art [4] and on the specific techniques, such as CAD/Feature based [5]. They also classified the CAPP systems according to the basic approaches used.

The basic CAPP approaches can be discussed in brief as follows:

2.2.1 The Variant Approach

The variant approach is comparable to the traditional manual approach where a process plan for a new component is created by recalling, identifying, and retrieving an existing plan for a similar part and making the necessary modifications for the new part.

In the variant CAPP system, parts are grouped into part families, a unique code is generated for each family, and a standard process plan is initially developed for each family. Most systems use well defined Group Technology based coding and classification systems to develop unique codes for various part families. This type of process planning system was first used to develop the code for a new part, to show which part family it belongs to, and then for retrieving and editing the standard process plan to reflect the characteristics of the new part.

The first CAPP system, CAPP (actual system name) was developed by the variant approach in 1976 under the direction and supervision of CAM-I(Computer Aided Manufacturing International) [87]. Another system, MIPLAN was developed by OIR (Organisation of Industrial Research)[39, 88]. Since then many variant systems have been reported.

The variant approach to process planning seems more feasible when the product design is fairly stable, lot size is medium to high, parts size variations within a family are minimal, material type is the same for all the members of the family, and few engineering changes are normally made. The investment is less and the development time is shorter, and thus it is feasible for medium sized companies to establish their own classification system.

There are some major limitations of the variant CAPP systems. It is difficult to maintain consistency in editing practice. A skilled process planner is still required to edit and generate new process plans. It is difficult to adequately accommodate various combinations of geometry, size, precision, material and quality. Significant on-line databases are required to accommodate stored plans and their modifications.

2.2.2 The Generative Approach

In the generative approach, the process plans are generated by means of decision logic, formulae, technology algorithms, and geometry based data [4].

There have been many attempts made to develop generative CAPP systems with traditional techniques [6,7,8,9,10].

With the introduction of AI techniques, many generative CAPP systems were developed as expert or knowledge based process planning systems [11,12,13,14].

The main difference in traditional and AI techniques is that in the former the algorithms dictate a deterministic procedure to solve a problem, whereas in AI, one searches for a solution in a state space that describes all configurations of the relevant objects [5]. The search is guided by rules and control strategies. If the direction of the search is from an initial state to the goal state, it is called forward chaining, and if it is from the goal state to the initial state, it is called backward chaining.

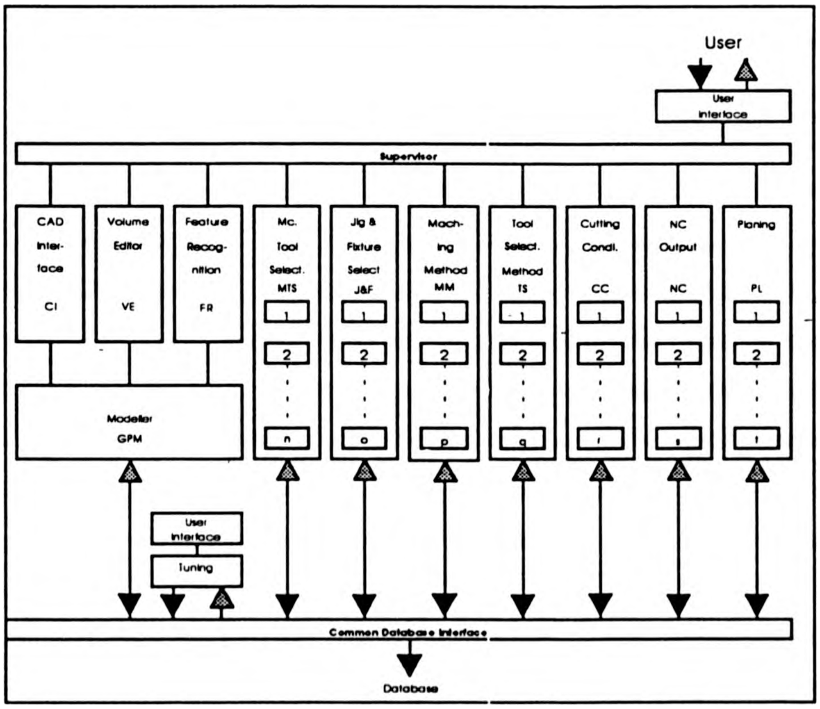


Figure 2.1: Schematic Representation of the PART System
[Ref.: PART, Hauten, Erve, Booijert, Nauta, Kals]

It is worth mentioning the significant and continuous research efforts in generative CAPP at the University of Twente in the Netherlands. The development efforts of ROUND [14], CUBIC and XPLANE [15] systems over the past decade have been incorporated into a new generative CAPP system, PART (Planning of Activities, Resources and Technology)[16, 17, 18, 19].

The PART system decomposes a B-rep solid model (Figure 2.4) of a component into manufacturing features by a feature recognition technique. The system has a modular structure. The basic technological functions of PART are grouped into six modules namely, MTS, J&F, MM, TS, CC and NC (Figure 2.1). The system is based on AI techniques. However, no expert system shell or typical AI programming languages are used. The knowledge based selection mechanisms are embedded in the procedural structure of the PART system. Interactive graphics are used to support the maintenance of a knowledge base. The inference engine is not fixed as in many expert systems, but the strategy and selection mechanisms can be specified and altered.

In order to achieve the required flexibility in sequencing the many decision steps according to a specific strategy, the PART system is implemented as a set of individually executable processes. A PART module comprises a group of processes called "phases". The phases communicate with each other via a common relational database. The PART system selects the set-ups, machining methods and their sequences, it calculates optimised tool paths and cutting conditions and generates NC programs.

The generative approach is naturally desirable as the process plans can be generated easily, rapidly and consistently. However, the development of a generative CAPP system is very complex task, and requires a long term investment. For example, the PART System project was sponsored by the Dutch Ministry of Economic Affairs, with a total budget of $3 million and twelve full time researchers working for four years from 1987 to 1991 [16].

2.2.3 The Semi-Generative Approach

This is a combination of generative and variant approaches. It offers several options, one of which is to make suitable changes to the standard process plan as in the variant approach. The second option is to begin with an incomplete process plan and complete it for a specific part. The third is to start from the beginning and completely create a new process plan using the various process descriptions logically stored in the computer as in the generative approach.

GENPLAN [20] from Lockheed is an example of such a system. Many commercial CAPP systems can also be classified into this category. Since it is a practically oriented system, the semi-generative approach may remain a popular choice until practical generative systems are commercially available.

2.3 Commercial CAPP Systems

A market review shows that commercial CAPP systems are being encouraged by the industry [21]. PC based systems such as CIMPLAN [22] and CEEQUEL [23] are being used by small to medium size machining, assembly and fabrication firms. These are variant CAPP systems within the price range of £3000 - £5000 and claim to have a customer base of over 50 each.

The workstation based system such as C-PLAN from CadCentre, UK [24], can be classified as a semi-generative CAPP system. It supports five principle functions; user interactive process planning function, report generation and link to MRP, a variant based search function, standard data entry and system administration functions.

LOCAM from Pafec, UK [25, 26], is also a semi-generative CAPP system. It supports five levels of process planning functions; traditional, interactive, features, semi-automatic and automatic. It supports Oracle relational database and customised interfaces to CAD, NC and DNC.

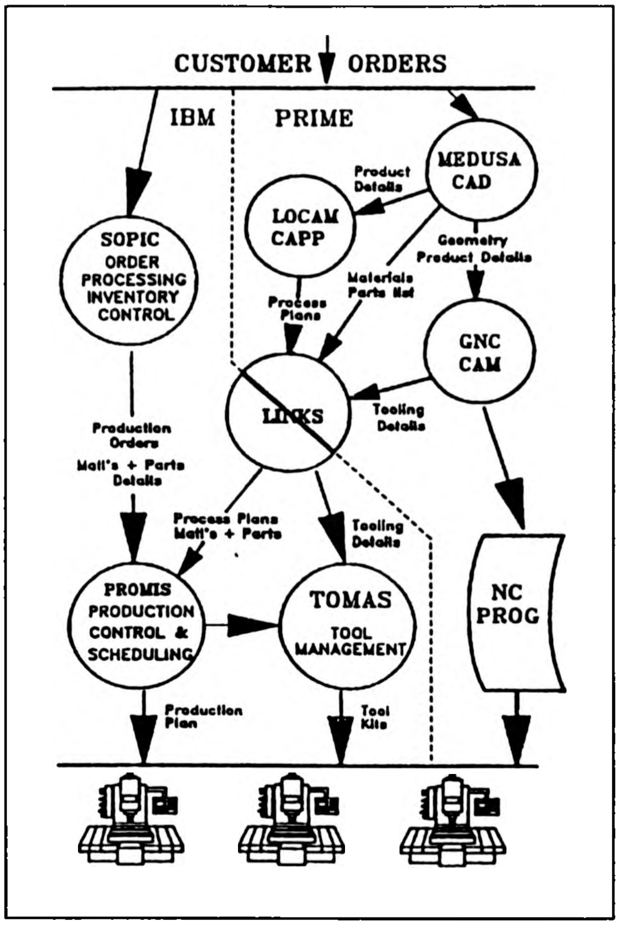


Figure 2.2: Sandvik COROCIM System
[Ref.: Hargreaves, 'CIM, a Users Experience']

Sandvik, UK has successfully achieved a CIM environment by using LOCAM through customised interfaces [27]. The system is popularly known as COROCIM System (Figure 2.2). It involved the integration of Prime Medusa CAD, LOCAM CAPP and GNC Computer Aided NC part programming. These systems were also linked to the SOPHIC order processing and inventory control and PROMIS production control and scheduling system. The system claimed to have achieved upto

50% reductions in lead times on typical product families. Significant benefits were achieved by increasing the utilisation of high-cost CNC machine tools. This was performed by reducing the set-up times and by using a Tool Management system, THOMAS, to control the preparation of tools, jigs, fixtures and gauges.

SUPERCAPES from SD-Scicon, UK [28], can be classified as a semi-generative CAPP system. It runs on HP9000 workstations and consists of a core module plus three main modules; machining, general purpose for non machining and costing.

The machining module produces process plans for all types of metal cutting machines including NC.

The general purpose module consists of non machining processes such as, fabrication, welding, rolling, marking out, mechanical and electrical assembly fitting, etc. It also consists of motion sequence macros covering assembly processes. The Motion Sequence Macro (MSM) covers human motion activities based on MTM (Methods Time Measurement) or any other Predetermined Motion Time System (PMTS). The sets for these standard elements covering commonly occurring human movements and activities have been supported. Thus the planner can build up plans for detailed assembly/repair operations without going down to fine elemental detail.

The costing module computes cost estimates based on labour, overhead rates and material costs from the SUPERCAPES database.

Supplementary modules are available for archiving, MRP transfer, project supervision, MICLASS coding and classification, and base data. The data is held in structured hierarchical form.

SUPERCAPES has a user base of over 50 companies including Jaguar Cars and GEC Alsthom, UK; Texas Instruments, USA; and Nikon and NEC Japan.

HMS-CAPP from Houtzeel Manufacturing systems, USA [29] is a variant based CAPP system. It uses CIMLINK's LINKAGE product to seamlessly connect this manufacturing information handling system to relational databases, CAD files, information files and other design and manufacturing software products such as MRP, time standards, tool control and shop floor control. HMS also uses the LINKAGE product to take advantage of graphical user interface, WYSIWYG, multiple RDBMS interfaces, and report generation capability. It also supports an advanced Group Technology system which has been developed recently for the US Air Force. It proposes three phases of installations. Phase one of eight weeks, for installing the process planning system into an open system client server environment. Phase two of twelve weeks, for review of representative samples of customers' drawings and process plans to customise Group Technology Classification. Phase three is of 10 to 25 weeks, for analysis of families of parts using the GT analyser and the establishment of the manufacturing rule based system, for frequently occurring families of parts to generate process plans semi-automatically.

MetCAPP from the Institute of Advanced Manufacturing Sciences (IAMS), USA (formerly, MetCut Research) [30], represents a new generation of commercial CAPP systems. It is a PC based system, works under DOS 5.0 and Microsoft Windows 3.1 and can be implemented on a PC Network. It supports a relational database option to provide variant process planning capabilities. It is based on an industry-standard machining database. It provides planning capabilities at three levels:

1. Part level via the CUTPLAN process planning module. It consists of a GT module which describes the part attributes with a series of fields. These attributes enable a search of a particular plan.
2. Feature level via the CUTTECH 'Expert Machining' module.
3. Cut level via CUTDATA machining recommendations module.

MetCAPP supports interfaces to CAD, CAM, MRP, document management and report writing system. An Application Program Interface (API) is also included for custom development.

MetCAPP is being implemented with a CIM project at the Singapore Polytechnic, with technical support from Warwick Manufacturing Group, University Of Warwick.

The U.S. Navy logistics project RAMP (Rapid Acquisition of Manufactured Parts) uses the MetCAPP interface as a user defined process and resource knowledge and forms a complimentary component of the GPPE (Generative Process Planning Environment) system [31].

Conventionally programmed CAPP systems have been developed by individual companies for their own work planning. SISPA, SIB from Siemens, ROVA from Daimler Benz, are some examples [32]. However, because of coded planning logic, they cannot be easily updated or maintained.

A considerable amount of time has to be spent in customising commercial systems as the systems do not have efficient tools to implement changes. For example, at Rockwell International, USA, a GT based CAPP system was implemented within four weeks. After a while, out of the system's 36 programs, 7 were no longer supported, 6 were rewritten and 20 were rarely used [33].

2.4 CAPP System Development Techniques

Development of a fully functional CAPP system has been regarded as a very complex task. It is necessary to consider all the important issues in design, manufacturing planning, and resources management together, and with rapid developments in computer technology, new tools and techniques are always giving a new challenge to the research community. An overview of various techniques used in computerising process planning is presented below.

2.5 Design Representation Techniques

The traditional way to communicate the design to manufacturing, is the engineering drawing. The designer's description of an object as a set of features positioned and blended together, merely becomes a geometry on the drawing sheet. It means that the intelligence in the design information is lost, while putting it on the drawing sheet. The manufacturing engineer reads the drawing and uses his 'intelligence' to understand the design.

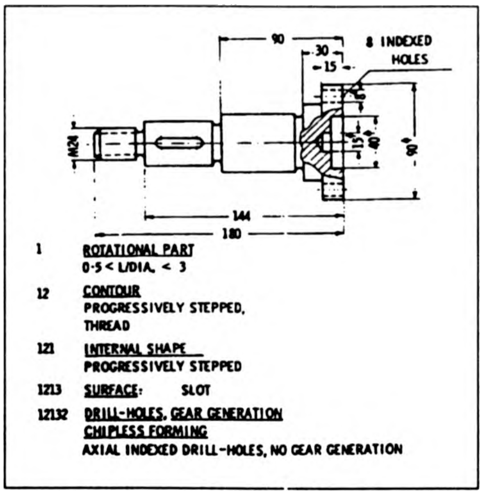
The early efforts in capturing the design intelligence are quite apparent in using Group Technology (GT) techniques.

2.5.1 Group Technology

GT seeks to exploit the similarities between parts or products for efficiencies in design and manufacturing. Besides visual inspection and Production Flow Analysis (PFA), the most widely acceptable method for implementing GT is classification and coding (CC) which organises similar entities into groups (classification) and then assigns a symbolic code to these entities (coding) in order to facilitate information retrieval [37].

There are three types of CC systems based on design attributes, manufacturing attributes and a combination of design and manufacturing attributes. The Overall shape, external shape, internal shape, length/diameter ratio, surface finish, special features (such as thread, hole pattern, etc.), dimension tolerance and raw material type and shape are some of the design attributes commonly seen. Major process, minor operations, operation sequence, machine tool, cutting tools, production time, batch size, annual production and required fixture, are some of the manufacturing attributes used.

The types of coding system have different structures. These are, the hierarchical (monocode), chain-type (polycode) and hybrid structure (combination of hierarchical and chain-type). In hierarchical structures, the interpretation of each succeeding symbol depends on the value of the proceeding symbols whereas in chain-type, the interpretation of each symbol in the sequence is fixed, and does not depend on the value of preceding symbol. Most of the commercial systems use hybrid structure [35].



1	<u>ROTATIONAL PART</u> 0·5 < L/DIA. < 3
12	<u>CONTOUR</u> PROGRESSIVELY STEPPED, THREAD
121	<u>INTERNAL SHAPE</u> PROGRESSIVELY STEPPED
1213	<u>SURFACE</u>: SLOT
12132	<u>DRILL-HOLES, GEAR GENERATION CHIPLESS FORMING</u> AXIAL INDEXED DRILL-HOLES, NO GEAR GENERATION

Figure 2.3: Coding of a Round Part
[Ref.: Opitz, 'GT and Mfg Systems']

One of the pioneering efforts in GT was from H. Opitz from University of Aachen in West Germany. In Opitz's system the basic code consisted of five digits and are referred to as a form code [40]. These digits are used to describe a component class, external shape, internal shape, surface machining and auxiliary holes/gear teeth respectively. A supplementary code of four digits is used to describe diameter or length value, material, raw material shape and accuracy in coding digits. Figure 2.3 illustrates the Opitz coding method for the round component

The MICLASS (Metaalinstitute Classification) System became a commercial success throughout Europe and North America [34]. MICLASS was commercially offered by TNO, the Netherlands organisation for applied scientific research [87]. The same system was later upgraded as Multiclass II, combining the GT and ORACLE

Relational Data Base technology, and became part of MultiCapp II, a variant CAPP system from OIR, USA [39]. The MICLASS CC number can range from 12 to 30 digits. The first 12 digits are a universal code that can be applied to any part. 18 supplementary digits can be used to code data that are specific to the particular company, such as lot size, cost data, etc. The system also supports the user interactive mode in various languages. To classify a given part, the user responds to a series of questions. A survey showed, that the time taken to generate a CC number for a new part varied from 10 minutes to 2 hours, depending on the complexity of a part [36, 37].

The other popular CC systems are : Brisch Birn, CODE from Manufacturing Data Systems, Inc. (MDSI), Michigan, USA, DCLASS from Bringham Young University, USA [50], CimTelligence [53], Deere Tech Services and General Electric; all from the USA. Custom built CC system are also being used [36].

Expert systems techniques have also been used to develop the CC system. A predicate logic was used to develop a CC system for round and prismatic parts at the Kobe University, Japan [38].

Recently, GTA (Group Technology Assistant), an intelligent, Object Oriented PC based system built with PDES compatible taxonomy has been developed by the U.S. Air Force, and is commercially available. Its main features include graphic user interface (GUI), classification using graphics and industry terminology, commercial standards from industry handbooks, and an object oriented shell, that validates instructions and issues design and process advice [34].

The major success of GT has been in the area of design retrieval where all previous drawings or models can be indexed using GT code. The designer, can select the existing designs as examples for making a new design, thereby eliminating the time for groundwork. It also contributes immensely towards design standardisation. A survey of 56 US companies revealed that 62% of them used GT concepts in the creation and retrieval of process plans [36]. The current research in CAPP shows that various GT techniques are being pursued. It has been observed that the traditional process type layout of a factory no longer exist [41]. Instead, by applying the principles of GT, PFA, and various advances in flexible production technology such as FMS, industrial engineers have in many companies created a new type of manufacturing system layout, that effectively combines flexibility and high capacity utilisation and short turnaround time. Such a manufacturing layout has been named as the workshop oriented factory [42]. Since early 1990, the Brite-Euram project no. BE-3528, Manufacturing Cell Operator's Expert System (MCOES) has been involved in the design and implementation of a prototype level process planning system for workshop oriented factory.

GT coding schemes cannot be used as accurate design representation methods. It is difficult to describe complex parts with coding digit combinations. The CC system needs customising efforts to code the special type of parts. Upgrading the system involves significant efforts. There are cases wheie the old CC systems were totally scrapped before installing a new one [36, 37]. Training and coding strategies become an important issue, which otherwise may cause proliferation of parts and manufacturing methods.

2.5.2 Computer Aided Design (CAD)

Using computers, the methods of design description have changed significantly over the past two decades. Initially, computer aided drafting was introduced. It was only an electronic or digital drafting board with major benefits in editing the drawings and improving productivity in the design department. With rapid developments in computer technology, it became possible to represent an object on the computer screen as it looks in the real world. The process of such representation is called three dimensional (3D) modelling. Model geometry representation methods have gone through significant changes, from wire frame (skeleton) to surfaces and solids.

The use of solid modelling in design and manufacturing is increasing rapidly because of reduced computing costs, fast computing hardware, and increased capabilities of solid modelling [43]. It is relatively easy to model the part by solid modelling techniques. The complete definition of part shape (geometry and topology) through solid models has been recognised as a key to integrating design and manufacturing functions.

There are several types of Solid data representations. The most widely used on commercial CAD/CAM systems are Boundary Representation (B-Rep) and Constructive Solid Geometry (CSG).

If information is added to the surface model, for example, concerning surface connectivity between surfaces (faces), and in addition the solid side of any face is identified, then this forms the elements of the boundary representation (B-rep) scheme [44]. Such information is referred to as topology (Figure 2.4).

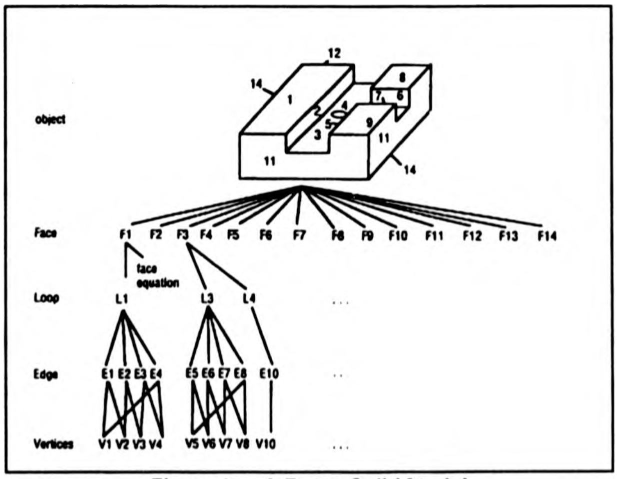

object
Face
Loop
Edge
Vertices
F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12 F13 F14
face equation
L1 L3 L4
E1 E2 E3 E4 E5 E6 E7 E8 E10
V1 V2 V3 V4 V5 V6 V7 V8 V10

Figure 2.4: A B-rep Solid Model
[Ref.: T. C. Chang, 'Expert process planning']

In practical systems the topological consistency is achieved by using a data structure in which faces are linked with edges, which are in turn linked to their bounding vertices (end points). Since on a face there may be more than one loop of edges, it is common to add another entity, a loop in the B-rep [45]. The B-rep models store information about the faces and edges of a model explicitly, in what is known as an evaluated form. This confers performance advantage on the method, because the information for certain applications can be extracted directly from the data structure. Such applications include hidden surfaces removed object, interactive NC part programming, surface area calculations, etc. A relatively large data storage space is required for B-rep models.

In CSG representation, the object Oi is represented by a binary tree consisting of geometrical primitives Pi, transformations Ti and symbols representing boolean operators (Figure 2.5). Leaf nodes can either be a primitive or a primitive with transformation. Intermediate nodes can either be a boolean operator or an element. Primitives used in CSG modeller can either be a standard primitive such as cylinder, sphere, box, etc. or user defined by sweeping a closed 2D profile [45].

The CSG models are compact, but may be stored in an unevaluated form in which the edges and faces that result from the combination of primitives have to be computed when required. The CSG method enables one to build the complex shapes quickly, but only within the limitations of the set of primitives available within the system. Many features found on the engineering component such as fillets, blend, draft to allow the component to withdraw from the mould or die, may be difficult or time consuming to produce [44]. The CSG model is not, in general, a unique representation of an object. The model may be constructed in various ways.

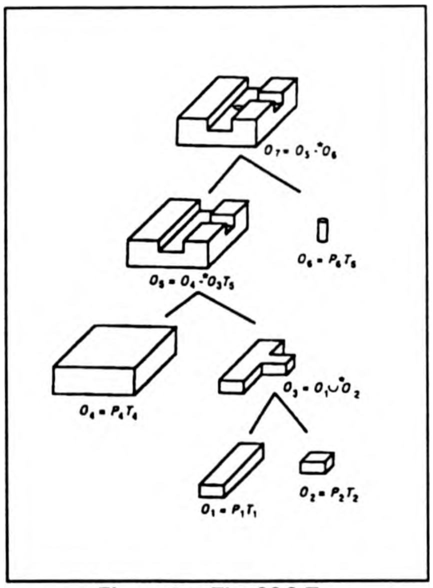

$O_7 = O_5 \overset{*}{-} O_6$
$O_6 = P_6T_6$
$O_5 = O_4 \overset{*}{-} O_3T_5$
$O_4 = P_4T_4$
$O_3 = O_1 \overset{*}{\cup} O_2$
$O_1 = P_1T_1$
$O_2 = P_2T_2$

Figure 2.5: The CSG Tree
[Ref.: T.C.Chang, 'Expert process planning']

Because of advantages and disadvantages of B-rep and CSG representations many practical systems offer the hybrid approach [43]. Usually a B-rep model is constructed using the CSG style boolean operator front end. The model is then evaluated into a B-rep model. The non-unique CSG models can actually be evaluated into unique B-rep. This may be a good reason for using a B-rep model in feature recognition technique [45].

Parametric solid modelling is the latest technique available on commercial CAD/CAM systems. Instead of describing the dimensional attributes of a modelled geometry by numbers, they are described by parameters/variables. The modelling process is recorded as a parametric history. It also captures the geometric relationship between entities [46]. This enables ease in model modification by changing the parameters defining a model and re-executing the parametric history.

The parametric solid modelling environment forms a basis for feature based modelling described later. The implicit form feature representation is a parametric description of a geometry [47]. The form feature instances can represent the same shape with different geometric attributes/parameters. The explicit description of the feature instance geometry can be obtained by evaluating the model.

2.5.3 Computer Aided Manufacturing (CAM) from CAD

The use of CAD wire-frame, surface or solid geometry to generate Numerical Control (NC) or Computer Numerical Control (CNC) part programs, has contributed significant benefits in terms of quality and productivity of manufacturing parts [48]. The NC/CNC programming and Direct Numerical Control (DNC) functionality was typically referred to as CAM. DNC enables distribution of part programs through standard communication protocol to various NC/CNC machine tool networks on the shop floor.

The majority of CAD vendors provide user interactive NC programming functionality, and not the automated ones. This is mainly due to the fact that NC programming follows after the process planning function, or is done simultaneously. The process planning has received significant attention because it is the link between CAD and CAM [49, 51]. A high degree of automation was achieved by individual users through customising the NC programming environment [62].

A seamless interface between CAD, highly customised NC and DNC has contributed to the realisation of the Computer Integrated Manufacturing (CIM) environment. NC machine tools have also contributed to the concept of Flexible Manufacturing Systems (FMS) in small batch manufacturing. A typical FMS cell consists of tightly coupled NC machine tools with automatic material handling, tool handling and transport devices.

In order to develop the missing link, CAPP researchers tried to use feature recognition techniques on the CAD model data to recognise manufacturing features.

2.5.4 Feature Recognition from CAD Model

In feature recognition, a low level data structure of a CAD model is searched automatically or interactively for a combination of geometric elements, that corrospond to a feature form [52].

Several methods have been used to recognise features automatically from a CAD model [54]. These are,

a. Syntactic pattern recognition

b. State transition diagrams and automata

c. Decomposition approach

d. CSG approach

e. Graph based approach

A Syntactic pattern recognition method is most suitable for 2D pattern recognition [54]. In this method, a 2D picture is represented by some semantic primitives, such as line segments in a particular orientation. A set of grammars define a particular feature pattern. A parser is then used to apply the grammar to the picture. If the syntax of the picture matches the grammar, then the picture can be classified as belonging to a particular pattern class.

STOPP, a sequential tool oriented process planning system [57, 58] used the Syntactic pattern recognition approach on a 3D B-rep Solid model. 2½D machining features such as hole, slot, pocket and step were considered as machined surfaces. The 3D B-rep data was converted to 2½D edge representation. Various pattern matching procedures were written in PASCAL, and were then used to recognise the machining features. When a feature was recognised from the B-rep model, it was eliminated from B-rep data and stored in another feature data structure, called a machining surface file (MS file). XPLAN [15] and PART [16] systems have borrowed the same procedure to recognise features.

The state transition and automata method is very similar to the syntactic pattern recognition. The part geometry is described as a rotational swept volume and/or the union of swept volumes. The machining features are recognised by using a state transition diagram (automata), where instead of using grammars and primitives, the relationship between adjacent primitives is used. The convex adjacencies are assigned a value '0' and the concave adjacencies as '1'.

Milacic [64] discussed the use of the automata for part family formation and used it in developing the system for automatic planning technology (SAPT). The grammars for individual parts that will comprise the family are first developed. From these grammars the grammar for composite parts representing the complete family of parts is created and is used as an automata. The feature string to be recognised is fed to the automata and a feature class is recognised.

The decomposition approach uses the Destructive Solid Geometry (DSG) concept. Taking a workpiece solid model W and component solid model S, the material to be removed (DSG) by machining can be obtained as E = W - S. E is then cut into several small pieces / machining features by various strategies. By using a CSG tree, a DSG tree can be constructed by subtractive operation only. A primitive in a DSG tree can be viewed as a machining feature.

In the CSG approach, the features are extracted from the CSG solid model [60]. The CSG model is first converted into a DSG model. Li [60] used a pattern matching algorithm to identify the primitives (features) in the DSG model. The main problem with the extraction of features from CSG models is that CSG descriptions are not unique and so it is possible to have a number of different CSG strings all of which describe the same object. The LUMP (Loughborough University Manufacturing Package) system used rules to control rewriting CSG strings from design system into manufacturing features. Each rule contains a parameterised version of the CSG template which it tries to match with the whole CSG string. The matched features are then substituted for the corresponding segments of the CSG string. Multiple templates are used if multiple possibilities exist in building a CSG model, thereby overcoming the non unique CSG model problem to some extent.

In the graph based approach, a feature recognition logic is represented by rules. The rules are based on certain patterns of elements and relationships until some set of elements can be identified as a feature. For example [52],

```
IF
  a hole entrance face exists, and
  the face adjacent to the entrance is cylindrical, and
  the cylindrical face is concave, and
  the next adjacent face is a plane adjacent only to the cylinder
THEN
  the cylinder face, and planer face comprise a blind hole.
```

By using this technique, a vertex edge type feature graph can be extracted from the complete graph representations of the topological entities (edges, faces and vertices) in the B-rep solid model and used as a feature model [59].

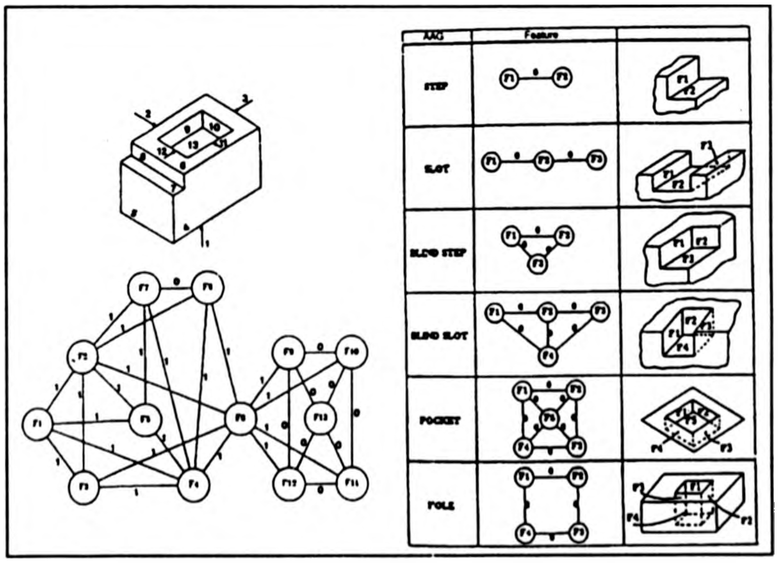


Figure 2.6: AAG for Sample Part & AAG for Feature Instances [Ref.: S. Joshi (54)]

The attributed adjacency graph based approach [54, 55, 56] for feature recognition uses the B-rep solid model. A B-rep model of an object can be defined as B = (V, E, F) where V is a set of vertices, E is a set of edges and F is a set of faces. The attributed adjacency graph (AAG) can be defined as G = (N, A, T) where N is a set of nodes, A is a set of arcs and T is a set of attributes to arcs in A. Each face of the part is represented as a node and each edge or face adjacency is shown as an arc. The attribute assigned is '1' if the two adjacent faces form a convex angle, and '0' if the angle is concave (Figure 2.6). The rules can then be written to recognise the feature graphs from the AAG of the complete graph of the model.

The ICAPP system at UMIST used feature recognition from wire-frame models of 2½D prismatic components, in IGES format [89]. The profiles at different depths are used to construct the hierarchy of depth levels. The space between the profile on the odd numbered depth level, and the profile on the even numbered depth level is the 2D space to be machined. The cross sections at different depths containing features are established, and the area of each cross section is subtracted from the next higher level layer to produce a volume to be machined.

In the user interactive mode of feature recognition the user interacts with the system by digitising a set of entities of a CAD model which imply a feature [62, 63]. The interactive graphics programming language of a CAD system with a suitable interface program interprets the selected entities as a feature such as a hole, a pocket, etc. This method is suitable for describing simple 2½D objects. However, one has to describe the object twice i.e. through CAD design and process planning stage. Any change to the CAD model will require another system interaction to describe the features.

The feature recognition techniques discussed above recognise manufacturing features by assuming the raw material as a billet or a solid block. The approach has been successful in producing a manufacturer's view, independent of the designer's view of a CAD model. Using feature recognition, it would be irrelevant if the model was constructed using design features, manufacturing features, or without using features at all [16].

2.5.5 Limitations of Feature Recognition Methods

When a designer starts a design process, most of the features are already known and in a process of creating a CAD model the feature information is lost. Feature recognition and extraction is a redundant effort which would be unnecessary if a method could be devised for retaining the design information in the CAD model data. The majority of feature recognition techniques are successful in a limited domain (2½D objects). Feature based design on the other hand, aims to supply the designer with a tool that can be used, while carrying out the design work, i.e. an integration tool for CAD and CAPP [65, 66].

2.5.6 Design By Feature

In the design by feature approach, the designer is provided with a feature library, similar to the primitives of a CSG solid modeller, that can be used with a set of operators such as add, delete and modify to create a feature representation. The feature representation maintains additional information such as feature instance name and attributes, that are not kept in a conventional solid modeller and this eliminates the need for feature recognition [65].

Since the design by feature involves using a library of features, the feature taxonomy schemes have been suggested to help the designer to access the feature data, and the manufacturing engineer to generate process plans for a group of features, that have some common geometric, topological and other properties. The feature taxonomy scheme is a way of classifying features into a hierarchy of classes and subclasses, depending upon the common attributes. Since the taxonomy is of a hierarchical nature, the attributes of a class can be inherited by its sub-class.

Pratt and Wilson [66] classified form feature representation into two types, implicit and explicit features. In the explicit type, all the geometry of a feature is fully described. The implicit type can be defined by a set of geometric parameters/attributes, but the full geometric form is evaluated when required. Features were further classified as simple and compound types. A compound feature is a combination of more elementary features.

Hummel and Brooks [63] also discussed feature taxonomy in their automated process planning system, XCUT and used the object oriented programming technique for implementation.

Gindy [68] presented a taxonomy where form features were classified as protrusions, depressions and surfaces. The form features were treated as volumes enveloped by entry/exit and depth boundaries. The feature classification is based on the external access direction (EAD), from which the feature volume could be machined by a cutting tool. A feature is 'through' if there is a pair of opposing external access directions (EADs). Gindy's scheme is specifically aimed at the needs of process planning in a machining environment for a prototype CAPP system GENPLAN [69, 70].

Shah & Rojers [71] introduced a feature based modelling system, which consists of an advanced feature based modelling shell, FMDS. It classified features into:

a. form features (nominal geometry)

b. material features (material composition & condition)

c. precision features (tolerances)

d. technological features (design constraints, etc.)

From the above classification, three modellers were developed; form, material and precision. Form features are used as the fundamental representations, and the other types can be attached to the form features. Each modeller works in two modes, set-up and modelling. The set-up mode can be used for customisation by the organisation using it to define the features needed by their designers. The object oriented programming approach to representing feature descriptions leads to property lists stored in a database. The solid representation of form features is stored as a feature producing volume (CSG sub-tree) and boolean operators. Realising that the interpretation of feature data is application dependent, the feature mapping shell is provided for the purpose of extracting and reformulating product data as needed by the applications. The information to be extracted is classified into three types:

Type 1: the parameters/attributes which are feature explicit. These are independent of other data in the model (model-invariant).

Type 2: the kind of parameters/attributes which can be calculated from those, that are explicit in feature data of a feature but model-invariant.

Type 3: the parameters/attributes which are dependent between two or more features.

Type 1 parameters can be obtained through simple GET functions. Type 2 and 3 require computations and logical operations on type 1 parameters/attributes. A language has been built which can be used for specific extraction procedures.

Pratt and Wilson [66] also studied the data handling in geometric modeller and application program environments. A much more intimate relationship between the feature processor and the modeller was advocated by suggesting a feature model database for the exchange of information with the geometric model database.

Chang's Quick Turnaround Cell [45], a generative CAPP system for one off prismatic parts, uses a feature based design system linked to, but not dependent on, the TWIN B-rep based solid modeller. After a design is completed, the design data is sent to the process planning system as a feature file. The feature refinement module in the process planner analyses the design model and further classifies design features into manufacturing features. Only the difference operator is used in the design system. Therefore, each feature corresponds to a cavity volume to be removed by machining.

The prototype feature based design system in the domain of the investment casting process was proposed by Chung and Simmons [73]. It consists of the solid modeller (I-DEAS from SDRC) and KEE rule based reasoning system. The user interacts with KEE through the user interface whilst I-DEAS runs in the background to build the model. The KEE represents the feature based design in the form of frame and slots. Objects of the application domain are represented in a hierarchical structure. The same system was used in aluminium casting processes by using macro and co-features [74]. Macro features are the classes of geometric forms such as boxes, U-Channels, L-brackets and slabs. Co-features are attachments of design details, such as holes, bosses, etc.

Dixon [75] proposed an intelligent CAD system based on these applications. However, the feature manipulation was limited, and common feature operations such as subtraction were not supported. Dixon's intelligent CAD system consists of two parts. The first part includes a user interface, a design with feature library, an operations library and a monitor, which allows the user to create primary representations of features. The features used by the designers were available in the design feature library. The operation library contained add, modify and delete options. The monitor ensured that the operations requested and performed by the user, were allowable and understandable to the system. The primary representation of an object was produced by the first part of the system. The second part of the system was used for converting the primary representations of the objects into secondary representations needed by respective activities, such as design and manufacturing activities. The conversion may involve some feature extraction to obtain more abstract features. A knowledge based system was used to assist in the design tasks. Visual display was also included and further design suggestions could be given.

The IWF in Berlin has been developing a feature modelling system, which is a part of an integrated system for product and process modelling [76]. The project is a part of a large European research project IMPPACT (Integrated Modelling of Products and Processes using Advanced Computer Technologies) within the ESPRIT program [77]. A Part Design Graph Language (PDGL) has been developed to describe features in an object oriented manner, including their technological semantics. PDGL uses the syntax of the STEP data modelling language, EXPRESS. The features are described in textual form and are called as feature templates. These textual features are stored in a library. When the user alters the parameters of the feature by taking the actual design into account, it is referred to as a generic feature. The feature modeller has a bi-directional interface with hybrid solid modeller. The feature modeller and geometric modeller each maintain their own databases. There is no standard feature library, but it can be constructed when required to meet the needs of a particular application. PDGL allows the creation of new user defined features.

An attributed node-label-controlled graph grammar (ANLCGG) for the FEAT-REP (FEAture-REPresentation) language has also been developed at the German Research Centre for Artificial Intelligence [78] A frame-like language describes features firstly by their kind, such as functional, qualitative (e.g. bars, solid workpiece), form, and atomic (shape tolerance, surface finish); and secondly by their application such as design features, manufacturing features, etc.

Research by the Japan Society of Precision Engineering [79] proposed a concept of structural modelling. It consists of a graph structure relating features to describe a product; a set of simple, special and compound features; and the geometric primitives to describe the shape attribute of the features.

Faux's CAM-I report summarises the design by feature methodologies with the dimension and tolerance models associated with feature models [80]. A primitive is a set of one or more faces which are treated as a single indivisible unit. The primitives are used for tolerance definitions which are classified as shape (size and form) and location (position and orientation).

Attempts have been made to represent dimensions & tolerances (D&T) by features. Roy and Liu [81] proposed a D&T model on the feature extraction principles. CAM-I also attempted a dimensioning and tolerancing model through features and solid model representation [82]. Desai & Pande [83] proposed a geometric feature modeller (GFM) for rotational components with D&T capability.

2.5.7 Product Modelling with Concurrent Engineering

The term concurrent engineering was coined in the US in 1989 [85]. It is the way of working where various engineering activities in product and production development processes are integrated and performed as much as possible in parallel rather than in sequence. Such a concept has been supported on a new generation of CAD/CAM systems such as Computervision, IBM CATIA, ProEngineer, etc. It is now possible to describe a product as a combination of assembly, subassembly and components, and develop concurrently by working as a team on a network of CAD/CAM systems. These systems are also supported by efficient engineering data management (EDM) tools.

Large scale product modelling firms, such as aerospace and automotive companies have been implementing concurrent engineering concepts. The systems have also been found to be very effective in the design modification process. They also bring in a cultural change in the way of functioning of an organisation, especially in working as a team and co-ordination of groups [86].

2.5.8 Knowledge Based CAD Modelling

The conventional CAD systems are considered to concentrate on providing means for representing the final form of a design. A design also includes a large amount of synthesis, analysis and decision making. Rule based systems reason with synthesis rules and generate proposals for solutic ns. A system for design analysis is based on constraint satisfaction. Constraint satisfaction can be performed by a truth maintenance system. The branch of AI for design applications, has been able to provide the design representation schemes to represent the above information in addition to CAD modelling. Such systems are known as knowledge based CAD system. They are aimed to assist the designer in design automation, design analysis, and to capture the design expertise [44].

The systems like ICAD [90] (An Intelligent Computer Aided Design System) fall into this category. A design can be described by the hierarchical structure in the form of components and their attributes (which include geometric representation), sub-assemblies and assemblies. The designer can use taxonomies and rules. Various constraints can be specified as a part of the product modelling process.

However, the ICAD system user interface is not very user friendly. This is mainly due to the way a design is described. The geometry is regarded as one of the attributes of the design and described by a number of primitive functions. To a new user this may appear like constructing a program. A conventional CAD user does not easily accept such a 'long-winded' approach. To achieve good skill level needs a considerable amount of training. The 'Design Expert' has to build a complete design domain for the end user [91].

A knowledge based CAD system could prove very effective in specific design domains such as tool design, design optimisation, etc. The US Navy logistics program, RAMP (The Rapid Acquisition of Manufacturing Parts) used features based models from ICAD as an input for the development of GPPE system [31].

2.5.9 CAD/CAD Data Exchange

There are a number of CAD systems in the market supporting different combinations of hardware and software. The CAD data storage structure varies from one system to another. Each system has its own speciality in terms of price and performance. In today's changing business environment, subcontracting is becoming important to stay competitive. Hence, the usage of different systems at various levels of product development/manufacturing is always a possibility. Under such circumstances, a great deal of product design data has to be exchanged between various systems.

The first effort in data exchange was initiated by major US CAD vendors by establishing the first version of Initial Graphics Exchange Specification (IGES) in 1979 [44]. It employed the format of an ASCII file capable of being exchanged between any two CAD systems. It marginally supported the exchange of 2D drawings [91].

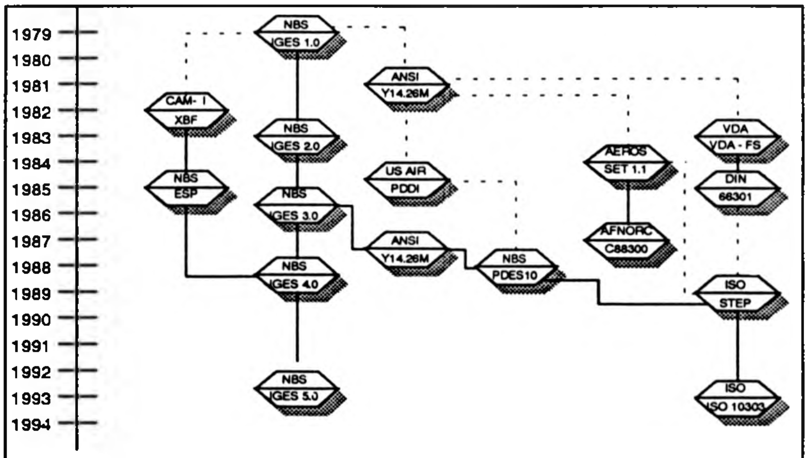
The format chosen for IGES produced long ASCII files. The early implementations of IGES translators by CAD system vendors also tended to be unreliable in part, because of vagueness in the specification and partial implementation by vendors. Such difficulties prompted different standards.

The German automotive industry developed a VDA/FS standard to overcome the limitation of IGES in exchanging higher degree surfaces.

SET standard was developed by the members of the European Airbus consortium. The standard was based on a similar format to IGES, but was more compact.

The development of IGES continued in the US. IGES revision 2.0 included entities from finite element and electrical system applications. The development of the standard for exchange of solid modelling data resulted in the Experimental Boundary File (XBF), which was eventually merged into IGES Experimental Solid Proposal (ESP). IGES revision 4.0 included facilities for the exchange of CSG solid models.

The transfer of B-rep models was included in IGES revision 5.0. The chronological details of various data exchange standard developments, are shown in Figure 2.7.



Organisations

NBS:	National Bureau of Standards
ANS:	American National Standards Institution
CAM - I:	Computer Aided Manufacturing - International
VDA:	Verband der Automobilindustrie (Germany)
AEROS:	Aerosapitale (France)
DIN:	Deutsches Institut fur Normung
ISO:	International Organisation for Standardization

Standards

IGES:	Initial Graphics Exchange Specification
XBF:	Experimental Boundary File
VDA - FS:	VDA Flachenschnittstelle
SET:	Standard D'Echange et de Transfert
ESP:	Experimental Solids Prosposal
PDDI:	Product Data Definition Interface
PDES:	Product Data Exchange Specifications
STEP:	STandard for the Exchange of Product model data

Figure 2.7: CAD Data Exchange Standards Development [Ref.: Dr. G. Warnecke (150)]

In order to merge various data exchange standards and develop an agreed international standard, the IGES organisation initiated PDES (Product Definition Exchange Specifications) in 1984. The European Community also funded a large project under the ESPRIT programme, called Project 322 - CAD Interfaces. These various projects, and the associated work in the area, have been drawn together into a single unified standard called STandard for the Exchange of Product model data (STEP). Although in US the PDES acronym has been retained for various reasons it now stands for Product Definition Exchange using STEP [44].

2.5.10 STEP/ISO 10303

STEP is the ISO standard for the exchange of product model data [92]. The long term aim of STEP is to support the product data exchange requirements of many industry sectors.

The IGES standard was based on the syntax of the exchange representation; whereas the STEP standard specifies the syntax and the semantics, i.e. the precise meaning of all data constructs.

STEP is based around information models described by a data modelling language called EXPRESS. The data description in EXPRESS is mapped to the physical file. The physical file does not need to have a definition of how, for example, a point should be represented. The EXPRESS language embodies techniques adopted from both relational and object oriented database development.

The STEP models are divided into two main elements - *application models*, which contain information related to a particular application, such as draughting or electrical product modelling and *resource models*, which provide facilities to the application models, for example, the description of a raw geometry or topology. The resource models that are being developed include geometry, topology, solids, features, tolerances and materials [44].

The development of the STEP standard is being undertaken by a number of committees, and working groups that deal with different aspects of the standard, which are divided into a number of separate standards. For example, Part 1 [94] is the overview, Part 2 [95] the EXPRESS language and Part 21 the physical file. Parts in the 40 series, involve the resource model, and the 100 series involves the application models. Associated with the application models are the application protocols in the 200 series, which describe the subset of the total STEP entities, which are relevant to particular applications, and also the constraints on application data and ways of conformance testing the data transfer. For example, AP 203 is an application protocol for configuration controlled design; AP 204 for Solid Design; AP 214 for Automotive Design; AP 224 for product definitions for process planning using form features, etc. [96]. The standard also provides a methodology and framework for the conformance testing of the implementations.

The standard is not complete yet. Because of the wide scope of STEP and because of the large number of people involved, the standardisation process is rather slow.

2.5.11 EXPRESS and Object Oriented Language

EXPRESS is similar to ordinary programming languages, but with the support of object oriented concepts. The main descriptive elements of EXPRESS are [95]:

- **Schema**
  - **Type (character of data used to describe entity)**
  - **Entity (describes the object of interest)**
  - **Algorithm**
    - **Function**
    - **Procedure**
  - **Rule (define constraints)**

Schema is a collection of information sets used by a discipline to describe a product data model. *Type* describes the attribute characteristics of the entities. *Entity* is designed to represent a set of information (The EXPRESS entity is analogous to a class in an object-oriented program). Entities are designed to have attributes and inheritance relationships with other entities. Entity sub_types and super_types are also supported. *Algorithm* is a sequence of statements that produces some desired end state. The body of the algorithm follows the local declarations. *Function* as in C, is an algorithm that operates on parameters, and produces and returns a single resultant value of a specific data type. *Procedure* is an algorithm that receives parameters from the point of invocation and operates on them in some manner to produce the desired end state. *Rule* defines constraints on entities and attributes that must be enforced by the information system.

2.6 Process Knowledge Representation

Process knowledge is knowledge about the capabilities of a manufacturing process. In a machining environment it is concerned about producing various features on a raw material within,

the shape and size limits,

the surface finish and tolerance limits,

the resources (tools, machine tools and fixtures),

the constraints.

The knowledge about the machining process is a combination of shop floor experience and knowledge from books. Knowledge representation techniques form a major component of generative process planning systems. Several methods have been used to represent process knowledge into format and structures that will facilitate program coding and documentation.

2.6.1 Decision Tables

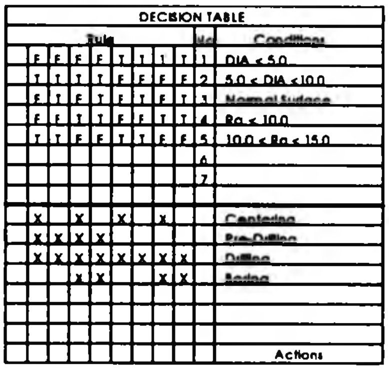
The use of decision tables for process planning was pioneered in the late 1960s by the consortia of Norwegian industrial firms, NAAK, which resulted in a system called AUTOPROS (automated process planning system) [50].

A decision table is partitioned into conditions and actions. Decision rules are identified by columns in the entry part of the decision table. When all the conditions are met, the marked action is taken (Figure 2.8). Decision tables can be linked and sequenced to evaluate many conditions and logical actions required in process planning. However, the system is difficult to maintain and is not being used commercially.

2.6.2 Decision Tree

A process logic is represented in the form of a tree structure. Each branch represents a possible course of action. If the condition specified at a branch is true, it can be traversed to the next node, and so on, until the goal is reached.

The DCLASS CAPP system uses a decision tree processor. Its further development, in establishing linkage between decision table and decision tree, has been the focus for the US Air Force project for Group Technology Software Support (GTSS) [50]. The tree logic is easily visualised and the given conditions are easily traceable. However, the trees may become quite large and difficult to display and maintain.



DECISION TABLE										
	Rule								No	Conditions
	F	F	F	F	T	T	1	T	1	DIA < 5.0
	T	T	T	T	F	F	F	F	2	5.0 < DIA < 10.0
	F	T	F	T	F	T	F	T	3	Normal Surface
	F	F	T	T	F	F	T	T	4	Ra < 10.0
	T	T	F	F	T	T	F	F	5	10.0 < Ra < 15.0
									6	
									7	
	X		X		X		X			Centering
	X	X	X	X						Pre-Drilling
	X	X	X	X	X	X	X	X		Drilling
			X	X			X	X		Boring
										Actions

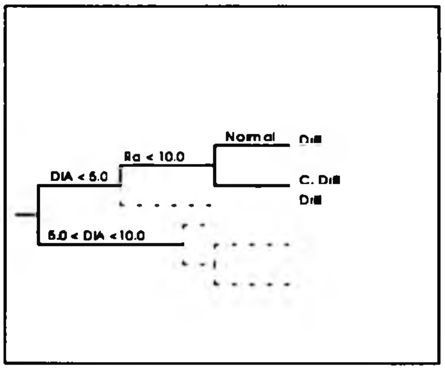


Figure 2.8: Decision Table and Decision Tree

2.6.3 Conventional Programming Languages

Based on the decision tree or table, knowledge can be represented in the form of rules in **If... Then...** format of a programming language. A CAPP survey in the eighties showed the usage of BASIC and FORTRAN as the most frequently used programming languages [4].

Conventional programming languages such as BASIC or FORTRAN, are limited to a sequential pre-defined programming style. The main concepts involve the use of algorithms, control structure and data, stored in files. There is no distinction between program processing and the technological oriented decision logic. This makes it very difficult or nearly impossible to adapt conventional programmed CAPP systems to specific needs [47]. Customising a system is very time consuming. Implementation of changes could be very difficult, especially if the original programmer has left the organisation.

2.6.4 Artificial Intelligence (AI) Techniques

AI based techniques are designed for capturing, representing, organising, and utilising knowledge by means of computers. Such systems have the capability to understand problem-specific knowledge and use the domain knowledge intelligently to suggest a path of action.

AI is not intended to replace human intelligence. It is rather a marketing name for new programming methods to create reasoning systems [97].

Various types of knowledge representation techniques have been used in expert systems [98]. These are,

a. State space representation: The state space representation represents the structure of a problem in terms of alternatives available at each possible state of a problem. It was developed for problem solving and game playing. It is not particularly well suited for expert system applications [99].

b. Predicate logic: The description of the real world is given in terms of logical clauses. Each fact needs to be represented only once irrespective of repeated usage. Logic schemes pose difficulty in procedural representation.

c. Procedural representation: The knowledge is represented in procedures, small programs about how to do things. Like conventional programming, the problem with this scheme is the difficulty in varying and changing procedural representation if modification is necessary [98].

d. Production rules: This is one of the most commonly used knowledge representation schemes. The knowledge is represented in the form of condition-action pairs, called production rules, whose general form is:

IF [some combination of condition is true]

THEN [draw some conclusions or take some action]

This approach has been used by several process planning systems and TOM (Technostructure of Machining)[11] was one of the earliest in metal cutting domain. The production rules in TOM were written in PASCAL language in the following format:

```
IF      <consequent geometry>
AND     <                      >
OR      <                      >
.
.
THEN    <recommendable machining type>
        <specified parameters and its values>
.
.
        <antecedent geometry>
```

The following example illustrates the production rule in TOM:

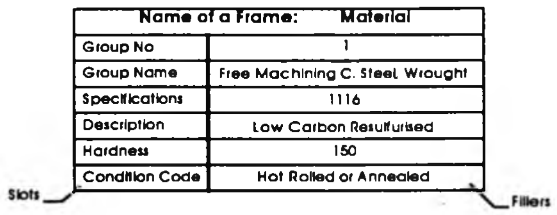
```
***** CONDITION PART OF THE RULE *****
IF          FORM IS CYLINDER
AND         HOLE IS ROUGH FINISHED
OR          HOLE IS FIN FINISHED
AND         HOLE IS THROUGH HOLE
AND         MAX TOL OF DIA - MIN TOL OF DIA >= 0.2
AND         DIA NOMINAL > 50.0
AND         DIA NOMINAL/DEPTH NOMINAL >= 4
***** PROCESS PART OF RULE *****
MAJORWORD       :       MILL
DIAMETER        :       DIA NOMINAL
DEPTH           :       DEPTH NOMINAL + 0.05 * DIA NOMINAL
RDEPTH          :       DEPTH OF UPPER HOLE - 1.0
FEED            :       200
SPEED           :       50.0
```

```
TOOL                :        3
TYPE                :        2
***** PREVIOUS STATE OF HOLE *****
          \SHAPE OF SIDE              \CYLINDER
          \DIA NOMINAL                \25.0
          \MAX                        \0.5
          \MIN                        \0
          \GROOVE                     \DONT EXIST
          \ROUGHNESS OF SIDE          \ROUGH(*)
          \DEPTH NOMINAL              \DEPTH NOMINAL
          \MAX                        \0.5
          \MIN                        \-0.5
          \FLAT BOTTOM                \FALSE
          \THROUGH HOLE               \THRU
```

TOM uses a backward-chaining inference mechanism to generate the machining sequence; that is, the searching process starts from the finished geometry. The backward chaining continues until a rule specifies no antecedent geometry; that is the machining needs no pre-machining. The rule conflicts are resolved by a backtrack search, for minimum machining time.

XCUT [63] uses the object oriented programming language Flavours to represent features and production rules for machining prismatic components. XPLANE [15] uses FORTRAN 77 to describe production rules. The COATS, the optimum tool selection for turning operations; KAPLAN [13], the optimal selection for turning operations; and FATA, the robotic assembly and disassembly, are the various CAPP systems developed at the university of Pisa, Italy. All the systems use production rules in PROLOG language and resolve the rule conflict with weight attribute [101].

e. Frames: Frames represent knowledge by use of prototypical objects. A class of an object is represented as a frame which consists of a collection of attributes and its values, also referred to as slots and fillers respectively (Figure 2.9). In some ways the frame representation is analogous to the storage of data in C or PASCAL structures [44]. The values or fillers could be associated by a particular procedure. A name of a frame is used for general identification of an entity or object. A particular object is described by an instance of the frame, in which the slot values are instantiated to those appropriate to a particular object. Frames generally allow the freedom to use different data types, such as lists or references to procedures. Furthermore, the values of slots may be obtained from other frames by inheritence: a particular class of a frame may itself be a subclass of a higher level class. For example, an End-Mill may be a subclass of Mill-Tools. A knowledgebase can be described as a collection of frames or entities.



Name of a Frame:	Material
Group No	1
Group Name	Free Machining C. Steel Wrought
Specifications	1116
Description	Low Carbon Resulfurised
Hardness	150
Condition Code	Hot Rolled or Annealed

Slots
Fillers

Figure 2.9: Freme Representation

SIPP (Semi Intelligent Process Planner) uses frame based knowledge representation scheme [102]. In most frame-based approaches to knowledge representation and reasoning, the problem solving knowledge that manipulates the frames is stored separately in the form of production rules. In SIPP, the problem-solving knowledge is represented not as rules, but hierarchically in the form of frames. The following example illustrates the knowledge representation scheme from SIPP:

```
type(hole_process, process).
relevant(hole_process, hole).
defaultvals(hole_process,[cost=1]). /*e.g. cost of twist_drill*/
restrictions(hole_process, H):-...various geometric restrictions


type(hole_improve_process, hole_process).
defaultvals(hole_improve_process,[
                precedence = 20,
                projected_cost = 1, /*e.g. cost of twist_drill*/
                cost = 3,/*e.g. cost of rough_bore*/
]).
restrictions(hole_improve_process, H):-H?special_features eq none


type(bore,hole_improve_process).
defaultvals(bore, [cost=3, precedence=22]).
restrictions(bore,H):-...various tolerance restrictions
```

```
type(rough_bore, bore).
restriction(rough_bore, H):-
H?pos_tolerance gte 0.002,
H?neg_tolerance gte 0.002.
actions(rough_bore, P, H):-
copy_item(H, G),
G:diameter gets H?diameter - 0.005,
G:pos_tolerance gets 1,
G:neg_tolerance gets 1,
G:straightness gets 1,
G:roundness gets 1,
G:parallelism gets 1,
G:true_position gets 1,
G:surface_finish gets 125,
subgoal(P, G).
```

In the above example, rough_bore is a subtype of bore, which is a subtype of hole_improve_process, which is a subtype of hole_process. This means that rough_bore is applicable for creating a particular surface H, only if the restrictions for hole_process, hole_improve_process, bore, and rough_bore are all satisfied for H. SIPP uses least-cost-first-search control strategy.

A frame based part description scheme which is also discussed by Zust [100] who represents a part as a model of a logical unit (features) and geometrical elements and workpiece to form the overall frame structure. XCAPP-P [103] uses framed based approach (coded in PROLOG) for feature based modelling of turned components. The same system also uses a frame based knowledge representation for machining feature recognition.

The frame based knowledge representation has led to a new programming style, named as Object Oriented programming.

f. Semantic nets: Semantic nets attempt to describe knowledge as a collection of objects(nodes) and binary relations(labelled edges) [99].

According to a semantic net representation, the knowledge is a collection of objects and associations represented as a label directed graph. A typical semantic net is shown in Figure 2.10. The figure represents the facts that the hole is a feature which is manufactured by drilling, and drilling is an operation. The semantic net concept could be used to describe structured objects or frames.

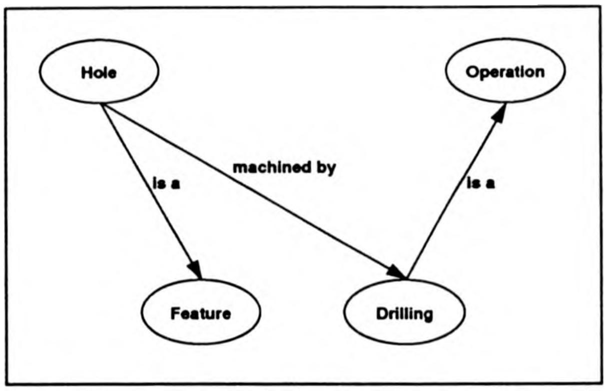


Figure 2.10: Semantic Nets

In AI based techniques, the symbol manipulation languages are preferred to conventional number manipulation languages. The builders of a majority of expert systems chose either LISP or PROLOG languages [45].

2.7 Databases

The main characteristic of process planning is the multiple input from all areas within a company. The design department provides the geometrical data, sales and purchasing give data on deliverable dates and lot sizes, and the factory itself incorporates information on tools, machine tools, fixtures, processes and availabilities.

Within the modern information technology strategy of a company, process planning is an element in a network of functions [104]. Other elements in the network may have their own data storage and management systems, for example, a tool management system. On a more general term such databases can be called resource databases. They are being used in the process planning environment either as a decision support system or a decision making system [105].

In recent years, the relational databases have been most widely used in the manufacturing industry. The principles of relational database models were first laid down by Dr. E. F. Codd [106]. A declarative knowledge can be viewed as a relational scheme. The identification of entity types facilitate classification of factual data into recognisable entities. The identification of entities, properties and associations can be used to represent a database.

The relational database model is perceived by its users as a collection of two dimensional tables which are easy to understand. The main element in relational databases are tables, mapping and relating the data. Each table has a unique name in the database. All rows are unique in the table. A column or a combination of columns, whose values uniquely identify the row in a table, are known as a primary key. A column or combination of columns whose values match that of a primary key in another table is known as a foreign key. The use of foreign keys allow one to relate information, which had no relations at the time the table was created. This feature represents high flexibility in the use of relational databases. The non-procedural English like language, SQL (Structured Query Language) is used to query the database. When applications must sort and search through large amounts of data, a query language like SQL can save hours of development time, that would otherwise be spent custom-programming reports [109].

The relational database however, can have difficulties in representing the semantic information in real-world applications [107]. When a real world entity cannot fit into the relational model directly, an artificial decomposition becomes necessary. This decomposition not only decreases efficiency, but also raises the possibility of losing the semantics associated with the data [108].

In a modern office information system, data includes not only text and numbers, but also images, graphics, digital audio and video Such multimedia data are typically stored as sequences of bytes with variable length. The variable length data structure can not fit well into the relational framework, which mainly deals with fixed format records [108].

The Object Oriented Databases are the new generation, with the major feature of being a logical single database composed of a hierarchy of objects. The real world entities are described as objects with attributes [110]. The object databases not only store data, but also the instructions for manipulating data, thereby providing a seamless integration between certain programming languages and the database itself. Some object databases provide fairly seamless integration between themselves and object-oriented programming languages C++, Smalltalk and Lisp [109].

Object oriented databases are considered superior to relational databases in structural knowledge representation, variant record structure, composite structure, revision control and declarative knowledge. The relational databases are better suited for report-intensive applications such as MRP, whereas object databases have more potential for CAD and CAE that are not report intensive but require high levels of performance [109].

Both, relational and object oriented databases represent the current trend in database technology.

2.8 Conclusions

The current state of techniques used in the development of CAPP systems have been highlighted and the state of commercial CAPP system have been reviewed.

Most commercial CAPP systems are either variant or semi-generative. These systems use primarily the group technology CC techniques for part design representation.

Solid modelling technique represent the CAD model unambiguously and completely, and are widely accepted to represent the part geometry. The B-rep solid model is preferred in downstream applications such as feature recognition and NC part programming, as it stores the evaluated solid geometry.

The low level data structure of a solid model cannot be used to apply reasoning methods in CAPP. A feature recognition technique constructs a manufacturing feature model of a part design. Most of the feature recognition techniques have been successful in a limited domain of process planning of 2 to 2½ axis machining. They were also successful in providing the independent object views to the designer and process planner.

Most feature recognition techniques used a solid block (or a billet) as a workpiece. However, in practical situations, the workpiece could also be a casting or a forging. The features produced by a casting or a forging may or may not need further machining.

Since most of the features are known to the designer, the feature recognition is a redundant effort. Feature based modelling techniques have been suggested to overcome feature recognition. However, mapping of the design features to manufacturing is a subject of current research. Both, feature recognition and geometric reasoning have been considered as possible techniques for feature mapping.

The feature based modelling systems, in general, comprise a feature processor and a geometric solid modeller. A bi-directional associativity between feature processor and geometric modeller has been advocated for implementing better geometric reasoning techniques.

AI based production rules and frames are being most widely used for knowledge representation techniques in CAPP research. The frame based knowledge representation is very similar to the object oriented programming technique.

Most of the knowledge based CAPP systems developed so far use custom made part knowledge description schemes. The use of CAD data exchange standards to represent the part knowledge has been seldom used. The STEP implemented form feature information model on commercial CAD systems need experimentation in CAPP.

The use of STEP form features could also be investigated for graphic visualisation of resources such as tool assemblies and modular fixture elements.

The majority of the CAPP research represents product knowledge as knowledge about an individual component. However, such consideration may be inadequate in a concurrent engineering environment. A structured representation of a product as an assembly, subassembly and components could be more effective. The recent product development tools such as product assembly modelling and knowledge based product modelling enable one to use such a representation.

The majority of CAPP research has not considered resource management as an independent issue. The action part of a production rule points to some resource in the process plan. However, the availability of the resource is rarely checked. The resource databases as a tool to suggest alternative resources have rarely been considered as a part of process planning strategy.

3. PRODUCT DESIGN REPRESENTATION

A CADDS5 system from Computervision (CV) has been used in this research for the product design representation. CV provides concurrent assembly design functionality for product development. In order to build CAD models, it provides both conventional CAD and feature based CAD modelling functionality. The product model data can be managed efficiently by the engineering data management system. CV has also been actively involved in providing STEP standard support for product data exchange.

One of the objectives of this research is to conduct a detailed study of the feature based CAD modelling functionality and devise a suitable strategy to communicate the manufacturing intent through features. It has also been felt necessary to study the overall product development process in a concurrent engineering environment, and find out the role of the design by feature and the CAPP in that context.

A product modelling strategy in a concurrent engineering environment is explained at the beginning of this chapter. Later, a detailed illustration of design by feature techniques is presented. The various strategies that have been tried, to communicate the manufacturing intent are discussed. The chapter concludes by suggesting a feature based design representation technique that has been used to suit the CAPP system reasoning processes.

3.1 Concurrent Assembly Design

According to the STEP standard and the relevant research projects, product models are defined as the logical information structure, which can be used to represent the entire information about the product [76, 77]. In this context the geometric models are only part (attributes) of such product models. Such a data modelling methodology is based on the object oriented data structure.

CV has provided the above product modelling methodology through a software product called, Concurrent Assembly Mock-UP (CAMU). It enables not only the concurrent assembly design but also an electronic/digital mock-up of a product assembly. With CAMU it is possible to design a product structure as a logical tree of assemblies, sub-assemblies and components (Figure 3.1). Both, top down and bottom up design approaches are supported. The designer can associate model geometry to various components in the product structure. Several users can work concurrently on the same assembly across the network. The designer can manage the overall development of a product by having access to multiple component models simultaneously. One can model a component while looking at other component models in the assembly, and switch readily between different component models. This enables one to design each component or sub-assembly in the context of the assembly. To build a CAD model of a component, the designer can use a wire-frame, surface, solid or even a feature based modelling approach [111].

In addition, it is possible to define and associate the non-graphic information as attributes to the assembly, sub-assembly or component nodes of a product structure. These attributes could be bill-of-materials information, physical properties, drawing notes, labour tracking data, or even special process information.

CAMU is developed using the object oriented methodology. The assemblies, sub-assemblies and components on a logical structure are described by object classes and instances (or occurrences) related to each other. Their relationships are also described by classes and instances. The relation class refers to the position and orientation of a component, sub-assembly or assembly. All the instances maintain the properties belonging to their class. For example (Figure 3.2), each wheel instance consists of one instance each, of the class disk, tyre and hub. The assemby structure is usually described in such a way that the leaf nodes are the component instances. The relation classes and instances ensure that the position and orientation of various components is precisely defined and maintained. For example, if the designer decides to move the position of axis_2, all the child nodes attached to it will also move together. Thus, such model data structures could be very efficient in manipulating design changes.

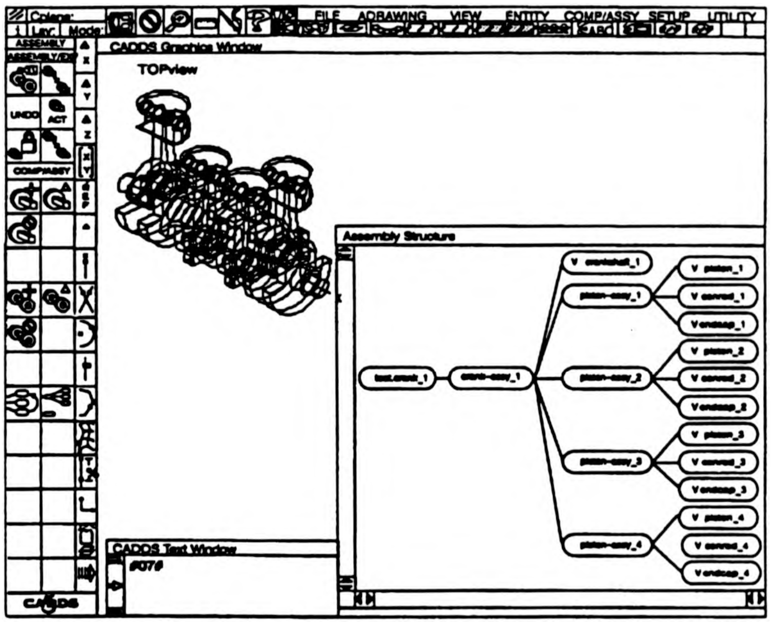


Figure 3.1: CAMU Windows Showing Sample Assembly
[REF: CAMU User Guide, Computervision]

With this approach, it is easier to conceptualise a product and represent its structure even before getting into any detailed design activity. Assembly design eliminates data redundancy, by associating a model geometry to a component on the logical structure. Several instances of the component can refer to the single model geometry.

If the various components on the assembly are associated with the solid model geometry, then it is possible to generate a shaded image of the entire assembly, check the clash detection between components and so on. The success of CAMU in the aerospace industry (Rolls-Royce, U.K., Shorts Brothers, U.K) for the past three years implies, that in future a physical mock-up may not be built at all, thereby reducing the product development times considerably [86].

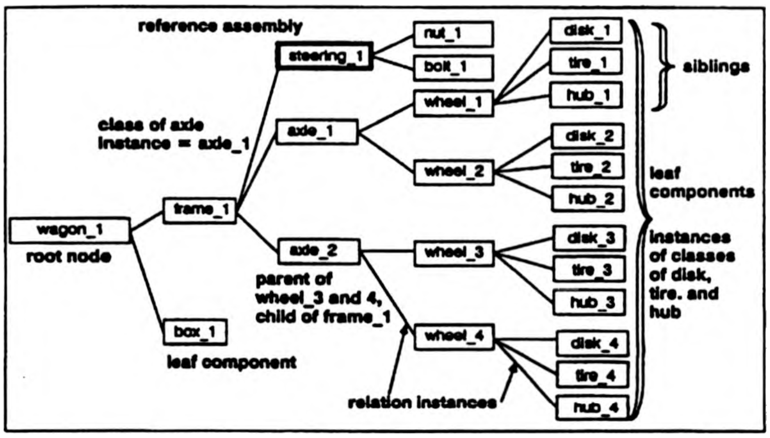


Figure 3.2: Assembly Structure for an Assembly named Wagon
[Ref: CAMU User guide, Computervision]

3.1.1 CAMU and Engineering Data Management System (EDM)

The product model data generated by CAMU in a large company may consist of hundreds of thousands of files. The data is heterogeneous (graphic, textual & mathematical) in nature. In general the designers spend 30% of their time in searching all sorts of data [44]. A system to assist engineers to index and search data, may be termed as an Engineering Data Management System (EDM) [114].

The EDM from CV is built by using the ORACLE relational database management system (RDBMS). It is the application software that sits on top of the ORACLE RDBMS [113]. EDM Client is a nodt on the network that uses TCP/IP protocol to communicate to the EDM server (Figure 3.3). Each node may be connected to the CAD system or any other application. The EDM receives requests for service from clients throughout the network. A typical EDM service may include a request for engineering data files, set-up authority schemes in a project team, manage the project status, release and revision control, back-up procedures, report generation and system security.

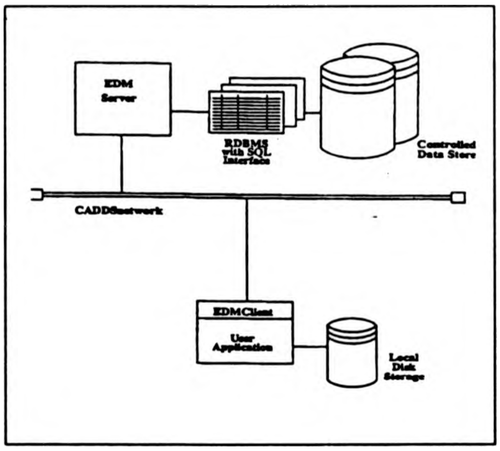


Figure 3.3: The EDM Volume of Information
[Ref: EDM Training Manual, Computervision]

The EDM Product Structure Navigator (PSN) is an add-on application software tool, which enables one to view and modify the CAMU product tree structure stored in the EDM vault. Most of the time the CAMU clients will be busy in creation/modification of CAD models of various components in a product design. PSN enables independent access of the product structure data for review without disturbing the clients (Figure 3.4).

EDM PSN also manages the non-graphic product information of a product assembly. Using EDM PSN one can display a product structure tree, the status of associated CAD models and files, highlight nodes according to specified attributes, generate reports, bills-of-materials and list instances of components with specified attributes, thereby enabling a simple 'what if' analysis.

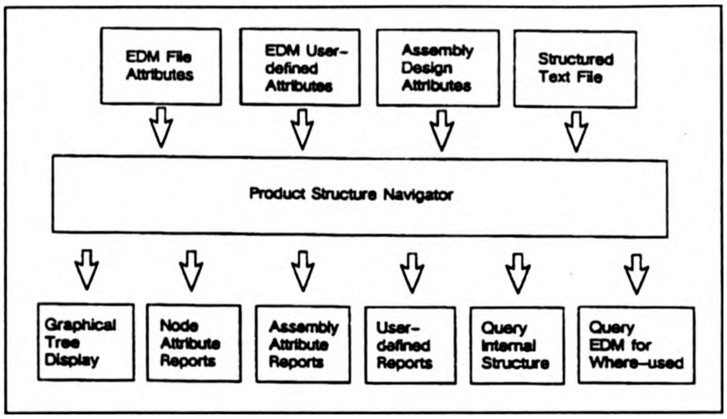


Figure 3.4: Product Structure Navigator Inputs and Outputs
[Ref: EDM Training Manual, Computervision]

3.1.2 CAMU and Configuration Access

Configuration Access is a graphical user interface which addresses a number of requirements in managing, manipulating and querying data, in a hierarchical product structure. The software is of significant benefit when working with EDM and CAMU. It provides a single interface to replace the CAMU/EDM and EDM/PSN interfaces [115].

EDM and CAMU are not a prerequisite in using Configuration Access. Configuration Access includes facilities to create and edit hierarchical structures; associate component CAD models, files and reference assemblies to the nodes within the structure; and edit class and instance attributes within the structure. Configurations may be saved as local product structures or as CAMU assembly trees or even in STEP AP203 for configuration control. Whenever the configuration data is sent to EDM, it may be exploded into EDM attributes so that its information is queried.

The AeroSTEP is a pilot project studying the phased use of STEP Application Protocol AP203 in the Aerospace industry. The Computervision STEP translator demonstrated the reading and writing of STEP Part 21 files (AP203) in November, 1993 [96]. The files were comprised of product structure and configuration data for the product, together with geometries of the selected components (Figure 3.5).

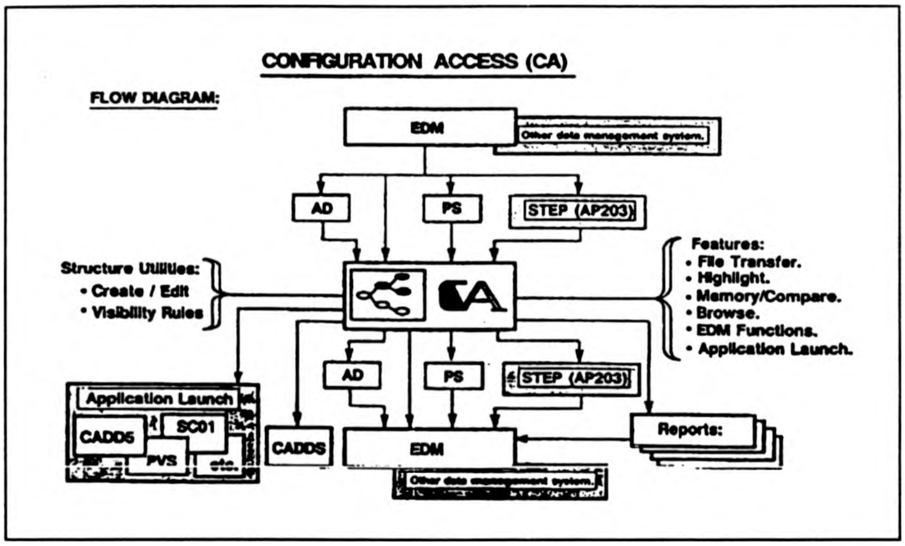


Figure 3.5: Product Structure, CA & STEP
[Ref: Computervision & Rolls Royce]

3.2 CAMU, EDM and CAPP

The non-graphic attribute association capability of CAMU could be used to assign the manufacturing process attributes of a component. This would enable the product design and manufacturing data integration within the EDM vault. This data could then be used by other applications such as MRP, CNC/DNC, etc. As explained above, the STEP application protocols should enable exchange of the data on any CAD/CAM systems, thus enabling a total integration of different CAD/CAM systems.

However, the manufacturing process attributes can only be generated by a CAPP system, unless assigned manually.

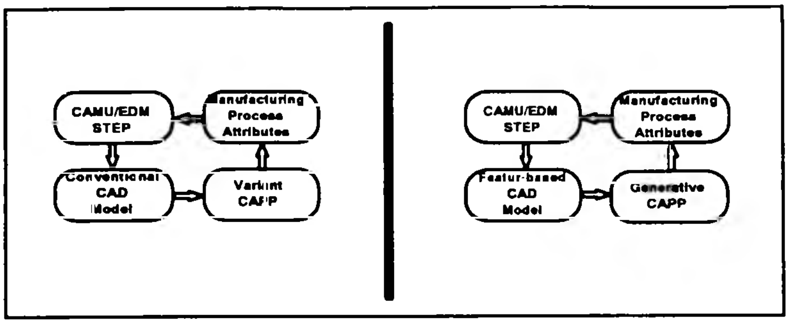


Figure 3.6: Variant or Generative CAPP options

There are two possible CAPP approaches, that can be used (Figure 3.6). The variant CAPP systems can be used until a generative CAPP system is fully developed. In order to develop a generative CAPP system, a feature based CAD modelling approach will be used [Figure 3.7].

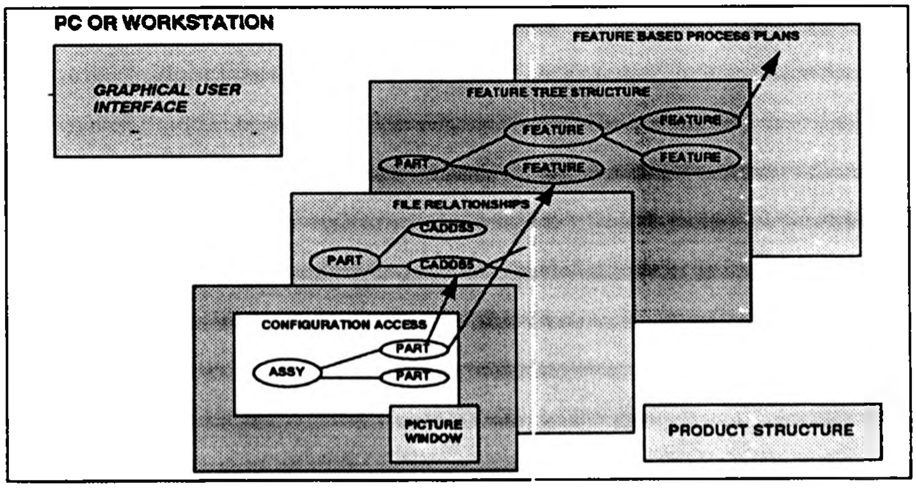


Figure 3.7: The Role of Configuration Access

3.3 Feature Based Modelling

The CADDS5 parametric solid modelling environment has been provided with the option of form features, that are available in the library. They are referred to as system features. The system also provides the capability to create the user defined features in the user library.

The feature based modelling is not mandatory on CADDS5. However, this research proposes the design representation by features so that the component model data could be used for geometric reasoning in the CAPP system. The following discussion illustrates the design by feature concepts.

3.3.1 The System Form Features

The system form features are provided in a library called SYSLIB. The feature processor that manages the feature environment is based on object oriented methodology, such that each feature can be regarded as an object. The taxonomy of the system features is based on STEP standard. The system feature taxonomy scheme is illustrated in Appendix A.

There are several reasons for classifying features. The number of possible features being unlimited, no system can support every feature required by every user. Once features are classified, it becomes easier to define a new feature by the 'something similar to' concept. A newly defined subclass automatically inherits the attributes of its superclass. A multiple inheritance concept of object oriented methodology enables one to define complex features with ease.

Each form feature in the SYSLIB comprises a definition and a geometry. The feature definition consists of its attributes, in addition to the attributes that would have been inherited from its super class. The attributes always carry the default values. The represention is also an attribute that corresponds to a parametric solid geometry of a form feature. The form features are also referred to as volume features, as their form is described by the solid geometry.

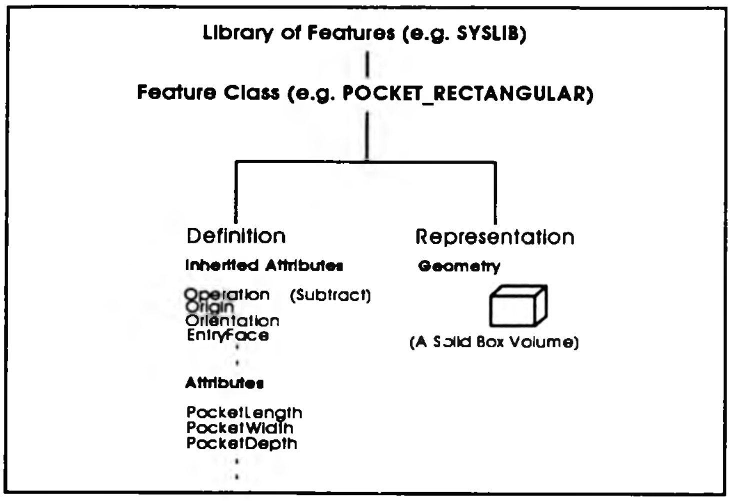


Figure 3.8: A Typical Form Feature Schema

The parameters that describe the form feature geometry correspond to the feature attributes in the feature definition. For example, a Pocket_Rectangular form feature geometry can be described by a hexahedron, that has length, width and height parameters. These parameters correspond to the attributes in the feature definition. This way, each feature is able to work as a generic feature [76].

In a modelling process, the designer creates a feature by overwriting the default attribute values to suit his/her needs. The feature processor passes the geometry attributes of a feature instance to the geometric modeller to evaluate the geometric form of a feature. While building the CAD model of a design, the user is not concerned with the feature processor and geometric modeller. The underlying principle is built in the software.

The attribute values of a feature instance can be modified by digitising the feature instance. In a parametric modelling environment, the form features are treated like any other parametric entity.

3.3.2 Modelling With Features

Systematic modelling steps should be followed to build a feature based model such as,

1. Identification of major feature influence in the component shape such as, flange, bracket, casing, handle, frame, plate etc.
2. Search for the major feature in the library or create it, if it is a new one.
3. Study of the major feature relationship with other features. Feature based modelling environment being parametric, this step is very important to communicate the design intent.
4. Insert other standard or user defined features to complete the design.

Appendix B illustrates the feature based modelling steps.

3.3.3 Model as a Feature Tree

The feature based model is built by attaching features to the entry or exit faces of existing features of a model. If the feature is merged with more than one feature, proximity face/s information will have to be provided besides entry or exit faces. Computervision feature modeller allows only entry and/or exit face selection while modelling. This enables to establish a parent/child relationship amongst feature instances and therefore, a design representation as a feature tree can be captured. A feature report file enables one to capture the attributes of the entire set of feature instances in a model. The report file data can be used by other applications such as CAPP.

The feature tree and the feature report file are illustrated in Appendix B.

3.3.4 Limitations of System Form Features for CAPP

The system form features are suitable mainly to represent the nominal form of a component design. Although their names are manufacturing implicit they may or may not imply the same meaning in manufacturing. For example, a Hole_Blind form feature in the design may imply a drilled hole or the existing hole that needs to be enlarged by boring. The manufacturing 'view' of the feature can only be inferred by considering the raw material condition from which it has to be manufactured. Most of the research in feature based CAPP systems have assumed the raw material to be a solid block (or a billet), which implies a predictable manufacturing 'view' for at least the subtractive types of form features. This enables automation in process planning as well as NC programming [45].

It is however, felt necessary to have broader raw material state assumptions. In conventional NC programming, a stock statement has been used to imply a raw material. Similarly, there should be a method to communicate the raw material stock on the feature.

The system form features represent the nominal solid geometry. A suitable tolerancing scheme is necessary to reason the form feature design representation. They also do not have a manufacturing attribute set, which can communicate the manufacturing intent, such as surface finish, clamping locations, etc.

Due to the above limitations, the system form features are unsuitable for their usage in CAPP system reasoning. Their definitions will have to be modified by user defined feature techniques with a suitable strategy.

3.3.5 User Defined Features

The user defined features can be created in the user library. Like system form feature, the user defined feature has a definition part and its solid geometry representation part. While defining a new feature the user is able to use the system form feature taxonomy. Usually, the user defines a new feature definition as a sub-class of existing feature (or features) with additional attributes that define its characterstics. The additional attributes are used to describe the features geometry parameters or to assign any other information to the feature. The object oriented methodology enables one to automatically inherit the attributes of a super-class.

A parametric solid geometry is constructed to represent the form (shape) of user defined feature. Finally, the definition attributes that describe geometry are associated to the solid geometry parameters.

The above procedure has to be followed to create a user defined feature, which then could be used as a generic feature like any other system form feature.

3.4 Form Features and Manufacturing Intent

As discussed in 3.3.4, the definition of the system form feature is inadequate to communicate the manufacturing intent. The research review also indicates that there is no common agreement on communicating the manufacturing intent [52, 65, 116].

This research proposes that the user defined feature technique can be used to communicate the application specific manufacturing intent (e.g. machining). It explores various user defined feature techniques with the following objectives:

a. Convenience in user interaction while modelling

b. Ease in interpretation and modification

c. Direct interpretation of such information for CAPP

d. Preserve designer's creativity/flexibility while modelling

The following discussion illustrates the various techniques that are proposed to communicate the manufacturing intent for the machining of prismatic components.

3.5 The Manufacturing Intent

In order to interpret the manufacturing intent, the design data should communicate the following information:

a. Tolerance

b. Surface Finish

c. Raw material condition

d. Machining intention

e. Fixturing

On a conventional CAD system, the above information is assigned by the detail drawing preparation methods such as, dimensioning and tolerancing commands, symbols, notes, and labels. It is assigned to the same model but on different layers. By layer manipulation, the designer and the manufacturing engineer are able to see their respective 'views' of a model data.

It was decided to use similar technique through user defined features to communicate the manufacturing intent. The information was categorised into tolerance and manufacturing feature types.

3.6 Tolerancing by Features

The feature tolerancing is necessary to ensure the functional obligations of the design, which also becomes a guideline to select a manufacturing process capability. The development of the dimensioning and tolerancing (D&T) model is a special research domain, and it was beyond the scope of this research. However, the various research efforts in the development of D&T model have been cited earlier[80, 81, 82, 83]. It was observed that Faux's CAM-I report [80] and the subsequent work for CAM-I by Ranyak and Fridshal [82] was based on tolerance features.

An attempt was made to create the tolerance feature library based on the CAM-I initiative [80, 82]. The idea behind the experiment was to compare the suitability of tolerance feature with the traditional detailing technique on a conventional CAD system.

Faux [80] proposed primitives to assign tolerances. A primitive is a set of one or more faces which are treated as a single individual unit for the purpose of defining and tolerancing a feature shape (size and form) and location (position and orientation).

In this context a form feature is a set of faces of a component, structured into one or more primitives and subfeatures, and whose faces, primitives and subfeatures obey the rules of its declared generic feature class.

A primitive is said to be resolved when the shape of its surface(s) are defined separately from its location and orientation. Geometric tolerancing classes for the resolved primitives are:

a. Location (Position and Orientation)

b. Shape (Size and Form)

An unresolved primitive is a surface or a set of surfaces whose shape and location are toleranced by a single profile tolerance. The geometric tolerance class for unresolved primitives is profile, which could be surface or curve profile. The above tolerance classes can be represented by a suitable feature representation scheme (Figure 3.9). In general, these could be thought of as 'labels' assigned to various faces (primitives) of a feature based model. The following discussion describes the various tolerance classes.

3.6.1 Datum Feature

The first step in tolerancing is to decide datums. Datums are used to establish the relationship of geometrically toleranced features. It was defined as a subclass of a class primitive. It assigns Datum_Name and Datum_Face attributes to the model.

The Datum feature was classified into two subclasses, Datum_Axis and Datum_Face.

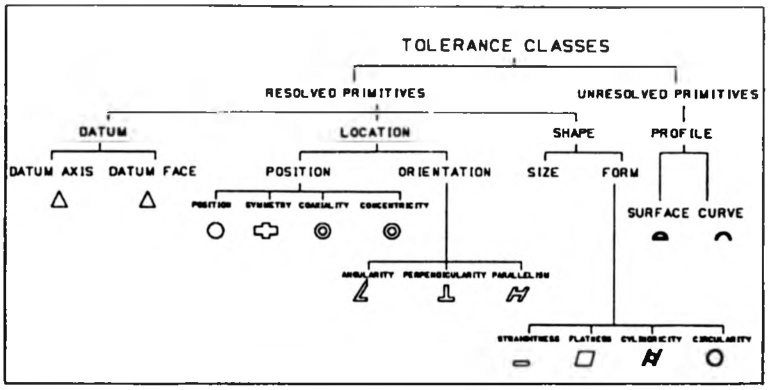


Figure 3.9: Geometric Tolerance Features

3.6.2 Location Tolerance Feature

The Location Tolerance Feature was defined as a primitive class which assigns positional or orientational tolerance values and a datum reference frame. The datum reference frame determines the possible coordinate system depending on the datum chosen.

The Location tolerance feature was classified into four subclasses, Position, Symmetry, Coaxiality and Concentricity.

3.6.3 Shape Tolerance Feature

A primitive class assigns a size or form tolerance value. It is observed that most of the form features that are provided in the SYSLIB can be modified to add size tolerance attributes such as, Tol_PocketLength, Tol_PocketWidth, etc. Form tolerances were defined into four subclasses; Straightness, Flatness, Cylindricity and Circularity.

3.6.4 Profile Tolerance Feature

The Profile Tolerance Feature is a primitive class that assigns a profile tolerance attribute value and symbol.

3.6.5 Tolerancing the Feature Based Model

A suitable layering scheme can be used to assign tolerance features to the model (Figure 3.10). These layers could be blanked for design manipulations/modifications. Any parametric changes to the model should cause a corresponding change in tolerance feature symbol location.

3.7 Manufacturing Features

The definition of manufacturing features and their associated information (attributes) depend upon their application [78]. As mentioned above for machining application, information such as surface finish, raw material, etc. is necessary to plan the machining process. A suitable manufacturing feature representation scheme is proposed (Figure 3.11).

3.7.1 Surface Finish

The surface finish depends mainly on the tolerance grade and thereby machining process and material grade. It is defined as a subclass of a class primitive with the attributes of a value, and a symbol.

3.7.2 Machine or Not_Machined

A form feature machining intention could be communicated through these features. If a raw material is a forging or a casting, some form features need not be machined. It is defined as a subclass of a class primitive with the attributes of a logical value and a symbol.

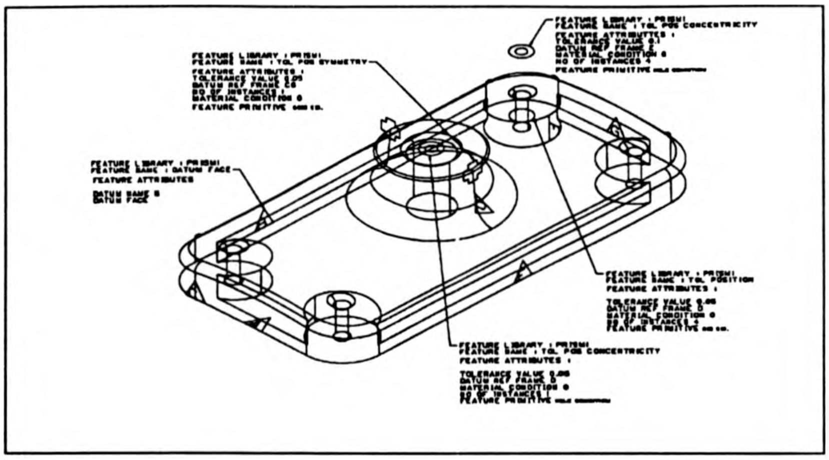


Figure 3.10 FBM with Tolerance Features

3.7.3 Raw Material Stock

The Raw Material Stock is defined as a subclass of a class primitive with the attribute, stock. By default the stock is always 'solid'. In the case of castings or forgings, the attribute value is modified for the uniform stock thickness.

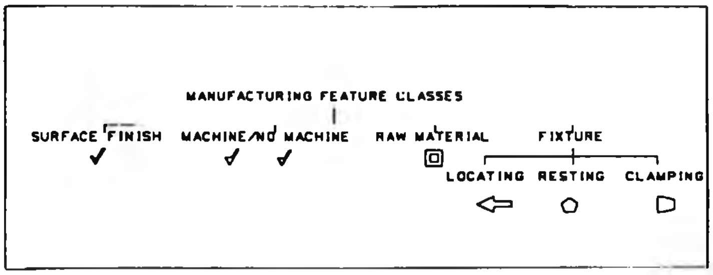


Figure 3.11: Manufacturing Features

3.7.4 Fixture Feature

Three subclasses are proposed to assign the intended resting, locating and clamping faces and locations. Automatic fixture design is a research subject in its own right, however, it was thought that the manufacturing engineer should be able to communicate this information at the design stage.

3.7.5 Assigning Manufacturing Features

Like tolerance features, a suitable layering scheme can be used for assigning manufacturing features to the feature based model (Figure 3.11). The tolerance and manufacturing feature symbols enable to interpret the model for its manufacturing intent and may help to eliminate the need for detailed drawing preparation [140].

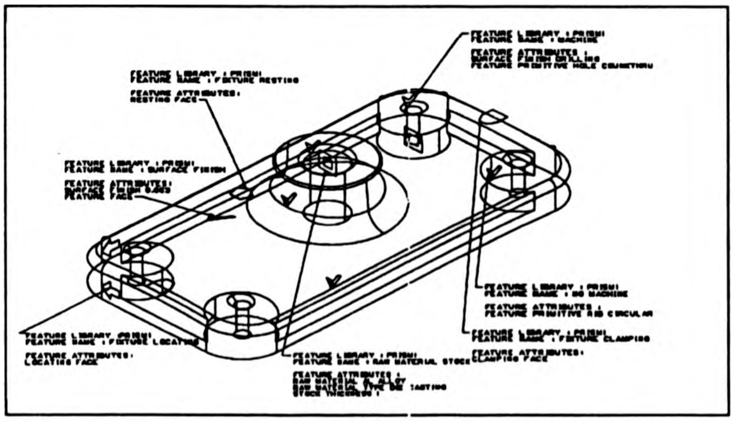


Figure 3.12: FBM with Manufacturing Features

3.8 Features and Creativity

One of the major criticisms about feature based design has been that it hinders the creativity of a designer by providing a library of fixed forms for design [117]. A detailed study of feature based modelling strategy reveals that the feature based design could be as flexible as the conventional CAD.

A case study [Appendix C] has revealed that the designer is free to conceptualise the geometric shapes as in conventional CAD [141]. Such shapes could be formed by sweeping profiles and thereby obtaining the new primitives, sculpting the primitives by a curve or a surface, boolean operations and fillets, etc. All the parametric solid/surface modelling techniques could be used to describe a new shape. The geometric shape can then be described as a representation of the user defined feature. The important design/manufacturing parameters of a geometric shape become the attributes of the feature.

3.9 CADDS5 System User Limitations

Free access to the design data is the fundamental requirement to develop a design based application, such as CAPP. Most CAD vendors use a specific method to represent data. The CADDS5 system has implemented the STEP based form feature definitions only. The feature data retrieval as per STEP format, is not implemented yet. The STEP committee has been working on this aspect through an application protocol AP224, and the details are not available yet.

For the time being, the CADDS5 system enables the user to access the feature based design data in the form of a report file [Appendix B]. The report file provides the information about the feature instances, their attribute values and the entry/exit face numbers for feature attachment to other features. The detailed geometry and topology of the feature instances cannot be accessed. This is a major limitation to interpret a design as a feature tree. Recognition of a design as a feature tree is a fundamental requirement for the geometric reasoning that is necessary for CAPP.

The tolerance and manufacturing features have been proposed on the assumption that the complete design data access would be possible within a short period of time. Because of the above limitations, they cannot be used in the proposed

CAPP system. However, the experiment was worthwhile, to foresee how future modellers might provide the detailing capabilities with features.

3.10 Modified Design Representation Scheme

Due to the geometry and topology access limitations, the manufacturing intent had to be communicated with the attributes that can be assigned to individual feature classes.

The shape tolerance attributes have been assigned to the form features. On a similar basis, the location tolerance feature attributes could also have been assigned. This would result in a long list of attribute selections while modelling. Hence it has been decided to develop the process plan strategy on the basis of shape tolerance only.

Instead of using the proposed manufacturing features, it was decided to assign manufacturing attributes. The attributes, 'surfaceFinish', 'rawMaterialCondition' and 'mcNotmc' have been assigned to each form feature to communicate manufacturing intent.

It has been decided to assign a 'childOf' attribute to each form feature in the library. While inserting a feature instance, the user assigns a parent feature instance name to the 'childOf' attribute. This should enable one to interpret the feature tree from the report file.

The modified feature based design approach would enable one to interpret the design data, as a feature tree where each feature consists of geometry, tolerance and manufacturing attributes.

4. CAPP SYSTEM DEVELOPMENT PLANNING

The development of a CAPP system involves the study and analysis of effective communication strategies, between product, process and resource domain knowledge. It is a complex task because of the volume of multi-domain information involved while planning. Ideally, it is essential to involve a team of experts in the development of such a system. A team should consist of the system users, the system developers and the domain experts. Since this research has been an individual effort, it relied on the prior knowledge of the individual and the current research trends described in the literature. The final CAPP system plan was evolved through study, planning, experiment and refinement process.

4.1 CAPP System Planning Considerations

One of the main criticisms about CAPP system planning has been the lack of consideration of global objectives in planning [1, 118]. An integrated planning system for all the business functions is the ultimate goal. While the complexity of manufacturing activities makes it impossible to achieve that goal for some time, its awareness would help to plan and develop the systems with broader perspectives.

In today's changing environment, a CAPP system should be adaptable, flexible and modular to allow addition of new applications.

Globally, the product domain represents the product knowledge; the process domain represents the manufacturing process knowledge; and the resource knowledge such as tools and machine tools, etc. is represented by the resource domain.

Different types of systems are being used in industry in the above domains, especially in product and resource domains.

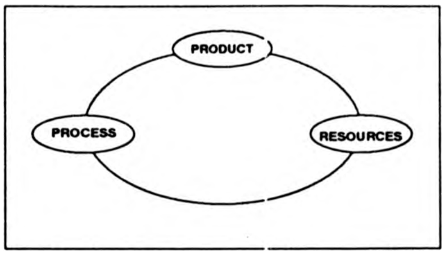


Figure 4.1 The Global Domains in Manufacturing Planning

The global issues that are relevant to these domains should be considered while planning the CAPP system. For example, a component design by feature may be a local input to an automated CAPP system. However a global issue of the component relationship to the product assembly, and the management of product model data, may influence the system planning. The global issues may pose questions to the planner, such as, 'what should be the process if a component on the assembly is not designed by features?', 'will the CAPP system be able to upgrade to the process planning of turned/round and sheet metal components?', 'is it possible to use a standard material and machinability database?', etc. The planning of a CAPP system should be flexible enough to accommodate such issues.

An application specific CAPP system will need the local representation of manufacturing process knowledge. In addition, it will also need the local representation of the product component design. It should be able to generate a feasible process plan based on these knowledge representations. However, the resources suggested by the plan will have to be checked in the resource domain by querying the resource knowledge (database). The global to local design representation of a component, will have to be generated by using the data exchange standard such as STEP or a customised interface.

4.2 CAPP System Objectives

The CAPP system to be developed, will generate a process plan for one-off prismatic parts. It should use the existing system process knowledge to suggest the process as a set of operation sequences, with the required resources. The resources will be searched in the resource database for their availability. The user interactive interfaces will be developed to add, delete and modify the process and resource knowledge.

4.3 The Programming Tool

The major problem in the planning of a CAPP system is the selection of a programming tool. The knowledge based programming tools such as LISP, PROLOG, Smalltalk, KEE etc. have been recognised to have a great potential [47]. Despite this, the majority of CAPP systems have been developed in FORTRAN and BASIC [4]. Individual research such as this, had to rely upon the available programming tools. The programming tools such as FORTRAN, BASIC, C and PASCAL are readily available with most of the academic and research institutions and hence, they become the automatic choice. It is even possible to develop an expert system application by using these languages with more programming efforts. One of the earliest expert CAPP systems, TOM was developed by using PASCAL [11].

It was apparent that the selection of a programming tool would directly affect the planning of a process domain. However, it was equally important to ensure that such a tool would be suitable for interacting with the product and resource domain. The CADDS5 system for the product domain and the ORACLE relational database management system for the resource domain, were the automatic and appropriate selections as per the current research trends.

The CADDS5 system runs on a workstation with the UNIX operating system. The C compiler is usually available on such a system. It was also possible to access the

ORACLE database management system through the network on the workstation. Once the database is designed, it could be queried and manipulated through the C programming tool (Pro C). C is the most popular programming language in engineering applications. A study of the C programming language in AI applications had revealed that it can be used for expert system development [119].

With the above considerations, the C programming language was selected to develop the CAPP system.

4.4 The Initial System Planning

The initial planning was focused on the process domain. For the various benefits that were discussed in the literature review, it was decided to use a knowledge based approach to generate a machining process plan. The initial plan of a CAPP system was conceived as in figure 4.2.

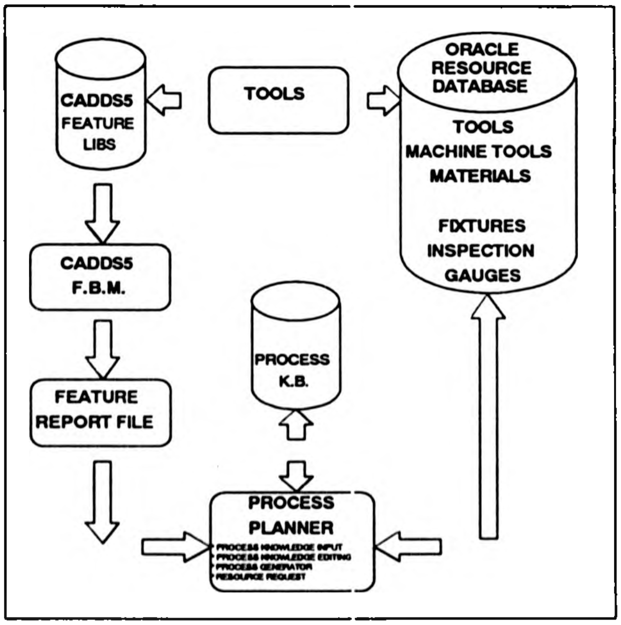


Figure 4.2: The Initial CAPP System Plan

The form feature based report file of a component model was used for local design representation in the CAPP system. A suitable interface program was developed to convert the report file into a frame like knowledge representation. This step was necessary due to the lack of direct access of feature data from the CADDS5 system.

As discussed in the previous chapter, each feature in the component design representation consists of design and manufacturing related attributes. Based on these attributes, the feature appearance conditions (attributes), for example, size range, tolerance range, etc., can be computed. The overall attributes of the feature instance were represented in the CAPP system as a manufacturing feature by a frame like structures, e.g.:

```
struct MFG_FT {
char class[30];
char library[20];
struct orientation cpl;
struct geometry {
struct lng length[20];
struct dia diameter[10];
struct ang angle[10];
struct rad radius[10];
}geom;
struct appearance {
char raw_matl_stock_type_id;
char raw_matl_stock_type[50];
char size_range_id;
int size_range[2];
char depth_range_id;
int depth_range[2];
char tol_range_id;
double tol_range;
char sf_range_id;
double sf_range
char mc_nome;
}mc;
}
```

The above representation enabled the mapping of design features into manufacturing features.

The determination of the manufacturing process was conceived as the matching process between the feature appearance conditions of a manufacturing feature instance and the process instance in the process knowledge base [141].

The process knowledgebase was developed by proposing that each process could be described as a set of operations for specific feature appearance conditions. For example:

```
struct process {
int process_id_num;
char raw_matl_stk_sype;
char size_range;
char depth range;
char tol_range;
char sf_range;
struct opr[10];
}
```

A user interface was developed, to describe the process knowledge for specific feature appearance conditions. A knowledge saving, editing and loading scheme was also developed.

The inference mechanism consisted of simple matching rules to match the feature appearance conditions between the feature and process instances. The system would select a process_id that matches with all the feature appearance conditions (attributes) of a feature instance.

The user interaction with the system is illustrated in Appendix D. The plan generation at this stage was limited to individual feature instances. The tool selection strategy was intended to be planned on the basis of the geometric attributes, and also on the available resources in the resource database. A detailed study of the resource requirements was deemed essential to develop the resource database.

4.4.1 The Resource Database

A resource database should include the information about cutting tools, machine tools, materials, machining parameters, fixtures and inspection gauges.

In a business environment, the resources are usually managed by a standard database management system. Most of the databases that have been used today are the relational ones. Instead of developing a database, a standard application database from the market was investigated. A material and machinability database from MetCut research [30] was available. However, its cost could not be justified for this research. Hence, it was decided to study and design a database for the CAPP application.

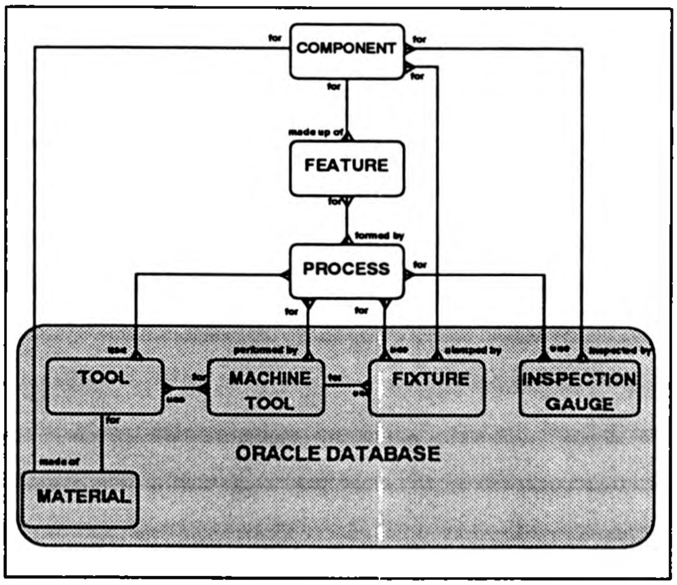


Figure 4.3 Entity Relationships

It was recognised that a detailed function, entity and data analysis was necessary to decide what and how the information about the above entities should be represented in the database [120]. An entity is anything of significance about which the information is to be held. A function analysis is necessary to identify what activities are done in the manufacturing environment, how are they done and what will be done. The function analysis enables one to identify the relationship amongst entities. Entity analysis is done to define and understand the things of significance about which the domain needs to know or hold information, and the relationship between them.

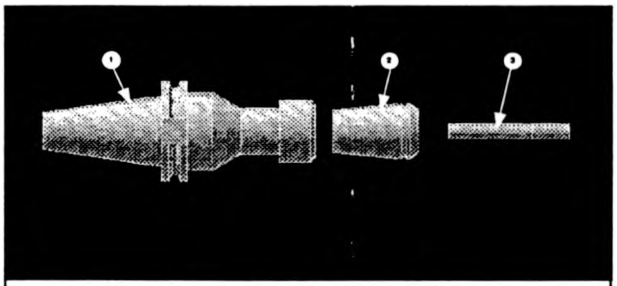
The resource entity model diagram (Figure 4.3) is a starting point for the database design. It was ensured that it represents facts, relationships and the requirements without any ambiguity. Finally, a data analysis is carried out to review the entity analysis output in the context of known data values that can be studied. On the basis of the detailed system analysis, the entities in the resource domain can be described as follows:

a. ***Process:*** A process is considered as an external entity to the resource domain. A process knowledgebase is searched to match the process that corresponds to the feature appearance conditions (feature instance attribute values) in a component. A process comprises a set of operations, which are to be carried out by a cutting tool or tools of a given size, with a desirable cutting and clearance height on a specific machine tool. The operation parameters would serve as the query input to the resource database.

b. ***Tool:*** Based on their operations, capability, types and holding components, these can be classified into a library. Cutting tools can be classified as either solid or insert type. Each of them can be used for machining operations by using one or more tool holding components such as basic holder, an intermediate, and/or an adapter.

It was observed that on a rotating tool type machine tool a basic tool shank is held by the machine tool spindle. Similarly, there is direct relationship between the cutting tool and the corresponding tool holding component (Figure 4.4).

Frequently, the similar type of cutting tools are used on number of machine tools. For example, a drill may be used on a drilling, jig boring, milling machine or a machining centre. Hence, it is possible to use the same tool assembly on different types of machine tools if the basic tool holder shank taper of the tool assembly matches the machine tool spindle taper. Such relationships can be used to design a relational database, such that there will be an optimum utilisation of tooling resources, thereby minimising the tooling inventory.



TOOL ID	HOLDER ID
1	0
2	1
3	2

Figure 4.4: Tool Component Relationships

The feature based modelling concept can be used to visualise the graphic representation of the standard resource components. It was observed that a proper resource database design should enable the feature based tool and tool assembly representation in the database. Each tool feature instance may have different attribute values in the database. For a given tool, an automated tool assembly generation and its graphic visualisation should be possible.

c. ***Material:*** A material has a relationship with a component and a cutting tool. Based on this relationship, a machining parameter relationship can be established. It was decided to use almost the same relations as described in the standard material and machinability data handbook of Metcut Research Centre. This should enable easy data mapping if required.

d. *Machine Tool:* Each machine tool uses a tool and a fixture to perform a machining process. It also represents the basic attributes in terms of size and accuracy.

e. *Fixture:* A fixture has a direct relationship with a machine tool, a component and a process. The importance of fixture planning is recognised. however, computer aided fixture planning is a special application, and it is not considered in detail in this research.

f. *Inspection Gauges:* Each inspection gauge can be related to a geometric tolerance between component features and operations. Not being a detailed part of the research like fixture, it will be represented as an independent entity.

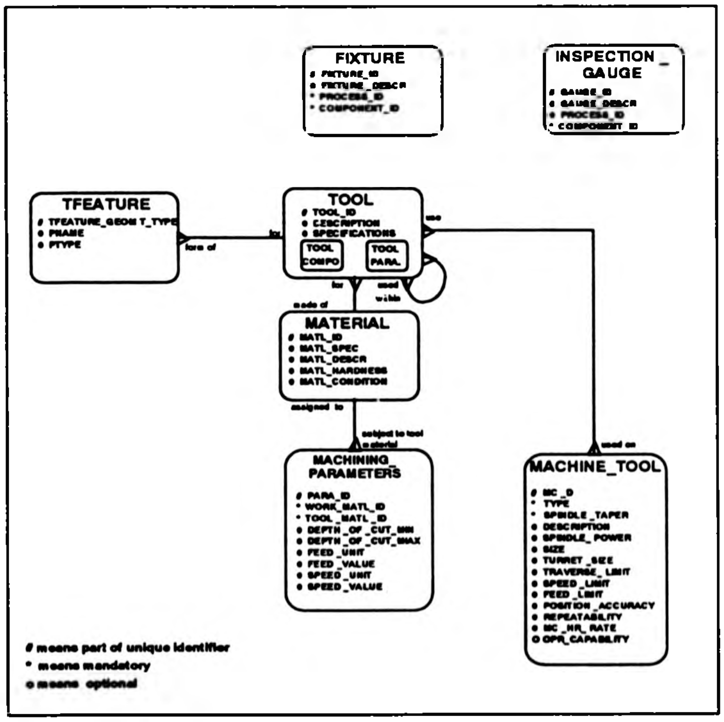


Figure 4.5: Resource Entity Relationships

After a detailed study, a unique identification key and a set of attributes for each entity were decided. A resource entity relationship diagram is illustrated in figure 4.5.

A normalisation procedure to a third normal form was carried out. Normalisation is a formalised procedure for eliminating data redundancy by progressive use of 'non-loss decomposition', which involves splitting records without losing information (Appendix E). It is based on the idea that an attribute may depend on another attribute in some way.

On the basis of the normalised entity model, the individual table designs were made. It involved choosing the primary keys, mapping the entities to tables, attributes to columns and related attributes to foreign keys.

4.4.2 Resource Database Input/Output

The user data entry and access is provided in ORACLE through SQL plus, a Structured Query Language. It was also possible to develop the resource database input/output forms by using the SQL form generator.

The SQL is a non-procedural language. That is, most statements are executed independently of the preceeding or following statements. The C language is a procedural language and based on constructs such as loops, branches and if/then pairs. Having specifically designed SQL to be a non-procedural language, and thus understanding the limitations, the originators of SQL have also explicitly designed SQL constructs, to be embedded in procedural programming languages such as C. The ProC tool provided with ORACLE, is designed to convert a C program that includes SQL statements into a C program that can access and manipulate data in an ORACLE database [121]. ProC converts the EXEC SQL statements found in its input file to appropriate ORACLE calls in an output file. The output file can subsequently be compiled, linked and executed in the normal manner for C programs.

4.4.3 Resource Queries

As discussed earlier in this chapter, the process domain was partially developed to generate a process as a set of operations. The same technique was intended to be used to derive the tool attributes such as tool type, cutting height, tool material and machine tool.

Based upon the tool attributes, a typical query to the resource database should return the detailed requirements from the resource database. For example, a tool query should return the resource details of an assembly of a tool component that is suitable for use on a specified machine tool. Optional assemblies may also be listed. All the details such as gauge length, clearance length, cutting parameters, tool component details, tool assembly graphics etc., should be automatically reported for decision making in the process or product domain.

The sample tool query procedure was developed by using C and ProC. The results are illustrated in Appendix E.

4.5 Review Of The Initial Plan

While using the overall CAPP system plan as discussed above, and developing it further, the first limitation of the system development was observed in the use of C as a programming tool. After developing the initial prototype of the process domain expert system environment, it was realised that the volume of programming effort for further enhancements would be enormous. It was observed that the programming efforts would be two fold. Firstly, to develop an expert system shell, and secondly, to develop the application by using it. Despite the effort required, such an expert system would serve the purpose of the process domain alone. The multi-domain queries such as process to resource or design to process, etc., would have to be developed in a procedural form using C. The limitations of a procedural/structural language like C in such cases could be quite significant. For example, any change to the logic of a specific procedural function/subroutine would mean debugging, compilation and linking each time.

The system also lacked the intelligent user interface. In order to keep the human expert as the key factor in the planning process, the role of intelligent user interface is increasing. Highly complex application programs can only be executed with interaction from the human planner [47]. Today, windows, frames, panes, menus, display control, etc. are becoming the standard features in every system. Such features are not part of the standard C language.

After gaining experience in design and development of entity models for resource domains, it was observed that it had quite a similarity with the frame based and object oriented knowledge representation. The same similarity was observed in the structured data representation in C while developing the process knowledgebase.

The process knowledge is symbolic in nature and hence, symbolic manipulation tools such as object oriented programming would suit better for process knowledge representation and reasoning [63]. It was noticed that STEP uses the object oriented concepts in EXPRESS [95]. The object oriented approach has also been recommended in CAPP research[47, 76, 122]. Hence, it was realised that the object oriented approach would be more appropriate and would be in tune with the current research trends.

4.6 The Object Oriented Programming Tool

The EXPRESS language was not available for use. A PC based object oriented programming tool, Smalltalk was available. With object oriented programming, the details about objects can be ignored while designing new objects, and structural properties can be assumed through property inheritance [63]. Each object is an entity combining both procedural and declarative knowledge associated with some particular concept.

A common notion in all object oriented languages is the uniform message passing between conceptual objects. Message passing is a form of indirect procedural call, which activates procedures associated with objects.

The message passing enables data abstraction. This principle groups together all the procedures that are associated with a given data type, enabling changes to the underlying implementation, without making changes to the calling programs. This is particularly important in implementing effective change mechanisms in applications such as concurrent engineering.

4.7 Final CAPP System Plan

It has been decided to use the same framework as the initial plan, but with the object oriented approach for the CAPP system knowledge representation of the product, process and resource domains.

The product domain is planned to be represented as an object hierarchy of product, subassembly, component and feature. The product, subassembly and component objects are planned to be defined on the basis of CADDS5 assembly design functionality. The form feature taxonomy is planned to be represented as per the CADDS5 feature library that is STEP based, but modified to communicate the manufacturing intent. It is also planned to develop a procedure to read the CADDS5 feature report file and transpose it into a component feature instance tree in the CAPP system.

The process domain is planned to be represented as an object hierarchy of various operations, such as drilling, boring, milling, etc. It is thought that geometric reasoning procedures could be developed to segregate features or features clusters into various machining set-ups. Based on the feature instance appearance condition, a rule based procedure could be implemented to suggest a machining process as a set of operations.

The resource domain is planned to be represented as per the resource entity model. The entities would be represented as object hierarchy, as per the entity relationship diagram. It is thought that the set of procedures could be developed to search the resources while suggesting a process.

The detailed development of the CAPP system with the object oriented approach is discussed in the next chapter.

5. CAPP SYSTEM DEVELOPMENT

This chapter discusses the CAPP system development by the object oriented language, Smalltalk. The first step in the development involved the abstract knowledge representation schemes for product, process and resource domains. The representation schemes use the definition of object hierarchies. The specific knowledge can be added by creating an instance of similar objects. The input of specific knowledge has been proposed by developing a good user interface that should also be suitable for the concurrent engineering environment.

Based on the knowledge that is represented in the system, the development of geometric and process reasoning schemes have been explained. Finally, the process plan report generation and the concluding observations on the proposed development technique have been discussed.

5.1 Overview of Smalltalk

In Smalltalk terms (Figure 5.1), an *object* is a package of information (data) and its manipulation (methods) [124, 125]. A class is a description of one or more similar objects. The *methods* are the library of self-contained subroutines unique to each class that describe a sequence of actions to be performed upon receiving a message. An *instance* is an object that is described by a particular class.

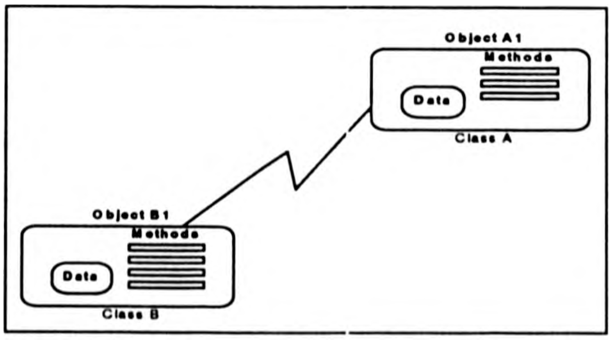


Figure 5.1: Object Concepts

The object's internal variables are called *instance variables*. They are data containers and can themselves contain other objects. They are used to describe the object attributes. The *class variables* are the global variables accessible to all instances of a class.

Objects have the characteristics of *encapsulation*, that is, data inside an object is protected from the outside world. The only way to access an object's data is by sending a message to that object. Upon receiving a message, the object activates the corresponding method and returns the result to the sender. Objects also have the characteristic of *inheritance*. A class may have several sub-classes; a subclass may inherit all or part of the properties from its super-class. Different objects can respond uniquely to the same message. This characteristic is known as *polymorphism* and gives the advantage of not having to remember the unique vocabulary for each class.

One of the characteristics of object oriented programming is the persistent data to the programming environment such that the data elements exist after the program terminates. This is possible by making a provision to hold these data instances, otherwise the garbage collector would remove the data instances after the program termination. This results in a system with the power of object oriented programming and the features of databases [108].

In Smalltalk essentially everything is an object. The system provides the hierarchy of system classes and subclasses along with the standard methods. A typical application development involves the initial planning similar to the entity relationship diagram discussed in the previous chapter. The abstract knowledge representation is usually done by defining objects and/or the object taxonomies. The user interface for system communication is designed with windows panes and menus. The application can be developed in a modular fashion by implementing various methods.

5.2 Product Knowledge Representation

The product design representation, discussed in chapter three, has been used as the basis for product knowledge representation. The product knowledge is represented as the object hierarchy of product, sub-assembly, component and feature classes. The Product class abstraction can be described by its *name*. Optionally, one may wish to record its *specifications* and a *group* to which it belongs. The product may be a *parentOf* subassemblies and components. Based on the abstraction, a class Product is defined as:

```
Object subclass: #Product
instanceVariableNames:
'name parentOf specifications group'
```

A subassembly is described as a subclass of a product and therefore it will inherit all the attributes and methods from its superclass, Product. In addition, a subassembly could be described as *childOf* a product. Hence, a class SubAssembly can be defined as:

```
Product subclass: #SubAssembly
instanceVariableNames:
'childOf'
```

The attributes *parentOf* and *childOf* will be used to establish the relationship between Product and SubAssembly class instances and they will also be used to relate components to assemblies and features to components. The parent child relationship is also referred to as aggregation.

A component has been defined as a subclass of SubAssembly. In addition to the inherited attributes and methods, a component is described by its *material*, manufacturing *process* and optionally the surfaceFinish method (e.g. Heat treatment, Anodise, Electroplate, etc.)

```
SubAssembly subclass: #Component
instanceVariableNames:
'material process surfFinish'
```

A component is described by its features. Based on CADDS5 STEP taxonomy, a class Feature is defined as:

```
Component subclass: #Feature
instanceVariableNames:
'library, datumPlane, origin, orientation, workpiece, representation, operation, application,
rawMatlCondition, mcNotmc'
```

As discussed in chapter three, the attributes *rawMatlCondition, surfFinish* and *mcNotmc* have been used to communicate the manufacturing intent.

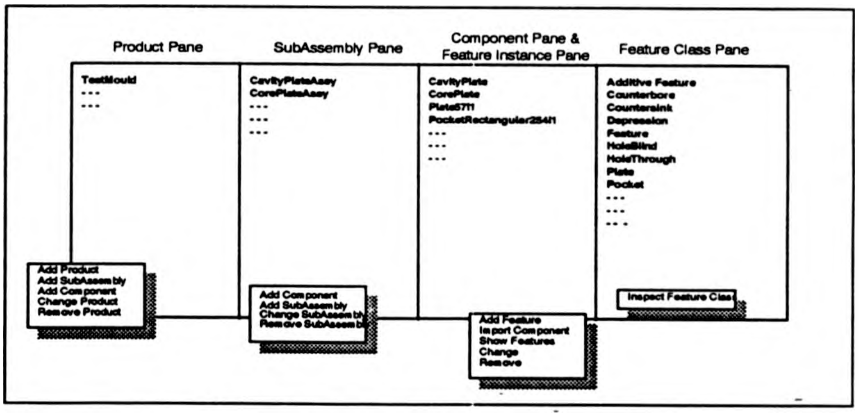


Figure 5.2: The Product Definition User Interface

The entire CADDS5 STEP based Feature taxonomy scheme (Appendix A) has been defined by the same concept as above. For example, a class HoleBlind is defined as,

```
Depression subclass: #HoleBlind
nstanceVariableNames:
'diameter depth'
```

The object definitions described above represent the abstract knowledge of the product domain. The user interface has been designed to add the specific knowledge (product data). Windows are designed to display the user defined product, subassembly and component instances. The feature classes are displayed in a separate window. Specific window pane menus are designed for the user commands (Figure 5.2).

In order to represent the feature based component design, two types of user interfaces have been developed. The local user interface, 'Add Feature', enables the local feature based design representation i.e. within the CAPP System. The local user interface can be used for system development and testing purposes. The other interface, 'Import Component', reads the CADDS5 feature based design report file into a feature based design representation in the CAPP System.

5.3 Process Knowledge Representation

The knowledge about the machining processes and their capabilities has been represented by a mixed approach of frames and rules which, in Smalltalk terms, are objects and methods respectively. The process knowledge was considered mainly at the universal level i.e. the knowledge available from the textbooks and handbooks. The knowledge was represented as an object hierarchy. The process knowledge was not mixed with the specific shop or the machine tools or cutting tools. That type of knowledge will be discussed under resource knowledge representation. The following discussion illustrates the process object hierarchy.

The machining process at the highest level is thought of as a set of operations which are performed on one or more manufacturing feature instances of a component within a particular set-up by using a particular fixture. The feature *processMode* can be manual or automated depending upon the method of process planning. The class MfgProcess can be defined as:

```
Object subclass: #MfgProcess
instanceVariableNames:
'processMode setUpNo oprNo fixtureForOpr mfgFtInstances'
```

The Feature process (FtProcess) is defined as a subclass of MfgProcess and represents the attributes that are common to a machining operation.

```
MfgProcess subclass: #FtProcess
instanceVariableNames:
'ftOprNo oprType oprName childOf parentOf
 preMcState oprStock toolForOpr mcForOpr reqdPosAccy
 speed feed depthOfCut cutTime moveTime totalOprTime
 precedsOpr'
```

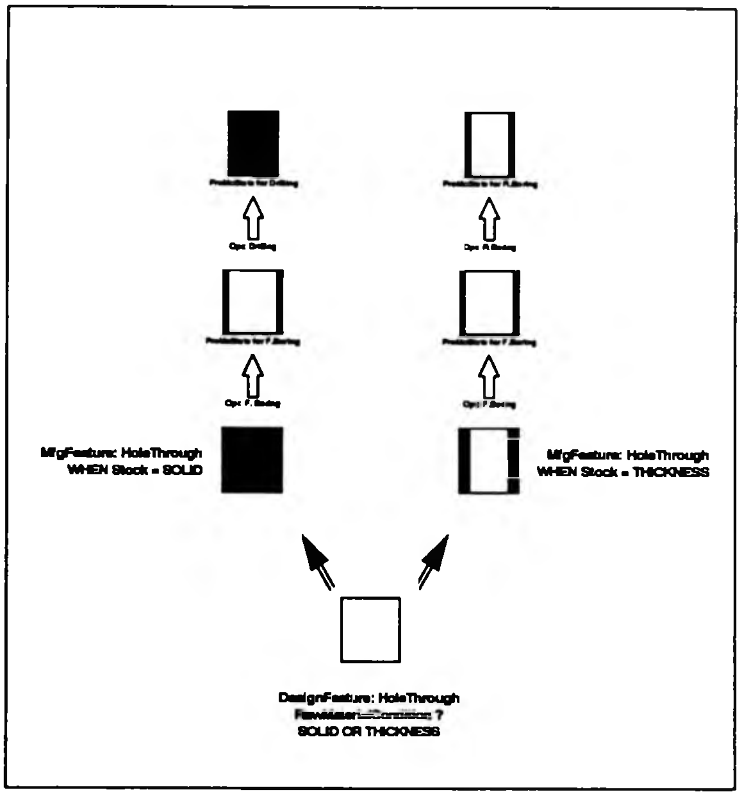


Figure 5.3: Operations and Pre-Machining State

The *ftOprNo* represents the operation number. The *oprType* could be roughing or finishing. The *oprName* describes the operation class name, e.g. TwistDrilling, Reaming, EndMilling, etc. The *childOf* and *parentOf* attributes describe the operation relationships. They are used to capture feature process as an operation tree. A machining process for a feature instance has been planned to be generated from the finished state to the raw state. Thus, a finishing operation will always be the root node of the operation tree (Figure 5.3). The *preMcState* holds the feature stock attribute for the preceding operation. The *oprStock* indicates the stock to be removed in the operation. The *toolForOpr* describes the tool class, maximum, actual and minimum tool diameter, and minimum tool height. The *mcForOpr* describes the valid machine tool classes for the operation. For example, an EndMilling operation for the Pocket feature can be performed on a Milling machine, a CNC Milling or a Machining Centre but not on a Radial Drilling machine or a Jig Boring machine. The *reqdPosAccy* describes the acceptable positioning accuracy of the machine tool to perform the operation.

The *speed*, *feed*, *depthOfCut*, *cutTime*, *moveTime* and *totalOprTime*, describe the cutting parameters and various times to perform the operation. The *precedsOpr* describes the operation precedence in relation to the machining operations of the parent features.

The HoleMakingProcess represents the abstraction of all the hole making processes:

```
FtProcess subclass: #HoleMakingProcess
instanceVariableNames:
'holeRefPt positionPt approachPt plungePt dwellTime returnPt
clearPt'
classVariableNames:
'processKnowledgeRef'
```

The instance variables describe the tool positions in the hole machining operations. The class variable, *processKnowledgeRef* holds the hole roughing and finishing knowledge instances of various hole making processes. The Hole Finishing Knowledge can be described as:

```
Object subclass: #HoleFinishingKnowledge
instanceVariableNames:
'mfgEngineer knowledgeRefNo processClass diaRange tolRange
depthRange sfRange reconFinStock'
```

It describes the capability of the hole making process in terms of diameter, tolerance, depth, surface finish and recommended machining stock for finishing. The Hole Roughing Knowledge is described as a subclass of Hole Finishing Knowledge:

```
HoleFinishingKnowledge subclass: #HoleRoughingKnowledge
instanceVariableNames:"
```

The inherited attributes from Hole Finishing Knowledge can be used to describe the hole making process capability for roughing.

The HoleMakingProcess is classified into a Hole Creating and Enlarging Process. The HoleCreatingProcess ended up in further classification into Drilling, TwistDrilling, CentreDrilling, etc. Similarly, the HoleEnlargingProcess is classified into Boring, Reaming, CounterBoring, CounterSinking, etc.

Similar concepts have been used to describe Milling and Grinding Processes. Figure 5.4 illustrates a typical machining process classification that has been used in organising the process knowledge.

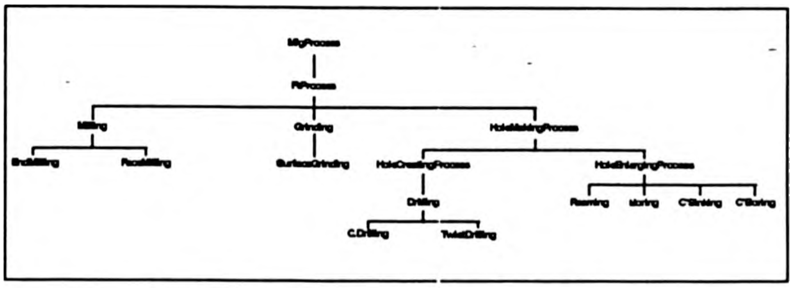


Figure 5.4: Machining Process Classification

The complexity in applying a particular process to produce a particular shape varies from process to process. For example, it is relatively easy to represent knowledge instances for hole making processes under which a process can be applicable. The shape producing capability of these processes is predictable. However, this may not be the case in other types of processes, such as milling. Under such circumstances the various methods that comprise the set of rules can be developed. These methods may represent both the universal and the experience based process knowledge to machine a particular shape. The proposed process reasoning techniques are discussed later in this chapter.

A user interface has been developed to display and inspect the machining process classes. The process window pane menu has been designed to provide the options for adding the process knowledge, and for generating the manual or automated process plans (Figure 5.5).

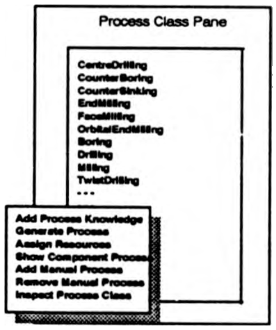


Figure 5.5: The Process User Interface

5.4 Resource Knowledge Representation

The resource knowledge representation comprises the knowledge about cutting tools, machine tools, fixtures, materials and machining parameters. The entity relationship diagram, discussed in chapter four has been used as the basis for resource object class definitions. However, multiple entities of that diagram are combined into one or more objects for convenience. The tool hierarchy superclass is defined as:

```
Object subclass: #Tool
instanceVariableNames:
'name toolDia shankType material toolSpec toolParameters
 toolEdgeType insertSpec insertGrade holderRef workApplication
 vendor cost toolQty'
```

The Tool subclasses are defined as per the types and characteristics of the tool, e.g. Bore, CounterBore, Drill, Mill, EndMill, etc.

The machine tool superclass is defined as:

```
Object subclass: #MachineTool
instanceVariableNames:
'name type description controlSystem spindleTaper spindlePower size traverseLimit speedLimit operations
positionAccuracy repeatability machineHourRate'
```

The entire machine tool class hierarchy is illustrated in Figure 5.6.

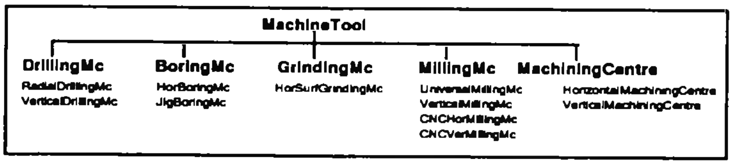


Figure 5.6: Machine Tool Classification

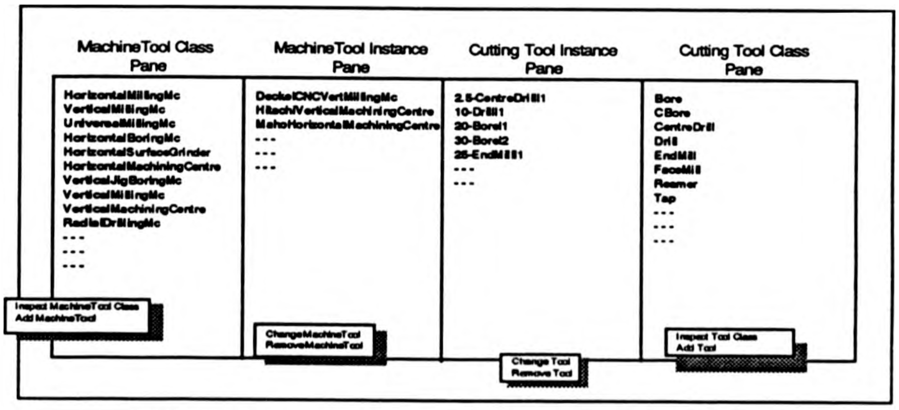


Figure 5.7: The Machine and Tool Resource User Interface

Like process classes, the machine tool and cutting tool class definitions are based on the universal knowledge. The specific knowledge is represented by the class instances. The user interface has been developed to inspect, add or delete the instances (Figure 5.7).

The fixture class has been defined by the same method that was discussed in chapter four while planning the resource entity model. The Fixture class is defined as:

```
Object subclass: #Fixture
instanceVariableNames:
'name description fixtureRefNo usedOn'
```

The user interface has been designed to add, remove and inspect the feature instance (Figure 5.8). It was observed that the fixture links the component machining in a particular setup on a particular machine tool. Hence, the 'Assign Fixture' command has been developed to assign a fixture instance for machining the component in the set-up on the machine tool. The ***usedOn*** attribute records the set of component, set-up and machine tool attributes, which are referred to by the 'Assign Resources' method.

The materials and machining parameters have been defined on the material and machinability database principles of Metcut Research. The class Material is defined as:

```
Object subclass: #Material
instanceVariableNames:
'materialGroupNo groupName specifications description hardness
conditionCode conditionDescr machinability mcParameters'
classVariableNames: 'MatInst'
```

The class Material represents the work material. The tool material is represented as a subclass of Material:

```
Material subclass: #ToolMaterial
instanceVariableNames:
'coatingStatus grade'
ClassVariableNames:
'ToolMatInst'
```

The user interface has been developed to represent the specific material knowledge and also to attach the machining parameters to the work material (Figure 5.8). The machining parameter class is defined as:

```
Object subclass: #McParameter
instanceVariableNames:
'toolMaterial toolClass depthOfCut feedUnit feedvalue
speedUnit speedValue
classVariableNames:
'McParaInst'
```

The entire resource knowledge representation is independent of product or process knowledge. In a concurrent engineering environment, the resources may be referred to by a design or manufacturing team. The process reasoning methods derive the valid machine or cutting tool for the operation based on the universal knowledge. A subsequent command, 'Assign Resources', would refine the initial plan by searching the available resources and selecting them if available, or generate a warning message. The planner may then decide to define the new resources.

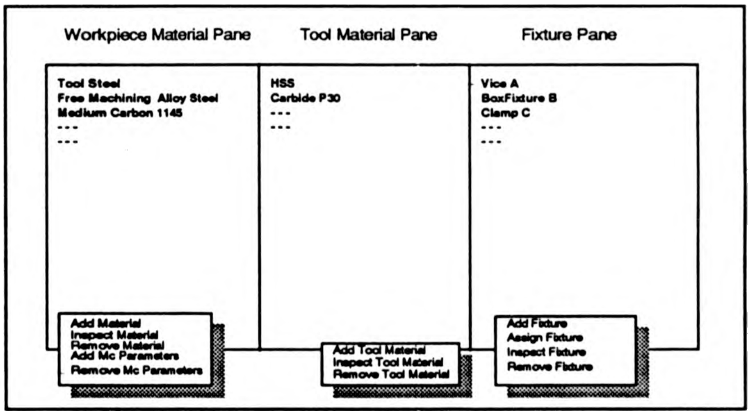


Figure 5.8: Material Resource User Interface

5.5 The System Communication Interface

The system user interface development is the main constituent of the CAPP system. The display and menu design of product, process and resource knowledge representation is controlled by the object class, ProductCom. It consists of various display panes. Each pane has been associated with the specific object class as discussed in section 5.2, 5.3 and 5.4. The pane display size, the corresponding menu options and the pane digitised cursor text have been controlled through the number of specific methods that are developed for each pane. The CAPP system window is an instance of a class ProducCom. For example, the following command at system level will open the CAPP system window:

```
Test := ProductCom new.
Test open.
```

The command 'Test open' at the system level will display all the CAPP system panes simultaneously. This enables the CAPP system user to work with the system without bothering with the complexity of the system design.

5.6 Geometric Reasoning Technique

The first step in process planning is obviously to understand the component geometry. The term geometric reasoning has been used in CAPP research to interpret the geometry, which helps in deriving a suitable manufacturing process. The term is used to mean any reasoning process related to the component geometry: recognising manufacturing features, identifying feature relationships, extracting feature dimensions, finding feasible approach directions for machining and determining geometric constraints [126].

Since the design by feature approach has been used and only higher level data could be extracted from the model, the various feature recognition techniques that are discussed in the literature review, are not used. The CAPP system feature interpretation is based on higher level data abstraction of a feature based model. The manufacturing feature for the corresponding design feature could be reasoned from the manufacturing attributes of the design feature. The principal attribute that describes the manufacturing feature is the raw material condition (Figure 5.3).

The raw material condition attribute of a feature instance that can either have a 'Solid' or 'Thickness' value, may be interpreted differently in terms of additive and subtractive features. Hence, certain assumptions were made to understand the raw material condition of a feature. In terms of subtractive features the raw material condition 'Solid' was assumed as the entire feature volume to be removed by the machining process. The same attribute value for the additive feature was assumed to imply the projected stock along the Z axis upto the highest face when an additive parent exists (Figure 5.9).

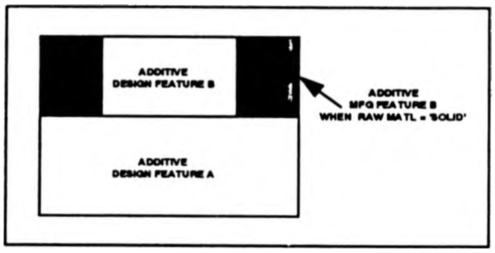


Figure 5.9: Interpretation of the Additive Feature for 'Solid' Stock

The following rules were assumed to interpret the raw material condition of additive features:

General Stock definition:

Stock := #('Stock Type' 'ZStock' 'SideStock').

1. If feature is a Primitive (e.g. Plate) and raw material condition is 'Solid', then Stock at: 2 & 3 is 0. Hence, no machining is required.

2. If feature is a Primitive and raw material condition is 'Thickness' and the feature has no Protrusion child (e.g. Tab_Rectangular), then ZStock = 'Thickness'. Hence, machining will occur on the top face of the feature.

3. If feature is a Primitive and raw material condition is 'Thickness' and the feature has Protrusion child, then the Stock at: 2 = 'Thickness', and at: 3 = 0. Hence, machining will occur on the top face of the feature.

4. If feature is a Protrusion and raw material condition is 'Solid', and the feature has no Protrusion child, then the SideStock is computed from parent child geometric attributes and the ZStock = 2 mm (assumption). Hence, the feature machining will occur on side faces and the top face.

5. If feature is a Protrusion and raw material condition is 'Solid', and the feature has Protrusion child, then the ZStock will be zero and the SideStock will be computed as per 4.

6. If feature is a Protrusion and raw material condition is 'Thickness' and a feature has no Protrusion child, then the SideStock is equal to 'Thickness' and the ZStock = 2 mm.

7. If feature is a Protrusion and raw material condition is 'Thickness and a feature has Protrusion child, then the SideStock and ZStock is equal to 'Thickness'.

The above rules have been used to develop a method, 'getAdditiveFtStock', which returns the appropriate stock attributes. The process reasoning is based upon the stock attributes.

The component feature tree helps in reasoning the feature relationships. Various methods have been developed to interpret the feature relationships such as, show child features; show parent feature; show component to which the feature instance belongs; show leaf instances (a feature instance that has no child instances) of the feature; etc. These methods help in reasoning the set of features to be machined together. For example, if the Hole feature has a subtractive parent such as Counterbore or Pocket and the Hole needs a drilling operation, then the drilling operation should have a precedence over any operation that has to be performed on the subtractive parents. The influence of feature relationships on process reasoning is also referred to as geometric constraint [45].

The feature dimensions that describe its shape, shape tolerances, origin, orientation, etc., were automatically available as feature instance attribute values. Simple methods such as diameter, tolDiameter, origin, datum, etc. enabled direct access to the feature instance attribute values. Specific methods have been developed to enable computation of indirect attributes such as area of a pocket, etc.

The external access direction to machine the features has been derived from the orientation plane or the datum in which they were created. For example, the Z axis of the hole features will always be along its axis; for a pocket, normal to the pocket cross-section (along the depth); for a boss, along height, etc.(Appendix A). Although any orientation plane could be used in building a model, the most commonly used orientation planes are, TOP, FRONT, RIGHT, REAR, LEFT and BOTTOM. Only these planes are considered for process reasoning at the moment. They are referred to as datum planes. They also imply a unique set-up number, from one for TOP to six for BOTTOM respectively.

The following steps are followed for the geometric reasoning of the component feature tree:

1. Read all the additive feature datum planes in the component feature tree.

2. For each datum plane, read all the additive feature instances.

3. From all the additive instances in step 2, select the instance that has the highest Z value. Send mfgProcess message to the instance for simulteneous geometric and process reasoning to generate feature machining process.

4. Search for subtractive children of the additive feature instance in step 3. Send mfgProcess message to the subtractive instance for simultaneous geometric and process reasoning. Traverse the subtractive feature instance tree from root to leaf node direction so that the parent process is generated first.

5. Repeat steps 3 and 4 for the next highest Z value additive feature and so on until all the additive feature instances are machined.

6. Repeat step 3, 4 and 5 for all the datum planes.

The mfgProcess method comprises a technique of simulteneous geometric and process reasoning. The geometric reasoning part interprets the manufacturing view of a feature from the raw material condition (refer to Figure 5.3), and the process reasoning part attempts to match that view with the process knowledge. The process generation methods for the feature instances are discussed in the following section.

5.7 Process Reasoning Technique

As discussed in section 5.3, the machining process abstract knowledge is organised in the form of a process object hierarchy. The real knowledge is added by knowledge instances for hole making processes and by rules for other processes. The process reasoning is based on the methods that have been developed for machining the feature classes. The methods are developed in a backward planning approach (Figure 5.3).

Like conventional programming languages, there is no standard practice to prepare flow diagrams in object oriented programming. The Smalltalk/v286 programming user guide illustrates the application logic by listing various methods that are developed with various objects [125]. A similar technique has been used in this chapter to illustrate the CAPP system development. However, it was felt necessary to illustrate the process reasoning technique by using the flow diagram (Figure 5.10). It is worth mentioning that the process reasoning technique is not a single program. The process reasoning is achieved by a combination of methods that are developed with the feature objects and process objects.

The feature machining method sends a message to the appropriate process classes. The process is generated if the message returns with the process instance, otherwise the message returns with nil. For example:

```
Object: a HoleThrough
Method: mfgMethod

(self mcNotmc at:1) == $Y
            ifTrue:[HoleMakingProcess holeThrough: self].
```

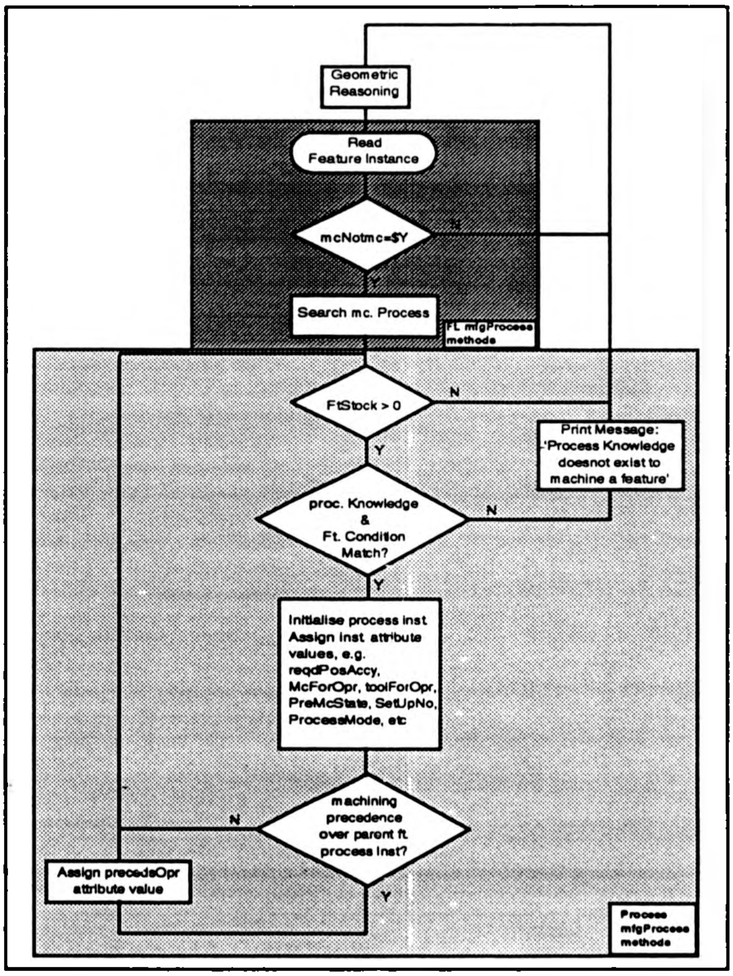

Geometric Reasoning
Read Feature Instance
mcNotmc=$Y
N
Y
Search mc. Process
Ft. mfgProcess methods
FtStock > 0
N
Y
Print Message: 'Process Knowledge doesnot exist to machine a feature'
proc. Knowledge & Ft. Condition Match?
N
Y
Initialise process inst. Assign inst. attribute values, e.g. reqdPosAccy, McForOpr, toolForOpr, PreMcState, SetUpNo, ProcessMode, etc
machining precedence over parent ft. process inst?
N
Y
Assign precedsOpr attribute value
Process mfgProcess methods

Figure 5.10: Data Flow Diagram for Process Reasoning Technique

Object: HoleMakingProcess
Method: holeThrough: aHole

```
"Search for hole finishing possibility by Boring, Reaming or
Drilling"
(proc := Boring genFinProcess: aHole) isNil
        ifFalse:[ aHole addProcess: proc].
(proc := Reaming genFinProcess: aHole) isNil
        ifFalse:[ aHole addProcess: proc].
(proc := TwistDrilling genFinProcess: aHole) isNil
        ifFalse:[ aHole addProcess: proc].
```

```
"Search the hole roughing possibility for the finishing"
aHole process do:[ :finProcess |
(proc := finProcess class genRfProcess: finProcess feature: aHole) isNil
            ifFalse:[finProcess parentOf: proc].
```

The process class methods interpret the feature appearance condition from the feature instance and attempt to match it with the process knowledge. The process is recommended if the matching of knowledge occurs. For example:

Object: Boring
Method: genFinProcess: aHole

```
featureStock := aHole holeStockValue
(feature at:2) > 0
            ifFalse:[Test answer: aHole name,' no stock for finish Boring'.
                        ^nil].
(mcStock := Boring matchFinKnowledge: aHole) isNil
            ifTrue:[Test answer: aHole name,' can not be finished by finish Boring'.
                        ^nil].
            ifFalse:["Initialise a process"
                  Proc := Boring new.
                  preMcStock at:2 put: (featureStock at:2 ) - mcStock.
                  proc initialize: aHole;
                        oprType: 'Finish';
                         oprStock: mcStock;
                     getFinTool: aHole;
                     preMcState: preMcStock.
                          ^proc]
```

Object: HoleEnlargingProcess
Method: matchFinKnowledge

"Search the HoleFinishingKnowledge instances for matching with the feature appearance condition. Return recommended stock or nil"

```
HoleMakingProcess procKnowledge do:[:each |
                (aHoleFeature diameter >= (each diaRange at:1) and:[
        aHoleFeature diameter <= (each diaRange at:2)])
  ifTrue:[(aHoleFeature tolDiaRange >= (each tolDiaRange at:1) and:[
                          aHoleFeature tolDiaRange <= (each tolDiaRange at:2)])
  ifTrue:[(aHoleFeature surfFinish >= (each sfRange at:1) and:[
                 aHoleFeature surfFinish <= (each sfRange at:2)])
  ifTrue:[(aHoleFeature depth >= (each depthRange at:1) and:[
                          aHoleFeature depth <= (each depthRange at:2)])]
  ifTrue:[^each recomFinStock]]]]]
```

Object: a Boring
method: getFinTool: aHole

```
getFinTool: aHoleFeature
      "selects the tooltype, tool dia and minimum
       tool height for finish boring"
```

```
  |toolValues minDepth|
   toolValues := OrderedCollection new.
   (minDepth := approachPt - plungePt) positive
      ifFalse:[minDepth negated].
   toolValues add: 'Bore';
            add: aHoleFeature diameter;
            add: aHoleFeature diameter;
            add: aHoleFeature diameter;
            add: minDepth.
toolForOpr := toolValues asArray.
```

The class method inheritance helps in process initialisation:

```
Object: a HoleEnlargingProcess
method: initialize: aHole

self reqdPosAccy: (HoleMakingProcess getTolRange: aHole)/2.
mcForOpr := #('BoringMc' 'DrillingMc' 'MillingMc' 'MachiningCentre').
super initialize: aHole

Object: a HoleMakingProcess
method: initialize: aHole

(aHole isKindOf: Passage)
            ifTrue:[self thruHoleCycleLocs: aHole]
   ifFalse:[ self blindHoleCycleLocs: aHole]
super initialize: aHole

Object: a FtProcess
method: initialize: aFeature

parentOf isNil
            ifTrue:[parentOf := OrderedCollection new].
self addSetUpNo: (aFeature getSetUpNo);
   processMode: 'Auto' asSymbol;
               oprName: self class.
```

The above technique has been used to generate a process operation sequence for individual features. The process generation tree is traversed from top to bottom, i.e. the parent processes are generated first. The knowledge of parent process helps in deciding the operation precedence of a child process. For example, if the HoleThrough is a child of a Counterbore and the Counterbore is finished by a CounterBore tool, then the Drilling or rough Boring operation for HoleThrough must have a precedence over the Counterboring operation.

Object: a HoleMakingProcess
Method: precedence: aHoleFeature

```
"Checks the parent feature instance machining process and returns the parent origin or nil"

parent := aHoleFeature childOf.

"Add the precedence rule for the Counterbore finishing"
(parent isKindOf: Depression)
        ifTrue:[parent rawMatlCondition = 'Solid'
        ifTrue:[parent procOpr = 'Finish'
        ifTrue:[self isKindOf: Drilling
        ifTrue:[precedsOpr := 'Parent Operation'.
                ^(parent origin].
                (self isKindOf: HoleEnlargingProcess) and:[
                self oprType = 'Rough'])
        ifTrue:[precedsOpr := 'Parent Operation'.
                ^(parent origin]]]].

"Add precedence rule for Pocket roughing"

((parent isKindOf: Pocket) and:[
  parent rawMatlCondition = 'solid'])
ifTrue:[self isKindOf: Drilling
ifTrue:[precedsOpr := 'Parent Operation'.
        ^(parent origin]]
```

Similar precedence rules may be written for machining closed tolerance feature instances.

The process options are also generated. For example, the hole may be machined by Reaming, Boring or Drilling. It should be noted that the available resources are not examined while generating the process or process options. Later, a technological process based on the universal knowledge has been used to search the available resources.

The process reasoning methods have been developed for HoleThrough, HoleBlind, CounterBore, CounterSink, Pocket, Plate, TabRectangular and BossCircular. Similar techniques can be applied for the other feature classes in STEP taxonomy such as, Slot, Notch, etc.(Appendix A).

5.8 Manual Planning Technique

It is obvious that the generative process planning technique, discussed above, can only be used on the feature based component design. In a concurrent engineering environment, the situation may arise that the design representation of a component has not been done by features and still the manufacturing process plan that has been considered by the product engineering team needs to be assigned to the component. Such a process plan may be either brief or detailed.

It was observed that two different approaches could be used for such an application. The first would be to expand the current system to accommodate the variant process planning capability. The second approach would be to develop the user interface to enable the user to interactively assign the process attributes. The first approach was not in line with the objectives of current research. The second approach was feasible because of the technique used in the design of the CAPP system user interface. The simultaneous display of product, process and resource domain knowledge representation has enabled digitised selection of the existing objects for inspection, deletion or addition. The digitised object selection technique has been used to assign the machining process instances to the component instance (Figure 5.11). The user interactive mode guides the user for specific attribute selection. The typical methods that are developed for such applications are illustrated below:

```
Object: a ProductCom
Method: initYourProcess

(curProduct isMemberOf: Component) and:[
(replyStream isKindOf: MfgProcess)])
            ifTrue:[procInstance := replayStream initYourProcess.
                        curProduct addProcess: procInstance].
Object: a MfgProcess
Method: initYourProcess

"Initialise manual process"
processMode := 'Expert' asSymbol "Expert means manual"
(setUpNo := Prompter
prompt: 'Set-Up Number ?'
defaultExpression : '1') isNil
```

```
ifTrue:[^self].
(fixtureForOpr := Prompter
prompt: 'Fixture For Operation ?'
default: '') isNil
ifTrue:[^self].

(mfgFtInstances := prompter
prompt: 'Batch Quantity ?'
defaultExpression: '1') isNil
ifTrue:[^self].
```

Object: a FtProcess
Method: initYourProcess

```
(ftoprNo := Prompter
prompt: Operation Number?
defaultExpression: '1')isNil
ifTrue:[^self].

(mfgFtInstances := Prompter
prompt: Number Off?
defaultExpression: '1')isNil
ifTrue:[^self].
```

Object: a EndMilling
Method: initYourProcess

```
super initYourProcess.

toolForOpr := #('EndMill' 50 'Dia' 5 'minHgt')
(toolForOpr  := Prompter
prompt: EndMill Diameter?
defaultExpression: '25')isNil
ifTrue:[^self].

(cuttingLength := Prompter
prompt: 'EndMill Minimum Tool Length ?
defaultExpression: '75') isNil
ifTrue:[^self].

toolForOpr at:3 put: toolDia;
                at:5 put: cuttingLength.
mcForOpr := #('MillingMc 'MachiningCentre').

(oprName := Prompter
prompt: 'Operation Description ?
default: 'End Mill') isNil
ifTrue:[^self].

(totalOprTime := Prompter
prompt: 'Estimated operation time (min) ?
defaultExpression: '1')isNil
ifTrue: [^self].
```

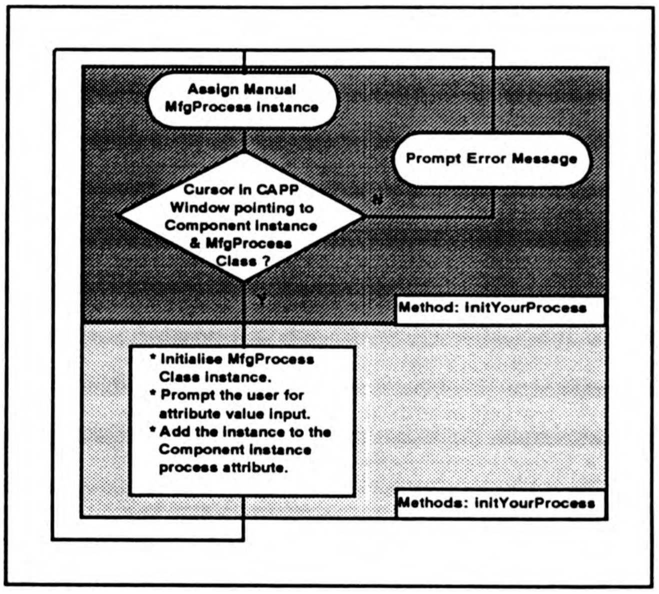

Assign Manual MfgProcess Instance
Prompt Error Message
Cursor in CAPP Window pointing to Component Instance & MfgProcess Class ?
N
Y
Method: InitYourProcess
* Initialise MfgProcess Class Instance.
* Prompt the user for attribute value input.
* Add the Instance to the Component Instance process attribute.
Methods: InitYourProcess

Figure 5.11: Data Flow Diagram for Manual Process Assignment

Similar to EndMilling, the methods have been developed for other process classes. The inheritance characteristics of the object oriented approach helped in the rapid development of the manual planning technique.

The user interactive mode assigns the set of process instances to the component process attribute. The manual process plan can be inspected, displayed, printed or saved in a file. The following section discusses the CAPP system process plan report generation.

5.9 Process Plan Report Generation

As discussed in section 5.7, the generative process reasoning technique generates and assigns the process instances to the process attribute of various feature instances. The process plan report generation involves the systematic output of various process instance attribute values.

A multipurpose text display pane was designed for displaying various types of system output. For example, the command 'Show Allchildren' displays all the child feature instances of a requested instance. The 'Generate Process' command prints various messages of a process reasoning scheme. These messages help in studying the process reasoning technique as well as bug fixing. The same text display pane has been used for process plan report generation.

The text display was designed with the menu command options such as restore, print, save and show-it. Most of these are standard Smalltalk text pane features and can be incorporated with ease. The 'restore' command erases the current text display. The 'print' command generates the hardcopy printout of the text display. The 'save' command saves the text display as a text file. The 'show-it' command executes the Smalltalk command. It can be used as a calculator. The standard text editing commands such as cut, paste and copy, have also been provided to edit the process plan if necessary.

The process plan report generation is activated by digitising the component instance and selecting the command, 'Show Component Process'. Various methods have been developed to display the process plan for each machining set-up. This plan comprises the display of operation number, operation name, operation type, tool for operation, machine tool for operation and operation precedence. The NC machining cycle location output capability has also been developed. The hole processing and pocket cycle locations can be displayed as a process plan.

The following example illustrates the typical process plan report display:

```
Automated Process Plan
Component name: NewComp
SetUp No: 1
Fixture For SetUp: MachineVice

Machining Process for HoleThrough11
Opr No: 1; Opr Name: Rough CentreDrilling
Tool For Opr: ('TwistDrill' 3.5 4.0 4.5 6.0)
Machine For Opr: 'BokoVerticalMachiningCentre11'
```

```
Rough CentreDrilling Cycle Locations...
PositionPt: (100 75 50)
ApproachPt: 53.0
PlungePt  : 44.0
RetractPt : 58.0
Dwell(Sec): 1.0
```

The same command will also display the manual process plan.

5.10 Conclusions

One of the advantages in object oriented approach is that the object schema, which is a notion used to specify the desired object as a collection of entities, can be modified relatively easily. The modifications such as addition or removal of an attribute from the object definition does not affect the overall schema. This helps in following a 'design-prototype-refine' approach in the application development.

The object oriented approach to CAPP system development was found to be very flexible. The entire system development could be achieved with a modular approach.

The detailed study of system objects and methods could significantly reduce the programming efforts needed for application development. Hence, the initial learning curve is rather steep compared with conventional programming languages.

The detailed initial system planning has helped in reducing the system development efforts. The study of entity relationship diagrams, discussed in chapter four for resource database design has helped in developing object hierarchies for knowledge representation.

Smalltalk/V286 does not support multiple inheritance, whereas CADDS5 feature definitions can describe composite features by multiple inheritance. For example, the feature taxonomy (Appendix A) defines a feature, Hole_Counterbore by inheriting the attributes of a feature class, HoleBlind and a Counterbore. Composite features can speed up the design representation. The composite features will have to be decomposed into its constituent features before constructing the component feature tree. Hence, the 'Import Component' method will need appropriate modifications. The geometric and process reasoning technique, however should remain the same.

The development of new methods by using the existing ones with modifications had enabled rapid system development.

The system user interface development tools were found to be very powerful and efficient compared to conventional programming languages such as C.

6. CAPP SYSTEM RESULTS

This chapter illustrates the test results that have been achieved by using the CAPP system. The illustrations are mainly focused on the primary research objectives i.e. the generation of automated process plans for feature based models. The secondary objectives such as geometric and process reasoning techniques, system integration with good user interface and manual process association, were discussed in detail whilst developing the system and hence will be mentioned here only briefly.

The integrated CAPP system that has been developed comprises the Computervision CADDS5 workstation and a Personal Computer 486 (PC 486) to run Smalltalk. The form feature library, based on the STEP standard and modified to add manufacturing attributes, has been developed to design the component as a feature based model. Beside the mandatory use of features, it enables the user to use all the standard CADDS5 commands such as view manipulations, shaded image, hidden line removal, detail drawing preparation, etc. At the end of the design process, the command, 'Verify Feature All Instances Report', outputs a feature report text file in 'UNIX' format. The system converts this file to 'DOS' format and then transfers it through the network to the PC 486. The CAPP system command, 'Import Component' converts the feature report text file into the corresponding feature instances and constructs a component feature tree. The feature tree has been used to apply geometric and process reasoning techniques to generate a component process plan. The following discussion illustrates the CAPP system capability with examples.

6.1 Product Design Representation

The product design is represented in the form of hierarchical tree structure of the Product, SubAssembiles and Components. The user describes the design by 'Add Product', 'Add SubAssembly' and 'Add Component' commands. The parent/child relationship (aggregation) is ensured by digitising the instance to which the new instance is to be added. All the instances are displayed in the Product, SubAssembily and Component display panes. The 'Change' and 'Remove' commands enable product design modification if necessary.

As discussed in chapter three, the new generation of commercial CAD systems describe the product design that is similar to the above concepts. This helps in achieving a single product data model and concurrent engineering. In future, an interface can be developed to read STEP product configuration data file into the CAPP system. The CAPP system product design representation scheme has ensured that the above issues have been given due consideration while developing the system.

The CAPP system represents the component design as a feature tree. The 'Add Feature' command enables building a component feature tree within the CAPP system. This helps in an understanding of basic feature based design principles as well as testing and developing the geometric and process reasoning schemes. The CADDS5 feature based model of a component is represented into the CAPP system by 'Import Component' command. It reads the feature report file, initialises the feature instances and generates a feature tree (refer to Appendix F, page 216, 241 and 265). The feature instances and the feature tree can be inspected by 'Change' and 'Show All Children' commands respectively.

The 'Import Component' command is based on the text file structure of a CADDS5 feature report file. In the future, it can be developed using the STEP application protocol for CAPP so that the feature based models of different CAD systems are represented within the CAPP system.

The literature review shows that various shapes of test components have been used by researchers to demonstrate the feature based design representation. The process planning program of CAM-I and their subsequent contract to Deere and Company to develop the first ever feature hierarchy, upon which the STEP form feature taxonomy is based, used the CAM-I approved test component, ANC 101-M

[127]. The test part comprises features such as HoleBlind, Pockets, U-Shaped Slots and a Boss. The same test component could not be used in this application because of the CAPP system limitations in geometric tolerancing and topological data representation. However, on a similar approach, three test components having similar features have been designed to illustrate process plan generation. These are referred to as TestComp1, TestComp2 and TestComp3 in Appendix F.

6.2 Process Plan Generation

As discussed in the previous chapter, the process plan generation is based on the proposed geometric and process reasoning techniques. The feature tree and feature attributes have been used to interpret the manufacturing view of a feature upon which the process reasoning techniques are developed. The application of a component feature tree has also been demonstrated for process operation precedence (refer to 5.7 and also Appendix F, page 244- 245 and 252).

The plan generation has been developed on two types of resource knowledge, universal and shop specific. The universal knowledge based process plan is generated by digitising the component instance and issuing the command, 'Generate Process'. The system outputs various process reasoning based messages in the text window while generating a process. This enables the system planner to interpret the planning logic and debug the system reasoning methods if necessary. The universal knowledge based process plan can be displayed in the text display window by digitising the component instance and issuing a command, 'Show Component Process'. The process reasoning messages and the universal knowledge based process plans for the test components are illustrated in Appendix F (page 217-227, 242-248 and 266). As can be seen from the illustrations, the process generation does not consider the shop specific resource knowledge at this stage. The possible machining set-ups, the resource options such as machine tools and cutting tools as well as the process options to machine the component are displayed.

By referring to the universal knowledge based process plan, the user may decide to check the shop resource and allocate the fixture for the specific machining set-up and the machine tool by using the 'Add Fixture' and 'Assign Fixture' commands. The manual fixture allocation links the machine tool from the Shop to the machining process. A method for Shop specific technological validation of a machine tool that is based on its size, power, speed and feed limits, process capability and economics of machining has not been developed yet.

The shop specific process can be generated by the command, 'Assign Resources' that enables a search of the shop specific resources for the process, such as fixture for set-up, machine tools and cutting tools and machining parameters. If available, the resources are allocated to the corresponding process attributes, otherwise the system prints a message in the text display window about the unavailability of various resources. The sample printed message for the test components is illustrated in Appendix F (page 228, 249 and 267).

The CAPP system has been able to demonstrate the calculations of NC machining cycle parameters from the form feature attributes (Appendix F page 229 - 235). Additional methods can be developed to generate CADDS5 CVNC command file for automated NC tool path generation, similar to the approach used by Foong [62]. This way an integrated CAD/CAPP/CAM system can be demonstrated.

The universal and shop specific process plan generation capability that has been developed in the CAPP system is unique. The approach should enable quick implementation of the CAPP in various types of machine shops. The universal plan generation capability would provide instant process plan generation based on the universal knowledge. The shop specific process refinment would be possible by defining the shop specific resources by using the various resource definition interfaces that have been implemented. Special methods may also be developed to link the company specific resource databases. The object oriented open architechure of the system should also enable the user to refine the existing methods in the CAPP system by referring to the existing methods and using the Copy, Cut and Paste features.

The process reasoning technique in the CAPP system assigns the process attributes to all the feature instances that have been considered for machining. This results in a repeated display of a process plan for identical but multiple feature instances that are present in the component. For example, four instances of similar size HoleThrough feature (refer to TestComp1, Appendix F, page 220-227) will display the process plan four times. This is overcome by the piovision of Cut and Paste commands with the text display window to edit the process plan if necessary. The final printout of the process plan for test components is illustrated in Appendix F (for TestComp1 on page 229 - 235).

The user interactive manual process planning technique has been developed to assign process attributes to the component that has not been designed by features. This assessment has been conducted to emphasis the need for assigning the manufacturing intent based on standard manufacturing terminology at an early stage in the design(conceptual design). The technique also highlights the importance of a good user interface in the CAPP system.

6.3 The Resource Representation

The shop specific resource knowledge representation has been achieved by the provision of a good user interface. The 'Add Machine Tool', 'Add Cutting Tool', 'Add Fixture', 'Add Work Material', 'Add Tool Material' and 'Add Machining Parameters' commands have been provided with the modification options such as 'Change' and 'Remove'.

As discussed in chapter four, the resource knowledge representation has also been implemented by using the Oracle relational database system. The automated tool assembly generation and the use of form features for graphic display of tool assemblies has been demonstrated (Appendix E).

A number of organisations have been using some form of database management system for resource knowledge representation. The implementation of a feature based CAPP system like the one proposed by this research would also mean the exchange of resource knowledge representation between different systems. The exchange of resource knowledge from the Oracle resource database to the CAPP system has not been implemented. However, it has been identified as work for future development.

Unlike most CAPP systems this research has emphasised on a good user interface. The CAPP system communication interface enables simultaneous display of product, process and resource information. This enables the review of process plan in the context of design and resources or a design in the context of manufacturing process and the resources. Hence, the proposed CAPP system should prove to be a valuable tool in a Concurrent Engineering environment.

6.4 Conclusions

The functionality of CAPP system has been illustrated by using the system to generate process plans for sample components.

The consistent results conclude that the feature based CAD models on commercial CAD systems can be used to automate downstream applications such as CAPP.

A two tier universal and shop specific process plan generation strategy should help in customising the CAPP system or adding new application modules to it such as Shop Scheduling.

A good user interface has been emphasised and developed. It enables simultaneous display of product, process and resources and should meet the requirements of concurrent engineering environment.

The system is by no means complete. There is scope for improvement as well as opportunities for further research. These issues will be discussed in detail in the next chapter.

The system has been developed from basic principles and is being used for the demonstration of feature based modelling concepts and its applications on the Computervision advanced parametric design training course at the University Of Warwick.

7. PROBLEMS IN FEATURE BASED TECHNIQUES

This research has demonstrated that the feature based modelling technique can be used in implementing a CAPP system with design for manufacturing approach (Appendix C & F). Feature based modelling typically allows associativity between geometric entities, construction procedures and feature dimensions. This associativity is helpful in defining parts rapidly, and making design changes quickly. However, when the feature based database is considered for the automation of downstream applications such as process planning, the meaning of features becomes viewpoint dependent. It becomes necessary to transform them from one view point to the other.

7.1 Documented, Complementary and Undocumented features

The documented features are the ones that are pre-defined in the design feature library and the applications have prior knowledge of them.

The undocumented features are the ones that are user defined features but the applications have no prior knowledge of them.

Considering the workpiece definition, the mapping of design features to manufacturing features may result in unknown machining volumes. These are called as complementary features [52].

Sometimes, the combination of documented design features due to feature interaction may be viewed or mapped differently in manufacturing. For example, the designer may use the HoleBlind and PocketRectangular feature instances from the feature library (documented features) to construct the form shown in figure 7.1.

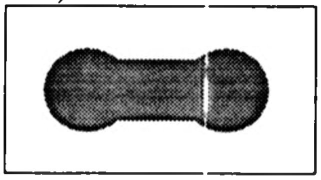


Figure 7.1: Interacting Feature Cluster Formed by Documented Features

The current CAPP system approach can recognise interacting features as individual feature instances that are attached to other features, e.g., plate, pocket, etc. However, their proximity interaction cannot be recognised. The current geometric and process reasoning methods are unable to view them as anything but individual manufacturing features. In reality, the process planner may view feature interactions in many different ways. The research proposes a user defined feature creation technique (undocumented features) to create such features and append the process planning logic to the CAPP system. The technique works well but it may be difficult to control the proliferation of features in a large manufacturing environment. Moreover, the user library may change frequently making it difficult to maintain the application knowledge-bases. Hence, the user defined feature techniques alone may not be used as a general purpose, automatic and robust method for mapping design features to manufacturing.

Similarly, application programs would be unable to reason undocumented features unless the application knowledge-base is appended simultaneously. The production rule based process reasoning schemes that are developed in the CAPP system may be inadequate to determine feasible processes for undocumented features. A suitable strategy to deal with the undocumented features may be necessary while considering STEP/ISO part feature data input to the application programs. Therefore, it may be necessary to characterise manufacturing knowledge in fundamental terms based on tool and workpiece motion [52].

7.2 Feature interaction

The feature interaction is defined as the relationship between manufacturing features. This relationship may exist due to proximity, overlap, geometric tolerance or other considerations that are important to the manufacturing engineer. The result of interactions between features is not directly related to the structure of specific features themselves but the planning method of the whole component.

Clearly feature interaction has no topology and must be thought of as an entity quite different to design and manufacturing features. Feature interactions are the cause of some of the most serious problems in the development of generative CAPP systems [132].

The interaction of features within a set-up can introduce three possible geometric constraints which may influence machining [117]. These are feature precedence, thin walls and machining feature overlap (stock overlap).

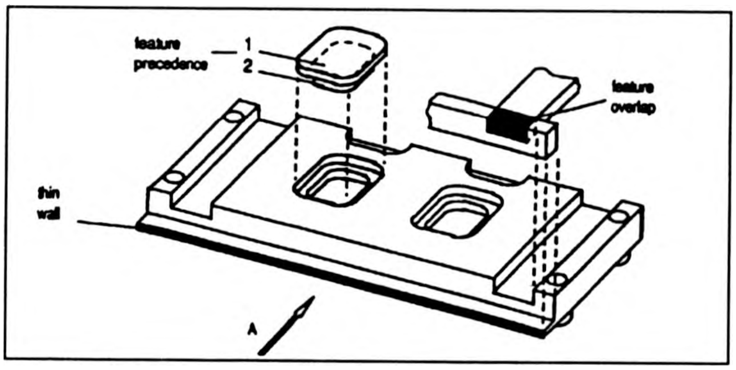


Figure 7.2A: Overlap, Precedence & Thin Wall Interactions

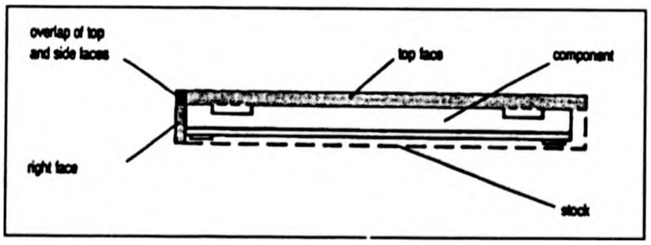


Figure 7.2B: View 'A' of the model in 7.2A showing stock around component [Ref.: Young & Bell]

In a multiple set-up, two additional feature interactions may have to be considered; stock relative to the component, and the prospect of partial machining of a feature in one set-up. The stock relative to the workpiece must be defined such that the surfaces of the workpiece will cleanup to achieve the component dimension.

The machining feature may also interact with fixtures and datums. They also interact through precision constraints such as tolerances and surface texture. However, the current feature based modelling systems cannot store inter feature dimensions and tolerances.

Mill [132] also discusses various feature interactions with reference to the Edinburgh composite component which is a description of hypothetical component, meant to be used as a test piece for feature based design systems, feature recognition systems and advanced CAPP systems.

7.3 Feature Based Modelling Systems

As this research has shown, the feature based modelling systems like Computervision can be used to build domain dependent feature based models. However, as discussed in chapter four, the model database transformed into the CAPP system is incomplete in terms of geometric tolerances, feature topology and workpiece definition. The bi-directional associativity between the feature modeller and the CAPP system does not exist. Hence, precise determination of the machining stock size and shape is not possible while machining a complex cluster of features.

On a more general basis researchers have used feature based modelling systems with bi-directional associativity between a modeller and the application [71, 75, 76]. However, there is not enough research evidence of sound methodology on representing feature interactions. The use of object oriented data structures and the mechanism of inheritance seems to be well suited for the description of feature libraries

[142, 151]. The feature modeller limitations in specifying the feature interactions are quite evident in the implementation of CAPP systems in this research.

In general, the current feature based models are domain dependent and the component feature data structures are incomplete for their usage in applications such as process planning.

7.4 Workpiece selection & stock thickness

Considering a design for manufacturing approach, it may be assumed that the workpiece can be defined or selected by the designer. However, it may be desirable or more flexible if the process planner makes the choice of a workpiece with best practices such as near net techniques, group technology, etc. Before any mapping of design features to manufacturing features, it is necessary to select the workpiece from which a component will be manufactured. Even positioning the component model in the workpiece may be necessary [143].

This research has attempted to suggest a method to specify a feature raw material condition as a solid material or a uniform stock thickness with some assumptions. However, while dealing with the machining of a group of features together, e.g. PocketRectangular with BossCircu:ar as an island, some of the stock may be shared by more than one feature. Therefore, stock refinement methods that are based on feature machining precedence will have to be considered.

7.5 Plan optimisation for features and complete part

The focus of the research objective has been to develop a feature based CAPP system to generate process plans for single piece parts. Hence, the CAPP system results may not reflect process plan optimisation methods such as minimum set-ups, minimum tools, etc. Small to medium size batch production companies would benefit more when optimisation methods are implemented in CAPP systems.

Plan optimisation for feature groups or patterns does not necessarily mean the plan optimisation for the whole component. The component may be machined in one or more set-ups. Each set-up may suggest similar operations on a group with different tool sizes. The roughing, semi-finishing and finishing operation may contribute to different tool sizes, and also act as constraints depending upon the feature interactions. The component plan optimisation may have to be based on minimum set-up, then minimum tool change, and finally on minimum tools while feature interactions and available machine tools act as constraints.

7.6 Research approach to the problems in feature based techniques

Various research efforts have addressed the above mentioned problems in feature based process planning techniques. Some of the significant contributions have been summarised below.

7.6.1 Design to Manufacturing feature mapping

Shah's feature test bed in Arizona State University, USA, are investigating the feasibility of using the **open set feature modeller** (that allows a user to define new features). A methodology is being developed whereby the application programs will recognise design features. This is proposed by developing a system to decompose design features into some generic fundamental terms and compare this representation to the capabilities of common machining operations [52].

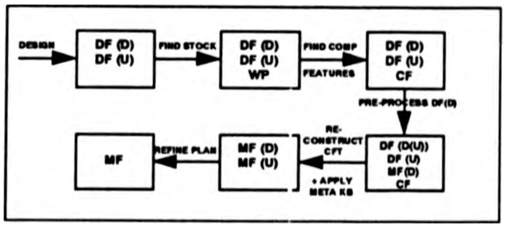


Figure 7.3: Major Steps In Shah's Design to Machining Feature Mapping

The system development has been decomposed into several steps (Figure 7.3).

1. Automatic workpiece (WP) selection.
2. Classification of mapping for documented features (D).
3. Methods for finding complementary features (CF).
4. Geometric reasoning for feature classification/ mapping. It has been observed that not all the documented design features DF(D) can be resolved into documented machining features MF(D). There may be unresolved documented design features DF(D(U)).
5. Reconstruction of CF tree (CFT) for machine understanding of undocumented features.
6. Meta Knowledge Base (KB) for machining process.
7. Selection of machining sequences for undocumented features (U).

The documented feature mapping problems have been classified into generic classes by proposing that the mapping rules could be pre-determined for "standard" cases. Five classes, namely one-to-one mapping, variant parameterisation, discrete aggregation, discrete decomposition and conjugate mapping have been specified.

To recognise complementary features, two recognition methods have been developed, one for explicit features (B-rep), and the other for procedural features (CSG).

The mapping of undocumented features to generate a suitable machining process has been discussed by matching the geometric properties with the fundamental characteristics of the machining process.

7.6.2 Young's feature interaction using SDSM queries

Young and Bell [117] used a library of machining features to build a spatially divided solid model (SDSM). The model provides a cell decomposition model with a range of basic geometric routines which provide information on specific cells in the model. Information on cell classification, cell position, and faces contained in a cell can be identified. Feature interaction queries could be developed by using this data. For example, queries such as, is the surface under clamp clear, is there a thin wall around the feature, have been successfully implemented.

7.6.3 Mapping machining features by recognition approach

As illustrated in figure 7.1, the combination of features may result in a new class of machining feature. Shah (see 7.6.1) proposed mapping rules to identify the generic classes. However, the possible permutations and combinations of various feature interactions will be very difficult to predict and control in such a way. Researchers have attempted to solve this problem by using feature recognition techniques.

Ferreira and Hinduja [144] have achieved good results in recognising certain classes of interacting pockets by using the convex hull based feature recognition method for 2½D components. The method uses a component model from a B-Rep solid modeller (GPM). Initially, it recognises features that originate from edges within the convex hull of each face. Next, the features originating in inner loops of edges are extracted. Later, the concave edges are analysed and additional faces to those not included by the above steps are appended to the features already recognised. Finally, a gluing method is applied, in which two or more features are merged, provided they have certain geometric characteristics. The method has been successfully demonstrated on subtractive features such as slots, compound slots, closed and open pockets, stepped pockets, notches. The method also recognises the complementary features for machining protrusions from the workpiece.

Tseng and Joshi [145] have successfully demonstrated a volume decomposition and reconstruction approach to recognise multiple interpretations of interacting 2½D pockets with complex depressions. The algorithm has been implemented by programming in C language on a 486-25 personal computer. It uses a B-Rep solid modeller (I-DEAS). The topological data of a component model is retrieved and used as input to the algorithms.

The method uses two stages. In stage one it decomposes the machinable volume. It assumes a raw material block as the smallest rectangular prism that bounds the component model. The total machinable volume (Delta volume) is identified by subtractive Boolean operations between a raw material block and the component model. All the bounding faces (which can only be planar faces) of the machinable volume are used as cutting planes that cut and slice the machinable volume into small blocks called Basic Removable Blocks.

The second stage reconnects and glues the Basic Removable Blocks to construct pocket volumes by using a recursive two block connection, coplanar connection and vertical connection algorithms. The algorithm terminates by listing the sets of multiple interpretations of recognised pockets and when all the available basic removable blocks are used for recognition. The recognised pocket lists have a precedence relationship ordered from left to right. The precedence represents the sequence in which the pockets are recognised, and can be considered as a precedence for machining in process planning.

Kulkarni and Pande [146] have classified nested and interacting features as compound features. The features which are not physically connected but logically related, for example, a through slot broken down into two because of the proximity shape of a model, has been classified as relative features. A feature recognition method has been developed for a B-Rep solid model of a component by using syntactic pattern recognition techniques. The algorithm processes the face boundaries (loops) to derive boundary strings which are syntactically analysed to detect feature signatures. The method of signature generation includes filleted and arc shaped features and therefore recognition of features originating from planar as well as non-planar surfaces is possible. The detected signatures are geometrically verified, and the position and dimension parameters of features are calculated.

An application based query facility has been developed, for instance, orientation of parent faces associated with the feature can be obtained to decide the approach direction of the tool for machining the feature; information such as hole groups and holes lying on the pitch circle diameter can be used to select CNC machining cycles; the abstract entities can be used to interactively identify and associate tolerance information.

One of the limitations of the syntactic pattern recognition technique seems to be that it is not data driven, like the convex hull technique of Ferreira and Hinduja, or volume decomposition and reconstruction approach of Tseng and Joshi. The data driven feature recognition techniques are able to recognise collections of features that are not predetermined.

7.6.4 Hybrid feature modelling system

Timo Laakko and Martti Mantyla [147] have developed a system which implements a hybrid of the feature based design and feature recognition approach. The system consists of a feature modeller that enables generic feature geometry instance creation in a B-Rep solid modeller. The solid model of a component at any stage of a design process can be used to recognise machining features by the feature recognition method. The feature recognition method uses a surface based attribute adjacency graph (SAAG) which is based on the AAG recognition technique of Joshi and Chang [54]. Each recognisable feature type is specified by means of a feature definition language which facilitates the addition of new feature types into the system.

The system is based on a novel feature recognition method that provides incremental feature recognition. The system allows changes to a geometric model to be recognised as new or modified features while preserving previously recognised features that remain unchanged in the geometric model. The additions or changes to the geometric model are parsed through the recognition method, causing the feature model tree to update automatically. It successfully argues the case that modelling a shaft could be more straightforward by a rotational sweep than the design by feature approach. Therefore, the designer may find it useful if a component can be modelled using both, featureless or feature based modelling.

7.6.5 Explicit feature interaction modelling

Pedley and Ehrmann [148] classify feature interactions of a component model as implicit or explicit. Geometric reasoning or recognition techniques are necessary to evaluate implicit feature interactions whereas a data structure and modelling format must be developed for explicit interactions.

Dimensions and tolerances are explicitly specified by the designer. An explicit feature based modeller with dimensioning and tolerancing capability has been developed under the auspices of the Brite/Euram scheme of the EU (Project no 4539) [148]. The system is aimed at Feature based design, process planning and NC programming of components typically manufactured on machining centres. Only explicit feature based modelling with dimensioning and tolerancing capability has been demonstrated so far. The use of the model data for process planning decisions has not been implemented yet

7.6.6 Workpiece selection

A good method of workpiece selection should contribute to reductions in a component machining cost. In practice, the workpiece selection depends upon the historical data, minimum material wastage, and minimum set-ups. Shah's feature test bed [52] has been investigating several methods such as GT coding, silhouette projection, bounding box, convex hull of key points and principle machining directions. Because each of these methods has limitations, a hybrid approach is being investigated by ranking the methods according to the complexity, and GT coding approach has been chosen for the first step.

7.6.7 Plan Optimisation Strategies

Any attempt in component process plan optimisation should be viewed as a "company" specific strategy for machining resource utilisation [69]. The shop scheduling information which ultimately decides the availability of machining resources, should be included in this decision. This is only possible if the process planning and scheduling systems are integrated.

However, the quality of process plan that is generated by a generative CAPP system can only be accessed if the system is able to provide the options for various strategies of plan optimisation.

As mentioned before, this research has pursued the objective of developing the CAPP system to generate process plans for single piece parts, i.e. the feasible plan with the available resources. Such plans may be useful in machining a single piece part in a tool room. For batch production, economics of machining will always be applicable and hence, plan optimisation strategies will be necessary.

The optimisation strategy based on the use of minimum number of machine tools for machining a component is discussed by Gindy [69]. At feature level optimisation, the minimum set of resources needed to produce each component feature are determined. Next, at the component level optimisation, taking into account the clustering constraints that may exist due to feature relationships, the group of operations is determined which are used for producing all component features into a minimum number of set-ups. Clustering is performed while observing the precedence relationship that may exist between component features.

The PART system [17] (see figure 2.7) has implemented different optimisation strategies as a set of independently executable programs, called 'phases' which can be executed according to a user defined 'scenario'. The scenario describes the sequence in which the process and operation planning steps are executed. The phases communicate to each other via the relational database in which all information on machine tools, tools, fixtures, features, methods, strategies, etc. is stored.

PART selects machining methods and tools per set-up. The first step selects the machining operations and cutting tool characteristics (ranges for the tool attribute values) for the complete set of simple and compound features. Subsequently, the tool selection module executes the second step; the actual selection of cutting tools. During tool selection, sets of tool descriptions can be combined in order to minimise the number of tools. And finally, the third step executes the determination of the machining sequence.

Different modules of the PART system use the tool database. Each module has a different 'view' of the tool. The machining method selection module, for example, is primarily interested in geometric description of the cutting part of the tool. The tool selection module is interested in more detailed description such as, location of the individual tools and the possibilities to assemble the tool with the tool holder/adaptor for use on the machine tool. Another view could be a tool capacity planning module which is interested in the availability of tools during the planning period. The cutting conditions view needs information on complete tool assemblies to calculate tool paths and cutting parameters for machining, and collision detection.

Hinduja and Huang's OP-PLAN module [149] of TECHTURN CAPP system has used a more scientific approach in optimising operation planning. It plans the operation sequence by using the backward planning technique (i.e. from the finish state to the raw state). However, at each operational level (node of the operation tree), it computes the possible options, performs the detailed machining cost calculations and selects the one with minimum machining cost. It displays the final plan with complete details of estimated cost and time.

These are some of the significant results that have been achieved by the feature based CAPP research community.

7.7 Probable Solutions to Problems in Feature Based CAPP

Section 7.5 and 7.6 discusses the problems in the feature based process planning technique and the research attempts to solve them. Despite the fact that this research has successfully demonstrated the use of design for manufacturing approach in feature based process planning of a single piece part, it is appropriate to discuss the probable solutions to the above problems.

7.7.1 Feature Proliferation and User Interface

The feature classification scheme used in ·he CAPP System is based on the STEP compatible form feature taxonomy scheme which is implemented by CADDS5 Feature Based Modeller (refer to Appendix A). The object oriented hierarchy of standard features (documented features) make features inherit common properties from features higher up in the hierarchy. The standard features are available in a system feature library, called SYSLIB. User defined features (undocumented features) can only be created in the user defined library. While creating these feature the user is allowed to refer to the properties of standard feature classes. Therefore, the user defined features will expand the standard feature taxonomy and should be a cause for concern about feature proliferation. The following measures may help overcome the feature proliferation problem.

7.7.1.1 Training on Standard Feature Taxonomy

The design and manufacturing engineers should be trained on the contents and capabilities of standard features. In addition, the fundamentals of object technology, e.g. object class and instances, data encapsulation, inheritance, polymorphism, etc., should be a part of the training. It is necessary to emphasis that a feature class has unique attributes. The instances always follow the property of its class. A possible variation of feature form should be studied thoroughly before it is defined. For example, a Boss_Circular could be cylindrical or tapered depending upon the Draft_Angle attribute value of 90 or less than 90 degrees respectively (Refer to Boss_Circular form attributes in Appendix A). The training would prevent users from creating redundant features. For example, a tapered cylindrical feature form should not be created as a new feature.

7.7.1.2 Graphic User Interface

The graphic user interface of a feature based modeller should support the functionality- such as menu customisation, select and display, etc. A menu customisation capability enables the user to quickly develop an icon menu with graphic display that implies a particular form feature. The select and display function allows the user to select the feature class and verify feature attributes and its graphic form, almost instantly.

7.7.1.3 The Rover Experience

Last year, the author was involved in a consultancy assignment to develop the user defined form feature library at Rover's Swindon and Cowley plants for tool design applications. The Feature based modeller from Computervision was used to develop the feature library. The project was divided into three phases. The first phase of one week involved everyone from design to manufacturing, to identify unique shapes that could be used in tool design. About seventy shapes were identified. The next phase of two weeks comprised of training and development of the features. There were about eight tool engineers and two system engineers involved in this activity. Their training, progress, problems and difficulties were continuously monitored by the author. In the first week of training and development, considerable discussions and modifications were necessary. The engineers appeared to be more confident in the second week and managed to complete the feature development task well ahead of schedule. Finally, the last phase of one week comprised of development of the user interface. Since the feature library had to be shared between Swindon and Cowley users, it was made available on the internet by using Netscape software. The HTML language was used to develop the user interface that would appear as if the user was referring to the standard part catalogue on a CAD system. The library has been in use for eight months and also used in new car tool design projects. According to Rover engineers, the approach has been extremely beneficial in reducing the tool design lead time. There has been no report on feature proliferation as each design and manufacturing engineer within the tool engineering team clearly understands the feature fundamentals and guidelines to create a new feature in the library.

This research proposes the above design for manufacturing approach in controlling the feature proliferation.

7.7.2 Feature Interaction

If a design form is described by a combination of documented features (refer to Figure 7.1) from the feature library, the feature mapping in the CAPP system could be many to many or many to one type. This may be implemented by developing rule based methods for proximity features and their attributes. The Feature (refer to page 94) class definition will have to be modified as follows:

```
Component subclass: #Feature
instanceVariableNames:
```

'library, datumPlane, origin, orientation, workpiece, representation, operation, application, **proximityFeature**
rawMatlCondition, mcNotmc'

Where, the proximityFeature attribute describes the coplanar proximity feature adjacency conditions. The proximityFeature attribute should have the following sub-attributes to describe the feature adjacency:

proximityFeature := (intFeature to on past in out right left top bottom at-angle).

Each proximity feature instance should be described by its position and orientation attributes. The position with respect to existing feature instance local origin can be specified by one of the sub-attributes:

to	Feature instance profile is tangent inside to the parent feature profile.
on	Feature instance profile origin is on the parent feature profile.
past	Feature instance profile is tangent outside to the parent feature profile.
in	Feature instance profile origin is inside the parent feature profile.
out	Feature instance profile origin is outside the parent feature profile.

The **to**, **on** and **past** attributes can be used in the same way as the APT NC programming language uses them to describe the circular form of a tool with respect to part profile (See Figure 7.4).

The orientation of the proximity feature with respect to feature instance local origin should be described with respect to the sub-attributes: **right**, **left**, **top**, **bottom** or **at_angle**.

It should be assumed that the proximityFeature attribute is used to describe coplanar feature clusters that are either additive or subtractive. Also, only basic feature shapes such as Hole_Blind or Hole_Through, Counterbore, Pocket_Rectangular, Cutout_Circular or Cutout_Rectangular, Slot_Rectangular, Boss_Circular and Tab_Rectangular, should be allowed to describe the feature interactions. These assumptions will be necessary to minimise the complexity in feature mapping rules in the CAPP system. From the designer's point of view, a more complex shape may be described easily by creating a complex user defined feature.

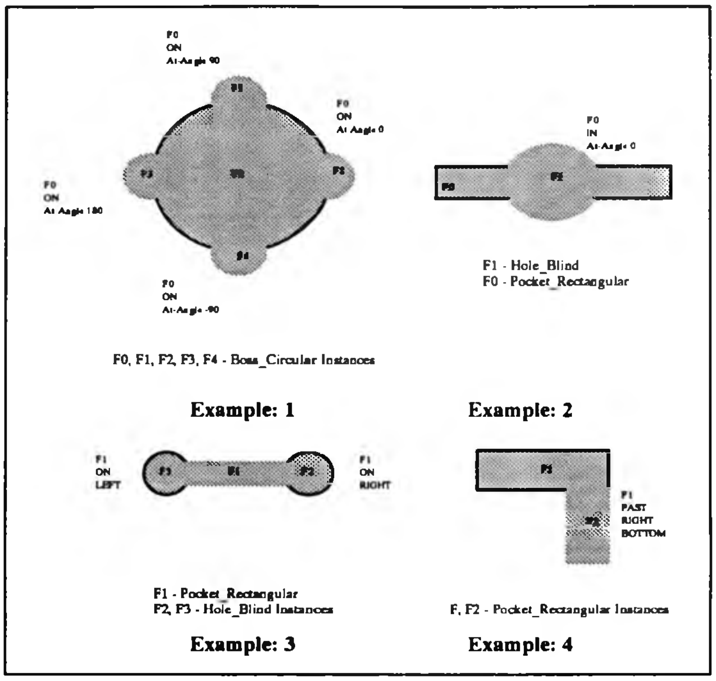


Figure 7.4: Illustration of side feature interaction

Figure 7.4 illustrates some of the forms described by combining basic form feature shapes.

7.7.3 Feature Mapping

The proximity feature attributes, as discussed above should make it possible to segregate feature clusters which interact. The current CAPP system method, generateProcess, segregats features on the basis of their machining datum plane and is as follows:

```
Class :     Component
Method:     generateProcess
      "Reason the feature instances of the component
      instance to generate a machining process"
    |addDatumId parentFeatures addFirstSameDatum highestFt |

    (curProduct isMemberOf: Component)
       ifFalse: [self error: 'Select Component instance'].

    "1. Clear previously generated Auto process"
    curProduct resetAutoProcess.
```

```
 "2. Read the datum id of additive feature instances"
 addDatumId := Set new.
 curProduct withAllChildren do: [:each |
           (each isKindOf: AdditiveFeature)
              ifTrue:[addDatumId add: each datum]].

 "3. Select same datum additive pearentFeature instances"
 addFtrsSameDatum := OrderedCollection new.
 addDatumId do: [:eachDatum |
              curProduct withAllChildren do: [:each |
              (each isKindOf: AdditiveFeature)
                 ifTrue:[(each datum = eachDatum)
                         ifTrue:[addFtrsSameDatum add: each]]].

 "4. While addFtrsSameDatum is not empty,
  For each addFtrsSameDatum, generate a process"
 [addFtrsSameDatum isEmpty]
    whileFalse:[
           highestFt := addFtrsSameDatum showHighZ.
           highestFt addMcProcess.
           addFtrsSameDatum remove: highestFt.
           ]]
```

```
Class:      AdditiveFeature
Method:     addMcProcess
     "generates a mc process for a single additive parent
        instance. Also requests a process for its subtractive
        child feature instances"
  |subChildren nextSubChildren allSubChildren aDatum datums|
     datums := #('TOP' 'FRONT' 'RIGHT' 'BOTTOM' 'LEFT' 'RIGHT').
     nextSubChildren := OrderedCollection new.

     datums do:[:aDatum| self mfgMethod: aDatum.
               subChildren := self getSubChildrenSameDatum:aDatum.
             allSubChildren := self getAllSubChildrenSameDatum: aDatum.
             [allSubChildren isEmpty]
          whileFalse: [subChildren do:[:aFeature |
                            aFeature mfgMethod.
                            (aFeature withAllChildren) isEmpty
                             ifFalse:[nextSubChildren addAll:
  (aFeature getSubChildrenSameDatum:aDatum)].
                           ].
                 allSubChildren removeAll: subChildren.
                 subChildren := nextSubChildren]].
```

As illustrated above, the generateProcess which basically segregates the additive features, will remain the same. However, the additive feature plan generation method, addMcProcess, will have to be modified to segregate the additive features into single additive feature or additive feature clusters, based on proximityFeature attributes. Similarly, subtractive features or subtractive feature clusters should be segregated. Here, the term, 'feature cluster' is being referred to as a set of interacting features.

As discussed in section 7.5.1, the current machining methods (mfgMethod) are not valid for side interacting features. The interacting additive and subtractive type of feature clusters should be machined either by pocket milling or by pocket milling with island. For this reason, the additive and subtractive cluster pocket milling methods should be developed for the EndMilling class.

The proximityFeature sub-attributes may not be able to recognise the precise topology of interacting feature clusters. However they would help in determining the overall size. Moreover, the shape of a cluster depends on the feature instance size attributes. From the designers viewpoint, feature based modelling is a subset of parametric modelling and hence the design modifications are likely to be implemented by changing the feature size attributes. Therefore, any feature cluster machining method should first check the validity of a cluster and then apply the appropriate machining method. It should respond to different types of machining methods, e.g., pocket milling, stepped pocket milling, pocket milling with island, open profile, closed profile, etc.

7.7.4 Operation Planning and Tool Selection

The operation planning capability that has been demonstrated in this research is based on machining one feature at a time. As discussed in section 5.7, the backward planning strategy has been used and the generic automated tool selection capability has been demonstrated in the CAPP system. Tool parameters are selected from the geometric attributes of form features. For example, the following methods explain a tool selection strategy for rough pocket milling and finish profile milling. The strategy works fine, as demonstrated in Appendix F (TestComp1).

```
Class:        EndMill
Method:       getToolRoughPocketSingle: aSubFeature

     "returns tool parameters for  Pocket finishing"
   | getMinPara toolMinDia toolMaxDia toolDia toolHgt toolValues|

   aSubFeature length < aSubFeature width
      ifTrue:[getMinPara := aSubFeature length]
      ifFalse:[getMinPara := aSubFeature width].

   "Decide tool parameters"
    toolMaxDia := getMinPara - 0.2 .
    toolMaxDia >= 50.0
      ifTrue:[ toolMaxDia := 50.0].
```

```
  toolDia    := getMinPara / 4.0.
  toolDia >= 50.0
   ifTrue:[toolDia := 50.0].

  toolMinDia := getMinPara / 8.0.
  toolMinDia >= 50.0
   ifTrue:[toolMinDia := 50.0].

  toolHgt    := aSubFeature depth + 10.0 .

 toolValues := #('EndMill' 'MinDia' 'Dia' 'MaxDia' 'MinHgt').
 toolValues at: 2 put: toolMinDia;
         at: 3 put: toolDia;
         at: 4 put: toolMaxDia;
         at: 5 put: toolHgt.
 toolForOpr := toolValues
```

```
Class:        EndMill
method:       getToolProfile: aFeature

    "Selects the tool for profile milling
     based on the total side stock"
  | toolValues sideStock toolDia|
  toolValues := #('EndMill' 10 'Dia' 50 'MinHgt').
  sideStock := aFeature additiveStockValue at: 3.
  sideStock >= 100
     ifTrue:[toolDia := 50].
  (sideStock < 100 and:[sideStock > 50])
     ifTrue:[toolDia := 30].
  sideStock <= 50
     ifTrue:[toolDia := 25].
  toolValues at:3 put: toolDia;
          at:5 put: (aFeature height).
  toolForOpr := toolValues
```

In case of interacting feature clusters, the tool selection methods will not be as simple as above. It will depend on individual feature instance attributes such as diameter, length and width, and the decomposition of machining features, based on roughing and finishing operation strategy. Assuming that interacting feature clusters will need a pocket or profile milling operation, the machining volume removal capability should be considered at elementary level. In a 2½ axis milling operation, the elementary area of cross-section for tool motion could be one of the following:

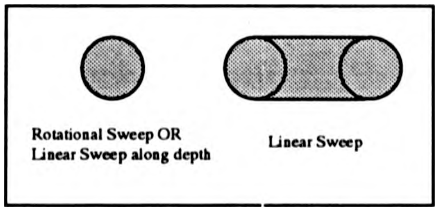


Figure 7.5 Elementary Tool Motion Cross-sections

The tool selection method should depend on the following rules:

1. Assume standard range of roughing cutter diameters, e.g., 50, 25, 20. 10, etc.

2. Check individual feature geometry attributes and select the individual feature maximum cutter diameters and lengths.

3. A strategy should be implemented to machine features from shallow to deep.

4. Certain assumptions should be made based on factors such as percentage of total roughing area that the tool diameter can machine, and its machinable depth, the resultant pinch off area, if any, and the subsequent number of tools to machine the pinch off area, the total area of a pocket, etc. Powerful geometry query functions such as, 'find total area of a cluster', 'find percentage of feature instance area within a cluster, etc., would have been very useful. This is where the bi-directional associativity between a Feature Modeller and the CAPP system would have helped. Under the existing system limitations, certain assumptions will be necessary to implement similar strategies such as, the area of a feature instance could be compared with the sum of the areas of all feature instances within a cluster. For example, if the area to be machined by the smallest tool diameter is 50% or more, then the same tool could be used for machining the entire cluster. These volume computations may not be accurate due to the lack of Boolean operators to compute the partial feature instance area in the CAPP system.

The optimum tool selection may also involve machining parameters selection, machining power computations and the availability and rigidity of machine tool and fixtures. This is discussed later in process plan optimisation section.

In case of additive feature clusters, the operation planning should be able to compute complementary machining features (Figure 7.6).

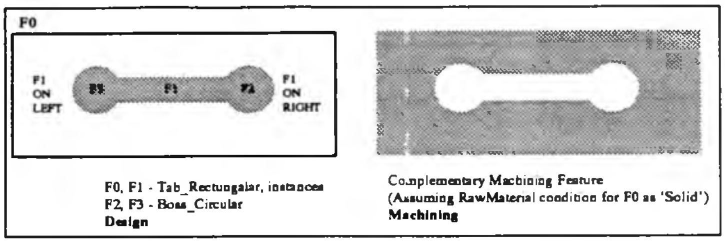

F0
F1 ON LEFT
F3
F1
F2
F1 ON RIGHT
F0, F1 - Tab_Rectungalar, instances
F2, F3 - Boss_Circular
Design
Complementary Machining Feature
(Assuming RawMaterial condition for F0 as 'Solid')
Machining

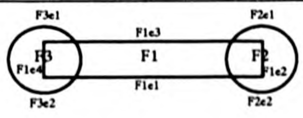
Figure 7.6: Illustration of Complementary Machining Feature

7.7.5 Process Plan Input to NC Programming

The CAPP system has demonstrated the capability of generating geometric data for NC machining such as hole machining cycle locations and pocket vertices and machining planes (refer to Appendix F, TestComp1, 2 & 3). If one has to extend the interacting feature cluster process plan input to NC programming, the cluster profile must be determined. Despite the fact that the feature topology is not an input to the CAPP system, the following approach may be implemented. The Feature class definition should be modified further by adding the instance variable, edgeTopology.

```
Component subclass: #Feature
instanceVariableNames:
'library, datumPlane, origin, orientation, workpiece, representation, operation, application, proximityFeature,
edgeTopology, rawMatlCondition, mcNotmc'
```

The edge topology of a feature instance is not available at present from the CADDS5 report file. For basic features, the edge topology can be defined from the attributes values. Figure 7.7 illustrates the edge topology description for feature instances in Figure 7.4, Example 3.



F1e1, F2e2, F3e3, F4e4	: Edges of PocketRectangular instance, F1.
F2e1, F2e2	: Edges of HoleBlind nstance, F2(0 -180, 180 - 360).
F3e1, F3e2	: Edges of HoleBlind nstance, F3(0 -180, 180 - 360).

Figure 7.7: Edge Topology of Interacting feature instances

The edges F1e2 and F1e4 are totally merged within HoleBlind feature instances (assuming that all the features have identical depth attribute value) and therefore, will not be a part of interacting feature cluster. The edges, F1e1 and F1e3 will have to be trimmed to form a corner with the edges of F2 and F3. The correct topology of the interacting feature cluster would have been available had the appropriate Feature modeller been used in this research.

One of the techniques that could still be implemented to describe the edge contour is the APT (Automated Programming Tool) based NC programming approach. The grammar of CVNC part programming software from Computervision supports the APT based NC programming commands. The tool motion statements are described by using modifiers such as TO, ON and PAST. When the tool is tangential **to** the edge, its position is described by the **TO** modifier. The **ON** modifier defines the tool centre as lying **on** the edge. While the tool is moving along the edge, if it has to be positioned **past** the intersecting edge, the **PAST** modifier is used. In order to implement this approach, an edge intersection method should be developed in the CAPP system.

The tool motion statements for interacting feature cluster profile in Figure 7.7 (In counter-clockwise direction, CCW) could be described by TO, ON and PAST modifier approach as follows:

1. MOVE XYLOC X0Y0 (F1 origin)
2. APPROACH
3. PLUNGE (to suitable depth of cut)
4. CUT TO F1e1
5. CUT TO F1e1 PAST F2e2 (intersection detected)
6. CUT TO F2e2 CCW TO F2e1
7. CUT TO F2e1 CCW PAST F1e3 (intersection detected)
8. CUT TO F1e3 PAST F3e1 (intersection detected)

9. CUT TO F3e1 CCW TO F3e2

10. CUT TO F3e2 PAST F1e1 (intersection detected)

11. CUT TO F1e1 TO F2e2

12. CUT XYLOC X0Y0

13. RETRACT

14. MOVE HOME

The logic in the above approach demands both the proximityFeature attributes and the edge intersection method. The edge, F1e1 along which the tool moves is referred to as drive edge and the intersecting edge to the drive edge becomes the check edge. Assuming that the edge topology of all the feature instances are to be travelled in a counter-clockwise direction, the drive edge intersection with the proximity feature edges is detected (F2e2) to find the check edge. If the intersection occurs, then that edge becomes the check edge for the current drive edge. For the next tool motion, the current check edge (F2e2), becomes the drive edge and its intersection with the proximity feature is detected. If no intersection occurs, then the next edge (F2e1) of the feature instance becomes the check edge and so on.

7.7.6 Mapping of Undocumented Features

As explained in 7.5.1, the undocumented features are the ones that are user defined features but the CAPP system has no prior knowledge of them. For these features the design for manufacturing approach has been proposed. The user interface has been developed in the CAPP system to add the process knowledge manually but efficiently (refer to section 5.8). This approach may be criticised for hindering the independence of design and manufacturing. It may be argued that the additional degrees of freedom for manufacturing experts may help them select optimal processes for a designed product. In the auther's opinion, if better computer based tools are available to aid design and manufacturing processes, one must use them to gain competitive advantage. In order to use them efficiently, some of the existing product engineering methods may need changes or reorganising. In the age of information technology, the integration goes beyond the boundaries of a single company. There is an increase in companies working in co-operation and partnerships with other companies, vendors, subcontractors creating what is commonly termed a virtual enterprise [151]. In this environment, the type of information that may be exchanged and used efficiently could be controlled by information technology management. For example, instead of passing a feature based design to the CAPP system that may not recognise one or more features, pass it to the system that does recognise them, generate a process plan and NC instructions in a neutral format (e.g. CLFILE). Instead of design information exchange one should try manufacturing information exchange.

If one still has to suggest a solution to the undocumented features in the CAPP system, it has to be based on feature recognition strategy. The undocumented feature volume should be recognised and mapped into machining features. The B-Rep solid model data of a feature based model may be captured in a text file. The recognition method may be developed on volume decomposition and reconstruction strategy of Tseng and Joshi (As discussed in section 7.6.3).

7.7.7 Process Plan Optimisation

The CAPP system has demonstrated the capability of generating process plans for machining a single piece part. The process plan refinement strategy has been demonstrated to assign the available resources by using the following method:

```
Class       Component
Method      assignResources

    "Checks the available resources &
    Assigns it to the process instances"
  |toolStat workMatlInst|
  "1. Assign MachineTool"
  self withAllChildren do:[:each |
        each process do:[:eachProc |
                   eachProc getProcSeq do:[:proc |
                                      proc mcForOpr: (self
              getMcTool: (proc setUpNo))]]].
    "2. Search fot Cutting tools"
    self withAllChildren do:[:each |
          each process do:[:eachProc |
                      eachProc getProcSeq do:[:proc |
                      (toolStat := proc toolSearch) isNil
                        ifTrue:[Test answer: (proc toolForOpr)  printString, ' Not available']
                       ifFalse:[proc assignNewtool: toolStat.
                             (workMatlInst := Material getMatlInst:
   (self showComponentInst) material) isNil
                             ifFalse:[proc mcParameters: toolStat
                  workMatl: workMatlInst]]]]]


class          McProcess
method         getMcTool: aSetUpNo
      "Search for the fixture and return if matched with the
       aSetUpNo"
    | |
    Fixture instances do:[:each |
                   (each whereUsed) size == 0
                    ifFalse:[each whereUsed do:[:each1 |
                          ((each1 at: 1) = self name and:[
                           (each1 at: 2) = aSetUpNo])
                             ifTrue:[^(each1 at:3)]]]].

    ^nil

class          McProcess
method         assignNewtool: aValue
      "assignes a new tool to the receiver"
    | temporaries toolAttrib|
    toolForOpr := nil.
    toolAttrib := OrderedCollection new.
    toolAttrib add: (aValue at:1);
            add: (aValue at:2).
    toolForOpr := toolAttrib asArray .
```

The method performs a search on the available resources to assign machine tools and cutting tools to various machining operations.

If one has to consider the process optimisation strategy, it should be viewed at two different levels, machining feature level and component level. The optimisation goal should be to minimise cutting time, set-up time and cutting cost; and the available machine tools, process capability (e.g. Roughing and Finishing) and operation precedence should act as constraints.

7.7.7.1 Feature Level Optimisation

The feature level operation planning decides what needs to be machined by what cutting tool and on what machine tool. Usually, machine tools are capable of mounting various types and sizes of cutting tools. Therefore, their selection, though important, may be regarded as secondary to that of cutting tools. While selecting a cutting tool, various factors such as roughing or finishing, tool approach direction and machining depth, minimum machining radius and pinch off areas, cutting parameters and machining power requirements, are considered to be important. These factors should be systematically analysed to implement the optimum tool selection strategy.

Certain type of tool selection could be straight-forward, e.g. in hole finishing operations the tool parameters could be determined easily from machining feature attributes. The suitability of a cutting tool for a given machine tool should be evaluated and may involve complex procedures. For finishing operations one may check the machine tool process capability parameters such as positioning and repeatability tolerance and required spindle speed. For hole roughing operations such as drilling, detailed power calculations may be necessary. For example, the volume of material removed in drilling operation can be computed as:

$$Vol = \frac{1}{4}(\Pi \times (toolDia)^2) \times N \times F \ mm^3/min.$$

Where, Vol = Volume of material removed in mm^3/min.

toolDia = Drill diameter.

N = Machine tool spindle RPM.

F = Cutting Feed in mm per revolution.

Therefore, HP = Vol x K

Where, HP = Horse Power required for drilling operation.

K = Specific power consumption for a given material.

The tool selection method should check the power availability on machine tools and append the mcForOpr attribute with the list of valid machine tools. If the machine tool constraint is forced, due to optimisation strategy at component level, and the power requirement exceeds the available machine tool horse power, the operation may be split into multiple drilling operations.

For machining cost calculation, the machining and set-up time calculation methods should be implemented. Cost calculation capability can be demonstrated easily in the CAPP system for hole machining operations For non hole machining operations, especially for interacting feature clusters, the machining tool path length, and cutting time calculation methods should be developed. Provision has been made to assign machine hour rate attributes to the machine tool (refer to section 5.4, page 99). The Fixture class should be modified to assign setupTime attribute. A more detailed costing may need labour costing data, etc. which has not been implemented.

The problems with detailed tool selection criteria is that the information such as cutting parameters, specific power consumption for various materials, etc. should be available within the resource knowledge-base of a CAPP system. It may be that the most commonly used range of cutting tools and machining parameters could be pre-defined. The system should derive the parameters for a new tool by searching the knowledge-base and interpolating the higher and lower diameter tool parameters. This strategy could be easily implemented in the CAPP system. The cutting parameter interface has already been developed. The workpiece material class will have to be modified to add the instance variable, spPower to input specific power consumption attribute.

In the case of pocket and profile milling, the basic tool parameter selection may not be as simple as hole machining. However, certain heuristics should be used (as discussed in section 8.2.3) to decide the cutting tool parameters. In addition it may also need a strategy to minimise the number of tools to machine the entire feature cluster for a given machining datum. To summarise, the feature level optimum tool selection strategy should implement the following steps:

1. Read part level optimisation goals, e.g. cutting time, set-up time, cost, etc.
2. Read part level constraints, e.g. machine tool, fixture machining datum, etc.
3. Select tool parameters based on feature geometry attributes.
4. Check tool parameter validity against part level constraints.
5. If necessary, modify tool parameters to satisfy constraints.
6. Experienced based heuristics rules may be used whenever appropriate.

As mentioned above in step 2, the fixture machining datum attribute should be useful in deciding the degree of freedom in tool approach direction. The manual fixture planning has been assumed in this research. The fixture class should specify the allowable machining datum list for a given component which would help in feature level plan optimisation. For example, if a fixture can be used with front or rear machining datums, and horizontal machining centre with indexible table is a machine tool constraint, then features such as through holes, cutouts, and feature clusters with opposite end openings may provide optimum machining plans. If the feature depth forbids the tool approach in one datum because of tool length/diameter ratio, and tool chatter, then the partial depths of the feature should be approached in complementary datums by indexing the machine tool table through 180 degrees.

Machining Features such as notches, and complementary types may have multiple approach directions normal to each other. In this case, tool collusion detection capability with workpiece and fixture will be necessary. The Fixture planning and modelling capability should be developed.

7.7.7.2 Component level plan optimisation

The feature level process plans are set-up (machining datum) dependent. The optimisation strategy at component level could be focused on minimising the number of cutting tools and therefore, reducing the tool changing time and tool inventory cost. For a given machining operation, the tool for operation attribute in the CAPP system generates the following tool parameters:

toolForOpr := (tool type, minToolDia, toolDia, maxToolDia, minCutHgt).

where, toolDia is the optimum recommended tool diameter at feature level planning.

minToolDia & maxToolDia are the minimum and maximum allowable tool diameters respectively.

minCutHgt is the minimum cutting height.

Therefore, a suitable method can be developed at component level to reduce the number of cutting tools by parsing the minimum and maximum tool diameters, and minimum cutting height attributes for similar tool types. For optimum machining cost, the component batch quantity and available tool inventory should be considered. The component class definition should be modified to assign the batch quantity attribute. The minimum cost justification approach should provide further refinement of cutting tools by proposing special purpose multiple operation tools. This is where the author believes that the design for manufacturing approach should be beneficial, because process optimisation will need both process planning and process design capability. Therefore, the manual process assignment interface has been developed to assign special processes.

The other issue at component level plan optimisation deals with sequencing feature machining operations to minimise tool changing and tool approach times. In optimising process sequence, factors such as roughing and finishing operations, operation precedence in feature machining and explicit tolerances between features and therefore finish machining operation precedence, are considered to be important. The method, 'show machining process', generates the process sequence for a given set-up at feature machining level. The method is illustrated below:

```
class       Component
method      printAutoProcess

    "prints an auto process of a Component instance"
  |setUpNums addFtrs procSetUp subChildren allSubChildren nextSubChildren workMatlInst|
  setUpNums := #(1 2 3 4 5 6).
  nextSubChildren := OrderedCollection new.

  "1. Get all the additive features of a component"
  addFtrs := self showAllAddFeatures.
  addFtrs isEmpty
     ifTrue:[Test answer: self name, 'No additive features in the model!']
    ifFalse:[Test answer: 'Automated Process Plan';
            answer: 'Component Name: ',self name.

        (workMatlInst := Material getMatlInst: (self showComponentInst) material) isNil
          ifFalse:[Test answer: 'Component Material: ',workMatlInst name;
                answer: 'Matl Description: ',workMatlInst description, ' ','Matl Condition: ',workMatlInst
                    conditionDescr,
                    answer: 'Material Specifications: ',workMatlInst spec,' ','Material Hardness: ',workMatlInst hardness
                                printString].
            Test answer: ' '.
            setUpNums do:[:setUpNumber|
                        addFtrs do:[:addFtr |
                        (addFtr matchSetUp:setUpNumber)isNil
                          ifFalse:[Test answer: 'SetUp No: ', setUpNumber printString;
                                        answer: 'Fixture For SetUp: ', (self getFixture: setUpNumber) ;
                                        answer: ' '.
                              addFtr printFtProcess: setUpNumber.
                              subChildren := addFtr getSubChildrenSameDatum: (self getDatum: setUpNumber).
                              allSubChildren := addFtr getAllSubChildrenSameDatum: (self getDatum: setUpNumber).
                              [allSubChildren isEmpty]
                                  whileFalse: [subChildren do:[:aFeature |
                                              printFtProcess: setUpNum
   "Prints the feature process/es in a setUpNo"
 |option|
  option := 1.
 (self process isEmpty)
   ifTrue:[^nil]
   ifFalse:[((self process at:1) setUpNo = setUpNum)
            ifFalse:[^nil]].

 (self process) size > 1
   ifTrue:[Test answer:(self process) size printString,' ','Options Exist to machine ',self name.
        self process do:[:each|
                    Test answer:'Option no: ',option printString.
                    each printProcess: self.
                    option := option + 1]]
   ifFalse:[Test answer: 'Machining Process for ',self name.
        (self process at:1) printProcess: self]
```

class **FtProcess**
method **printProcess: aFeature**

```
   "prints the process attributes on the text pane."
  ||
     self getProcSeq do: [:each |
                    Test answer: ' '.
                    Test answer: 'Opr No: ',each ftOprNo printString, '; Opr Name: ',(each oprType),' ',(each oprName)
                         printString.
                    (each precedsOpr) isNil
                      ifFalse:[each precedsOpr = 'Parent Finish'
                            ifTrue:[Test answer: (each oprType),' ',(each oprName) printString,' ', 'MUST PRECEDE ',
                                   (aFeature childOf) name, ' Finishing'].
                          each precedsOpr ='Parent Operation'
                            ifTrue:[Test answer: (each oprType),' ',(each oprName) printString,' ', 'MUST PRECEDE ',
                                   (aFeature childOf) name, ' Operation'].
                          ].
                    (each toolForOpr) isNil
                      ifFalse:[Test answer: 'Tool For Opr: ',(each toolForOpr) printString].

                    (each speed) isNil
                      ifFalse:[Test answer: 'Recommended Cutting Speed: ',(each speed) printString,' ','Feed: ',(each feed)
                                printString].
                    (each mcForOpr) isNil
                      ifFalse:[Test answer: 'Machine For Opr: ',(each mcForOpr) printString].

                    (each isKindOf: HoleMakingProcess)
                      ifTrue:[Test answer: (each oprType),' ',(each oprName) printString,' Cycle Locations...';
                            answer:'PositionPt: ',(each positionPt) printString;
                            answer:'ApproachPt: ',(each approachPt) printString;
                            answer:'PlungePt  : ',(each plungePt) printString;
                            answer:'RetractPt : ',(each returnPt) printString;
                            answer:'Dwell(Sec): ',(each dwellTime) printString;
                            answer:''].
                    ((aFeature isKindOf: Pocket) and:[(each isKindOf: EndMilling)])
                      ifTrue:[Test answer: (each oprType),' ',(each oprName) printString,' Cycle Locations...';
                            answer:'PositionPt  : ',(each positionPt) printString;
                            answer:'ApproachPlane: ',(each approachPlane) printString;
                            answer:'ZWorkPlane  : ',(each zWorkPlane) printString;
                            answer:'RetractPlane : ',(each retractPlane) printString;
```

```
answer:'ClearPlane  : ',(each clearPlane) printString;
answer:'']].
Test answer: ''.
```

The above method prints the feature machining operations for the additive feature and its subtractive children. On a similar approach, component optimum process sequence generation methods can be developed. The following guidelines should be followed:

1. For a given set-up all roughing operations should precede the finishing operations, based on opration type attribute.

2. The machining sequence should follow from shallow to deep feature machining. In addition, proximity feature machining operations should be checked to avoid thin wall machining constraint.

3. The machine tool and fixture constraints should be checked to decide the degrees of freedom for feature approach. This would enable to minimising of the number of set-ups.

4. For each cutting tool parameter, the number of machinable features should be listed and the optimised path of tool approach should be determined to minimise non-cutting time of a tool.

5. Sometimes the finishing operation sequence depends upon explicit tolerances between features which may be machined in the same or different set-ups. The explicit tolerances between different features cannot be assigned within the CAPP system. The roughing and finishing plan generation capability has been demonstrated on the basis of feature size tolerance attributes. Heuristic rules could be developed to assign the precedence to the finishing operations such as, facing and profile finishing first, followed by hole finishing. Also, the features with closed tolerance should be machined last.

These are some of the component level process plan optimisation issues that should considered to implement them in the CAPP system.

8. CAPP FOR SIDE FEATURE INTERACTION

As discussed before, feature based component model data can only be imported into the CAPP system as features and their attributes. The attribute information does not include feature geometry and topology data which is vital for developing the mapping methods for interacting features. A method has been proposed in section 7.2 to define a feature geometry and topology for basic feature classes from their origin and size attributes. The implementation of geometry and topology classes and methods should be carried out in the CAPP system. This chapter discusses the implementation procedure and the solution to the side feature interaction problem that has been discussed in the previous chapter.

8.1 Side feature interaction attributes

Each feature in the feature library has been appended with side feature interaction attributes which are, intFeature, to, on, past, in, out, right, left, top, bottom and atAngle. In the object oriented approach, this kind of modification takes only few minutes. The modification of superclass definition would automatically modify its subclasses.

The designer uses one or more side feature interaction attribute when he/she intends to build a portion of a model form by a combination of basic features. It has been observed that individual feature instances within the combination may have different depths. The attribute **intFeature**, is the most important attribute to communicate feature interaction. While building a model, the designer types in the name of a parent feature instance which interacts with the feature to be inserted. The designer may use "Verify Feature" command to know the parent feature instance name.

Once the design is complete, the report filt is generated to capture the model feature attribute data (refer to Appendix G, section 9).

8.2 Modifications to Import Feature interface

As discussed in section 5.2, the CAPP system communication interface uses the "Import Component" command to convert feature report file into feature instance hierarchy. The import component method should accommodate side feature interaction attributes and therefore, needs minor modifications. These modifications are illustrated in Appendix G, section 2.

A component report file records the model co-ordinates (e.g. feature instance origin) with respect to TOP orientation plane or datum (or world co-ordinate system). The world co-ordinate system is the co-ordinate system such that the model co-ordinates are recorded with respect to it. In addition, the report file records the local orientation planes in which the individual features were inserted in the model. The local orientation plane unit vectors **i**, **j** and **k**, are also recorded (refer to Appendix G, section 9). As discussed in section 5.6, the CAPP system segregates features into various machining set-ups based on their orientation plane attribute. In order to determine feature origin locations with respect to their local orientation planes (or local orientation planes) such as, FRONT, RIGHT, etc., the co-ordinate transformation method will be necessary.

A model location vector (a, b, c) with respect to TOP orientation plane can be written in alternative form as:

$(a, b, c) = a\mathbf{i} + b\mathbf{j} + c\mathbf{k}$.

where, $\mathbf{i} = (1, 0, 0)$; $\mathbf{j} = (0,1, 0)$ and $\mathbf{k} = (0, 0,1)$.

Therefore, the scalar product of local orientation plane **i**, **j** and **k** vector with feature origin co-ordinates (w.r.to TOP orientation) should transform them as per local orientation plane co-ordinate system. The necessary Vertex transformation methods have been developed and are illustrated in Appendix G, section 3.7.

8.3 Feature topology and geometry

As discussed in section 2.5.10, the STEP standard uses EXPRESS language for data modelling. Its application for graphical form representation is referred to as EXPRESS-G [95]. In order to pursue the objective of providing a neutral mechanism capable of describing the product data, the topology and geometry based object definitions for the EXPRESS language have been studied. Due to the resource limitations, the possibility of interfacing the EXPRESS data structure to the CAPP system is not viable. Hence, it has been decided to implement the geometry and topology class definitions on similar approach to that of EXPRESS language [95, page 134].

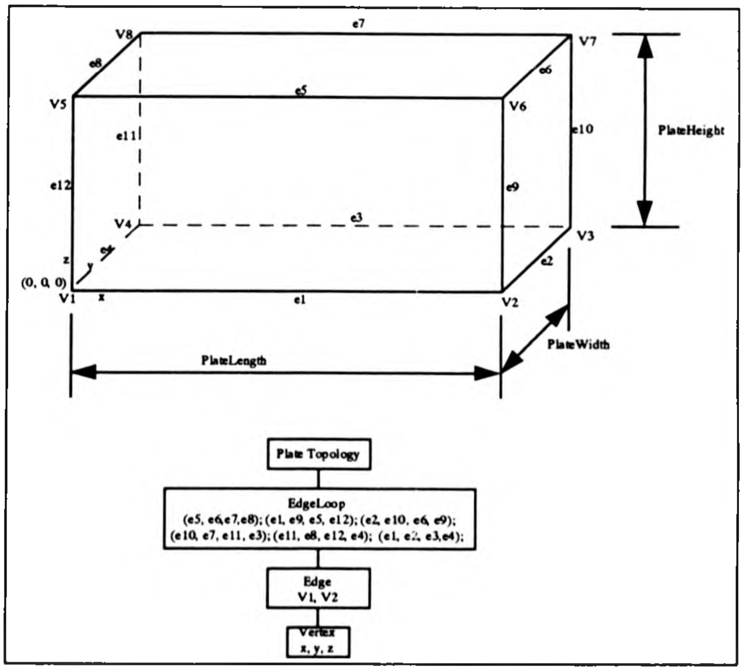


Figure 8.1: Feature edge topology

A feature topology comprises of a collection of faces. Each face is described by an edge loop which is an ordered collection of edges. The vertices are the end points of an edge. Hence, the feature topology related object classes namely, **loop**, **edge** and **vertex** have been defined. The Line and Arc class objects have been defined to describe the Edge geometry. The class definitions and relevant methods are illustrated in Appendix G, section 3.1 and 3.5.

The relationship between a feature (e.g. Plate feature), its attributes (e.g. length, width, height and origin (0, 0, 0)) and its edge topology is illustrated in figure 8.1.

As discussed in section 7.2.4, the feature class definition has been modified to accommodate the **edgeTopology** attribute. The edge topology assignment method has been developed for each feature class within the CAPP system. These methods are illustrated in Appendix G, section 4. The CAPP system initialises the edge topology of a component feature instances before executing any process plan generation methods.

8.4 Detection of Interacting Feature Cluster

The term cluster is used to describe the combination of features whose interaction describes a new form or shape. As discussed in the geometric reasoning section (section 5.6), the features are segregated into various machining set-ups, depending on their datum attributes (local orientations). The previous method, addMcProcess has been modified to detect the side interacting feature cluster (refer to Appendix G, section 5). It uses the intFeature attribute value to detect the interacting feature aggregation.

8.5 Feature Validation

According to Shah and Mantyla [151], when the features are created, modified or deleted, their validation must be determined. The validation conditions may be based on size limits, location or orientations. In the author's opinion, a feature based modeller should include the validation constraints. As discussed in section 7.1.3, the designer should be fully trained in the capabilities and constraints of a feature based modeller.

As far as the Computervision's CADDS5 feature based modelling system is concerned, the form validation constraints can be applied on individual feature classes. For example, if the feature is HoleCounterbore, then CboreDiameter > HoleDiameter and CBoreDepth < HoleDepth (Refer to Appendix A). However, if the designer uses the combination of feature instances to describe an interacting form, its validation depends entirely on the intent. The range of feature origin locations and their size attributes should be the responsibility of a designer. Cluster specific validation rules should be developed to counter-check the intent. If the interacting feature cluster satisfies a set of rules, then it is a valid cluster and therefore, it should be forwarded to process plan generation methods. Appendex G, section 5.1 illustrates a sample feature validation method, based on feature interaction attribute values. It checks the **on, past** or **atAngle** attribute values of the interacting feature. The interaction is valid if one of them is described by the designer. The validation rules may be custom specific and could include number of rules.

8.6 Feature Mapping

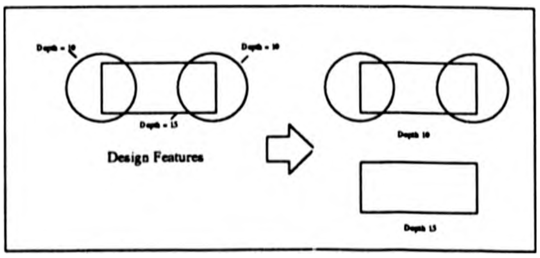
As discussed in the introduction section 1.1, features are regarded as application dependent. A method of transforming features from one application to another is referred to as feature mapping [151]. The feature mapping method uses the existing feature information and recognises the application specific features from it. For example, if a PocketRectangular and HoleBlind feature instances form a KeyHole shape due to their interaction, then the determination of KeyHole form in the context of machining is a mapping method. The mapping depends on various factors. Feature mapping for machining applications may depend on tool selection strategy, variant depths of feature instances, stock attributes, etc. As discussed in section 7.6.1, Shah

[52] has defined several mapping classes based on the feature transformation form one application to another. However, there is not much research evidence about the applications and test results on these classifications. Shah and Mantyla [151] have discussed the heuristic, intermediate-level structure, Cell-Based and Graph based mapping techniques. Most of these techniques are based on feature recognition. Feature recognition is a method of recognising features from a featureless model. Feature mapping takes place at a higher level than feature recognition and it is a topic of current research.

The method of feature mapping proposed here should be called as **tool-feature interaction** based mapping. The machining features are produced by machining operations and each operation is controlled by cutting tool selection. Therefore, the features should be mapped by using design feature attributes (including interaction), tool selection strategy and workpiece information. As discussed in section 5.2, there is no independent workpiece model available and therefore the feature stock attribute has been assumed to be 'Solid' by default. It means the machining features are produced from the solid block of workpiece.

For interacting feature clusters, the mapping method executes the following procedure:

1. Select the varying depths of features within a cluster, if any.
2. Rearrange depths in ascending order (machining from shallow to deep).
3. Map features as per depth.



As illustrated in Figure 8.2, the design feature may be split into multiple machining features (PocketRectangular in this case) in depth based mapping. The depth based mapping method is illustrated in Appendix G, section 6. It may also be used to detect disjoint features.

4. Select cutting tools for each depth based mapped features. The depth based mapped features may split further depending on the tool selection strategy (Figure8.3).

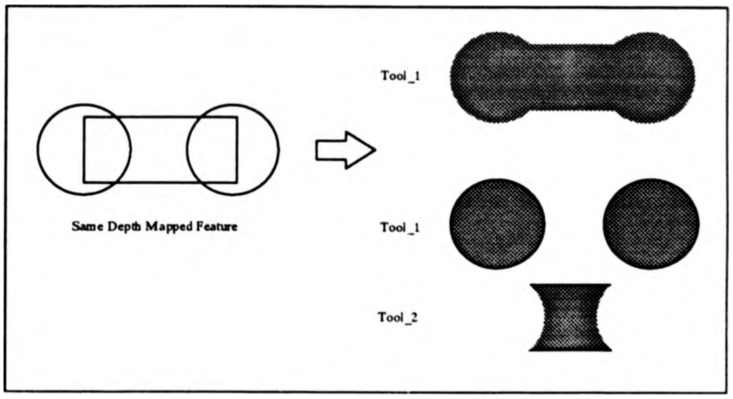


Figure 8.3: Tool based Mapping

8.6.1 Tool Selection Method

The cutting tool selection method for a given same depth feature cluster would require the following:

a. Select tool diameter for individual features. The methods, getToolDiaRoundProf and getToolDiaPolygonProf have been developed and illustrated in Appendix G, section 6.1.

b. Sort the selected tool diameters in ascending order. The smallest diameter tool will always be able to machine the entire cluster.

c. Compute the approximate cluster area as the sum of the areas of individual features.

d. For each tool diameter, compute the approximate percentage of machining area.

e. The decision about selecting the number of tools should depend upon the diameter range and percentage of total area to be machined by a tool. Depending on the shop specific machining strategy, custom specific rules should be used for the tool selection method. For example, it is assumed that if the minimum diameter from the tool diameter range is 20 or more, then it should be used for machining the entire cluster. If the minimum tool diameter is less than 20 mm and the difference between minimum and maximum tool diameter is more than 20 mm, and the percentage of area machined by higher diameter tool is more than 25%, then the higher diameter tool should also be used.

f For a pocket roughing operation, various drilling/orbital milling options should be considered if the hole features exist within a cluster. The pocket roughing methods are illustrated in Appendix G, section 7.

8.6.2 Feature profile union or subtraction

The tool-feature interaction based mapping should be supported by profile union or subtraction methods, usually referred to as 2D Boolean methods. As illustrated in Figure 8.3, the mapped feature profile computation is necessary to avoid tool non-cutting motion as well as for NC toolpath generation.

As discussed in section 7.4.3, the edge intersection method had been proposed to be used along with feature interaction attributes, to, on, past, etc., to determine the APT based machining contour. However, the feature interaction attribute based contour determination method was found to involve unnecessary complications. It is observed that there are a number of possible permutations and combinations to position and orient the feature profile within the cluster. Therefore, it has been decided to develop the profile geometry based union and subtraction methods to determine the mapped feature profile.

In order to develop the edge intersection methods, the basic curve intersection methods such as line to line, line to arc, and arc to arc intersection has been developed. Hill [152] explained the line to line and line to arc methods by using vector algebra. The arc to arc method has been developed by using basic geometry principles. For a given edge, the edge intersection method computes single or multiple intersections with another edge. Supplementary methods have been developed to compute the edge intersection with feature profile, edge inside a feature profile, intersection point nearest to edge start point, etc. The codes for intersection methods have been illustrated in Appendix G, section 3.2.

The union of feature profiles follows the following method:

1. Compute immersed edges in the union of feature profiles & remove them from total edges.

2. Check if the start or end points of the remaining edges are inside the other feature profiles. If yes, modify them to feature intersection points.

3. Generate new edges if the edge start and end points are outside a feature but the edge intersects it.

4. Recheck the edges for double intersection and follow step 3.

5. Step 4 completes the computation of edges for profile union. The resultant edges are reordered such that the preceding edge end point is followed by succeeding edge start point.

6. The method outputs the result as an ordered collection of edges.

The subtraction of features profiles **profl** from **prof2** follow the following method:

1. Get union of profiles, prof1 and prof2 as **prof3**.

2. Compare start and end points of prof1 with prof3. If they exist in both then the edge belongs to the subtraction profile. If either start or end point exists in both, then that edge is a part of subtraction profile.

3. Check if the start or end points of prof1 edges are inside prof2. If yes, trim them to intersection point and reverse the edge direction.

4. Reorder total edges from step 2 and 3. The method outputs the result as the ordered collection of edges.

The codes for union and subtraction methods are shown in Appendix G, section 6.3 and 6.4 respectively.

The net result of feature mapping is the ordered collection of absolute and relative depths, mapped feature profiles and the tool parameters to machine them. Based on the mapped feature attributes, the roughing and/or finishing operation attributes are assigned.

8.7 Process plan generation for cluster pockets

For each depth within a feature cluster, the mapped feature attributes are used to determine the finish and rough pocket milling operations. Since the raw material condition for the cluster is assumed to be 'solid', the rough pocket milling and finish contouring operations are assumed to be mandatory.

The finishing stock is assumed to be 0.4 mm on depth as well as on the side of the mapped feature. As discussed in section 5.3, the finished stock computes the pre-machining stock, which will be the stock for the roughing operation. The side stock is assumed to apply on the mapped feature profile. Therefore, the communication of side stock from roughing to finishing would require offsetting the mapped feature profile by a finished stock value. The profile offset method has not been developed because the NC programming systems can offset the toolpath Hence, only depth stock has been used to communicate stock values from finishing to roughing.

The tool for operation and machining edge profile is assigned from the mapped feature attributes. Various depth planes in pocket milling such as, approach, retract, Zwork and clear plane, are computed from absolute and relative depths of a mapped feature. The cluster pocket finishing and roughing methods are illustrated in Appendix G, section 7.

8.7.1 Pocket corner finishing operation

While finishing a sharp cornered pocket profile, additional corner finishing operations may be necessary when the angle between corner edge tangents is concave, i.e. less than 180 degrees. A method has been developed to compute the angle between profile edge tangents. It computes the start and end tangents of a corner edge. A dot product of tangent vectors computes the angle between the tangents. In order to decide the concave or convex angle, the left or right turn detection method has been used from vector algebra [152]. If the dot product of unit vector in Z direction (i.e. **k**) with a vector, which is a result of the cross product of corner edge tangents, i.e. **k.(a x b)** is greater than zero, then the edge turn is **left**, otherwise, **right**. Therefore, the angle between tangents should be 180 minus the angle between tangents or 180 plus the angle between tangents respectively. The relevant methods are illustrated in Appendix G, section 3.3 and 3.4. A tool diameter of 3.0 mm. is assumed whenever corner finishing is necessary.

8.8 Process plan Report Generation

The process plan report generation method remains the same as discussed section 5.9. However, some modifications are necessary to output the edge profile of interacting features. The edge profile/s along with pocket machining plane data could be used for NC toolpath generation. Unfortunately, due to the limitations of graphic display methods in the current Smalltalk system, the mapped profile/s could not be displayed graphically. The modified codes for process plan report generation methods are illustrated in Appendix G, section 8.

8.9 Test Results

As illustrated in Appendix G, section 9, a component model has been built on Computervision's CADDS5 feature based modelling system. The features report file has been imported into the CAPP system. The process plan report is generated by executing the 'Generate Process', 'Assign Resources' and 'Show Component Process' commands. Upon close inspection, it has been observed that the automated process plan results comply with methods that have been discussed in this chapter. One of the distinct features of the process plan is the interacting feature profile edge topology output, which could be used for NC toolpath generation.

The current version of Smalltalk system does not provide advanced graphics display methods and therefore, it is not possible to display the operation plan specific profile edge topology. Hence, the process plan for TestComp4 in Appendix G has been illustrated with relevant figures of profile edge topology.

The side feature interaction solution could be further experimented with the combination of additive and subtractive feature interactions.

9. DISCUSSION

The feature based CAPP system study, analysis, planning, development and subsequent test results have demonstrated that form feature based component models on commercial CAD system can be used for automated process planning applications. The context of today's best equipped commercial CAD systems, their use in real world product development and the emerging STEP standard have been considered while studying features. The modular development of the CAPP system has been demonstrated by object oriented product, process and resource knowledge representation and their integration. In a way, this research work presents a broad overview of the various issues that need to be considered for the development of a CAPP system. This chapter discusses the advantages and limitations of the proposed approach and concludes with recommendations for further work to improve and extend the scope of the current implementation.

9.1 The Product Design Representation

The product structure/configuration as a hierarchy of assemblies, sub-assemblies, components and features has been used to represent product knowledge in the CAPP system. The product design representation approach is in line with the modern industry practice for concurrent product engineering. The implementation of STEP based form feature definitions has made the CAPP system adaptable to use the standard product model data.

The work on the Object Oriented Product Modelling approach using Smalltalk has recently been published [130]. However, the part design is interactively described in the Smalltalk system and then converted into AutoSolid for visual display, whereas this research follows the opposite but logical approach by using a feature based CAD system for design and subsequently using the design data in the CAPP system. In addition, interactive part description within the Smalltalk system has also been provided for testing and development purposes.

Some work on object oriented STEP compatible part definition schema that consists of nominal shape, form features, shape tolerance, surface and materials information and its use to develop the CAPP application protocol has recently been published[128, 129]. However, the work appears to have addressed the product definition only. Its application in geometric and process reasoning has not been discussed. The work is being tested by developing a CAPP system. Such a protocol could also be tested in the CAPP system proposed by this research by developing an interface to convert a STEP object file into a part feature tree.

One of the fundamental issues that seems to have been overlooked by the above work is the interpretation of user defined features (special form features to suit particular needs) from the STEP file to the CAPP system. More illustration is necessary to elaborate the strategy in dealing with undocumented features in a CAPP system.

This research pre-defines an object in the CAPP system that corresponds to the user defined feature in the modeller. Hence, interpretation of the CADDS5 report file into the CAPP system as a feature tree is possible. The import component method of the CAPP system can be modified to automatically define the unknown features that may be present in the STEP file. While generating a process plan for the unknown feature, the CAPP system will issue a warning message so that the manufacturing engineer could assign the appropriate manufacturing method. The 'Change/Inspect' command can be used to study the attributes of a new feature while developing a manufacturing method. If the user defined feature is a one off and unlikely to be repeated again then a technique similar to manual process assignment (refer to 5.8) can be implemented.

The above approach illustrates the ease in customising the CAPP system which is essential for its implementation in different Machine shops. A recent survey on Integrated Design and Manufacturing in Japan has revealed that Japanese companies, both large and not so large, buy software and customise it to suit their own manufacturing style [136].

This research has been able to demonstrate the application of user defined features in the CAPP system. The geometry of a feature is one of its attributes (representation). While representing that geometry, the designer can have complete freedom as in conventional CAD modelling. Hence, the feature based modelling approach should not be viewed as an approach that hinders the designer's creativity. A paper on a case study of feature based tool design was published by the author [141] (refer to 3.8 and Appendix C). It is convenient to assign a manufacturing method for a complex user defined feature by considering all its features and their relationship to each other rather than developing methods for them. This is because these features may not repeat again, e.g. machining surfaces in die design or a special feature to suit the needs of a particular organisation.

The user defined feature approach can also be used while considering different types of workpieces, e.g. a casting or a forging (refer to TestComp3, Appendix F, page 254-270) and not just a billet as most research efforts seems to have considered. The workpiece becomes a starting point for a feature based model. The rest of the model can be built by using other features in the library. In some cases the workpiece feature may not need any machining to form its shape as it is assumed to be a casting. All the children features of the workpiece can be machined as per the process plan generated by the CAPP system. This illustration also supports the reason behind keeping the STEP form feature classification limited to standard machinable forms such as hole, slot, etc.

One of the limitations of the user defined feature approach in a large manufacturing organisation could be the possibility of proliferation. Strict control on feature taxonomy schemes and organising application specific feature libraries may help in preventing it. The users should be trained adequately. The other preventive measure could be the provision of a GT classification and coding module for features. A designer should generate a GT code for the feature that he/she intends to define as a new feature. The feature should be defined if the code is unique. The automated GT classification and coding module for the entire feature based model can also be used to automate variant CAPP systems [131]. This should considerably narrow the gap between variant and generative CAPP systems.

The study on symbolic representation of tolerance and manufacturing features (refer to 3.6 & 3.7) could not be pursued further because of the data access problems from the CADDS5 system. However, the display of tolerance and manufacturing related attributes in the form of symbols on the model would help in addressing the detailed drawing related issues at the design stage. It would provide better interpretation of a feature based model. A paper on this approach has been published by the author [140].

The product definition user interface (Figure 5.2) in the CAPP system should have been provided with the nodal display of product structure (as in Figure 3.2) to enable better interpretation. At the moment, 'Show All Children' and 'Change/Inspect' commands have to be used for a number of times to interpret the product structure.

9.2 Geometric Reasoning

Unlike most CAPP systems, this research has considered two types of raw material stock attributes, a solid (billet) and a thickness. The interpretation of the stock attribute is based on certain assumptions that are discussed in chapter five. The thickness attribute can be used on castings, forgings or pre-machined workpieces. The approach has made the system more versatile as demonstrated through results in Appendix F.

It is assumed that the raw material (workpiece) is known at the design stage (which is usually the case in the design for manufacturing approach). The designer builds a feature based model based on the raw material knowledge. The overall workpiece definition is implied from the primitive type feature and a raw material condition attribute of a feature. For example, the workpiece for TestComp1 (Appendix F) is the primitive feature Plate; for TestComp2, a primitive feature Plate with the height of a protrusion feature and 2 mm machining stock added to its height; and for TestComp3, a primitive feature Bracket which is implied as a Casting by declaring its mcNotMc attribute to be 'N', which means machining is not required to form its shape.

It is worth mentioning that the STEP based feature definition has an attribute workpiece that enables the selection of solid volume in which the feature is inserted (Appendix B, page 176). This research has not used the workpiece attribute because of the feature geometry access problems on the CADDS5 system. Instead, a rawMatlCondition attribute has been defined and used to interpret the raw material state of a feature.

This research accepts that there may be a one to many relationship between the design feature and the manufacturing feature as observed by Mills, Shah and Pratt [132, 52, 71, 133]. While studying the problems in feature based process planning, a solution to side feature interaction problem has been discussed in chapter 7. The solution to the problem has been implemented in the CAPP system and the test results in Appendix G are very promising. The CAPP report generation capability would have been much better had the graphic display methods been available in Smalltalk system.

It is realised that the proposed assumptions (refer to 5.6) that enable geometric reasoning of the additive feature clusters will not be adequate. The feature recognition techniques that are discussed in the literature review will be essential. The geometric reasoning would be more effective if both the recognised manufacturing features (or delta volumes) as well as the component feature tree are considered. None of these issues could be demonstrated as the approach needs two way interaction between the feature based modeller and the CAPP system, as recommended by Pratt and Wilson [66], and the access to the underlying geometry and topology of the features and workpiece to compute feature extrinsic information as proposed by Shah[71, 131]. This observation becomes interesting in the context of STEP file product data input to the CAPP system. The geometry engine will have to be the integral part of the CAPP system to compute the feature extrinsic information that is necessary for geometric reasoning purposes. The geometry engine should use standard geometry data and not the proprietary one. Computervision has recently introduced an object oriented CAD tool kit called Pelorus [134]. Pelorus is STEP compliant and it will free users from proprietary CAD databases, even for geometry data. Systems like Pelorus should be considered for developing applications such as CAPP.

The component feature tree and feature attributes have been used as the basis in proposing systematic steps for geometric reasoning of a component (refer to 5.6). The approach has shown consistency in plan generation (Appendix F) and it will work provided the designer ensures the feature validity.

The term validation is used to imply that the designer uses form features true to their definition and rules [133]. For example, a HoleBlind feature is used to represent a blind hole and not a through hole, Pocket is used to represent a pocket and not a slot or a notch, etc. However, maintaining the feature validity while designing a complex part cannot be left alone to the designer. For example, a designer may insert a HoleBlind feature in the side of a Plate feature. Next he may insert a Pocket feature in the top face of a Plate feature. If the Pocket feature intersects a HoleBlind feature then the HoleBlind feature becomes invalid as it becomes a HoleThrough feature conforming to different class definition and rules. The feature validation and revalidation should be built into a feature based modeller. This should be supported by efficient user defined feature definition languages such that the features conform to a set of rules if used in modelling applications. The CADDS5 user defined feature capability enables one to enforce constraints in the feature definition, and maintain them in the model. However, it does not check the proximity obstructions caused by other feature instances.

9.3 Process Knowledge and Reasoning

The object oriented approach has made it possible to use a mixed approach for process specific knowledge representation. It is represented by a frame based approach for hole making processes and rule based approach for other types of process (for reason, refer to chapter 5 page 98). This enables flexibility in process knowledge representation.

Organising the feature machining knowledge by object oriented approach was found to be very convenient. Various feature machining methods have been developed that are relevant to the process class. For example, a Boring process 'knows' how to respond when the rough or finish boring search is requested; an EndMilling process 'knows' how to respond to pocket milling, profile milling or face milling. These methods respond to the backward process reasoning search on the manufacturing view of a design feature i.e. from finished state to raw state. The relevant process reasoning search is assigned by the manufacturing engineer (refer to 5.3). Hence, there is no blind search and the search is quicker. If the search provides multiple feature machining options then all the options are generated and displayed (refer to Appendix F, page 220-221). The 'Assign Resources' command refines the process plan by searching the shop specific resources. It was observed that there could be other issues that may influence the economics of machining, e.g. available resources, shop schedule, etc.

Hence, it was decided to provide the options to the user. The user selects the appropriate option.

A typical expert system forward or backward chaining search tends to resolve the conflict (when multiple solutions exists) by assuming some form of a weight, e.g. minimum machining time, minimum machining cost, etc. In an object oriented approach the search is controlled by methods that could be simple or complex and may depend on one or many parameters. Hence, it was found to be more suitable for a process reasoning application, where decisions are based on knowledge as well as experience, and its relevance within the context of a particular feature.

The development of new methods in object oriented programming is also found to be very convenient. A typical procedure involves copying a similar method, introducing modifications to suit the need, and saving it under the appropriate method name. Like conventional programming, the program compilation and linkage is not necessary.

Recently, a unique work on modifying an existing generative CAPP system for the purpose of integration with the scheduling system has been published [135]. The modifications are made to the process decision model files and the machine tool files as the old process decision model was based on a particular machine tool. If similar modification were considered on the CAPP system in this research, it will not need process reasoning modifications, because the process generation is based on the universal knowledge. Moreover, the process plan per set-up is generated as a network of operations maintaining aggregation between operations, a technique that has been used by Tonshoff for scheduling techniques [122, 138]. Each operation records all the valid machine tool classes (Appendix F, page 219-227). The shop specific machine tools that belong to the valid machine tool classes for the operation can be searched to allocate the best possible machine tool for the machining process. A machine tool scheduling and rescheduling method can be developed based on the factors such as load, breakdown, average cycle time, machine hour rate, number of alternate machine tools. etc. The work involved will be 'add on' type and not the modification to the CAPP system.

The CAPP system process reasoning capability is applied to shape tolerance and surface finish attributes of a feature and not to location tolerance. It may be argued that the close shape tolerance can also imply closer location tolerance. However, a location tolerance relates different features as per the functional obligations of a design. Sometimes the process planner has to manipulate location tolerance when the related features are machined in different set-ups. Hence, lack of consideration of location tolerance is a limitation of the CAPP system.

The generative feature based process planning capability of the CAPP system is based on the assumption of machining a single piece part. It does not consider the batch quantity, group layout of the Shop and special-toolings such as multi-spindle drilling attachment, multiple bore roughing tool, drilling and counterbore tool, etc. However, the automated plan generation capability of the CAPP system could be used together with batch quantity information to develop methods for justification of special-toolings based refined process plans. The group layout of the shop could be considered while developing the shop scheduling capability. In both cases the work involved will be 'add on' type and not the modification to the CAPP system.

One of the important aspects that has not been dealt with this research is the self learning and teaching capability of the CAPP system. Some work has been published recently that compares the five step process planning methodology of process, sub-process, feature process, methods & equipments, and operations & tools, to learning methods such as learning by example, observation, analogy, explanation based generalisation and advice taking [137]. The research in this field seems to be at an early stage. However, its importance in the self learning capability of the CAPP system to make the manufacturing knowledge more meaningful and then passing it on to the new generation should not be ignored. As far as the teaching ability of the CAPP system is concerned, elaborate comment statements have been used at the beginning of a specific method development so that it clearly explains the logic behind the method development. The 'Change/Inspect' commands have been developed to explain the various object definitions (refer to 5.2, 5.3 and 5.4).

The machining process classification (refer to figure 5.4) could have been defined as machining operation classification. Each feature machining process comprises one or more machining operations. Hence, the term Operation could have be used in place of the term Process.

The feature process operation sequence, operation precedence and set-up numbers are automatically generated by the CAPP system. However, in the context of the overall process, an operation sequence optimisation will be necessary and it should be linked to further research efforts in developing scheduling techniques.

9.4 Resource integration

The resources in industry are usually managed by databases (relational, object oriented, etc.). The CAPP system in this research has not used any database to represent resource knowledge. Instead, resource knowledge representation has been developed in Smalltalk, comprising a number of object hierarchies and suitable user interfaces (refer to 5.4). Object oriented databases provide seamless integration with object oriented programming languages [139]. Hence, object oriented approach to application development such as CAPP benefits from simultaneous considerations of database requirements and programming needs.

The resource knowledge representation is independent of process and product knowledge which enables modularity in its development, modification and application. Various methods have been developed to communicate between product, process and resource objects.

The resource knowledge is Shop specific and hence it is not used as the basis for process generation. It is used for process refinement by developing the 'Assign Resources' method. This approach should help in customising the CAPP system.

One of the major problems that have been observed in customising CAPP systems is the utilisation of existing company specific resource databases. Suitable methods could be developed to connect to various databases and get the attributes that are relevant to a particular resource object. This concept has not been implemented yet. However, it has been identified as work for further research.

As discussed in 4.4.1, a resource database has also been developed by using the Oracle relational database system, as a part of initial CAPP system development. Considerable effort has been expended in developing automated tool assembly generation and graphic visualisation by using form features. The work is unique that for a given tool request from the database, an application program links the relational information for tool components to generate possible tool assemblies for a specific type of machine tool spindle (refer to Appendix E). In addition, it enables a graphic display of tool assemblies that is based on the tool feature attributes and also computes tool cutting, clearance and total length. The work should have a significant importance in tool management and scheduling applications. Similar situations exist in various industries where the existing systems capabilities and applications cannot be ignored in preference to the new system installations. Hence, the assessment of the interaction between different databases should prove to be a benchmark in investigating the database response times. The response time would make it possible to decide whether to have a single object oriented database or not. The current ESPRIT project 2165 on Integrated Modelling of Product and Process Using Advanced Technologies, proposes a single object oriented database, IBASE (IMPPACT database) [77].

Considering the existing resource database of the CAPP system which has about one hundred resource instances, the response time for resource allocation for

TestComp1 (Appendix F, page 226) using 'Assign Resources' command has been observed to be about two minutes.

9.5 General

This research has been able to highlight the limitations of conventional programming languages for developing the CAPP system (refer to 4.5).

The need for a good user interface has been consistently emphasised while developing the CAPP system. The integration of product, process and resource domains has been successfully demonstrated by developing various intelligent menus for system user communication. A built-in intelligence has been provided in the menu design. For example, if the user selects a wrong input to execute a particular command, the system displays messages with necessary corrective action that is relevant to the command. This makes the system safe to operate and makes it easy to learn its functionality.

The overall CAPP system communication window has been developed by recognising the needs of Concurrent Engineering where referencing and cross referencing relevant information between various engineering domains is a way of working.

The process planning function also needs graphic support for applications such as graphic tool assembly generation, fixture planning and display, process layout preparation, NC tool path simulation, etc. The Smalltalk/v286 has a graphical capability called bit mapped graphics or raster graphics, e.g. a line is drawn with a continuous vector of dots, a character is formed with a block of dots, etc.[125]. However, the graphic library support has been found to be inadequate and considerable effort will be required to develop the graphic methods that would enable the CAPP functionality in the above mentioned graphic applications.

9.6 Future Research

Considering the knowledge and experience gained in this research, it is appropriate to propose two types of recommendations for further research, first the recommendations for further work to improve and extend the scope of the current implementation, and secondly the recommendations for global CAPP research.

9.6.1 Enhancing the Current Implementation

Further work is necessary to conduct a detailed study of STEP application protocols for product modelling and process planning and the development of suitable methods to represent CAPP system product knowledge from the STEP file.

A soulution to the side feature interaction problem has been implemented only for subtractive features. It could be enhanced for additive features as well as a combination of additive and subtractive features.

The development of automated GT coding and classification methods in the CAPP system by using the component feature tree could lead to the possibility of linkage to the commercial variant CAPP systems to automate them.

It would be possible to add further enhancements in the resource knowledge representation by adding Method Time Measurement (MTM) knowledge and implementation of methods to compute cut time, move time, set-up time and total cycle time for the component machining. In addition the methods in process reasoning that are based on the component batch quantity and the scheduling techniques could also be developed.

It would be possibile to develop suitable methods to exchange resource information from the Oracle resource database to the CAPP system. Further possibilities could be investigated to use standard resource databases in the CAPP system such as the machinability database from Metcut Research, Sandvik tool management system, etc. The study of STEP application protocols for resource information exchange could also be carried out. The data response time could be investigated in case the database plug-in approach is implemented in the CAPP system.

The various types of CAPP applications such as round components, sheet metal, assembly planning, etc. could be implemented by using a similar approach to investigate the consistency and shortfalls, if any.

9.6.2 Global CAPP research

The object oriented CAD tool kits like Pelorus that are STEP compliant and independent of proprietary CAD databases should be used in CAPP research. They would automatically establish the two way link between CAD and CAPP systems which is essential for powerful geometric reasoning.

Considerable research efforts are necessary to enhance the feature based CAD systems to automatically generate the various types of workpiece models from the design model. It is encouraging to see the powerful commands on sophisticated CAD systems such as 'Generate thickness for the entire model' or Generate thickness excluding/including faces of a model', 'Offset face of a model', 'Offset face of a model with a draft angle'. More research is necessary in geometric reasoning techniques that are based on feature based design and workpiece model and that use both the feature tree and the feature recognition techniques.

It is essential to include dimensioning and tolerancing (D&T) along with the above information. Suitable techniques will be necessary to display this type of information on the model as symbols or labels. D&T information of a feature based model should also be investigated in the context of automated detailed drawing preparation.

Further work is needed in the development of feature languages to provide the user defined feature techniques with constraint management that would make it possible to check their validation and revalidation while using them for feature based modelling.

Continuous and significant amount of research work is necessary in feature mapping. Powerful open geometry systems such as ACIS [151] should be considered as a standard platform for such research. Systems like ACIS with powerful geometry functions make it possible to focus on relevant areas of research at an early stage. Otherwise, a significant amount of effort is expended on developing such functions.

Now that the object oriented branch of AI has been finding wider acceptance in the CAPP research community, its applications in process knowledge representation and reasoning need further research, and knowledge based learning techniques should become a part of it.

10. CONCLUSIONS

1. A detailed study has been carried out to understand the tools and techniques that have been used in computerising the process planning function. The relevance of various techniques and their limitations have been observed.

2. Component design data in terms of its constituent features is essential to develop a generative CAPP system. The design feature may have a different meaning in manufacturing. In order to interpret that meaning both geometric reasoning and feature recognition methods are necessary.

3. A detailed study has been carried out to understand the advanced concurrent product development techniques by using sophisticated CAD/CAM and Engineering Data Management systems. The importance of the STEP standard for free sharing of product data between design and manufacturing has been emphasised.

4. The STEP form feature information model that has been supported on the commercial CAD system has been studied in detail. It consists of form feature attributes that describe their nominal shape. This information has been found to be insufficient for CAPP system application. In addition, the CAPP system needs tolerance, surface finish and raw material information. In order to overcome these limitations, the user defined feature capability of the CAD system has been investigated for its suitability to assign manufacturing related attributes while modelling with features.

5. The component feature report text file, the user accessible data of a feature based CAD model, has been investigated for its suitability as an input to the CAPP system. Its limitations in building feature aggregation have been observed. Without aggregation a component feature tree can not be built. A feature library has been developed that enables feature based modelling of prismatic components with the capability to assign manufacturing attributes and maintaining feature aggregation.

12. The feature tree, feature geometric attributes and feature manufacturing attributes that also include their raw material condition, have been used for implementing geometric reasoning techniques. The geometric reasoning techniques interpret the manufacturing view of a design feature and the feature machining set-up. The CAPP system is capable of generating process plans for different types of raw material conditions.

13. The problems in feature based process planning and their possible solutions have been discussed. The solution to the side feature interaction problem has been implemented in the CAPP system.

14. Process reasoning techniques have been developed that are based on the geometric reasoning of features. A process reasoning system involves matching of process knowledge with the manufacturing view of a feature. If the matching occurs, the process object instance is initialised. The 'initialise' methods are based on the universal process knowledge.

15. Each initialised process also computes the pre-machining state of a feature that describes the new manufacturing view of a feature based on its machining operation. This backward planning approach of process instance initialisation continues until the manufacturing view matches with the raw material condition attribute of a feature.

16. The machining process of a feature is generated by initialising one or more process objects that describe the machining operations for the feature machining process. The feature machining process operation sequence is established by process instance aggregation.

17. The feature process machining sequence as per process instance aggregation may prove to be incorrect in the context of machining all the features in a given setup. The operation preference method has been developed that evaluates the feature process operations in the context of a component feature tree and highlights the

feature operation preference that can be machined either before or after the machining operation of a particular feature instance.

18. The 'Assign Resources' method has been developed that refines the universal process plan in the context of Shop specific resources. This technique should enable ease in customising the CAPP system.

19. The capability of the CAPP system has been demonstrated by testing it on different types of test components. The success and consistency of results indicate that the feature based generative CAPP system can be developed to automate the process planning function by using commercial feature based CAD systems.

20. The object oriented approach should have a significant potential in CAPP research due to the analogy and convenience of representing features as objects. The EXPRESS language of the STEP standard also has an object oriented functionality. The commercial object oriented CAD/CAM systems have recently been introduced. CAPP systems need a powerful geometry engine to support geometric reasoning, and for providing the additional functionality such as graphic tool assembly generation, automated fixture planning and display, automated process layout preparation, NC tool path simulation, etc. Hence, the CAPP System should be developed by using the hardware and software that provides object oriented CAD/CAM tools and method libraries, such that CAD/CAPP/CAM becomes a single integrated system.

REFERENCES

[1]. Ham, I., Lu, S.C-Y., "New developments of CAPP in USA and Japan", Proceedings of CIRP International Workshop on Computer Aided Process Planning, September 21-22, 1989, pp. 1 - 23.

[2]. Ham, I., Lu, S.C., "Computer-Aided Process Planning: The present and the future", Annals of the CIRP, Vol. 37/2/1988, pp. 591 - 601.

[3]. Eversheim, W., Shulz, J. "Survey of Computer Aided Process Planning Systems", Annals of CIRP Vol 34/2/1985, pp. 607 - 709.

[4]. Alting, L., Zhang, H., "Computer aided process planning: State of the art survey", International Journal of Production Research, 1989, Vol. 27, No 4, 553 - 585.

[5]. Shah, J., Sreevalsan, P., Mathew A., "Survey of CAD/Feature based process planning and NC programming techniques", Computer Aided Engineering Journal, February 1991, pp. 25 - 33.

[6]. Eversheim, W., Esch. H., "Automated Generation of Process Plans for prismatic Parts", Annals of CIRP, Vol. 32/1/1983, pp. 361 - 364.

[7]. Eskicioglu H., Davis, B.J., 'An Interactive Process Planning System for Prismatic Parts (ICAPP)', Annals of the CIRP Vol. 32/1/1983, pp. 365 - 370.

[8]. Peklenik, J. Sekolonik, R., 'Development of Part Spectrum Database for Computer Integrated Manufacturing Systems (CIMS), Annals of CIRP Vol. 39/1/1990, pp. 471-474.

[9]. Pande, S.S., Palsule, N.H., 'GCAPPS-a computer assisted generative process planning system for turned components', Computer Aided Engineering Journal, August 1988. pp. 163 - 168.

[10]. Pande, S.S., Walvekar, M.G., 'PC-CAPP - a computer assisted process planning system for prismatic components', Computer Aided Engineering Journal, August 1989, pp. 133 - 138.

[11]. Matsushima, K., Okada, N.,Sata, T., 'The integration of CAD and CAM by application of Artificial Intelligence techniques (TOM), 'Annals of CIRP Vol. 31/1/1982, pp. 329 - 332.

[12]. Davies B. J., "Applications of Expert system in process planning", Annals of the CIRP VOL. 35/2/1986, pp. 451-452.

[13]. Giusti, F., Santochi, M., Dini, G., University Of Pisa, Itali, "KAPLAN: a Knowledge Based Approach to process planning of rotational parts", Annals of the CIRP VOL. 38/1/1989, pp. 481 - 484.

[14]. Houten, F.J.A.M., Kals, 'Strategy in generative planning of turning process', Annals of CIRP, Vol. 35/1, 1986, pp. 331 - 335.

[15]. Erve, Kals, 'XPLANE, Generative CAPP System for part manufacturing', Annals of CIRP, Vol. 30/1, 1986, pp.

[16]. Houten, F.J.A.M., Erve, A.H., Jonkers, F.J.C.M., Kals, H.J.J., "PART, a CAPP System with a flexible architecture", Proceedings of CIRP International Workshop on Computer Aided Process Planning, Hanover University, September 21-22, 1989, pp. 57 - 69.

[17]. Houten, F.J.A.M., Erve, A. H., Boogert, R.M., Nauta, Kals, H.J.J., "PART, Selection of Machining Methods and Tools", Proceedings of the 22nd CIRP International Seminar on Manufacturing Systems, University of Twente, June 11-12, 1990, pp. 1 - 19.

[18]. Jonkers, F.J.C.M., Kals, H.J.J., 'The development of the software architecture for PART, a CAPP system', 24th CIRP International Seminar on Manufacturing Systems, The Technical University of Denmark, Copenhagen, June 11-12, 1992, pp. 71 - 80.

[19]. Vin, L.J., Vris, J., Streppel, A.H., Kals, H.J.J., 'PART-S, a CAPP System for small batch manufacturing of sheet metal components', 24th CIRP International Seminar on Manufacturing Systems, The Technical University of Denmark, Copenhagen, June 11-12, 1992, pp. 171 - 182.

[20]. Tulkoff, J., 'Processing planning in the computer integrated factory', CIM Review, Fall 1987, pp. 24 - 27.

[21]. Process Planning: the best buys, 'Industrial Computing, June 1990, p 40.

[22]. CIMPLAN, Product information brochure, October, 1989, CIMTEL LTD, 2 Holt Count, Aston Science Park, Jennens Road, Birmingham, B7 4EJ.

[23]. CEEQUEL, CAPP, Product brochure, SCOTT-GRANT, Computer Services Ltd., 26 Cross Street, Manchester M2 7AN.

[24]. C-PLAN, Product brochure, CADCentre, UK, pp. 1 - 15.

[25]. LOCAM, The manufacturing decision system, Product brochure, Pafec Ltd, Strelly Hall, Nottingham NG8 6PE, UK, pp. 1 - 9.

[26]. Logan, F., 'Process Planning Via Manufacturing Codes', CAPP Conference, April 16-17, 1985, Anaheim, California, pp. 1 - 13.

[27]. Hargreaves, J.,'CIM, a users experience', Sandvik Coromant UK, pp. 1-5.

[28]. SUPERCAPES, Product brochure, SD SCICON, Wavendon Tower, Wavendon, Milton Keynes MK17 8LX, pp. 1 - 13.

[29]. HMS CAPP, Version 1.2, Product information brochure, Houtzeel Manufacturing Systems Inc., P.O.Box 1605, Waltham, MA 02254 -1605 USA,

[30]. MetCAPP, Product information brochure, Institute of Advanced Manufacturing Sciences, An Edison Technology Centre, 1111 Edison Drive, Cincinnati, Ohio 45216-2265, USA.

[31]. Lawler, B.D., Rogers, C.E., Amster B.R. "Generative process planning for PDES: RAMP goes beyond the routing Sheet.", pp. 1 - 17.

[32]. Warnecke, H.J., Mayer, Ch., Muthsam H., 'New tools in CAPP', Proceedings of CIRP International Workshop on CAPP, Hanovei University, Sept. 21-22, 1989, pp. 169 - 181.

[33]. Strohmeier, A.H., 'Implementing Computer Aided Process Planning: Rockwell International Case Study, CIM Reviw, Fall 1987, pp. 34-41.

[34]. Bakerjian, R., 'Tool and Manufacturing Engineers Handbook', Volume 6, Design for Manufacturability, SME publication, fourth edition,1992.

[35]. Groover, M.P., Zimmers, E.W., 'CAD/CAM: Computer Aided Design and Manufacturing', Prentice-Hall International Editions, 1984.

[36]. Hyer, N.L., Wemmerlov, U., 'Group Technology in US Manufacturing industry: a state of current practices', International Journal of Production Research, 1989, Vol. 27, No. 8, pp. 1287 - 1304.

[37]. Tatikonda, M.V., Wemmerlov, U., 'Adoptation and implementation of group technology classification and coding systems: inside from seven case studies', International Journal of Production Research, 1992, Vol 30, No. 9, pp. 2087 - 2110.

[38]. Iwata, K., Fukuda, Y., "Knowledge based flexible part classification system", Annals of the CIRP Vol. 36/1/1987, pp. 317 - 320.

[39]. MultiCAPP II , Product brochure, Organization for Industrial Research, 100 Crosby Drive, Bedford, Massachusetts, USA.

[40]. Opitz, H., Wiendahl, H.P., 'Group Technology and manufacturing systems for small and medium quantity production', International Journal of Production Research, 1971, Vol. 9, No. 1, pp. 181 - 203.

[41]. Burbidge, J.L., 'Change to group technology: process organisation is obsolete', International Journal of Production Research, 1992, Vol. 30, No. 5, pp. 1209 - 1219.

[42]. Opas, J., Kanerva, F., Mantyla, M., 'Automatic process planning system in an operative process planning system', International Journal of Production Research, 1994, Vol. 32, No. 6, 1347 - 1363.

[43]. Zeid, I., 'CAD/CAM Theory and Practice', McGraw-Hill, Inc., USA, 1991.

[44]. McMahon, C., Browne, J., 'CAD/CAM From Principles to Practice', Addison-Wesley, UK, 1993.

[45]. Chang. T.C., "Expert process planning for manufacturing", Addison-Wesley, USA, 1990.

[46]. Computervision Parametric Modelling User Guide, Revision 4.0.0, 1993.

[47]. Spur, G., Krause, F. L., Kempf, M., Institute for Machine Tools and Production Technology (IWF), Technical University Berlin, "Advanced Software Tools for the Development of CAPP - Systems", Proceedings of 22nd CIRP International Seminar on Manufacturing Systems, June 11-12, 1990.

[48]. Ralston, D., Munton, T., IBM, UK, 'Computer Integrated Manufacturing', Computer Integrated Engineering Journal, August, 1987, pp. 167 - 174.

[49]. Wolfe, P.M., 'CAPP is link between CAD and CAM', Industrial Engineering, August, 1985, pp. 72 - 77.

[50]. Allen D.K., 'An Introduction to Computer Aided Process Planning', CIM Review, Fall 1987, pp. 7 - 23.

[51]. Marchal, P., 'Computer Aided process planning and estimating as part of an integrated CADCAM system', Computer Aided Engineering Journal, October 1985, pp. 167 - 172.

[52]. Shah, J.J., Hsiao, D., Leonard, J., 'Geometric modelling for product realisation', IFIP Transaction, Vol. B-8, Elsevier, 1993.

[53]. Granville, C.S., President, CimTelligence 'Thinking CAPP', CIM Review, Spring, 1990, pp. 50 - 56.

[54]. Joshi, S, Chang, T.C., 'Feature extraction and feature based design approaches in the development of design interface for process planning', Journal of Intelligent Manufacturing 1990, 1, pp. 1-15.

[55]. Joshi, S., Vissa, N., Chang, T., 'Expert process planning system with solid model interface', Int. J.. Prod. Research, 1988, Vol. 2 6, No 5, pp. 863-885.

[56]. Lee, K., Lee,J., Lee, J.M., 'Pattern Recognition and Process Planning Prismatic Workpieces by knowledge based approach', Annals of CIRP Vol. 38/1/88, pp. 485 - 488.

[57]. Choi, B.K., Barash, M.M., Anderson, D.C., 'Automatic recognition of machined surfaces from a 3D solid model', Computer Aided Design, Vol. 16, No., 2, 1984, pp. 81 - 86.

[58]. Choi, B.K., Barash, M.M., 'STOPP: an approach to CADCAM integration', Computer Aided Design, Vol. 17, No. 4, 1985, pp. 162 - 168.

[59]. Chuang, S.H., Henderson, M.R., 'Three dimensional shape pattern recognition using vertex classification and vertex-edge type shape', Computer Aided Design, Vol. 22, No. 6, August 1990, pp. 377 - 387.

[60]. Perng, D., Chen, Z., Li, R., 'Automatic 3D machining feature extraction from 3d CSG solid input', Computer Aided Design, Vol. 22, No. 5, pp. 285 - 295.

[61]. Herbert, P.J., Hinde, C.J., Bray, D., Launders, A., Round, D., Temple, D.M., 'Feature Recognition within a truth maintained process planning system', International Journal of Computer Integrated Manufacturing, 1990, Vol. 3, No. 2, pp. 121 - 132.

[62]. Foong, F.L., Patil, D., 'Customising NC Programming on the CAD/CAM System, 'International Manufacturing Conference with China, March, 1993.

[63]. Hummel, K.E., Brook, S.L., 'Symbolic representation of manufacturing features for an automated process planning system', ASME pub. PED vol. 24 Dec. 1986, pp. 233-243.

[64]. Milacic, V.R., 'SAPT- Expert System for Manufacturing Process Planning', Computer Aided/Inteligent Process Planning Symposium, Miami Beach, November. 1985, pp. 43 - 53.

[65]. Case, K., Gao, J., 'Feature Technology- An overview', Symposium on Feature Based Approaches to design and process planning', Loughbourough University of Technology, UK, "4-"5 Sept. 1991, pp. 1 - 14.

[66]. Pratt, M.J., Wilson, P.R., ' Requirement for support of form features in a solid modelling system', CAM-I, R-85-ASPP-01, 1985.

[67]. Lenau, T., Mu, L., 'Features in integrated modelling of products and their production', Symposium on Feature Based Approaches to design and process plannning', Loughbourough University of Technology, Uk, 4 - 5 Sept. 1991.

[68]. Gindy, N.N.Z., 'A hierarchical structure of form features', Int. J.. Prod. Research, Vol. 27, No 12, pp. 2089-2103.

[69]. Gindy, N.N.Z., 'A reference model for generative Planning system', 24th CIRP International Seminar on Manufacturing Systems, The Technical University of Denmark, Copenhagen, June 11-12, 1992, pp. 61 - 69.

[70]. Gindy, N.N.Z, Huang, X., Rachev, T. M., "Feature based component model for computer aided process planning systems", Symposium on Feature Based approaches to design and process planning, 24-25 Sept., University Of Technology, Loughborough, UK, pp. 1 - 9.

[71]. Shah,J.J., Rogers, M, 'Functional requirements and conceptual design of the Feature-Based Modelling System, 'Computer Aided Engineering Journal February 1988, pp. 9 - 15.

[72]. Shah, J.J., 'Feature transformations between application specific feature spaces', Computer Aided Engineering Journal, December, 1988, pp. 247 - 255.

[73]. Chung, J.C.H, Cook, R.L., Patal, D., Simmons, M.K., 'Feature based geometry construction for geometric reasoning', Computers in Engineering, Vol. 1, 1988.

[74]. Luby, S.C., Dixon, J.R., Simmons, M.K., 'Designing with features: creating and using a feature database for evaluation of manufacturability of casting', Computers in Engineering, July 1986.

[75]. Cunningham, J.J., Dixon, J.R., 'Designing with featurtes: the origin of features', Computers in Engineering, Vol. 1, 1988.

[76]. Kruse, F.L., Reiger, E. , 'PDGL: A Language for Efficient Feature Based Product Gestaltung', Annals of the CIRP Vol.40/1/1991, pp. 135 - 138.

[77]. Eversheim, W., Cremer, R., Schneewind, J., 'A methodology for the flexible development of Integrated Process Planning System', 24th CIRP International Seminar on Manufacturing Systems, Copenhagen, Denmark, June 11-12, 1992. pp. 49 - 60.

[78]. Klauck, C., Legleitner, R., Bernardi, A., 'FEAT-REP: Representing features in CAD/CAM', Research Report RR-91-20, German Research Centre for Artificial Intelligence, Kaisertslautern, pp. 1 - 33.

[79]. Kimura, F., Suzuki, H., Inui, M., Ranta, M., "Product feature modelling for integrating product design and process planning", Proceedings of 22nd CIRP International Seminar on Manufacturing Systems, June 11-12, 1990.

[80]. Faux, I.D., 'Reconciliation of design and manufacturing requirements for product description data using functional primitive part features', CAM-I Report R86-ANC-GM-PP-01, 1986.

[81]. Roy, U., Liu, C., 'Feature based representational scheme of a solid modeller for providing dimensioning and tolerancing information', Robotics and Computer Integrated Manufacturing, Vol. 4, No 3/4, pp. 335-345.

[82]. Ranyak P.S., Fridshal, R., 'Features for tolerancing a solid model', General Dynamics Corporation, San Diego, California, USA, Computers in Engineering, 1988, Vol.1, pp. 275 - 279.

[83]. Pande, S.S., Desai, V.S., "GFM-an interactive feature modeller for CAPP of rotational components", Computer Aided Engineering Journal, Oct. 1991, pp. 217 - 220.

[84]. Lenau, T., 'Integrating process planning with product design (Design / CAPP integration), 24th CIRP International Seminar on Manufacturing Systems, The Technical University of Denmark, Copenhagen, June 11-12, 1992, pp. 21 - 34.

[85]. Sohlenius G., "Concurrent Engineering", Annals of CIRP Vol. 41/2/1992, pp. 645-655.

[86]. Murphy, G., 'Implications of Concurrent Engineering in Shorts', M.Sc. dissertation, Dept. Of Engineering, University of Warwick, Sept. 1994.

[87]. C.H. Link, 'CAPP - CAM-I Automated Process Planning System', CAM/NC Conference, CAM-I Arlington, Texas, USA, 1976. pp. 401 - 408.

[88]. Houtzeel, A., Schilperoort, B., A., TNO Waltham, Massachusetts, CAM/NC Conference, CAM-I Arlington, Texas, USA, 1976. pp. 383 - 400.

[89]. Park, M.W., Vosniakos, Davis, B.J., 'Automatic feature extraction in wire-frame models of 21/2-D Prismatic parts', proceedings of the 28th MATADOR, UMIST, 18-19 April 1990, pp. 83 - 89.

[90]. Mayers, L., Pohl, J., 'ICADS expert design advisor: an aid to reflective thinking', Knowledge Based Systems, Vol. 5, No. 1, March, 1992, pp. 41 - 54.

[91]. Houten, F.J.A.M., "Manufacturing Interfaces", Annals of the CIRP Vol. 41/2/1992, pp. 699 - 709.

[92]. Lewis, S. 'Get into STEP', CADCAM, March 1994, pp. 65 - 70.

[93]. Medhat, S, 'CE's Enabling Technologies', CADD, April 1994, pp. 20 - 22.

[94]. ISO 10303-1 : 1993 - Product Data Representation and Exchange Part 1 : Overview and fundamental principles.

[95]. ISO 10303-11: 1992 - Product Data Representation and Exchange Part 11: Description Methods: The Express Language Refe:ence Manual.

[96]. CV STEP Newsletter, Issue 4, April 1994.

[97]. Sohlenius G., Kjellberg, T., The Swedish Institute of Production Engineering Research, "Artificial Intelligence and its Potential use in the manufacturing system", Annals of the CIRP VOL. 35/2/1986, pp. 425 - 432

[98]. Wang, H., Wysk, R.,'A knowledge based approach for process planning', Int. J.. Prod. Research, 1988, Vol. 26, No. 6, pp. 999 - 1014.

[99]. Kumara S. R. T., Joshi, S., Kashyap, R., Moodie, C., Chang, T., "Expert systems in industrial Engineering",
Int. J.. Prod. Research, 1986, Vol. 24, No. 5, pp. 1107 - 1125.

[100]. Zust R., Taiber, J.,Schultschik. R., "Knowledge based process planning system for prismatic workpieces in a CAD/CAM environment., Annals of the CIRP VOL. 39/1/90, pp. 493 - 496.

[101]. Satochi, M., Dini, G., 'Flexible and process integrated knowledge bases as instruments for modern CAPP systems', Proceedings of CIRP International Seminar on Manufacturing Systems, June 11-12, 1990, pp. 1 - 13.

[102]. Nau, D.S., Chang, T.S., 'A knowledge-based approach to generative process planning', Computer Aided Intelligent Process Planning Symposium, Miami Beach, November 1985, pp. 65 - 70.

[103]. Zhang, K.F., Wright, A.J., Davis, J., 'A feature recognition knowledge base for process planning of rotational mechanical components', International Journal of Advanced Manufacturing Technology, 1989, pp. 13 - 25.

[104]. Spur, G., Krause, F.L., Major, F.W., 'Advanced software technologies for CAPP', Proceedings of CIRP International Workshop on CAPP, Hanover University, Sept. 21-22, 1989, pp. 121 - 137.

[105]. Vogt, H. G., Zaring, P.A., Dept. of Production Technology, Chalmers University of Technology, Sweden,
'IHOPE, The Interactive Hole Operation Planning Expert', Proceedings of CIRP International Seminar on Manufacturing Systems, June 11-12, 1990, pp. 1 - 13.

[106]. Date, C. J., 'An Introduction to Data Base Systems', Addison Wesley, 1986.

[107]. Prijic,A., 'Development of a tool management system using Oracle', M.Sc. dissertation, Dept. Of Engineering, University of Warwick, Sept. 1994.

[108]. Hurson A.R., Pakzad, S.H, Cheng,J.B., Object Oriented Database Management Systems: Evolution and Performance Issues', Computer (USA), Feb. 1994.

[109]. Gomsi, J., Desanti, M., 'A technical comparison of relational and object oriented databases for manufacturing applications', Data Resource Management, Vol. 3, Iss. 1, pp. 40 - 47.

[110]. Tonshoff, H.K., Dittmer, H., "Object instead of function oriented data management for tool management", Robotics and Computer Integrated Manufacturing, Vol 7, no 1/2, pp 133-141, 1990, pp. 133 - 141.

[111]. Computervision CADDS5 CAMU User Guide.

[112]. Computervision CADDS5 Feature Based Design User Guide.

[113]. Engineering Data Manager (EDM) Training Manual from Computervision, 1992.

[114]. Shongwe, P.M., 'Engineering Data Management', M.Sc. Dissertation, August 1993.

[115]. Hall, M.P., 'Configuration Access, Administrator guide', from Computervision, December, 1993.

[116]. Lenau, Torben, Alting Leo, Institute of Manufacturing Engineering, The Technical University Of Denmark, "Prerequisites for CAPP", Proceedings of CIRP International Seminar on Manufacturing Systems, June 11-12, 1990, pp. 1 - 9.

[117]. Young, R.I.M., Bell, R., 'Design by features, advantages and limitations in machine planning integration', Symposium on Feature Based Approaches to design and process plannning', Loughbourough University of Technology, Uk, 4 - 5 Sept. 1991.

[118]. Elmaraghy, H.A., Elmarghy, W.H., 'Bridging the gap between process planning and production planning and control', 24th CIRP International Seminar on Manufacturing Systems, The Technical University of Denmark, Copenhagen, June 11-12, 1992, pp. 1 - 7

[119]. Schildt H., 'Artificial Intelligence using C', the C programmer's guide to AI techniques', McGraw-Hill, Osbourne, USA.

[120]. Relational Systems Design, Oracle Course Notes

[121]. ProC Supplement to the Oracle Precompilers Guide, Version 1.5

[122]. Tonshoff, H.K., Beckendorff Ulrich, Anders Nico, Institute of Production Engineering and Machine Tools, University Of Hannover, "FLEXPLAN - A concept for intelligent process planning and scheduling", Proceedings of CIRP International Workshop on Computer Aided Process Planning, September 21-22, 1989, pp. 87 - 106.

[123]. Introduction to ORACLE: SQL, SQL Plus and PL/SQL, Course notes.

[124]. Latif, M., Boyd, R.D., Hannam, R.G., 'Studies on integrating manufacturing intelligence through features', Symposium on Feature Based Approaches to design and process plannning', Loughbourough University of Technology, Uk, 4 - 5 Sept. 1991.

[125]. Smalltalk/V286 Object Oriented Programming System, Tutorial and Programming Handbook, Digitalk Inc., Los Angeles California 90045, USA, May 1988.

[126] Chang. T.C., 'Geometric Reasoning - The key to integrated process planning', Proceedings of 22nd CIRP International Seminar on Manufacturing Systems, June 11-12, 1990.

[127] Butterfield, W.R., Green, M.K., Scott,D.C., Stoker, W.J., 'Part features for process planning', CAM-I Report R85-PPP-03, 1986.

[128]. Liu, T., Fischer, G.W., 'Developing Feature-Based Manufacturing Applications using PDES/STEP', Concurrent Engineering Research and Applications, Vol 1, 1993, pp. 39-50.

[129]. Qiao,L., Zhang, C., Liu, T.,Fischer, G.W.,'A PDES/STEP-based product data preparation procedure for computer aided process planning', Computers in Industry, Vol.21, 1993, pp. 11-22.

[130]. Gu, P., Zhang, Y., Norrie, D. H., "Object-Oriented product modelling for computer-integrated manufacturing", Int. J. Computer integrated Manufacturing, Vol. 7, No. 1, 17-28, 1994.

[131]. Shah, J.J., Bhatnagar, A.S., 'Group technology classification from feature based geometric models', Manufacturing review, Sept. 1989, pp 204 - 213.

[132]. Mill, F.G., Salmon, J.C., Pedley, A.G., 'Representation problems in feature-based approaches to design and process planning', International Journal Of Computer Integrated Manufacturing, 1993, Vol 6, Iss. 1-2, pp. 27-33.

[133]. Pratt, M.J., 'Automated Feature Recognition and its role in Product Modelling', Computing Supplement, 1993, Iss. 8, pp. 241-250.

[134]. CAD/CAM news, 'CV launches object oriented toolkit for design automation', CAD/CAM Journal, February, 1995.

[135]. Liao, T.W., Coates, E.R., Aghazadesh, F., Mann, L., Guha, N., 'Modification of CAPP Systems for CAPP/Scheduling Integration', Computers and Industrial Engineering, 1994, Vol 26, Iss 3 pp 451 - 63.

[136]. Whitney, D.E., 'Integrated Design and Manufacturing in Japan', Manufacturing Review, Vol 6., No. 4, December, 1993.

[137]. Horvath, L., Rudas, I.J., 'A Machine learning based approach to manufacturing process planning', IEEE International Symposium on Industrial Electronics, Budapest, Hungary, June 1-3, 1993.

[138]. Tonshoff, H.K., Kreutzfeldt, J., Hofschneider, D., 'Concurrent process planning and workshop control in batch production', Proceedings of CIRP International

Workshop on Computer Aided Manufacturing Systems, june 11-12, 1992, pp. 315 - 328.

[139]. Gomsi, J., Desanti M., 'Relational and Object Oriented: Understanding the database debate', CIM Review, Fall 1990.

[140]. Patil, D., Hill, J., 'Integrated Product Design Using Features', Proceedings of 6th International Manufacturing Conference with China, HongKong, March 10 - 12, 1993, Vol. 2, pp. 279 - 287.

[141]. Patil, D., Hill, J., 'Integrated Product Design Using Features', Proceedings of the International Conference on Design to Manufacture in Modern Industry, Bled, Slovenia, June 7-9, 1993, Vol. 1, pp. 334 - 344.

[142]. Gardan, Y., Minich, C., 'Feature Based models for CAD/CAM and their limits', Computers in Industry, 1993, pp. 3 - 13.

[143]. Xu, X., Hinduja, S., 'Positioning a 2½ - D component in its blank', IMechE conference transactions on the 11th International Conference on Computer Aided Production Engineering, 20-21 Sept. 1995, pp. 41 - 47.

[144]. Ferreira J.C.E, Hinduja, S., 'Convex hull based feature-recognition method for 2.5D components', Computer Aided Design, Vol 22, No 1, 1990, pp. 41 - 49.

[145]. Tseng, Y.J., Joshi, S.B., 'Recognising multiple interpretations in 2½D machining of pockets', International Journal of Production Research, May 1995, pp. 1063 - 1086.

[146]. Kulkarni V.S., Pande S.S., 'A system for automatic extraction of 3D part features using syntactic pattern recognition technques', International Journal of Production Research, 1995, Vol. 33, No. 6, pp. 1569 - 1586.

[147]. Laakko, T., Mantyla, M., 'Feature Modelling by incrimental feature recognition', Computer Aided Design, Vol. 25, No. 8, August 1993, pp. 479 - 492.

[148]. Pedley, A.G., Ehrmann, M., 'Explicit feature interaction modelling in CAD and CAM systems', IMechE conference transactions on the 11th International Conference on Computer Aided Production Engineering, 20-21 Sept. 1995, pp. 269 - 275.

[149]. Hinduja, S., Huang, H., 'OP-PLAN: an automated operation planning system for turned components', Proceedings of Institution of Mechanical Engineers, 1989, Vol. 203, pp. 145 - 158.

[150]. Warnecke, G., Martens, P., Schulz, C., 'A proposal for the integration of CAD and CAPP', Proceedings of CIRP International Workshop on CAPP, Hanover University, Sept. 21-22, 1989, pp. 185 - 196.

[151]. Shah, J.J., Mantyla M., 'Parametric and Feature based CAD/CAM', Wiley-Interscience Publication, 1995.

[152]. Hill, F.S., 'Computer Graphics', Macmillan Publishing Company, New York, 1990.

APPENDIX A

TAXONOMY OF SYSTEM FEATURES

This appendix details the taxonomy of the system-defined features for use within the CADDS5 Parametric environment[112].

Form Features Information Model

The Form Feature Information Model, or FFIM, presents a conceptual model of form feature information. The FFIM is based upon a structure developed for the Product Data Exchange Standard, or PDES/STEP. The FFIM is intended to provide a good foundation for user defined features, common methodology and data exchange. The following section describes some key terms.

Form Feature

A portion of a shape which, for the purpose of some application, is treated as an entity.

Volume Feature

A form feature that is view as a volume added to or subtracted from the previous state of the workpiece.

Additive Feature

A volume feature whose volume is added to the previous state of the workpiece. The value of the static attribute Operation, inherited from class Feature, is set to Add.

Protrusion

An additive feature that extends outward from the previous state of the Workpiece at a single place.

Connector

An additive feature whose added volume joins the previous state of the Workpiece at two distinct places. the feature may connect two separate volumes. In other cases, both connections may be with the same volume, so that the feature creates a handle on the Workpiece.

Primitive

An additive feature whose added volume does not connect to the previous state of the Workpiece.

Subtractive Feature

A volume feature whose subtracted volume emerges from the previous state of the Workpiece at a single opening.

Passage

A subtractive feature whose subtracted volume emerges from the previous state of the Workpiece at two distinct openings.

Void

A subtractive feature whose volume is completely enclosed by the previous state of the Workpiece.

System Features

A library of System features, named SYSLIB, is provided. The following section describes those system features.

Tab

A short thin strip projecting from the edge or surface of an object.

Boss

A flat surface or regular or irregular shape protruding slightly above the nominal Workpiece surface and often around fastening holes to provide a seat to which other components may be attached.

Circular Boss

A short cylindrical projection from the surface of a fabricated Workpiece.

Counterbore

Cylindrical enlargement on the end of a hole. Often used to provide a recess for cap screw heads.

Countersink

Conical enlargement on the end of a hole. Often used to receive the head of a screw or rivet.

Pocket

A shallow non-through-going cavity in the surface of a Workpiece. Often used for weight reduction or as a space for component parts.

Groove

A closed circular channel on the rotational surface of a workpiece. Usually for packing, seal, fluid channel, and so forth.

Flat

On an axial part, a planar surface on the workpiece, usually machined, and nominal to the centreline.

Slot

A long, narrow linear or circular channel of square or angular cross-section.

Cutout

In sheet metal, a cut opening, regular or irregular in shape which has a continuous periphery (it doesn't intersect the edge of the Workpiece). A cutout is often used as a view port, access way, or for weight reduction.

Charts

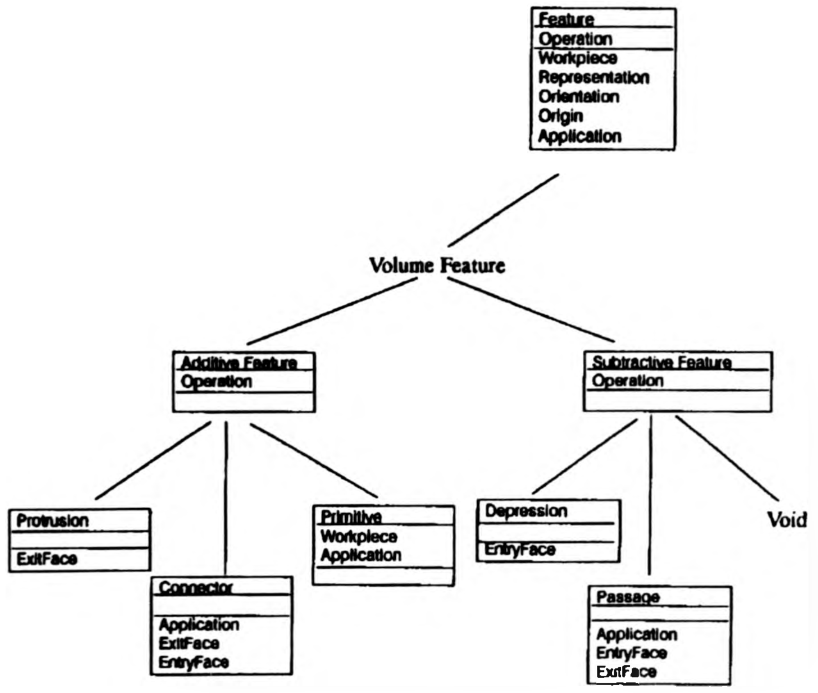
The following chart, Form Feature Information Model, shows the current FFIM classification of form features. Each box in the diagram represents a feature class and its corresponding attributes. The feature class boxes are divided into three sections:

1. The Class Name
2. The static or class attributes
3. The attributes, or instance attributes

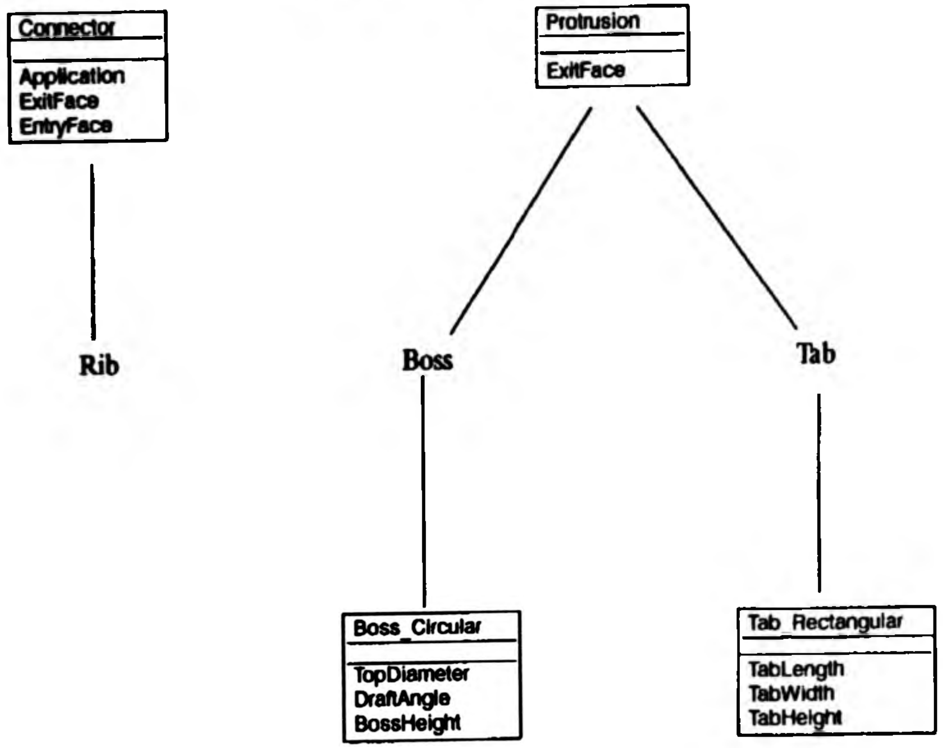
Features with no new attributes may be shown without an accompanying box.

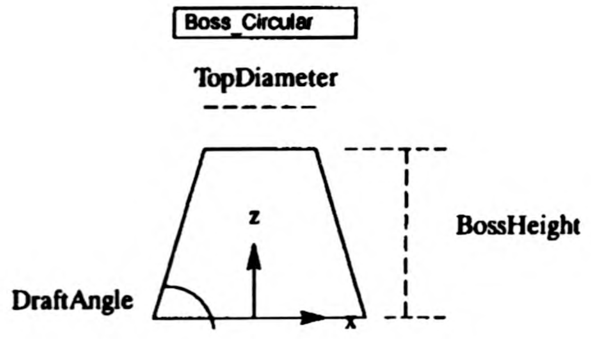
The subsequent chart, System features, spans several pages and shows the taxonomy of individual system feature; accompanying illustrations show how attributes of size relate to their representations.

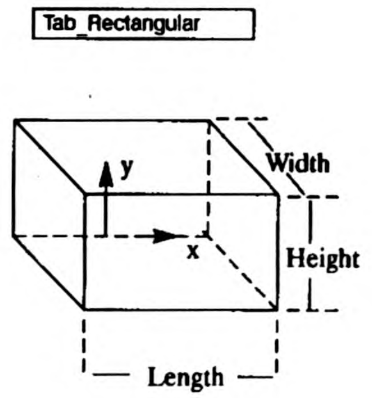
Form Feature Information Model (PDES)

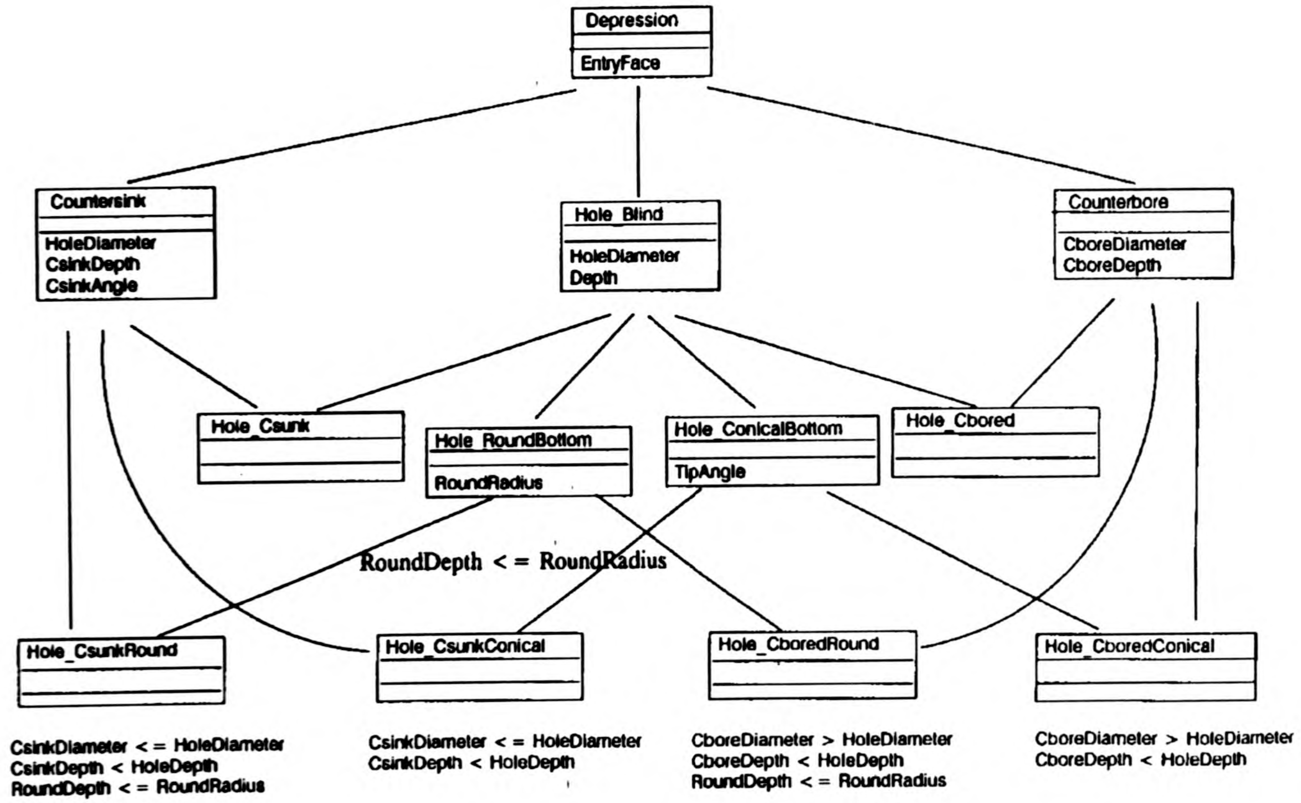


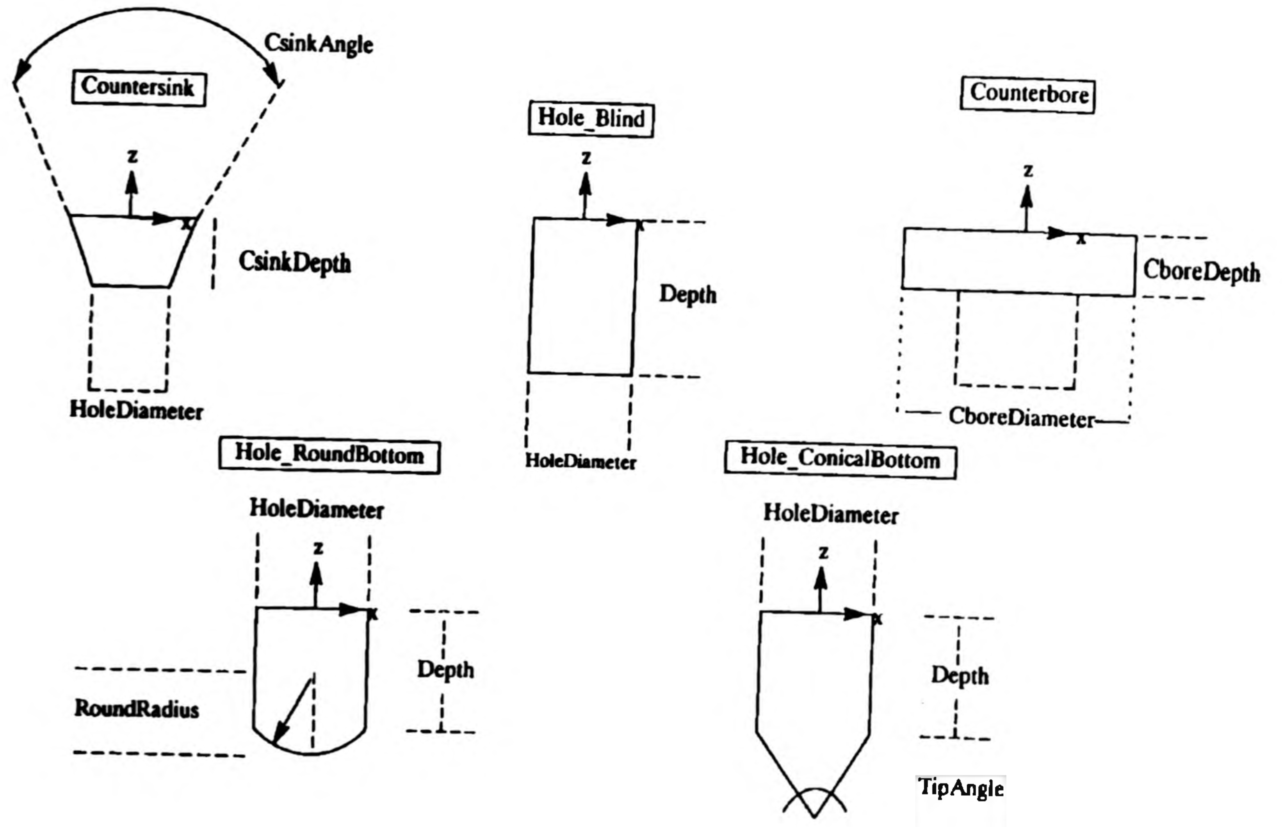
System Features

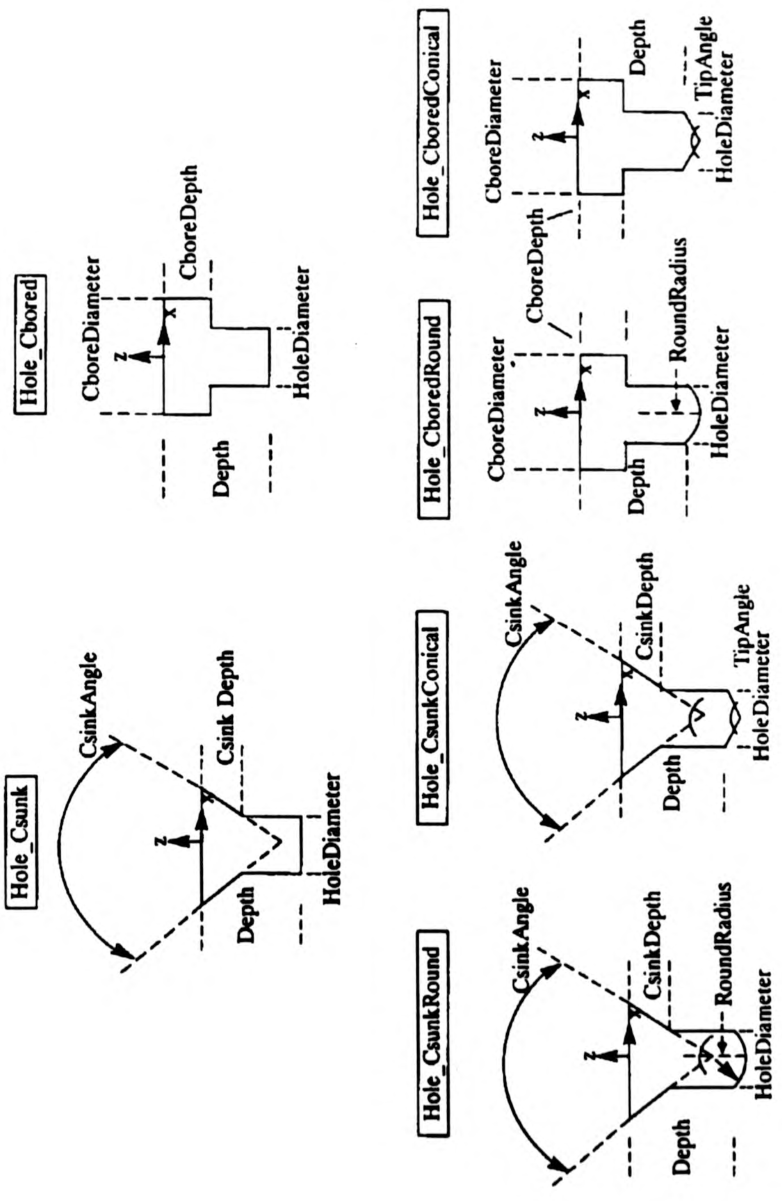


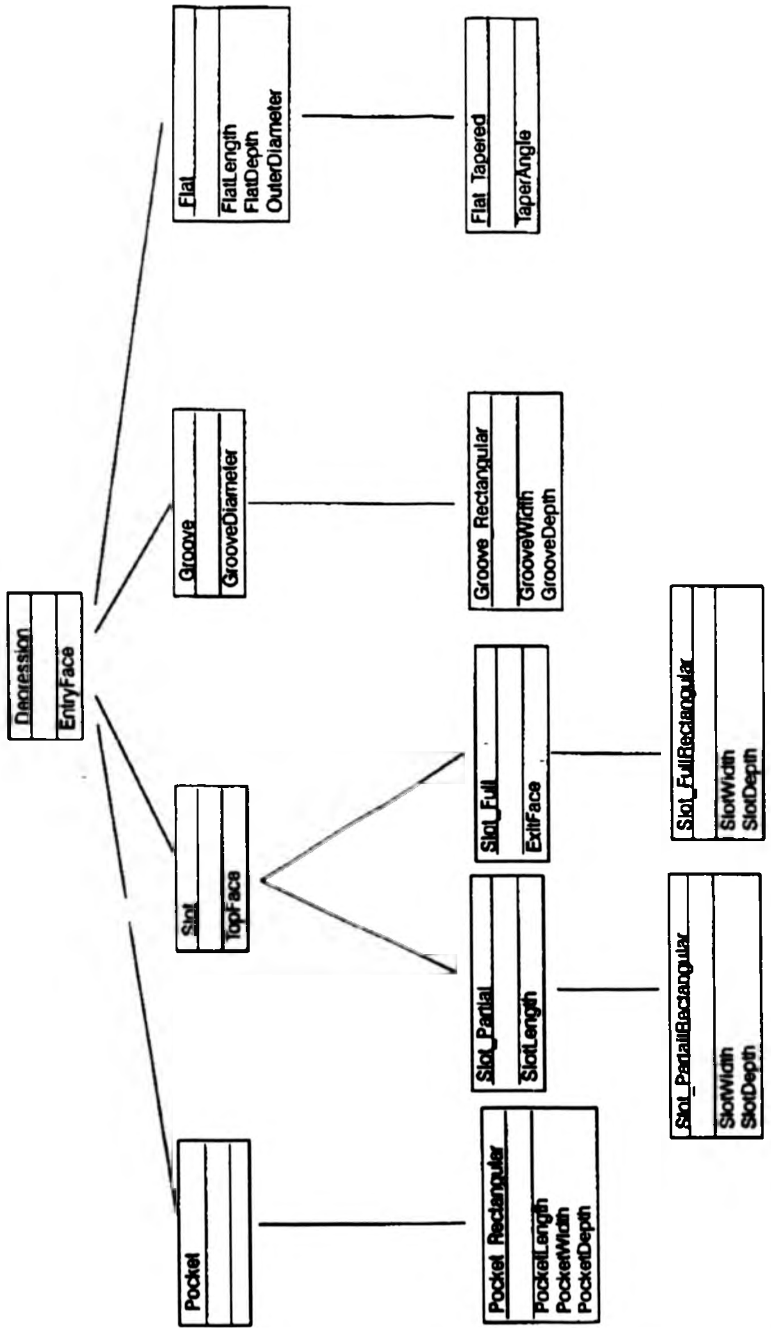

Boss_Circular
TopDiameter
z
BossHeight
DraftAngle
x


Tab_Rectangular
y
Width
x
Height
Length

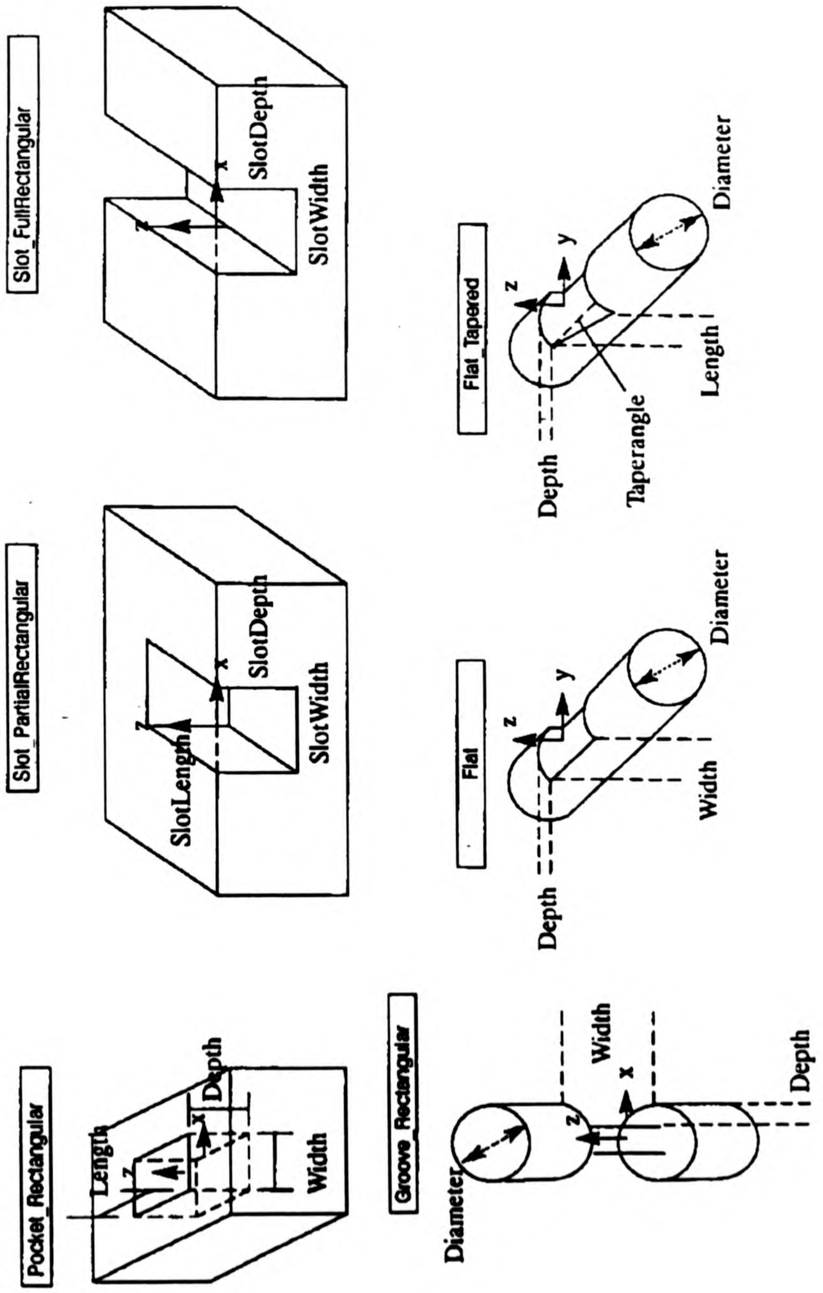

Depression
EntryFace
Countersink
HoleDiameter
CsinkDepth
CsinkAngle
Hole_Blind
HoleDiameter
Depth
Counterbore
CboreDiameter
CboreDepth
Hole_Csunk
Hole_RoundBottom
RoundRadius
Hole_ConicalBottom
TipAngle
Hole_Cbored
RoundDepth < = RoundRadius
Hole_CsunkRound
Hole_CsunkConical
Hole_CboredRound
Hole_CboredConical
CsinkDiameter < = HoleDiameter
CsinkDepth < HoleDepth
RoundDepth < = RoundRadius
CsinkDiameter < = HoleDiameter
CsinkDepth < HoleDepth
CboreDiameter > HoleDiameter
CboreDepth < HoleDepth
RoundDepth < = RoundRadius
CboreDiameter > HoleDiameter
CboreDepth < HoleDepth


CsinkAngle
Countersink
z
x
CsinkDepth
HoleDiameter
Hole_Blind
z
x
Depth
HoleDiameter
Counterbore
z
x
CboreDepth
CboreDiameter
Hole_RoundBottom
HoleDiameter
z
x
Depth
RoundRadius
Hole_ConicalBottom
HoleDiameter
z
x
Depth
TipAngle

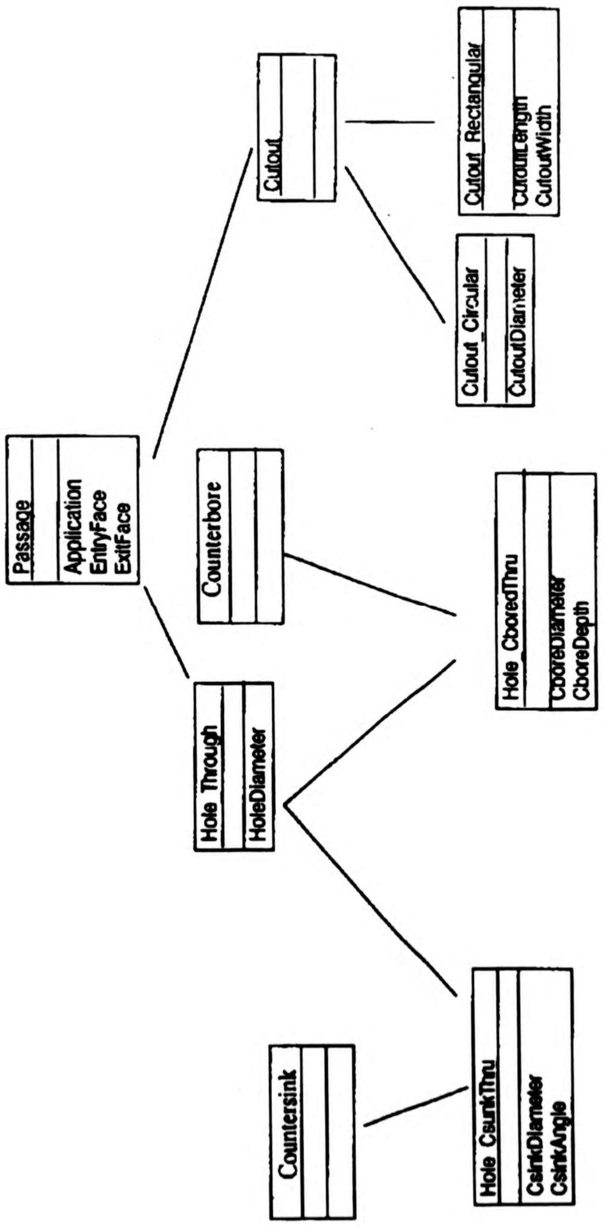

Hole_Csunk
CsinkAngle
Csink Depth
Depth
HoleDiameter
z
x
Hole_Cbored
CboreDiameter
CboreDepth
Depth
HoleDiameter
Hole_CsunkRound
CsinkAngle
CsinkDepth
RoundRadius
Depth
HoleDiameter
Hole_CsunkConical
CsinkAngle
CsinkDepth
TipAngle
Depth
HoleDiameter
Hole_CboredRound
CboreDiameter
CboreDepth
RoundRadius
Depth
HoleDiameter
Hole_CboredConical
CboreDiameter
CboreDepth
Depth
TipAngle
HoleDiameter



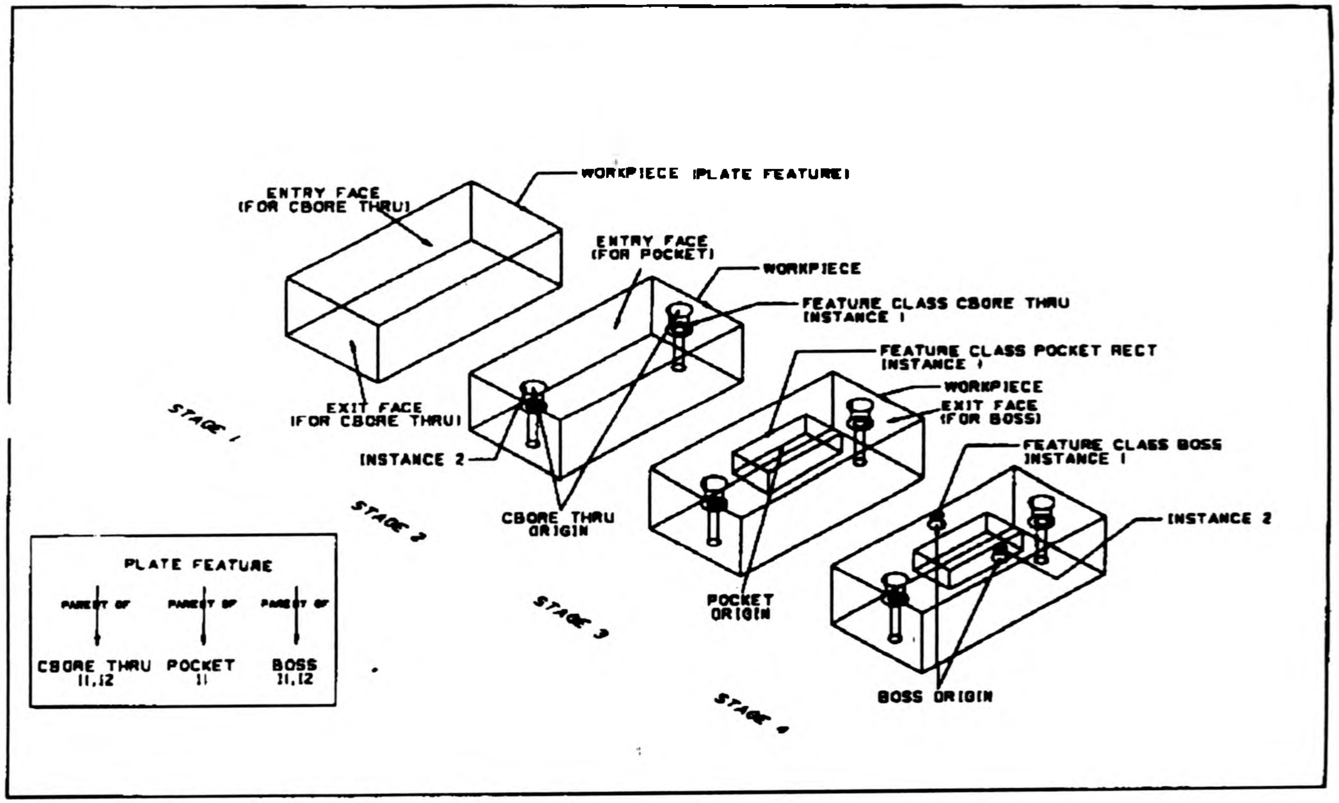
Depression
EntryFace
Flat
FlatLength
FlatDepth
OuterDiameter
Flat_Tapered
TaperAngle
Groove
GrooveDiameter
Groove_Rectangular
GrooveWidth
GrooveDepth
Slot
TopFace
Slot_Full
ExitFace
Slot_FullRectangular
SlotWidth
SlotDepth
Slot_Partial
SlotLength
Slot_PartialRectangular
SlotWidth
SlotDepth
Pocket
Pocket_Rectangular
PocketLength
PocketWidth
PocketDepth



Pocket_Rectangular
Length
Depth
Width
z
x
Groove_Rectangular
Diameter
Width
Depth
x
z
Slot_PartialRectangular
SlotLength
SlotDepth
SlotWidth
z
x
Flat
Depth
Width
Diameter
z
y
Slot_FullRectangular
SlotDepth
SlotWidth
z
x
Flat_Tapered
Depth
Taperangle
Length
Diameter
z
y


Passage
Application
EntryFace
ExitFace
Cutout
Cutout Rectangular
CutoutLength
CutoutWidth
Cutout_Circular
CutoutDiameter
Counterbore
Hole CboredThru
CboreDiameter
CboreDepth
Hole Through
HoleDiameter
Countersink
Hole CsunkThru
CsinkDiameter
CsinkAngle

APPENDIX B



CADDS5 Feature Based Modelling Process For a Sample Component

CADDS5 Report File For a Sample Component

```
*****     printing 1222 Boss_Circular1219_I1     *****

class :
    class : Boss_Circular                        library : SYSLIB
static attributes :
    BooleanOp  Operation                               = Add
attributes :
    Workpiece  Workpiece                               = 1179
    Name       Representation                          = rep
    CPlane     Orientation                          id = 6
                                                  name = TOP
                                                   i x = 1.000000e+00
                                                     y = 0.000000e+00
                                                     z = 0.000000e+00
                                                   j x = 0.000000e+00
                                                     y = 1.000000e+00
                                                     z = 0.000000e+00
                                                   k x = 0.000000e+00
                                                     y = 0.000000e+00
                                                     z = 1.000000e+00
    Location   Origin                                x = 1.000000e+02
                                                     y = 8.750000e+01
                                                     z = 5.000000e+01
    ApplyType  Application                             = Boolean
    Face       ExitFace                                = 1013
    Diameter   TopDiameter                             = 1.500000e+01 = millimeters
    Length     BossHeight                              = 1.000000e+01 = millimeters
    Angle      DraftAngle                              = 5.000000e+00 = degrees
*****     printing 1424 Boss_Circular1219_I2     *****

class :
    class : Boss_Circular                        library : SYSLIB
static attributes :
    BooleanOp  Operation                               = Add
attributes :
    Workpiece  Workpiece                               = 1179
    Name       Representation                          = rep
    CPlane     Orientation                          id = 6
                                                  name = TOP
                                                   i x = 1.000000e+00
                                                     y = 0.000000e+00
                                                     z = 0.000000e+00
                                                   j x = 0.000000e+00
                                                     y = 1.000000e+00
                                                     z = 0.000000e+00
                                                   k x = 0.000000e+00
                                                     y = 0.000000e+00
                                                     z = 1.000000e+00
    Location   Origin                                x = 1.000000e+02
                                                     y = 1.250000e+01
                                                     z = 5.000000e+01
    ApplyType  Application                             = Boolean
    Face       ExitFace                                = 1013
    Diameter   TopDiameter                             = 1.500000e+01 = millimeters
    Length     BossHeight                              = 1.000000e+01 = millimeters
    Angle      DraftAngle                              = 5.000000e+00 = degrees
*****     printing 450 Pocket_Rectangular447_I1     *****

class :
    class : Pocket_Rectangular                   library : SYSLIB
static attributes :
    BooleanOp  Operation                               = Subtract
attributes :
    Workpiece  Workpiece                               = 755
    Name       Representation                       = rep
    CPlane     Orientation                       id = 6
                                               name = TOP
                                                i x = 1.000000e+00
                                                  y = 0.000000e+00
                                                  z = 0.000000e+00
                                                j x = 0.000000e+00
                                                  y = 1.000000e+00
                                                  z = 0.000000e+00
                                                k x = 0.000000e+00
                                                  y = 0.000000e+00
                                                  z = 1.000000e+00
    Location   Origin                             x = 1.000000e+02
                                                  y = 5.000000e+01
                                                  z = 5.000000e+01
    ApplyType  Application                          = Boolean
    Face       EntryFace                            = 782
    Length     PocketLength                         = 5.000000e+01 = millimeters
    Length     PocketWidth                          = 1.000000e+02 = millimeters
    Length     PocketDepth                          = 2.000000e+01 = millimeters
*****     printing 204 Hole_CboredThru201_I1     *****

class :
    class : Hole_CboredThru                 library : SYSLIB
static attributes :
    BooleanOp  Operation                            = Subtract
attributes :
    Workpiece  Workpiece                            = 755
    Name       Representation                       = rep
    CPlane     Orientation                       id = 6
                                               name = TOP
                                                i x = 1.000000e+00
                                                  y = 0.000000e+00
                                                  z = 0.000000e+00
                                                j x = 0.000000e+00
                                                  y = 1.000000e+00
                                                  z = 0.000000e+00
                                                k x = 0.000000e+00
                                                  y = 0.000000e+00
                                                  z = 1.000000e+00
    Location   Origin                             x = 1.750000e+02
                                                  y = 5.000000e+01
                                                  z = 5.000000e+01
    ApplyType  Application                          = Merge
    Face       EntryFace                            = 782
    Face       ExitFace                             = 781
    Diameter   HoleDiameter                         = 1.000000e+01 = millimeters
    Diameter   CboreDiameter                        = 2.000000e+01 = millimeters
    Length     CboreDepth                           = 2.000000e+01 = millimeters
*****     printing 943 Hole_CboredThru201_I3     *****

class :
    class : Hole_CboredThru                 library : SYSLIB
static attributes :
    BooleanOp  Operation                            = Subtract
attributes :
    Workpiece  Workpiece                            = 755
    Name       Representation                       = rep
    CPlane     Orientation                       id = 6
                                               name = TOP
                                                i x = 1.000000e+00
                                                  y = 0.000000e+00
                                                  z = 0.000000e+00
                                                j x = 0.000000e+00
                                                  y = 1.000000e+00
                                                  z = 0.000000e+00
                                                k x = 0.000000e+00
```

```
                                                 y = 0.000000e+00
                                                 z = 1.000000e+00
    Location   Origin                            x = 2.500000e+01
                                                 y = 5.000000e+01
                                                 z = 5.000000e+01
    ApplyType  Application                         = Merge
    Face       EntryFace                           = 782
    Face       ExitFace                            = 781
    Diameter   HoleDiameter                        = 1.000000e+01 = millimeters
    Diameter   CboreDiameter                       = 2.000000e+01 = millimeters
    Length     CboreDepth                          = 2.000000e+01 = millimeters

*****     printing 54 Plate51_I1     *****

class :
    class : Plate                           library : USERLIB
static attributes :
    BooleanOp  Operation                           = Add
    Workpiece  Workpiece                           = 0
    ApplyType  Application                         = NoApply
attributes :
    Name       Representation                      = rep
    CPlane     Orientation                      id = 6
                                              name = TOP
                                               i x = 1.000000e+00
                                                 y = 0.000000e+00
                                                 z = 0.000000e+00
                                               j x = 0.000000e+00
                                                 y = 1.000000e+00
                                                 z = 0.000000e+00
                                               k x = 0.000000e+00
                                                 y = 0.000000e+00
                                                 z = 1.000000e+00
    Location   Origin                            x = 0.000000e+00
                                                 y = 0.000000e+00
                                                 z = 0.000000e+00
    Length     PlateLength                         = 2.000000e+02 = millimeters
    Length     PlateWidth                          = 1.000000e+02 = millimeters
    Length     PlateHeight                         = 5.000000e+01 = millimeters
```

APPENDIX C

[illegible] PRODUCT DESIGN USING FEATURES

ENABLES [illegible] MANUFACTURABILITY [illegible] GENERAL PRACTICE OF DESIGNING A PRODUCT WITH MANUFACTURING IN MIND

[illegible]
[illegible]
PRODUCT MANUFACTURING METHOD [illegible] SELECTION [illegible]

[illegible] DESIGN CREATION [illegible] PRIMITIVES TO REPRESENT PRODUCT SHAPE
[illegible] TOP AND [illegible] HANDLE [illegible] TOP AND [illegible] AIR [illegible]
[illegible] MANUFACTURING NEEDS WHILE DESIGN CREATION
[illegible]
[illegible] FEATURE
[illegible] PRIMITIVE
[illegible]

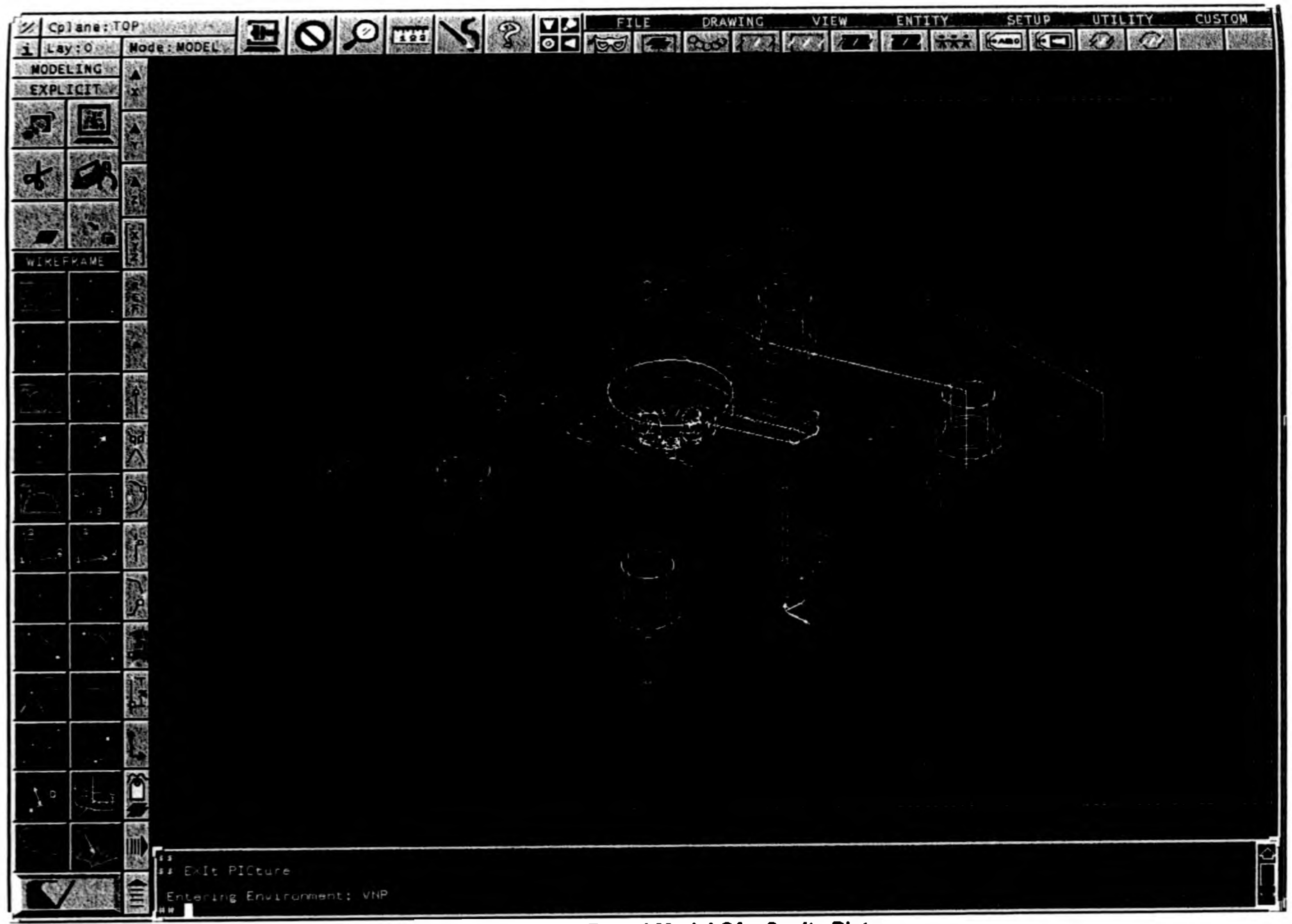
[illegible] FEATURE [illegible] PRIMITIVES
[illegible] ALL THE ATTRIBUTES [illegible] CLASS

CREATE FEATURE [illegible] REPRESENTATION BY APPROPRIATE BOOLEAN
OPERATIONS [illegible] THE PRIMITIVES OF THE FEATURE [illegible] PRIMITIVES
CREATE [illegible] MATERIAL [illegible] AND [illegible] PLATE
[illegible] FEATURES AND OTHER FEATURE [illegible] FROM THE FEATURE LIBRARY
[illegible] INTERPRET [illegible]

[illegible] MODEL [illegible]
[illegible] ATTRIBUTES [illegible] THE MODEL
[illegible] THE [illegible] ATTRIBUTES [illegible] FEATURE [illegible] TOLERANCE CONDITIONS
[illegible] MACHINED [illegible] MATERIAL CONDITION
[illegible] TOLERANCE RANGE
[illegible] SEQUENCE FOR THE FEATURE MACHINING PROCESS
[illegible] MACHINING OF THE COMPONENT

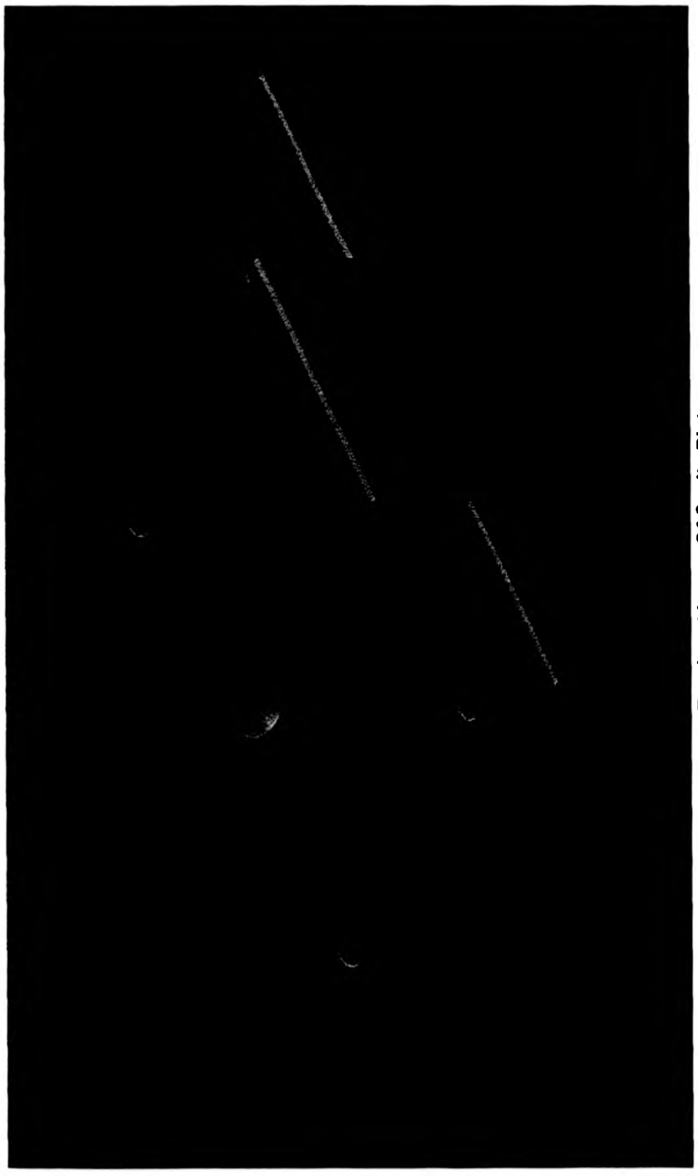
APPENDIX D

This appendix illustrates the user interaction with the CAPP System that was based on initial CAPP System planning. The knowledge and experience gained while developing this system has been used to develop the object oriented CAPP System.



```
## Exit PICture
Entering Environment: VNP
```

CADDS5 Feature Based Model Of a Cavity Plate


Rendered Image Of Cavity Plate

The following script illustrates the user interaction with the CAPP System

```
-----------------------------
WMG CAPP SYSTEM
-----------------------------

Main Menu:
Type 'E' to EnterKnowledgebase

Type 'S' to SaveKnowledgebase

Type 'L' to LoadKnowledgebase

Type 'G' to Generate Process Plan

Type 'X' to Exit
choose one: L

Select loading option -->

Type 'A' for loading process knowledge
Type 'B' for loading tooling knowledge
Type 'C' for loading machine tool knowledge
Type 'D' for loading A, B & C

choose one: A

loading process knowledge base

-----------------------------
WMG CAPP SYSTEM
-----------------------------

Main Menu:
Type 'E' to EnterKnowledgebase

Type 'S' to SaveKnowledgebase

Type 'L' to LoadKnowledgebase

Type 'G' to Generate Process Plan

Type 'X' to Exit
choose one:  G
```

```
Enter Part File Name :
/enterprise/usr5/dilip/parts/lowercase-cavity-plate.rpt

Total features in the database are 6
------------------------------------------------------------
                W M G - C A P P
------------------------------------------------------------
              P R O C E S  S - P L A N
Total 4 no of features need to be machined in setup 6
DATUM-> TOP
-------------------------
PROCESS TO MACHINE Hd_lowercase_cavity :
OPR 1 R. END MILL PARTING SURF
OPR 2 SEMI F. PARTING SURF
OPR 3 R. END MILL CAVITY SURF
OPR 4 SEMI F. CAVITY SURF
OPR 5 DRILL FOR CAVITY CUTOUT
OPR 6 R. & FINISH CAVITY CUTOUT
OPR 7 FINISH PARTING SURF
OPR 8 FINISH CAVITY SURF
-------------------------
PROCESS TO MACHINE Hole_CboredThru :
OPR 1 CENTRE DRILL
OPR 2 DRILL THRU
OPR 3 COUNTERBORE
OPR 4 CHAMFER
OPR 5 REAM HOLE THRU
-------------------------
PROCESS TO MACHINE Tab_Rectangular :
OPR 1 ROUGH END MILL
OPR 2 FINISH BORE
OPR 3 CHAMFER
-------------------------
PROCESS TO MACHINE Hole_CsunkThru :
OPR 1 CENTRE DRILL
OPR 2 DRILL
OPR 3 CHAMFER
OPR 4 FINISH BORE
-------------------------
Total 1 no of features need to be machined in setup 15
DATUM-> RIGHT
-------------------------
PROCESS TO MACHINE Hole_Through :
OPR 1 CENTRE DRILL
OPR 2 DRILL
-------------------------
```

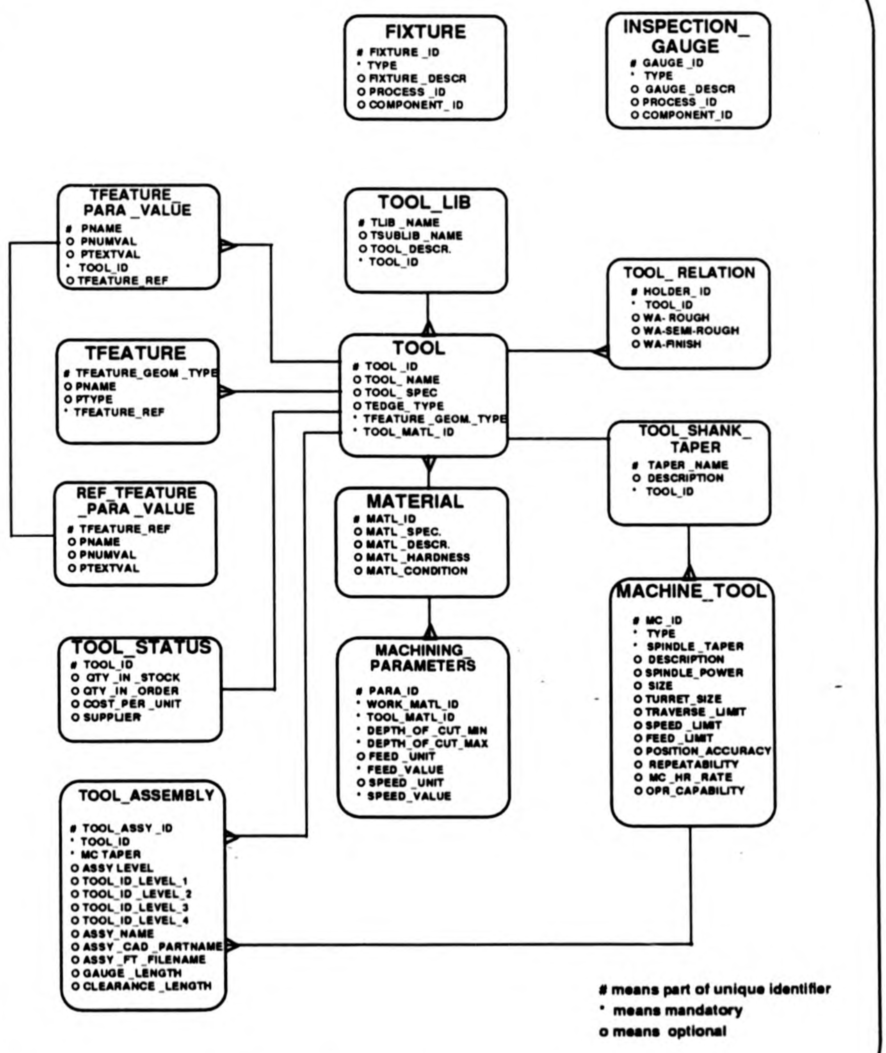
```
Total 1 no of features need to be machined in setup 11
DATUM-> BOTTOM
-------------------------
PROCESS TO MACHINE Counterbore :
OPR 1 COUNTERBORE
-------------------------
-------------------------------END-------------------------


------------------------------
WMG CAPP SYSTEM
------------------------------

Main Menu:
Type 'E' to EnterKnowledgebase

Type 'S' to SaveKnowledgebase

Type 'L' to LoadKnowledgebase

Type 'G' to Generate Process Plan

Type 'X' to Exit
choose one: choose one: E

Enter Options -->

Type 'P' to Enter Process Knowledge

Type 'T' to Enter Tooling Knowledge

Type 'M' to Enter Machine Tool Knowledge

Type 'X' to Return to Main Menu
choose one: P

Feature Name: Hole_Blind
Select Raw Material Stock Type-->

Type 'A' for INTERNAL Solid Volume
Type 'B' for INTERNAL Cast or forged or pre-machined withmax 3 mm
Type 'C' for INTERNAL Non Uniform or Merging with other features
Type 'D' for EXTERNAL Cast or forged or pre-machined withmax 3 mm
Type 'E' for EXTERNAL Non Uniform or Merging with other features
choose one: A
```

```
Select Feature Size Range-->

FOR ROUND FEATURES:

Type 'A' for DIAMETER RANGE [0 - 10] mm
Type 'B' for DIAMETER RANGE [10 - 25] mm
Type 'C' for DIAMETER RANGE [25 - 50] mm
Type 'D' for DIAMETER RANGE [50 - 100] mm
Type 'E' for DIAMETER RANGE [100 - 250] mm
Type 'F' for DIAMETER RANGE [250 - 500] mm
Type 'G' for DIAMETER RANGE [500 & MORE] mm

FOR PRISMATIC FEATURES:

Type 'K' for LENGTH x WIDTH RANGE [10 x 10] mm
Type 'L' for LENGTH x WIDTH RANGE [25 x 25] mm
Type 'M' for LENGTH x WIDTH RANGE [50 x 50] mm
Type 'N' for LENGTH x WIDTH RANGE [100 x 100] mm
Type 'P' for LENGTH x WIDTH RANGE [250 x 250] mm
Type 'Q' for LENGTH x WIDTH RANGE [500 x 500] mm
Type 'R' for LENGTH x WIDTH RANGE [500 & MORE] mm

choose one:  C

Select Feature Depth Range-->

Type 'A' for Depth Range [0 - 5] mm
Type 'B' for Depth Range [5 - 10] mm
Type 'C' for Depth Range [10 - 25] mm
Type 'D' for Depth Range [25 - 50] mm
Type 'E' for Depth Range [50 - 100] mm
Type 'F' for Depth Range [100- 150] mm
Type 'G' for Depth Range [150 - 200] mm
Type 'H' for Depth Range [200 &MORE] mm

choose one: D

Select achievable size tolerance range:

Type 'A' for [+_0.001 to +- 0.01 mm]
Type 'B' for [+_0.011 to +- 0.05 mm]
Type 'C' for [+_0.051 to +- 0.1 mm]
Type 'D' for [+_0.11 and above]
choose one: B

Select achievable surface finish range:
```

```
Type 'A' for [0.025 to  0.1m_mm (Grinding)]
Type 'B' for [0.15 to 0.63m_mm (Boring)]
Type 'C' for [1.0 to 2.5m_mm (Reaming, Milling)]
Type 'D' for [2.5 to 6.35m_mm (Drilling, Rough Milling)]
Type 'E' for [Above 6.35m_mm]

choose one: B

Select Process --->>

Type 'A' for CENTRE DRILL
Type 'B' for DRILL
Type 'C' for DEEP HOLE DRILL

Type 'D' for REAM
Type 'E' for ROUGH BORE
Type 'F' for FINISH BORE
Type 'G' for COUNTERBORE
Type 'H' for TAP

Type 'I' for CHAMFER

Type 'J' for ROUGH END MILL
Type 'K' for FINISH END MILL
Type 'L' for ROUGH FACE MILL
Type 'M' for FINISH FACE MILL

Type 'N' for INTERNAL GRINDING
Type 'P' for EXTERNAL GRINDING
Type 'Q' for SURFACE GRINDING

Type 'O' for OTHER TYPE

Type 'X' for Exit

operation no->1 (choose one): A

Select Process --->>

Type 'A' for CENTRE DRILL
Type 'B' for DRILL
Type 'C' for DEEP HOLE DRILL

Type 'D' for REAM
Type 'E' for ROUGH BORE
Type 'F' for FINISH BORE
Type 'G' for COUNTERBORE
Type 'H' for TAP

Type 'I' for CHAMFER

Type 'J' for ROUGH END MILL
Type 'K' for FINISH END MILL
Type 'L' for ROUGH FACE MILL
Type 'M' for FINISH FACE MILL

Type 'N' for INTERNAL GRINDING
Type 'P' for EXTERNAL GRINDING
Type 'Q' for SURFACE GRINDING

Type 'O' for OTHER TYPE

Type 'X' for Exit

operation no->2 (choose one): B

Select Process --->>

Type 'A' for CENTRE DRILL
Type 'B' for DRILL
Type 'C' for DEEP HOLE DRILL

Type 'D' for REAM
Type 'E' for ROUGH BORE
Type 'F' for FINISH BORE
Type 'G' for COUNTERBORE
Type 'H' for TAP

Type 'I' for CHAMFER

Type 'J' for ROUGH END MILL
Type 'K' for FINISH END MILL
Type 'L' for ROUGH FACE MILL
Type 'M' for FINISH FACE MILL

Type 'N' for INTERNAL GRINDING
Type 'P' for EXTERNAL GRINDING
Type 'Q' for SURFACE GRINDING

Type 'O' for OTHER TYPE
Type 'X' for Exit

operation no->3 (choose one): E

Select Process --->>

Type 'A' for CENTRE DRILL
Type 'B' for DRILL
Type 'C' for DEEP HOLE DRILL

Type 'D' for REAM
Type 'E' for ROUGH BORE
Type 'F' for FINISH BORE
Type 'G' for COUNTERBORE
Type 'H' for TAP

Type 'I' for CHAMFER

Type 'J' for ROUGH END MILL
Type 'K' for FINISH END MILL
Type 'L' for ROUGH FACE MILL
Type 'M' for FINISH FACE MILL

Type 'N' for INTERNAL GRINDING
Type 'P' for EXTERNAL GRINDING
Type 'Q' for SURFACE GRINDING

Type 'O' for OTHER TYPE

Type 'X' for Exit

operation no->4 (choose one): F

Select Process --->>

Type 'A' for CENTRE DRILL
Type 'B' for DRILL
Type 'C' for DEEP HOLE DRILL

Type 'D' for REAM
Type 'E' for ROUGH BORE
Type 'F' for FINISH BORE
Type 'G' for COUNTERBORE
Type 'H' for TAP

Type 'I' for CHAMFER
Type 'J' for ROUGH END MILL
Type 'K' for FINISH END MILL
Type 'L' for ROUGH FACE MILL
Type 'M' for FINISH FACE MILL

Type 'N' for INTERNAL GRINDING
Type 'P' for EXTERNAL GRINDING
Type 'Q' for SURFACE GRINDING

Type 'O' for OTHER TYPE

Type 'X' for Exit

operation no->5 (choose one): X

Enter Options -->

Type 'P' to Enter Process Knowledge

Type 'T' to Enter Tooling Knowledge

Type 'M' to Enter Machine Tool Knowledge

Type 'X' to Return to Main Menu
choose one: X

------------------------------
WMG CAPP SYSTEM
------------------------------

Main Menu:
Type 'E' to EnterKnowledgebase

Type 'S' to SaveKnowledgebase

Type 'L' to LoadKnowledgebase

Type 'G' to Generate Process Plan

Type 'X' to Exit
choose one: S

Select saving option -->

Type 'A' for saving process knowledge
Type 'B' for saving tooling knowledge
Type 'C' for saving machine tool knowledge

choose one: A

saving process knowledge

------------------------------
WMG CAPP SYSTEM
------------------------------

Main Menu:
Type 'E' to EnterKnowledgebase

Type 'S' to SaveKnowledgebase

Type 'L' to LoadKnowledgebase

Type 'G' to Generate Process Plan

Type 'X' to Exit
choose one: X
```

APPENDIX E

This appendix illustrates the Oracle Revision 7.0 Resource database design and development. The normalised resource entity model, as shown on the next page has been used to create the Oracle tables. A tool query program has been developed to query the resource database. The subsequent illustrations show the user interaction and the output of a program that generates the possible tool assemblies and the CADDS5 tool feature report files for graphic visualisation of tool assemblies.



NORMALISED RESOURCE ENTITY MODEL

The user interaction to tool query program

```
crocus > tool

Connected to ORACLE
Enter tool library name ?
drills
Enter tool sub_library name ?
twist_drill
Enter tool diameter ?
9.5
Enter required  tool height ?
75
Enter Machine Tool Spindle Taper ?
iso40
Enter Work material to be machined ?
ms
Level 3 assy generated
assembly added
part name is ->drills.twist_drill.dia9.5.1750
command file name is ->drills.twist_drill.dia9.5.1750.cmd
gauge_lng-> 220.100000
clearance_lng-> 62.000000
crocus > exit

script done on Wed Mar 22 19:15:31 1995
```

Cadds5 Command File For Tool Assembly of 9.5 Twist Drill

command file name is ->drills.twist_drill.dia9.5.1750.cmd

Select Library TOOLS

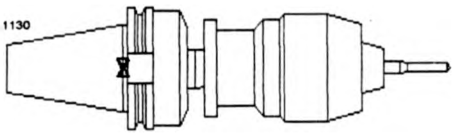
insert feature iso_shank b10 16.1 dia1 17 dia2 7 D1 63.55 D2 44.45 D4 56.25 D5 72.35 g1 16 h1 22.8 h2 25 l1 68.4 l3 8.2 l4 32 l5 3.2 l8 11.1 l9 19.1 V1 30 h3 18.5 Origin Loc [0.000000,0.000000,0.000000] Go

insert feature drill_chuck_holder h_length 60 B 19 l1 30 D 60 l3 15 taper 5 A 60 D1 20 Origin Loc [19.100000,0.000000,0.000000] Go

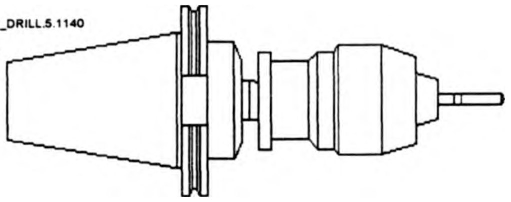
insert feature drill_chuck chk_length 109 B 19 l1 30 D 56 D1 50 l3 7.5 D2 40 l4 45 l5 75 l6 96 taper1 15 taper2 15 dia 20 taper 5 Origin Loc [49.100000,0.000000,0.000000] Go

insert feature twist_drill_ps cdia 9.5 csdia 10 tip_angle 118 clng 41 ctlength 91 Origin Loc [129.100000,0.000000,0.000000] Go

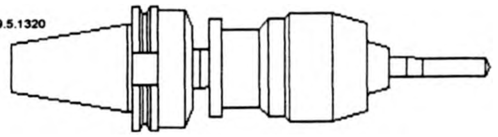
ASSY NAME : DRILLS.TWIST_DRILL.5.1130
ASSY COMPONENTS :
* iso40 drill chuck holder 3 -16
* 3 -16 drill chuck
* dia 5 twist drill



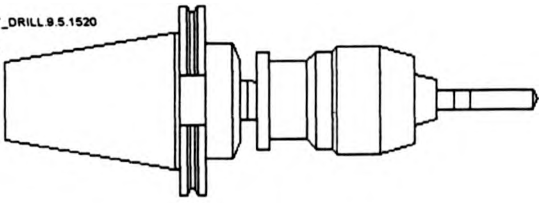
ASSY NAME : DRILLS.TWIST_DRILL.5.1140
ASSY COMPONENTS :
* iso50 drill chuck holder 3 -16
* 3 -16 drill chuck
* dia 5 twist drill



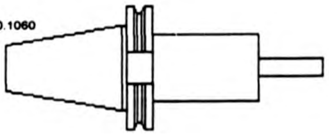
ASSY NAME : DRILLS.TWIST_DRILL.9.5.1320
ASSY COMPONENTS :
* iso40 drill chuck holder 3 -16
* 3 -16 drill chuck
* dia 9.5 twist drill



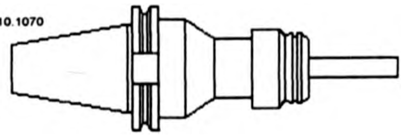
ASSY NAME : DRILLS.TWIST_DRILL.9.5.1520
ASSY COMPONENTS :
* iso50 drill chuck holder 3 -16
* 3 -16 drill chuck
* dia 9.5 twist drill



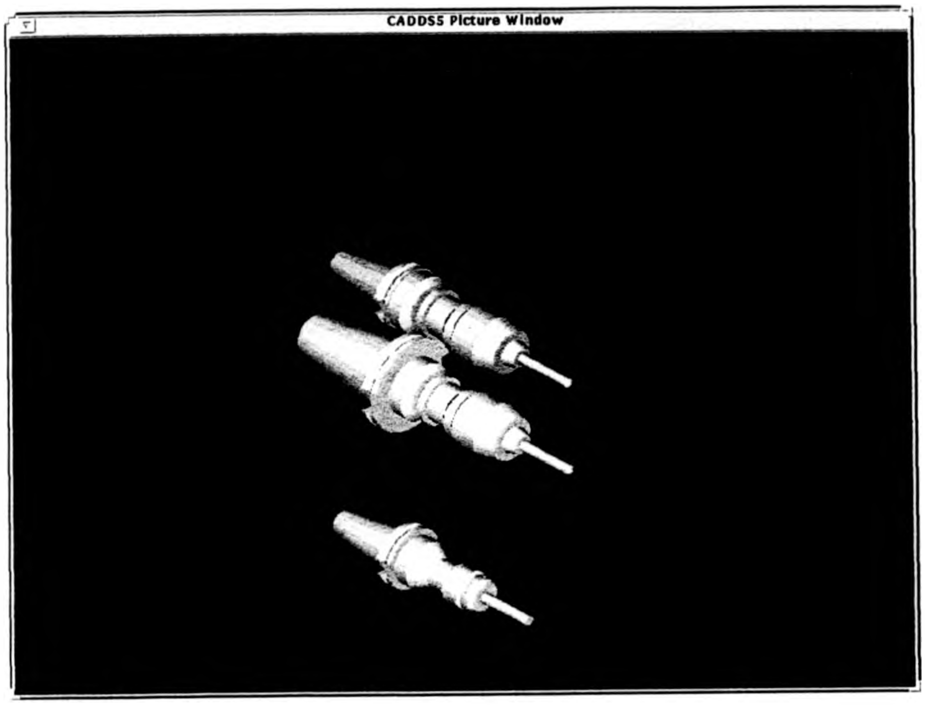
ASSY NAME : MILLS.END_MILL.DIA_10.1060
ASSY COMPONENTS :
* iso40 endmill holder
* dia 10 coronite endmill



ASSY NAME : MILLS.END_MILL.DIA_10.1070
ASSY COMPONENTS :
* iso40 collet chuck
* dia 10 double slotted collet
* dia 10 coronite endmill



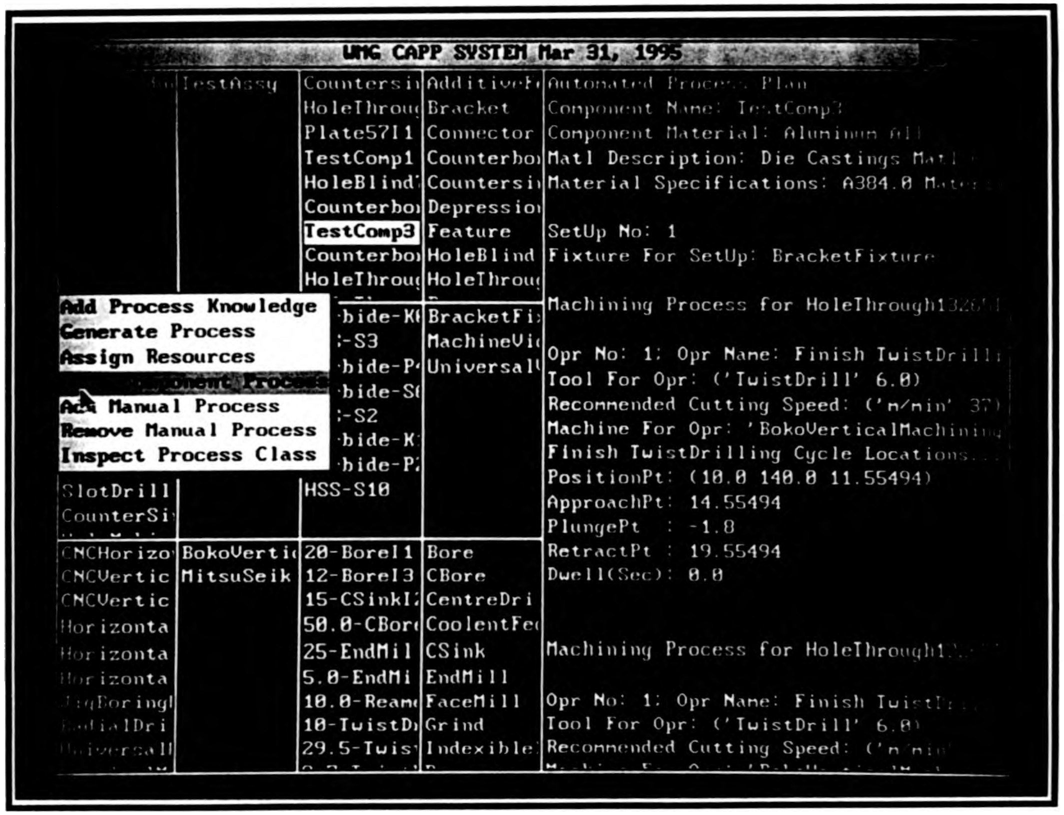
**TOOL ASSEMBLIES
GENERATED FROM RESOURCE DATABASE**



ISO VIEW OF
TOOL ASSEMBLIES
GENERATED FROM THE RESOURCE DATABASE

APPENDIX F

WMG CAPP SYSTEM TEST RESULTS

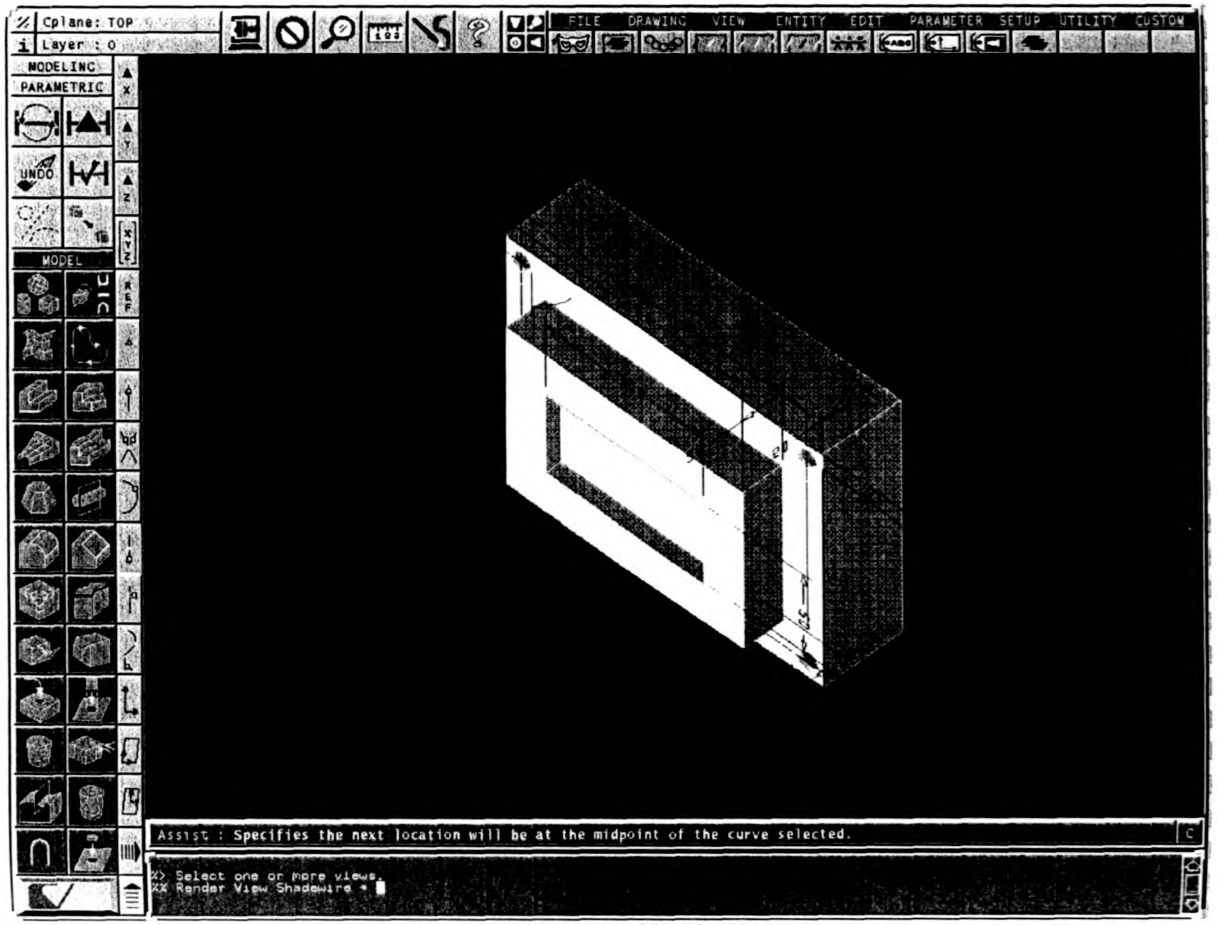


```
UWC CAPP SYSTEM Mar 31, 1995

   TestAssy   Countersi  AdditiveF   Automated Process Plan
              HoleThrou  Bracket     Component Name: TestComp3
              Plate5711  Connector   Component Material: Aluminum All
              TestComp1  Counterbo   Matl Description: Die Castings Matl
              HoleBlind  Countersi   Material Specifications: A384.0 Mate
              Counterbo  Depressio
              TestComp3  Feature     SetUp No: 1
              Counterbo  HoleBlind   Fixture For SetUp: BracketFixture
              HoleThrou  HoleThrou
                                     Machining Process for HoleThrough1326
              bide-K     BracketFi
              -S3        MachineVi   Opr No: 1: Opr Name: Finish TwistDrill
              bide-P     Universal   Tool For Opr: ('TwistDrill' 6.0)
              bide-S                 Recommended Cutting Speed: ('m/min' 37
              -S2                    Machine For Opr: 'BokoVerticalMachining
              bide-K                 Finish TwistDrilling Cycle Locations...
              bide-P                 PositionPt: (10.0 140.0 11.55494)
SlotDrill     HSS-S10                ApproachPt: 14.55494
CounterSi                            PlungePt  : -1.0
CNCHorizo  BokoVerti  20-Bore11  Bore        RetractPt : 19.55494
CNCVertic  MitsuSeik  12-Bore13  CBore       Dwell(Sec): 0.0
CNCVertic             15-CSink1  CentreDri
Horizonta             50.0-CBor  CoolentFe
Horizonta             25-EndMil  CSink       Machining Process for HoleThrough1
Horizonta             5.0-EndMi  EndMill
JigBoring             10.0-Ream  FaceMill    Opr No: 1: Opr Name: Finish Twist
RadialDri             10-TwistD  Grind       Tool For Opr: ('TwistDrill' 6.0)
Universal             29.5-Twis  Indexible   Recommended Cutting Speed: ('m/min'
```

Component Name: TestComp1

Old drawing A activated.

CADDS5 Feature Based Model Of TestComp1



Assist : Specifies the next location will be at the midpoint of the curve selected.

```
%> Select one or more views.
%% Render View Shadewire *
```

Rendered Image of TestComp1

CADDS5 Report File For TestComp1

```
*****     printing 8773 Countersink8770_I1     *****

class :
    class : Countersink                       library : CVOBJECTS
static attributes :
    BooleanOp  Operation                             - Subtract
attributes :
    Workpiece  Workpiece                             - 12056
    Name       Representation                        - rep
    CPlane     Orientation                        id - 6
                                                name - TOP
                                                 i x - 1.000000e+00
                                                   y - 0.000000e+00
                                                   z - 0.000000e+00
                                                 j x - 0.000000e+00
                                                   y - 1.000000e+00
                                                   z - 0.000000e+00
                                                 k x - 0.000000e+00
                                                   y - 0.000000e+00
                                                   z - 1.000000e+00
    Location   Origin                              x - 1.900000e+02
                                                   y - 1.000000e+01
                                                   z - 5.000000e+01
    ApplyType  Application                           - Boolean
    String     rawMatlCondition                      - Solid
    Real       surfFinish                            - 2.000000e+00
    String     mcNotmc                               - Y
    Face       EntryFace                             - 12004
    String     childOf                               - Plate52_I1
    Diameter   HoleDiameter                          - 1.000000e+01 - millimeters
    Length     CsinkDepth                            - 1.500000e+00 - millimeters
    Angle      CsinkAngle                            - 9.000000e+01 - degrees
    String     TolHoleDiameter                       - 0.1 / - 0.1
    String     TolCsinkAngle                         - 0.1 / - 0.1
    String     TolCsinkDepth                         - 0.1 / - 0.1

*****     printing 10125 Countersink8770_I5     *****

class :
    class : Countersink                       library : CVOBJECTS
static attributes :
    BooleanOp  Operation                             - Subtract
attributes :
    Workpiece  Workpiece                             - 13421
    Name       Representation                        - rep
    CPlane     Orientation                        id - 11
                                                name - BOTTOM
                                                 i x - 1.000000e+00
                                                   y - 0.000000e+00
                                                   z - 0.000000e+00
                                                 j x - 0.000000e+00
                                                   y - -1.000000e+00
                                                   z - 0.000000e+00
                                                 k x - 0.000000e+00
                                                   y - 0.000000e+00
                                                   z - -1.000000e+00
    Location   Origin                              x - 1.000000e+01
                                                   y - 1.400000e+02
                                                   z - 0.000000e+00
    ApplyType  Application                           - Boolean
    String     rawMatlCondition                      - Solid
    Real       surfFinish                            - 2.000000e+00
    String     mcNotmc                               - Y
    Face       EntryFace                             - 13211
    String     childOf                               - Plate52_I1
    Diameter   HoleDiameter                              = 1.000000e+01 =  millimeters
    Length     CsinkDepth                                = 1.500000e+00 =  millimeters
    Angle      CsinkAngle                                = 9.000000e+01 = degrees
    String     TolHoleDiameter                           = 0.1 / - 0.1
    String     TolCsinkAngle                             = 0.1 / - 0.1
    String     TolCsinkDepth                             = 0.1 / - 0.1

*****     printing 12315 Countersink8770_I9     *****

class :
    class : Countersink                        library : CVOBJECTS
static attributes :
    BooleanOp  Operation                                 = Subtract
attributes :
    Workpiece  Workpiece                                 = 12056
    Name       Representation                            = rep
    CPlane     Orientation                            id = 6
                                                    name = TOP
                                                     i x = 1.000000e+00
                                                       y = 0.000000e+00
                                                       z = 0.000000e+00
                                                     j x = 0.000000e+00
                                                       y = 1.000000e+00
                                                       z = 0.000000e+00
                                                     k x = 0.000000e+00
                                                       y = 0.000000e+00
                                                       z = 1.000000e+00
    Location   Origin                                  x = 1.900000e+02
                                                       y = 1.400000e+02
                                                       z = 5.000000e+01
    ApplyType  Application                               = Boolean
    String     rawMatlCondition                          = Solid
    Real       surfFinish                                = 2.000000e+00
    String     mcNotmc                                   = Y
    Face       EntryFace                                 = 12004
    String     childOf                                   = Plate52_I1
    Diameter   HoleDiameter                              = 1.000000e+01 =  millimeters
    Length     CsinkDepth                                = 1.500000e+00 =  millimeters
    Angle      CsinkAngle                                = 9.000000e+01 = degrees
    String     TolHoleDiameter                           = 0.1 / - 0.1
    String     TolCsinkAngle                             = 0.1 / - 0.1
    String     TolCsinkDepth                             = 0.1 / - 0.1

*****     printing 12344 Countersink8770_I10     *****

class :
    class : Countersink                        library : CVOBJECTS
static attributes :
    BooleanOp  Operation                                 = Subtract
attributes :
    Workpiece  Workpiece                                 = 12056
    Name       Representation                            = rep
    CPlane     Orientation                            id = 6
                                                    name = TOP
                                                     i x = 1.000000e+00
                                                       y = 0.000000e+00
                                                       z = 0.000000e+00
                                                     j x = 0.000000e+00
                                                       y = 1.000000e+00
                                                       z = 0.000000e+00
                                                     k x = 0.000000e+00
                                                       y = 0.000000e+00
                                                       z = 1.000000e+00
    Location   Origin                                  x = 1.000000e+01
                                                       y = 1.400000e+02
                                                       z = 5.000000e+01
    ApplyType  Application                               = Boolean
    String     rawMatlCondition                          = Solid
    Real       surfFinish                            = 2.000000e+00
    String     mcNotmc                               = Y
    Face       EntryFace                             = 12004
    String     childOf                               = Plate52_I1
    Diameter   HoleDiameter                          = 1.000000e+01 = millimeters
    Length     CsinkDepth                            = 1.500000e+00 = millimeters
    Angle      CsinkAngle                            = 9.000000e+01 = degrees
    String     TolHoleDiameter                       = 0.1 / - 0.1
    String     TolCsinkAngle                         = 0.1 / - 0.1
    String     TolCsinkDepth                         = 0.1 / - 0.1

*****     printing 12373 Countersink8770_I11     *****

class :
    class : Countersink                      library : CVOBJECTS
static attributes :
    BooleanOp  Operation                             = Subtract
attributes :
    Workpiece  Workpiece                             = 12056
    Name       Representation                        = rep
    CPlane     Orientation                        id = 6
                                                name = TOP
                                                 i x = 1.000000e+00
                                                   y = 0.000000e+00
                                                   z = 0.000000e+00
                                                 j x = 0.000000e+00
                                                   y = 1.000000e+00
                                                   z = 0.000000e+00
                                                 k x = 0.000000e+00
                                                   y = 0.000000e+00
                                                   z = 1.000000e+00
    Location   Origin                              x = 1.000000e+01
                                                   y = 1.000000e+01
                                                   z = 5.000000e+01
    ApplyType  Application                           = Boolean
    String     rawMatlCondition                      = Solid
    Real       surfFinish                            = 2.000000e+00
    String     mcNotmc                               = Y
    Face       EntryFace                             = 12004
    String     childOf                               = Plate52_I1
    Diameter   HoleDiameter                          = 1.000000e+01 = millimeters
    Length     CsinkDepth                            = 1.500000e+00 = millimeters
    Angle      CsinkAngle                            = 9.000000e+01 = degrees
    String     TolHoleDiameter                       = 0.1 / - 0.1
    String     TolCsinkAngle                         = 0.1 / - 0.1
    String     TolCsinkDepth                         = 0.1 / - 0.1

*****     printing 13549 Countersink8770_I12     *****

class :
    class : Countersink                      library : CVOBJECTS
static attributes :
    BooleanOp  Operation                             = Subtract
attributes :
    Workpiece  Workpiece                             = 13421
    Name       Representation                        = rep
    CPlane     Orientation                        id = 11
                                                name = BOTTOM
                                                 i x = 1.000000e+00
                                                   y = 0.000000e+00
                                                   z = 0.000000e+00
                                                 j x = 0.000000e+00
                                                   y = -1.000000e+00
                                                   z = 0.000000e+00
                                                 k x = 0.000000e+00
                                                   y = 0.000000e+00
                                                   z = -1.000000e+00
    Location   Origin                              x = 1.900000e+02
```

```
                                                           y = 1.400000e+02
                                                           z = 0.000000e+00
    ApplyType  Application                                   = Boolean
    String     rawMatlCondition                              = Solid
    Real       surfFinish                                    = 2.000000e+00
    String     mcNotmc                                       = Y
    Face       EntryFace                                     = 13211
    String     childOf                                       = Plate52_I1
    Diameter   HoleDiameter                                  = 1.000000e+01 = millimeters
    Length     CsinkDepth                                    = 1.500000e+00 = millimeters
    Angle      CsinkAngle                                    = 9.000000e+01 = degrees
    String     TolHoleDiameter                               = 0.1 / - 0.1
    String     TolCsinkAngle                                 = 0.1 / - 0.1
    String     TolCsinkDepth                                 = 0.1 / - 0.1

*****     printing 13578 Countersink8770_I13     *****

class :
    class : Countersink                        library : CVOBJECTS
static attributes :
    BooleanOp  Operation                                     = Subtract
attributes :
    Workpiece  Workpiece                                     = 13421
    Name       Representation                                = rep
    CPlane     Orientation                                id = 11
                                                        name = BOTTOM
                                                         i x = 1.000000e+00
                                                           y = 0.000000e+00
                                                           z = 0.000000e+00
                                                         j x = 0.000000e+00
                                                           y = -1.000000e+00
                                                           z = 0.000000e+00
                                                         k x = 0.000000e+00
                                                           y = 0.000000e+00
                                                           z = -1.000000e+00
    Location   Origin                                      x = 1.900000e+02
                                                           y = 1.000000e+01
                                                           z = 0.000000e+00
    ApplyType  Application                                   = Boolean
    String     rawMatlCondition                              = Solid
    Real       surfFinish                                    = 2.000000e+00
    String     mcNotmc                                       = Y
    Face       EntryFace                                     = 13211
    String     childOf                                       = Plate52_I1
    Diameter   HoleDiameter                                  = 1.000000e+01 = millimeters
    Length     CsinkDepth                                    = 1.500000e+00 = millimeters
    Angle      CsinkAngle                                    = 9.000000e+01 = degrees
    String     TolHoleDiameter                               = 0.1 / - 0.1
    String     TolCsinkAngle                                 = 0.1 / - 0.1
    String     TolCsinkDepth                                 = 0.1 / - 0.1

*****     printing 13607 Countersink8770_I14     *****

class :
    class : Countersink                        library : CVOBJECTS
static attributes :
    BooleanOp  Operation                                     = Subtract
attributes :
    Workpiece  Workpiece                                     = 13421
    Name       Representation                                = rep
    CPlane     Orientation                                id = 11
                                                        name = BOTTOM
                                                         i x = 1.000000e+00
                                                           y = 0.000000e+00
                                                           z = 0.000000e+00
                                                         j x = 0.000000e+00
                                                           y = -1.000000e+00
                                                           z = 0.000000e+00
```

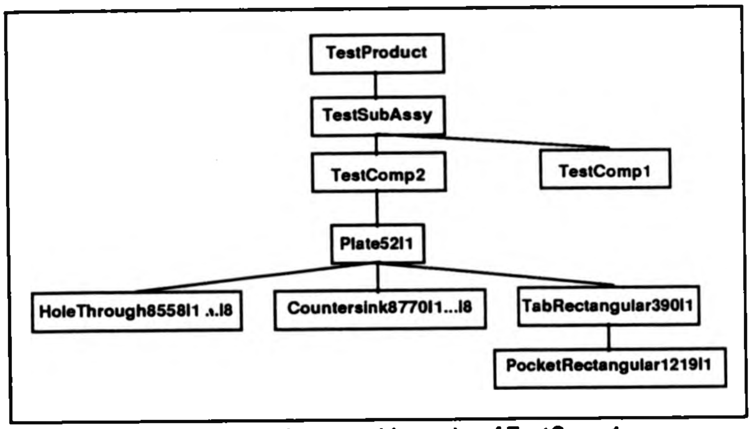
```
                                                          k x = 0.000000e+00
                                                            y = 0.000000e+00
                                                            z = -1.000000e+00
    Location   Origin                                       x = 1.000000e+01
                                                            y = 1.000000e+01
                                                            z = 0.000000e+00
    ApplyType  Application                                    = Boolean
    String     rawMatlCondition                               = Solid
    Real       surfFinish                                     = 2.000000e+00
    String     mcNotmc                                        = Y
    Face       EntryFace                                      = 13211
    String     childOf                                        = Plate52_I1
    Diameter   HoleDiameter                                   = 1.000000e+01 = millimeters
    Length     CsinkDepth                                     = 1.500000e+00 = millimeters
    Angle      CsinkAngle                                     = 9.000000e+01 = degrees
    String     TolHoleDiameter                                = 0.1 / - 0.1
    String     TolCsinkAngle                                  = 0.1 / - 0.1
    String     TolCsinkDepth                                  = 0.1 / - 0.1

*****     printing 8561 Hole_Through8558_I1     *****

class :
    class : Hole_Through                         library : CVOBJECTS
static attributes :
    BooleanOp  Operation                                      = Subtract
attributes :
    Workpiece  Workpiece                                      = 12056
    Name       Representation                                 = rep
    CPlane     Orientation                                 id = 6
                                                         name = TOP
                                                          i x = 1.000000e+00
                                                            y = 0.000000e+00
                                                            z = 0.000000e+00
                                                          j x = 0.000000e+00
                                                            y = 1.000000e+00
                                                            z = 0.000000e+00
                                                          k x = 0.000000e+00
                                                            y = 0.000000e+00
                                                            z = 1.000000e+00
    Location   Origin                                       x = 1.000000e+01
                                                            y = 1.400000e+02
                                                            z = 5.000000e+01
    ApplyType  Application                                    = Merge
    String     rawMatlCondition                               = Solid
    Real       surfFinish                                     = 2.000000e+00
    String     mcNotmc                                        = Y
    Face       EntryFace                                      = 12004
    Face       ExitFace                                       = 12046
    String     childOf                                        = Plate52_I1
    Diameter   HoleDiameter                                   = 1.000000e+01 = millimeters
    String     TolHoleDiameter                                = 0.1 / - 0.1

*****     printing 12113 Hole_Through8558_I5     *****

class :
    class : Hole_Through                         library : CVOBJECTS
static attributes :
    BooleanOp  Operation                                      = Subtract
attributes :
    Workpiece  Workpiece                                      = 12056
    Name       Representation                                 = rep
    CPlane     Orientation                                 id = 6
                                                         name = TOP
                                                          i x = 1.000000e+00
                                                            y = 0.000000e+00
                                                            z = 0.000000e+00
                                                          j x = 0.000000e+00
                                                            y = 1.000000e+00
```

```
                                                        z = 0.000000e+00
                                                      k x = 0.000000e+00
                                                        y = 0.000000e+00
                                                        z = 1.000000e+00
    Location   Origin                                   x = 1.900000e+02
                                                        y = 1.400000e+02
                                                        z = 5.000000e+01
    ApplyType  Application                                = Merge
    String     rawMatlCondition                           = Solid
    Real       surfFinish                                 = 2.000000e+00
    String     mcNotmc                                    = Y
    Face       EntryFace                                  = 12004
    Face       ExitFace                                   = 12046
    String     childOf                                    = Plate52_I1
    Diameter   HoleDiameter                               = 1.000000e+01 = millimeters
    String     TolHoleDiameter                            = 0.1 / - 0.1

*****     printing 12142 Hole_Through8558_I6       *****

class :
    class : Hole_Through                          library : CVOBJECTS
static attributes :
    BooleanOp  Operation                                  = Subtract
attributes :
    Workpiece  Workpiece                                  = 12056
    Name       Representation                             = rep
    CPlane     Orientation                               id = 6
                                                       name = TOP
                                                      i x = 1.000000e+00
                                                        y = 0.000000e+00
                                                        z = 0.000000e+00
                                                      j x = 0.000000e+00
                                                        y = 1.000000e+00
                                                        z = 0.000000e+00
                                                      k x = 0.000000e+00
                                                        y = 0.000000e+00
                                                        z = 1.000000e+00
    Location   Origin                                   x = 1.900000e+02
                                                        y = 1.000000e+01
                                                        z = 5.000000e+01
    ApplyType  Application                                = Merge
    String     rawMatlCondition                           = Solid
    Real       surfFinish                                 = 2.000000e+00
    String     mcNotmc                                    = Y
    Face       EntryFace                                  = 12004
    Face       ExitFace                                   = 12046
    String     childOf                                    = Plate52_I1
    Diameter   HoleDiameter                               = 1.000000e+01 = millimeters
    String     TolHoleDiameter                            = 0.1 / - 0.1

*****     printing 12171 Hole_Through8558_I7       *****

class :
    class : Hole_Through                          library : CVOBJECTS
static attributes :
    BooleanOp  Operation                                  = Subtract
attributes :
    Workpiece  Workpiece                                  = 12056
    Name       Representation                             = rep
    CPlane     Orientation                               id = 6
                                                       name = TOP
                                                      i x = 1.000000e+00
                                                        y = 0.000000e+00
                                                        z = 0.000000e+00
                                                      j x = 0.000000e+00
                                                        y = 1.000000e+00
                                                        z = 0.000000e+00
                                                      k x = 0.000000e+00
```

```
                                                                     y = 0.000000e+00
                                                                     z = 1.000000e+00
    Location   Origin                                                x = 1.000000e+01
                                                                     y = 1.000000e+01
                                                                     z = 5.000000e+01
    ApplyType  Application                                             = Merge
    String     rawMatlCondition                                        = Solid
    Real       surfFinish                                              = 2.000000e+00
    String     mcNotmc                                                 = Y
    Face       EntryFace                                               = 12004
    Face       ExitFace                                                = 12046
    String     childOf                                                 = Plate52_I1
    Diameter   HoleDiameter                                            = 1.000000e+01 - millimeters
    String     TolHoleDiameter                                         = 0.1 / - 0.1

*****     printing 1222 Pocket_Rectangular1219_I1     *****

class :
    class : Pocket_Rectangular                 library : CVOBJECTS
static attributes :
    BooleanOp  Operation                                               = Subtract
attributes :
    Workpiece  Workpiece                                               = 11880
    Name       Representation                                          = rep
    CPlane     Orientation                                          id = 6
                                                                  name = TOP
                                                                   i x = 1.000000e+00
                                                                     y = 0.000000e+00
                                                                     z = 0.000000e+00
                                                                   j x = 0.000000e+00
                                                                     y = 1.000000e+00
                                                                     z = 0.000000e+00
                                                                   k x = 0.000000e+00
                                                                     y = 0.000000e+00
                                                                     z = 1.000000e+00
    Location   Origin                                                x = 5.000000e+01
                                                                     y = 5.000000e+01
                                                                     z = 7.500000e+01
    ApplyType  Application                                             = Boolean
    String     rawMatlCondition                                        = Solid
    Real       surfFinish                                              = 2.000000e+00
    String     mcNotmc                                                 = Y
    Face       EntryFace                                               = 11836
    String     childOf                                                 = Tab_Rectangular390_I1
    Length     PocketLength                                            = 1.000000e+02 - millimeters
    Length     PocketWidth                                             = 5.000000e+01 - millimeters
    Length     PocketDepth                                             = 1.000000e+01 - millimeters
    String     TolPocketLength                                         = 0.1 / - 0.1
    String     TolPocketWidth                                          = 0.1 / - 0.1
    String     TolPocketDepth                                          = 0.1 / - 0.1

*****     printing 393 Tab_Rectangular390_I1     *****

class :
    class : Tab_Rectangular                    library : CVOBJECTS
static attributes :
    BooleanOp  Operation                                               = Add
attributes :
    Workpiece  Workpiece                                               = 11719
    Name       Representation                                          = rep
    CPlane     Orientation                                          id = 6
                                                                  name = TOP
                                                                   i x = 1.000000e+00
                                                                     y = 0.000000e+00
                                                                     z = 0.000000e+00
                                                                   j x = 0.000000e+00
                                                                     y = 1.000000e+00
                                                                     z = 0.000000e+00
```

```
                                                       k x = 0.000000e+00
                                                         y = 0.000000e+00
                                                         z = 1.000000e+00
    Location   Origin                                    x = 2.500000e+01
                                                         y = 2.500000e+01
                                                         z = 5.000000e+01

    ApplyType  Application                                 = Boolean
    String     rawMatlCondition                            = Soild
    Real       surfFinish                                  = 2.000000e+00
    String     mcNotmc                                     = Y
    Face       ExitFace                                    = 11746
    String     childOf                                     = Plate52_I1
    Length     TabLength                                   = 1.500000e+02 = millimeters
    Length     TabWidth                                    = 1.000000e+02 = millimeters
    Length     TabHeight                                   = 2.500000e+01 = millimeters
    String     TolTabLength                                = 0.1 / - 0.1
    String     TolTabWidth                                 = 0.1 / - 0.1
    String     TolTabHeight                                = 0.1 / - 0.1

*****     printing 55 Plate52_I1     *****

class :
    class : Plate                               library : CVOBJECTS
static attributes :
    BooleanOp  Operation                                   = Add
    Workpiece  Workpiece                                   = 0
    ApplyType  Application                                 = NoApply
attributes :
    Name       Representation                              = rep
    CPlane     Orientation                              id = 6
                                                      name = TOP
                                                       i x = 1.000000e+00
                                                         y = 0.000000e+00
                                                         z = 0.000000e+00
                                                       j x = 0.000000e+00
                                                         y = 1.000000e+00
                                                         z = 0.000000e+00
                                                       k x = 0.000000e+00
                                                         y = 0.000000e+00
                                                         z = 1.000000e+00
    Location   Origin                                    x = 0.000000e+00
                                                         y = 0.000000e+00
                                                         z = 0.000000e+00

    String     rawMatlCondition                            = Soild
    Real       surfFinish                                  = 2.000000e+00
    String     mcNotmc                                     = Y
    Length     PlateLength                                 = 2.000000e+02 = millimeters
    String     TolPlateLength                              = 0.1 / - 0.1
    Length     PlateWidth                                  = 1.500000e+02 = millimeters
    String     TolPlateWidth                               = 0.1 / - 0.1
    Length     PlateHeight                                 = 5.000000e+01 = millimeters
    String     TolPlateHeight                              = 0.1 / - 0.1
```

Report File Representation in WMG CAPP System



Feature Instance hierarchy of TestComp1

The CAPP System Process Plan Generation for TestComp1

The CAPP Command: Generate Process

Message in the Text Display Window:

```
TabRectangular390I1 Profile finish by End Milling
TabRectangular390I1 rough profiling by End Milling
TabRectangular390I1 Top face finishing by End Milling
TabRectangular390I1 Top face roughing by End Milling
PocketRectangular1219I1 finish by End Milling
PocketRectangular1219I1Pocket Corner finishing byEndMilling
PocketRectangular1219I1 roughing by End Milling
PocketRectangular1219I1 Pocket corner hole drilling
Plate52I1 No Stock, Surface Grinding not required
Plate52I1 No Stock, Face Milling not required
Plate52I1 No Stock, Face EndMilling not required
Countersink8770I1 CounterSinking required
Countersink8770I9 CounterSinking required
Countersink8770I10 CounterSinking required
Countersink8770I11 CounterSinking required
HoleThrough8558I1 can be finished by Boring
HoleThrough8558I1 can be finished by Reaming
HoleThrough8558I1 cannot be finished byTwistDrilling
HoleThrough8558I1 Rough Boring is not required
HoleThrough8558I1 Drilling required
HoleThrough8558I1 CentreDrilling not required
HoleThrough8558I1 Rough Boring is not required
HoleThrough8558I1 Drilling required
HoleThrough8558I1 CentreDrilling not required
HoleThrough8558I5 can be finished by Boring
HoleThrough8558I5 can be finished by Reaming
HoleThrough8558I5 cannot be finished byTwistDrilling
HoleThrough8558I5 Rough Boring is not required
HoleThrough8558I5 Drilling required
HoleThrough8558I5 CentreDrilling not required
HoleThrough8558I5 Rough Boring is not required
HoleThrough8558I5 Drilling required
HoleThrough8558I5 CentreDrilling not required
HoleThrough8558I6 can be finished by Boring
HoleThrough8558I6 can be finished by Reaming
HoleThrough8558I6 cannot be finished byTwistDrilling
HoleThrough8558I6 Rough Boring is not required
HoleThrough8558I6 Drilling required
HoleThrough8558I6 CentreDrilling not required
HoleThrough8558I6 Rough Boring is not required
```

```
HoleThrough8558I6 Drilling required
HoleThrough8558I6 CentreDrilling not required
HoleThrough8558I7 can be finished by Boring
HoleThrough8558I7 can be finished by Reaming
HoleThrough8558I7 cannot be finished byTwistDrilling
HoleThrough8558I7 Rough Boring is not required
HoleThrough8558I7 Drilling required
HoleThrough8558I7 CentreDrilling not required
HoleThrough8558I7 Rough Boring is not required
HoleThrough8558I7 Drilling required
HoleThrough8558I7 CentreDrilling not required
Countersink8770I5 CounterSinking required
Countersink8770I12 CounterSinking required
Countersink8770I13 CounterSinking required
Countersink8770I14 CounterSinking required
```

The universal knowledge based Process Plan for TestComp1

The CAPP Command: **Show Component Process**

```
Automated Process Plan
Component Name: TestComp1
Component Material: Tool Steel, Wrought
Matl Description: Cold Work Matl Condition: Annealed
Material Specifications: A2 Material Hardness: (200 250)

SetUp No: 1
Fixture For SetUp: MachineVice

Machining Process for Countersink8770I1

Opr No: 1; Opr Name: FinishCounterSinking
Tool For Opr: ('CSink' 5.0 90.0 15.0 4.5)
Machine For Opr: ('BoringMc' DrillingMc' MillingMc' MachiningCentre')
FinishCounterSinking Cycle Locations...
PositionPt: (180.0 10.0 50.0)
ApproachPt: 53.0
PlungePt  : 48.5
RetractPt : 58.0
Dwell(Sec): 1.0


Machining Process for Countersink8770I9

Opr No: 1; Opr Name: FinishCounterSinking
Tool For Opr: ('CSink' 5.0 90.0 15.0 4.5)
Machine For Opr: ('BoringMc' DrillingMc' MillingMc' MachiningCentre')
FinishCounterSinking Cycle Locations...
PositionPt: (180.0 130.0 50.0)
ApproachPt: 53.0
PlungePt  : 48.5
RetractPt : 58.0
Dwell(Sec): 1.0


Machining Process for Countersink8770I10

Opr No: 1; Opr Name: FinishCounterSinking
Tool For Opr: ('CSink' 5.0 90.0 15.0 4.5)
Machine For Opr: ('BoringMc' DrillingMc' MillingMc' MachiningCentre')
FinishCounterSinking Cycle Locations...
PositionPt: (10.0 130.0 50.0)
```

```
ApproachPt: 53.0
PlungePt  : 48.5
RetractPt : 58.0
Dwell(Sec): 1.0


Machining Process for Countersink8770I11

Opr No: 1; Opr Name: Finish CounterSinking
Tool For Opr: ('CSink' 5.0 90.0 15.0 4.5)
Machine For Opr: ('BoringMc' DrillingMc' MillingMc' MachiningCentre')
Finish CounterSinking Cycle Locations...
PositionPt: (10.0 10.0 50.0)
ApproachPt: 53.0
PlungePt  : 48.5
RetractPt : 58.0
Dwell(Sec): 1.0


2 Options Exist to machine HoleThrough8558I1
Option no: 1

Opr No: 1; Opr Name: Rough TwistDrilling
Tool For Opr: ('TwistDrill' 9.25 9.5 9.75 56.0)
Machine For Opr: ('BoringMc' DrillingMc' MillingMc' MachiningCentre')
Rough TwistDrilling Cycle Locations...
PositionPt: (10.0 130.0 50.0)
ApproachPt: 53.0
PlungePt  : -3.0
RetractPt : 58.0
Dwell(Sec): 0.0


Opr No: 2; Opr Name: Finish Boring
Tool For Opr: ('Bore' 10.010.0 10.0 54.5)
Machine For Opr: ('BoringMc' DrillingMc' MillingMc' MachiningCentre')
Finish Boring Cycle Locations...
PositionPt: (10.0 130.0 50.0)
ApproachPt: 53.0
PlungePt  : -1.5
RetractPt : 58.0
Dwell(Sec): 0.0
```

```
Option no: 2

Opr No: 1; Opr Name: Rough TwistDrilling
Tool For Opr: ('TwistDrill' 9.45 9.7 9.95 56.0)
Machine For Opr: ('BoringMc' DrillingMc' MillingMc' MachiningCentre')
Rough TwistDrilling Cycle Locations...
PositionPt: (10.0 130.0 50.0)
ApproachPt: 53.0
PlungePt  : -3.0
RetractPt : 58.0
Dwell(Sec): 0.0


Opr No: 2; Opr Name: Finish Reaming
Tool For Opr: ('Reamer' 10.010.0 10.0 54.5)
Machine For Opr: ('BoringMc' DrillingMc' MillingMc' MachiningCentre')
Finish Reaming Cycle Locations...
PositionPt: (10.0 130.0 50.0)
ApproachPt: 53.0
PlungePt  : -1.5
RetractPt : 58.0
Dwell(Sec): 0.0


2 Options Exist to machine HoleThrough855815
Option no: 1

Opr No: 1; Opr Name: Rough TwistDrilling
Tool For Opr: ('TwistDrill' 9.25 9.5 9.75 56.0)
Machine For Opr: ('BoringMc' DrillingMc' MillingMc' MachiningCentre')
Rough TwistDrilling Cycle Locations...
PositionPt: (180.0 130.0 50.0)
ApproachPt: 53.0
PlungePt  : -3.0
RetractPt : 58.0
Dwell(Sec): 0.0


Opr No: 2; Opr Name: Finish Boring
Tool For Opr: ('Bore' 10.010.0 10.0 54.5)
Machine For Opr: ('BoringMc' DrillingMc' MillingMc' MachiningCentre')
Finish Boring Cycle Locations...
PositionPt: (180.0 130.0 50.0)
ApproachPt: 53.0
PlungePt  : -1.5
RetractPt : 58.0
```

```
Dwell(Sec): 0.0


Option no: 2

Opr No: 1; Opr Name: Rough TwistDrilling
Tool For Opr: ('TwistDrill' 9.45 9.7 9.95 56.0)
Machine For Opr: ('BoringMc' DrillingMc' MillingMc' MachiningCentre')
Rough TwistDrilling Cycle Locations...
PositionPt: (180.0 130.0 50.0)
ApproachPt: 53.0
PlungePt  : -3.0
RetractPt : 58.0
Dwell(Sec): 0.0


Opr No: 2; Opr Name: Finish Reaming
Tool For Opr: ('Reamer' 10.010.0 10.0 54.5)
Machine For Opr: ('BoringMc' DrillingMc' MillingMc' MachiningCentre')
Finish Reaming Cycle Locations...
PositionPt: (180.0 130.0 50.0)
ApproachPt: 53.0
PlungePt  : -1.5
RetractPt : 58.0
Dwell(Sec): 0.0


2 Options Exist to machine HoleThrough855816
Option no: 1

Opr No: 1; Opr Name: Rough TwistDrilling
Tool For Opr: ('TwistDrill' 9.25 9.5 9.75 56.0)
Machine For Opr: ('BoringMc' DrillingMc' MillingMc' MachiningCentre')
Rough TwistDrilling Cycle Locations...
PositionPt: (180.0 10.0 50.0)
ApproachPt: 53.0
PlungePt  : -3.0
RetractPt : 58.0
Dwell(Sec): 0.0


Opr No: 2; Opr Name: Finish Boring
Tool For Opr: ('Bore' 10.010.0 10.0 54.5)
Machine For Opr: ('BoringMc' DrillingMc' MillingMc' MachiningCentre')
Finish Boring Cycle Locations...
PositionPt: (180.0 10.0 50.0)
```

```
ApproachPt: 53.0
PlungePt  : -1.5
RetractPt : 58.0
Dwell(Sec): 0.0


Option no: 2

Opr No: 1; Opr Name: Rough TwistDrilling
Tool For Opr: ('TwistDrill' 9.45 9.7 9.95 56.0)
Machine For Opr: ('BoringMc' DrillingMc'MillingMc'MachiningCentre')
Rough TwistDrilling Cycle Locations...
PositionPt: (180.0 10.0 50.0)
ApproachPt: 53.0
PlungePt  : -3.0
RetractPt : 58.0
Dwell(Sec): 0.0


Opr No: 2; Opr Name: Finish Reaming
Tool For Opr: ('Reamer' 10.010.0 10.0 54.5)
Machine For Opr: ('BoringMc' DrillingMc'MillingMc'MachiningCentre')
Finish Reaming Cycle Locations...
PositionPt: (180.0 10.0 50.0)
ApproachPt: 53.0
PlungePt  : -1.5
RetractPt : 58.0
Dwell(Sec): 0.0


2 Options Exist to machine HoleThrough855817
Option no: 1

Opr No: 1; Opr Name: Rough TwistDrilling
Tool For Opr: ('TwistDrill' 9.25 9.5 9.75 56.0)
Machine For Opr: ('BoringMc' DrillingMc'MillingMc'MachiningCentre')
Rough TwistDrilling Cycle Locations...
PositionPt: (10.0 10.0 50.0)
ApproachPt: 53.0
PlungePt  : -3.0
RetractPt : 58.0
Dwell(Sec): 0.0


Opr No: 2; Opr Name: Finish Boring
Tool For Opr: ('Bore' 10.010.0 10.0 54.5)
```

```
Machine For Opr: ('BoringMc' DrillingMc' MillingMc' MachiningCentre')
Finish Boring Cycle Locations...
PositionPt: (10.0 10.0 50.0)
ApproachPt: 53.0
PlungePt  : -1.5
RetractPt : 58.0
Dwell(Sec): 0.0


Option no: 2

Opr No: 1; Opr Name: Rough TwistDrilling
Tool For Opr: ('TwistDrill' 9.45 9.7 9.95 56.0)
Machine For Opr: ('BoringMc' DrillingMc' MillingMc' MachiningCentre')
Rough TwistDrilling Cycle Locations...
PositionPt: (10.0 10.0 50.0)
ApproachPt: 53.0
PlungePt  : -3.0
RetractPt : 58.0
Dwell(Sec): 0.0


Opr No: 2; Opr Name: Finish Reaming
Tool For Opr: ('Reamer' 10.010.0 10.0 54.5)
Machine For Opr: ('BoringMc' DrillingMc' MillingMc' MachiningCentre')
Finish Reaming Cycle Locations...
PositionPt: (10.0 10.0 50.0)
ApproachPt: 53.0
PlungePt  : -1.5
RetractPt : 58.0
Dwell(Sec): 0.0


SetUp No: 1
Fixture For SetUp: MachineVice

Machining Process for TabRectangular390I1

Opr No: 1; Opr Name: Rough EndMilling
Tool For Opr: ('EndMill' 10 25 50 25.0)
Machine For Opr: ('MillingMc' MachiningCentre')

Opr No: 2; Opr Name: FinishEndMilling
Tool For Opr: ('EndMill' 10 25 50 25.0)
Machine For Opr: ('MillingMc' MachiningCentre')
```

```
Opr No: 3; Opr Name: Rough EndMilling
Tool For Opr: ('EndMill' 12.5 25.0 50.0 15.0)
Machine For Opr: ('MillingMc' 'MachiningCentre')

Opr No: 4; Opr Name: FinishEndMilling
Tool For Opr: ('EndMill' 12.5 25.0 50.0 15.0)
Machine For Opr: ('MillingMc' 'MachiningCentre')

Machining Process for PocketRectangular121911

Opr No: 1; Opr Name: Rough TwistDrilling
Tool For Opr: ('TwistDrill' 2.0 5.05.0 13.0)
Machine For Opr: ('BoringMc' DrillingMc' MillingMc' MachiningCentre')
Rough TwistDrilling Cycle Locations...
PositionPt: ((50.0 50.0 75.0) (150.0 50.0 75.0) (150.0 100.0 75.0) (50.0
               100.0 75.0))
ApproachPt: 78.0
PlungePt  : 65.0
RetractPt : 83.0
Dwell(Sec): 1.0


Opr No: 2; Opr Name: Rough EndMilling
Tool For Opr: ('EndMill' 6.25 12.5 49.8 20.0)
Machine For Opr: ('MillingMc' 'MachiningCentre')
Rough EndMilling Cycle Locations...
PositionPt   : (75.0 50.0 75.0)
ApproachPlane: 95.0
ZWorkPlane   : 65.4
RetractPlane : 95.0
ClearPlane   : 145.0


Opr No: 3; Opr Name: FinishEndMilling
Tool For Opr: ('EndMill' 6.25 8.3333333 49.8 20.0)
Machine For Opr: ('MillingMc' 'MachiningCentre')
Finish EndMilling Cycle Locations...
PositionPt   : (75.0 50.0 75.0)
ApproachPlane: 95.0
ZWorkPlane   : 65.0
RetractPlane : 95.0
ClearPlane   : 145.0


Opr No: 4; Opr Name: FinishEndMilling
Tool For Opr: ('EndMill' 2.0 5.05.0 20.0)
```

```
Machine For Opr: ('MillingMc' 'MachiningCentre')
Finish EndMilling Cycle Locations...
PositionPt   : (75.0 50.0 75.0)
ApproachPlane: 95.0
ZWorkPlane   : 65.0
RetractPlane : 95.0
ClearPlane   : 145.0
```

SetUp No: 6
Fixture For SetUp: MachineVice

```
Machining Process for Countersink8770I5

Opr No: 1; Opr Name: FinishCounterSinking
Tool For Opr: ('CSink' 5.0 90.0 15.0 4.5)
Machine For Opr: ('BoringMc' 'DrillingMc' 'MillingMc' 'MachiningCentre')
FinishCounterSinking Cycle Locations...
PositionPt: (10.0 130.0 0.0)
ApproachPt: 3.0
PlungePt  : -1.5
RetractPt : 8.0
Dwell(Sec): 1.0


Machining Process for Countersink8770I12

Opr No: 1; Opr Name: FinishCounterSinking
Tool For Opr: ('CSink' 5.0 90.0 15.0 4.5)
Machine For Opr: ('BoringMc' 'DrillingMc' 'MillingMc' 'MachiningCentre')
FinishCounterSinking Cycle Locations...
PositionPt: (180.0 130.0 0.0)
ApproachPt: 3.0
PlungePt  : -1.5
RetractPt : 8.0
Dwell(Sec): 1.0


Machining Process for Countersink8770I13

Opr No: 1; Opr Name: FinishCounterSinking
Tool For Opr: ('CSink' 5.0 90.0 15.0 4.5)
Machine For Opr: ('BoringMc' 'DrillingMc' 'MillingMc' 'MachiningCentre')
FinishCounterSinking Cycle Locations...
PositionPt: (180.0 10.0 0.0)
ApproachPt: 3.0
```

```
PlungePt  : -1.5
RetractPt : 8.0
Dwell(Sec): 1.0


Machining Process for Countersink8770I14

Opr No: 1; Opr Name: FinishCounterSinking
Tool For Opr: ('CSink' 5.0 90.0 15.0 4.5)
Machine For Opr: ('BoringMc' DrillingMc' MillingMc' MachiningCentre')
FinishCounterSinking Cycle Locations...
PositionPt: (10.0 10.0 0.0)
ApproachPt: 3.0
PlungePt  : -1.5
RetractPt : 8.0
Dwell(Sec): 1.0
```

The Shop Specific Resource Search for TestComp1

The CAPP Command: **Assign Resources**

Message in the Text Display Window:

```
('Bore' 10.010.0 10.0 54.5) Not available
('Reamer' 10.010.0 10.0 54.5) Not available
```

The Shop Specific knowledge based Process Plan for TestComp1

The CAPP Command: **Show Component Process**

Automated Process Plan
Component Name: TestComp1
Component Material: Tool Steel, Wrought
Matl Description: Cold Work **Matl Condition:** Annealed
Material Specifications: A2 **Material Hardness:** (200 250)

SetUp No: 1
Fixture For SetUp: MachineVice

Machining Process for Countersink8770I1

Opr No: 1; Opr Name: FinishCounterSinking
Tool For Opr: ('CSink' 15)
Recommended Cutting Speed: ('m/min' 20) Feed: ('mm/rev' 0.1)
Machine For Opr: 'BokoVerticalMachiningCentreI1'
FinishCounterSinking Cycle Locations...
PositionPt: (180.0 10.0 50.0)
ApproachPt: 53.0
PlungePt : 48.5
RetractPt : 58.0
Dwell(Sec): 1.0

Machining Process for Countersink8770I9

Opr No: 1; Opr Name: FinishCounterSinking
Tool For Opr: ('CSink' 15)
Recommended Cutting Speed: ('m/min' 20) Feed: ('mm/rev' 0.1)
Machine For Opr: 'BokoVerticalMachiningCentreI1'
FinishCounterSinking Cycle Locations...
PositionPt: (180.0 130.0 50.0)
ApproachPt: 53.0
PlungePt : 48.5
RetractPt : 58.0
Dwell(Sec): 1.0

Machining Process for Countersink8770I10

Opr No: 1; Opr Name: FinishCounterSinking
Tool For Opr: ('CSink' 15)
Recommended Cutting Speed: ('m/min' 20) Feed: ('mm/rev' 0.1)

```
Machine For Opr: 'BokoVerticalMachiningCentre11'
Finish CounterSinking Cycle Locations...
PositionPt: (10.0 130.0 50.0)
ApproachPt: 53.0
PlungePt  : 48.5
RetractPt : 58.0
Dwell(Sec): 1.0


Machining Process for Countersink8770I11

Opr No: 1; Opr Name: FinishCounterSinking
Tool For Opr: ('CSink' 15)
Recommended Cutting Speed: ('m/min' 20) Feed: ('mm/rev' 0.1)
Machine For Opr: 'BokoVerticalMachiningCentre11'
Finish CounterSinking Cycle Locations...
PositionPt: (10.0 10.0 50.0)
ApproachPt: 53.0
PlungePt  : 48.5
RetractPt : 58.0
Dwell(Sec): 1.0


2 Options Exist to machine HoleThrough8558I1
Option no: 2 is selected

Opr No: 1; Opr Name: Rough TwistDrilling
Tool For Opr: ('TwistDrill' 9.7)
Recommended Cutting Speed: ('m/min' 14) Feed: ('mm/rev' 0.075)
Machine For Opr: 'BokoVerticalMachiningCentre11'
Rough TwistDrilling Cycle Locations...
PositionPt: (10.0 130.0 50.0)
ApproachPt: 53.0
PlungePt  : -3.0
RetractPt : 58.0
Dwell(Sec): 0.0


Opr No: 2; Opr Name: Finish Reaming
Tool For Opr: ('Reamer' 10.0)
Recommended Cutting Speed: ('m/min' 14) Feed: ('mm/rev' 0.18)
Machine For Opr: 'BokoVerticalMachiningCentre11'
Finish Reaming Cycle Locations...
PositionPt: (10.0 130.0 50.0)
ApproachPt: 53.0
PlungePt  : -1.5
```

```
RetractPt : 58.0
Dwell(Sec): 0.0


2 Options Exist to machine HoleThrough8558I5
Option no: 2 is selected

Opr No: 1; Opr Name: Rough TwistDrilling
Tool For Opr: ('TwistDrill' 9.7)
Recommended Cutting Speed: ('m/min' 14) Feed: ('mm/rev' 0.075)
Machine For Opr: 'BokoVerticalMachiningCentre11'
Rough TwistDrilling Cycle Locations...
PositionPt: (180.0 130.0 50.0)
ApproachPt: 53.0
PlungePt  : -3.0
RetractPt : 58.0
Dwell(Sec): 0.0


Opr No: 2; Opr Name: Finish Reaming
Tool For Opr: ('Reamer' 10.0)
Recommended Cutting Speed: ('m/min' 14) Feed: ('mm/rev' 0.18)
Machine For Opr: 'BokoVerticalMachiningCentre11'
Finish Reaming Cycle Locations...
PositionPt: (180.0 130.0 50.0)
ApproachPt: 53.0
PlungePt  : -1.5
RetractPt : 58.0
Dwell(Sec): 0.0


2 Options Exist to machine HoleThrough8558I6
Option no: 2 is selected

Opr No: 1; Opr Name: Rough TwistDrilling
Tool For Opr: ('TwistDrill' 9.7)
Recommended Cutting Speed: ('m/min' 14) Feed: ('mm/rev' 0.075)
Machine For Opr: 'BokoVerticalMachiningCentre11'
Rough TwistDrilling Cycle Locations...
PositionPt: (180.0 10.0 50.0)
ApproachPt: 53.0
PlungePt  : -3.0
RetractPt : 58.0
Dwell(Sec): 0.0
```

```
Opr No: 2; Opr Name: Finish Reaming
Tool For Opr: ('Reamer' 10.0)
Recommended Cutting Speed: ('m/min' 14) Feed: ('mm/rev' 0.18)
Machine For Opr: 'BokoVerticalMachiningCentreI1'
Finish Reaming Cycle Locations...
PositionPt: (180.0 10.0 50.0)
ApproachPt: 53.0
PlungePt  : -1.5
RetractPt : 58.0
Dwell(Sec): 0.0


2 Options Exist to machine HoleThrough8558I7
Option no: 2 is selected

Opr No: 1; Opr Name: Rough TwistDrilling
Tool For Opr: ('TwistDrill' 9.7)
Recommended Cutting Speed: ('m/min' 14) Feed: ('mm/rev' 0.075)
Machine For Opr: 'BokoVerticalMachiningCentreI1'
Rough TwistDrilling Cycle Locations...
PositionPt: (10.0 10.0 50.0)
ApproachPt: 53.0
PlungePt  : -3.0
RetractPt : 58.0
Dwell(Sec): 0.0


Opr No: 2; Opr Name: Finish Reaming
Tool For Opr: ('Reamer' 10.0)
Recommended Cutting Speed: ('m/min' 14) Feed: ('mm/rev' 0.18)
Machine For Opr: 'BokoVerticalMachiningCentreI1'
Finish Reaming Cycle Locations...
PositionPt: (10.0 10.0 50.0)
ApproachPt: 53.0
PlungePt  : -1.5
RetractPt : 58.0
Dwell(Sec): 0.0

Machining Process for TabRectangular390I1

Opr No: 1; Opr Name: Rough EndMilling
Tool For Opr: ('EndMill' 25)
Recommended Cutting Speed: ('m/min' 72) Feed: ('mm/tooth' 0.075)
Machine For Opr: 'BokoVerticalMachiningCentreI1'

Opr No: 2; Opr Name: FinishEndMilling
```

```
Tool For Opr: ('EndMill' 25)
Recommended Cutting Speed: ('m/min' 120) Feed: ('mm/tooth' 0.1)
Machine For Opr: 'BokoVerticalMachiningCentre11'

Opr No: 3; Opr Name: Rough EndMilling
Tool For Opr: ('EndMill' 25)
Recommended Cutting Speed: ('m/min' 72) Feed: ('mm/tooth' 0.075)
Machine For Opr: 'BokoVerticalMachiningCentre11'

Opr No: 4; Opr Name: FinishEndMilling
Tool For Opr: ('EndMill' 25)
Recommended Cutting Speed: ('m/min' 120) Feed: ('mm/tooth' 0.1)
Machine For Opr: 'BokoVerticalMachiningCentre11'

Machining Process for PocketRectangular121911

Opr No: 1; Opr Name: Rough TwistDrilling
Tool For Opr: ('TwistDrill' 5.0)
Recommended Cutting Speed: ('m/min' 14) Feed: ('mm/rev' 0.075)
Machine For Opr: 'BokoVerticalMachiningCentre11'
Rough TwistDrilling Cycle Locations...
PositionPt: ((50.0 50.0 75.0) (150.0 50.0 75.0) (150.0 100.0 75.0) (50.0
        100.0 75.0))
ApproachPt: 78.0
PlungePt  : 65.0
RetractPt : 83.0
Dwell(Sec): 1.0


Opr No: 2; Opr Name: Rough EndMilling
Tool For Opr: ('EndMill' 25)
Recommended Cutting Speed: ('m/min' 72) Feed: ('mm/tooth' 0.075)
Machine For Opr: 'BokoVerticalMachiningCentre11'
Rough EndMilling Cycle Locations...
PositionPt   : (75.0 50.0 75.0)
ApproachPlane: 95.0
ZWorkPlane   : 65.4
RetractPlane : 95.0
ClearPlane   : 145.0


Opr No: 3; Opr Name: FinishEndMilling
Tool For Opr: ('EndMill' 25)
Recommended Cutting Speed: ('m/min' 120) Feed: ('mm/tooth' 0.1)
Machine For Opr: 'BokoVerticalMachiningCentre11'
FinishEndMilling Cycle Locations...
```

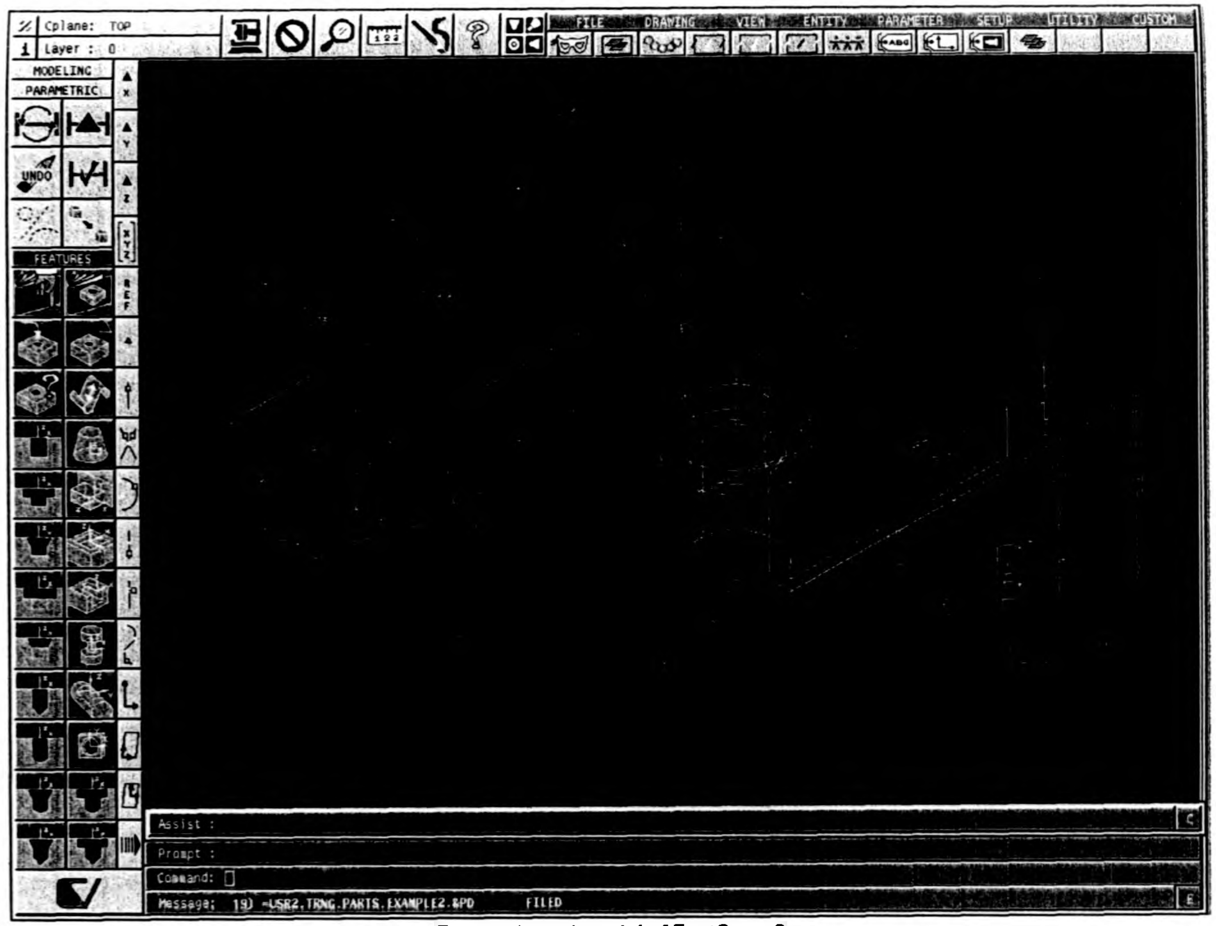
```
PositionPt   : (75.0 50.0 75.0)
ApproachPlane: 95.0
ZWorkPlane   : 65.0
RetractPlane : 95.0
ClearPlane   : 145.0


Opr No: 4; Opr Name: FinishEndMilling
Tool For Opr: ('EndMill' 5.0)
Recommended Cutting Speed: ('m/min' 50) Feed: ('mm/tooth' 0.08)
Machine For Opr: 'BokoVerticalMachiningCentre11'
FinishEndMilling Cycle Locations...
PositionPt   : (75.0 50.0 75.0)
ApproachPlane: 95.0
ZWorkPlane   : 65.0
RetractPlane : 95.0
ClearPlane   : 145.0
```

SetUp No: 6
Fixture For SetUp: MachineVice

```
Machining Process for Countersink8770I5

Opr No: 1; Opr Name: FinishCounterSinking
Tool For Opr: ('CSink' 15)
Recommended Cutting Speed: ('m/min' 20) Feed: ('mm/rev' 0.1)
Machine For Opr: 'BokoVerticalMachiningCentre11'
FinishCounterSinking Cycle Locations...
PositionPt: (10.0 130.0 0.0)
ApproachPt: 3.0
PlungePt  : -1.5
RetractPt : 8.0
Dwell(Sec): 1.0


Machining Process for Countersink8770I12

Opr No: 1; Opr Name: FinishCounterSinking
Tool For Opr: ('CSink' 15)
Recommended Cutting Speed: ('m/min' 20) Feed: ('mm/rev' 0.1)
Machine For Opr: 'BokoVerticalMachiningCentre11'
FinishCounterSinking Cycle Locations...
PositionPt: (180.0 130.0 0.0)
ApproachPt: 3.0
PlungePt  : -1.5
```

```
RetractPt : 8.0
Dwell(Sec): 1.0


Machining Process for Countersink8770I13

Opr No: 1; Opr Name: FinishCounterSinking
Tool For Opr: ('CSink' 15)
Recommended Cutting Speed: ('m/min' 20) Feed: ('mm/rev' 0.1)
Machine For Opr: 'BokoVerticalMachiningCentreI1'
FinishCounterSinking Cycle Locations...
PositionPt: (180.0 10.0 0.0)
ApproachPt: 3.0
PlungePt  : -1.5
RetractPt : 8.0
Dwell(Sec): 1.0


Machining Process for Countersink8770I14

Opr No: 1; Opr Name: FinishCounterSinking
Tool For Opr: ('CSink' 15)
Recommended Cutting Speed: ('m/min' 20) Feed: ('mm/rev' 0.1)
Machine For Opr: 'BokoVerticalMachiningCentreI1'
FinishCounterSinking Cycle Locations...
PositionPt: (10.0 10.0 0.0)
ApproachPt: 3.0
PlungePt  : -1.5
RetractPt : 8.0
Dwell(Sec): 1.0
```

Component Name: TestComp2



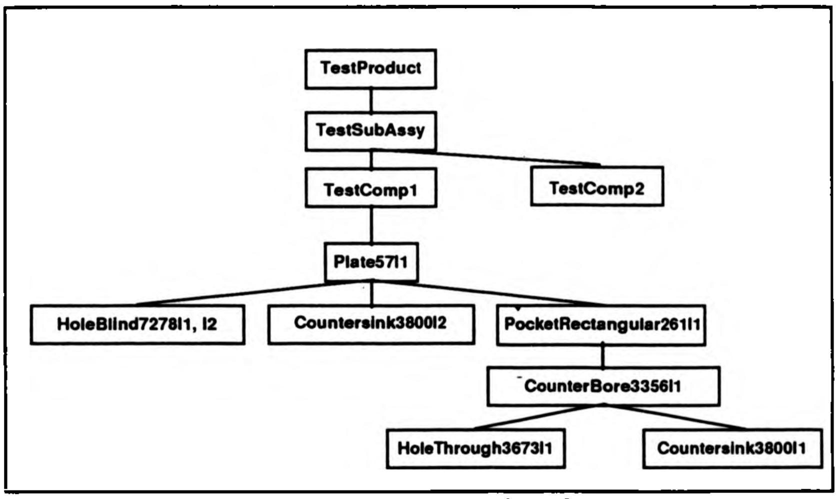
```
Assist :
Prompt :
Command:
Message:  19) -USR2.TRNG.PARTS.EXAMPLE2.$PD       FILED
```

Feature based model of TestComp2

CADDS5 Report File For TestComp2

```
*****     printing 585 Countersink582_I1      *****

class :
    class : Countersink                          library : CVOBJECTS
static attributes :
    BooleanOp  Operation                                  = Subtract
attributes :
    Workpiece  Workpiece                                  = 2143
    Name       Representation                             = rep
    CPlane     Orientation                             id = 6
                                                     name = TOP
                                                      i x = 1.000000e+00
                                                        y = 0.000000e+00
                                                        z = 0.000000e+00
                                                      j x = 0.000000e+00
                                                        y = 1.000000e+00
                                                        z = 0.000000e+00
                                                      k x = 0.000000e+00
                                                        y = 0.000000e+00
                                                        z = 1.000000e+00
    Location   Origin                                   x = 1.000000e+02
                                                        y = 7.500000e+01
                                                        z = 8.000000e+01
    ApplyType  Application                                = Boolean
    String     rawMatlCondition                           = Solid
    Real       surfFinish                                 = 2.000000e+00
    String     mcNotmc                                    = Y
    Face       EntryFace                                  = 2099
    String     childOf                                    = Pocket_Rectangular261_I1
    Diameter   HoleDiameter                               = 5.000000e+01 = millimeters
    Length     CsinkDepth                                 = 2.500000e+00 = millimeters
    Angle      CsinkAngle                                 = 9.000000e+01 = degrees
    String     TolHoleDiameter                            = 0.1 / - 0.1
    String     TolCsinkAngle                              = 0.1 / - 0.1
    String     TolCsinkDepth                              = 0.1 / - 0.1
*****     printing 946 Countersink582_I2      *****

class :
    class : Countersink                          library : CVOBJECTS
static attributes :
    BooleanOp  Operation                                  = Subtract
attributes :
    Workpiece  Workpiece                                  = 2404
    Name       Representation                             = rep
    CPlane     Orientation                             id = 11
                                                     name = BOTTOM
                                                      i x = 1.000000e+00
                                                        y = 0.000000e+00
                                                        z = 0.000000e+00
                                                      j x = 0.000000e+00
                                                        y = -1.000000e+00
                                                        z = 0.000000e+00
                                                      k x = 0.000000e+00
                                                        y = 0.000000e+00
                                                        z = -1.000000e+00
    Location   Origin                                   x = 1.000000e+02
                                                        y = 7.500000e+01
                                                        z = 0.000000e+00
    ApplyType  Application                                = Boolean
    String     rawMatlCondition                           = Solid
    Real       surfFinish                                 = 2.000000e+00
    String     mcNotmc                                    = Y
    Face       EntryFace                                  = 2324
    String     childOf                                    = Plate57_I1
    Diameter   HoleDiameter                         = 5.000000e+01 = millimeters
    Length     CsinkDepth                           = 2.500000e+00 = millimeters
    Angle      CsinkAngle                           = 9.000000e+01 = degrees
    String     TolHoleDiameter                      = 0.1 / - 0.1
    String     TolCsinkAngle                        = 0.1 / - 0.1
    String     TolCsinkDepth                        = 0.1 / - 0.1

*****      printing 481 Hole_Through478_I1      *****

class :
    class : Hole_Through                   library : CVOBJECTS
static attributes :
    BooleanOp  Operation                            = Subtract
attributes :
    Workpiece  Workpiece                            = 2143
    Name       Representation                       = rep
    CPlane     Orientation                      id = 6
                                              name = TOP
                                               i x = 1.000000e+00
                                                 y = 0.000000e+00
                                                 z = 0.000000e+00
                                               j x = 0.000000e+00
                                                 y = 1.000000e+00
                                                 z = 0.000000e+00
                                               k x = 0.000000e+00
                                                 y = 0.000000e+00
                                                 z = 1.000000e+00
    Location   Origin                            x = 1.000000e+02
                                                 y = 7.500000e+01
                                                 z = 8.000000e+01
    ApplyType  Application                          = Merge
    String     rawMatlCondition                     = Solid
    Real       surfFinish                           = 2.000000e+00
    String     mcNotmc                              = Y
    Face       EntryFace                            = 2099
    Face       ExitFace                             = 2133
    String     childOf                              = Pocket_Rectangular261_I1
    Diameter   HoleDiameter                         = 5.000000e+01 = millimeters
    String     TolHoleDiameter                      = 0.1 / - 0.1

*****      printing 264 Pocket_Rectangular261_I1      *****

class :
    class : Pocket_Rectangular             library : CVOBJECTS
static attributes :
    BooleanOp  Operation                            = Subtract
attributes :
    Workpiece  Workpiece                            = 1982
    Name       Representation                       = rep
    CPlane     Orientation                      id = 6
                                              name = TOP
                                               i x = 1.000000e+00
                                                 y = 0.000000e+00
                                                 z = 0.000000e+00
                                               j x = 0.000000e+00
                                                 y = 1.000000e+00
                                                 z = 0.000000e+00
                                               k x = 0.000000e+00
                                                 y = 0.000000e+00
                                                 z = 1.000000e+00
    Location   Origin                            x = 2.500000e+01
                                                 y = 2.500000e+01
                                                 z = 1.000000e+02
    ApplyType  Application                          = Boolean
    String     rawMatlCondition                     = Solid
    Real       surfFinish                           = 2.000000e+00
    String     mcNotmc                              = Y
    Face       EntryFace                            = 2009
    String     childOf                                  = Plate57_I1
    Length     PocketLength                             = 1.500000e+02 = millimeters
    Length     PocketWidth                              = 1.000000e+02 = millimeters
    Length     PocketDepth                              = 2.000000e+01 = millimeters
    String     TolPocketLength                          = 0.1 / - 0.1
    String     TolPocketWidth                           = 0.1 / - 0.1
    String     TolPocketDepth                           = 0.1 / - 0.1

*****     printing 60 Plate57_I1       *****

class :
    class : Plate                                   library : CVOBJECTS
static attributes :
    BooleanOp  Operation                                = Add
    Workpiece  Workpiece                                = 0
    ApplyType  Application                              = NoApply
attributes :
    Name       Representation                           = rep
    CPlane     Orientation                           id = 6
                                                   name = TOP
                                                    i x = 1.000000e+00
                                                      y = 0.000000e+00
                                                      z = 0.000000e+00
                                                    j x = 0.000000e+00
                                                      y = 1.000000e+00
                                                      z = 0.000000e+00
                                                    k x = 0.000000e+00
                                                      y = 0.000000e+00
                                                      z = 1.000000e+00
    Location   Origin                                 x = 0.000000e+00
                                                      y = 0.000000e+00
                                                      z = 0.000000e+00

    String     rawMatlCondition                         = Solid
    Real       surfFinish                               = 2.000000e+00
    String     mcNotmc                                  = Y
    Length     PlateLength                              = 2.000000e+02 = millimeters
    String     TolPlateLength                           = 0.1 / - 0.1
    Length     PlateWidth                               = 1.500000e+02 - millimeters
    String     TolPlateWidth                            = 0.1 / - 0.1
    Length     PlateHeight                              = 1.000000e+02 - millimeters
    String     TolPlateHeight                           = 0.1 / - 0.1
```

Report File Representation in the CAPP System:



Feature Instance hierarchy of TestComp2

The CAPP System Process Plan Generation for TestComp2

The CAPP Command : Generate Process

Message in the Text Display Window:

```
Plate57I1 No Stock, Machining not required
PocketRectangular261I1 finish by End Milling
PocketRectangular261I1Pocket Corner finishing by EndMilling
PocketRectangular261I1 roughing by End Milling
PocketRectangular261I1 Pocket corner hole drilling
Counterbore3356I1 can be finished by CounterBoring
Countersink3800I1 CounterSinking required
HoleThrough3673I1 can be finished by Boring
HoleThrough3673I1 cannot be finished by Reaming
HoleThrough3673I1 cannot be finished by TwistDrilling
HoleThrough3673I1 Rough Boring is not required
HoleThrough3673I1 Drilling required
HoleThrough3673I1 CentreDrilling required
Plate57I1 No Stock, Machining not required
Plate57I1 No Stock, Machining not required
Plate57I1 No Stock, Machining not required
HoleBlind7278I1 can be finished by Boring
HoleBlind7278I1 can be finished by Reaming
HoleBlind7278I1 can be finished by TwistDrilling
HoleBlind7278I1 CentreDrilling not required
HoleBlind7278I1 Rough Boring is not required
HoleBlind7278I1 Drilling required
HoleBlind7278I1 CentreDrilling not required
HoleBlind7278I1 Rough Boring is not required
HoleBlind7278I1 Drilling required
HoleBlind7278I1 CentreDrilling not required
HoleBlind7278I1 Rough Drilling not necessary
HoleBlind7278I2 can be finished by Boring
HoleBlind7278I2 can be finished by Reaming
HoleBlind7278I2 can be finished by TwistDrilling
HoleBlind7278I2 CentreDrilling not required
HoleBlind7278I2 Rough Boring is not required
HoleBlind7278I2 Drilling required
HoleBlind7278I2 CentreDrilling not required
HoleBlind7278I2 Rough Boring is not required
HoleBlind7278I2 Drilling required
HoleBlind7278I2 CentreDrilling not required
HoleBlind7278I2 Rough Drilling not necessary
Countersink3800I2 CounterSinking required
Plate57I1 No Stock, Machining not required
Plate57I1 No Stock, Machining not required
```

The universal knowledge based Process Plan for TestComp2

The CAPP Command : Show Component Process

```
Automated Process Plan
Component Name: TestComp2
Component Material: Free Machining Carbon Steel
Matl Description: Medium carbon ResulfurizedMatl Condition: Hot rolled
                                                            or Cold drawn
Material Specifications: 1140 Material Hardness: (175 225)

SetUp No: 1
Fixture For SetUp: UniversalVice

Machining Process for PocketRectangular261I1

Opr No: 1; Opr Name: Rough TwistDrilling
Tool For Opr: ('TwistDrill' 2.0 5.0 5.0 23.0)
Machine For Opr: ('BoringMc' 'DrillingMc' 'MillingMc' 'MachiningCentre')
Rough TwistDrilling Cycle Locations...
PositionPt: ((25.0 25.0 75.0) (175.0 25.0 75.0) (175.0 125.0 75.0) (25.0
            125.0 75.0))
ApproachPt: 78.0
PlungePt  : 55.0
RetractPt : 83.0
Dwell(Sec): 1.0


Opr No: 2; Opr Name: Rough EndMilling
Tool For Opr: ('EndMill' 12.5 25.0 50.0 30.0)
Machine For Opr: ('MillingMc' 'MachiningCentre')
Rough EndMilling Cycle Locations...
PositionPt   : (87.5 62.5 75.0)
ApproachPlane: 95.0
ZWorkPlane   : 55.4
RetractPlane : 95.0
ClearPlane   : 145.0


Opr No: 3; Opr Name: Finish EndMilling
Tool For Opr: ('EndMill' 12.5 16.666666 50.0 30.0)
Machine For Opr: ('MillingMc' 'MachiningCentre')
Finish EndMilling Cycle Locations...
PositionPt   : (87.5 62.5 75.0)
ApproachPlane: 95.0
ZWorkPlane   : 55.0
```

```
RetractPlane : 95.0
ClearPlane   : 145.0


Opr No: 4; Opr Name: Finish EndMilling
Tool For Opr: ('EndMill' 2.0 5.0 5.0 30.0)
Machine For Opr: ('MillingMc' 'MachiningCentre')
Finish EndMilling Cycle Locations...
PositionPt   : (87.5 62.5 75.0)
ApproachPlane: 95.0
ZWorkPlane   : 55.0
RetractPlane : 95.0
ClearPlane   : 145.0


Machining Process for Counterbore3356I1

Opr No: 1; Opr Name: Finish CounterBoring
Tool For Opr: ('CBore' 50.0 50.0 50.0 30.0)
Machine For Opr: ('BoringMc' 'DrillingMc' 'MillingMc' 'MachiningCentre')
Finish CounterBoring Cycle Locations...
PositionPt: (100.0 75.0 55.0)
ApproachPt: 75.0
PlungePt  : 45.0
RetractPt : 80.0
Dwell(Sec): 2.0


Machining Process for Countersink3800I1

Opr No: 1; Opr Name: Finish CounterSinking
Tool For Opr: ('CSink' 25.0 90.0 35.0 6.0)
Machine For Opr: ('BoringMc' 'DrillingMc' 'MillingMc' 'MachiningCentre')
Finish CounterSinking Cycle Locations...
PositionPt: (100.0 75.0 45.0)
ApproachPt: 48.0
PlungePt  : 42.5
RetractPt : 53.0
Dwell(Sec): 1.0


Machining Process for HoleThrough3673I1

Opr No: 1; Opr Name: Rough CentreDrilling
Rough CentreDrilling MUST PRECEDE Counterbore3356I1 Finishing
Tool For Opr: ('CentreDrill' 4.5 5.0 5.5 8.0)
```

Machine For Opr: ('BoringMc' 'DrillingMc' 'MillingMc' 'MachiningCentre')
Rough CentreDrilling Cycle Locations...
PositionPt: (100.0 75.0 55.0)
ApproachPt: 58.0
PlungePt : 47.0
RetractPt : 63.0
Dwell(Sec): 1.0

Opr No: 2; Opr Name: Rough TwistDrilling
Rough TwistDrilling **MUST PRECEDE** Counterbore3356I1 Finishing
Tool For Opr: ('TwistDrill' 29.25 29.5 29.75 67.0)
Machine For Opr: ('BoringMc' 'DrillingMc' 'MillingMc' 'MachiningCentre')
Rough TwistDrilling Cycle Locations...
PositionPt: (100.0 75.0 55.0)
ApproachPt: 58.0
PlungePt : -9.0
RetractPt : 63.0
Dwell(Sec): 0.0

Opr No: 3; Opr Name: Finish Boring
Tool For Opr: ('Bore' 30.0 30.0 30.0 49.5)
Machine For Opr: ('BoringMc' 'DrillingMc' 'MillingMc' 'MachiningCentre')
Finish Boring Cycle Locations...
PositionPt: (100.0 75.0 45.0)
ApproachPt: 48.0
PlungePt : -1.5
RetractPt : 53.0
Dwell(Sec): 0.0

SetUp No: 6
Fixture For SetUp: UniversalVice

3 Options Exist to machine HoleBlind7278I1
Option no: 1

Opr No: 1; Opr Name: Rough TwistDrilling
Tool For Opr: ('TwistDrill' 9.25 9.5 9.75 18.0)
Machine For Opr: ('BoringMc' 'DrillingMc' 'MillingMc' 'MachiningCentre')
Rough TwistDrilling Cycle Locations...
PositionPt: (175.0 125.0 0.0)
ApproachPt: 3.0
PlungePt : -15.0
RetractPt : 8.0

```
Dwell(Sec): 1.0


Opr No: 2; Opr Name: Finish Boring
Tool For Opr: ('Bore' 10.0 10.0 10.0 18.0)
Machine For Opr: ('BoringMc' 'DrillingMc' 'MillingMc' 'MachiningCentre')
Finish Boring Cycle Locations...
PositionPt: (175.0 125.0 0.0)
ApproachPt: 3.0
PlungePt  : -15.0
RetractPt : 8.0
Dwell(Sec): 1.0


Option no: 2

Opr No: 1; Opr Name: Rough TwistDrilling
Tool For Opr: ('TwistDrill' 9.45 9.7 9.95 18.0)
Machine For Opr: ('BoringMc' 'DrillingMc' 'MillingMc' 'MachiningCentre')
Rough TwistDrilling Cycle Locations...
PositionPt: (175.0 125.0 0.0)
ApproachPt: 3.0
PlungePt  : -15.0
RetractPt : 8.0
Dwell(Sec): 1.0


Opr No: 2; Opr Name: Finish Reaming
Tool For Opr: ('Reamer' 10.0 10.0 10.0 18.0)
Machine For Opr: ('BoringMc' 'DrillingMc' 'MillingMc' 'MachiningCentre')
Finish Reaming Cycle Locations...
PositionPt: (175.0 125.0 0.0)
ApproachPt: 3.0
PlungePt  : -15.0
RetractPt : 8.0
Dwell(Sec): 1.0


Option no: 3

Opr No: 1; Opr Name: Finish TwistDrilling
Tool For Opr: ('TwistDrill' 10.0 10.0 10.0 18.0)
Machine For Opr: ('BoringMc' 'DrillingMc' 'MillingMc' 'MachiningCentre')
Finish TwistDrilling Cycle Locations...
PositionPt: (175.0 125.0 0.0)
ApproachPt: 3.0
```

```
PlungePt  : -15.0
RetractPt : 8.0
Dwell(Sec): 1.0


3 Options Exist to machine HoleBlind7278I2
Option no: 1

Opr No: 1; Opr Name: Rough TwistDrilling
Tool For Opr: ('TwistDrill' 9.25 9.5 9.75 18.0)
Machine For Opr: ('BoringMc' 'DrillingMc' 'MillingMc' 'MachiningCentre')
Rough TwistDrilling Cycle Locations...
PositionPt: (25.0 25.0 0.0)
ApproachPt: 3.0
PlungePt  : -15.0
RetractPt : 8.0
Dwell(Sec): 1.0


Opr No: 2; Opr Name: Finish Boring
Tool For Opr: ('Bore' 10.0 10.0 10.0 18.0)
Machine For Opr: ('BoringMc' 'DrillingMc' 'MillingMc' 'MachiningCentre')
Finish Boring Cycle Locations...
PositionPt: (25.0 25.0 0.0)
ApproachPt: 3.0
PlungePt  : -15.0
RetractPt : 8.0
Dwell(Sec): 1.0


Option no: 2

Opr No: 1; Opr Name: Rough TwistDrilling
Tool For Opr: ('TwistDrill' 9.45 9.7 9.95 18.0)
Machine For Opr: ('BoringMc' 'DrillingMc' 'MillingMc' 'MachiningCentre')
Rough TwistDrilling Cycle Locations...
PositionPt: (25.0 25.0 0.0)
ApproachPt: 3.0
PlungePt  : -15.0
RetractPt : 8.0
Dwell(Sec): 1.0


Opr No: 2; Opr Name: Finish Reaming
Tool For Opr: ('Reamer' 10.0 10.0 10.0 18.0)
Machine For Opr: ('BoringMc' 'DrillingMc' 'MillingMc' 'MachiningCentre')
```

```
Finish Reaming Cycle Locations...
PositionPt: (25.0 25.0 0.0)
ApproachPt: 3.0
PlungePt  : -15.0
RetractPt : 8.0
Dwell(Sec): 1.0


Option no: 3

Opr No: 1; Opr Name: Finish TwistDrilling
Tool For Opr: ('TwistDrill' 10.0 10.0 10.0 18.0)
Machine For Opr: ('BoringMc' 'DrillingMc' 'MillingMc' 'MachiningCentre')
Finish TwistDrilling Cycle Locations...
PositionPt: (25.0 25.0 0.0)
ApproachPt: 3.0
PlungePt  : -15.0
RetractPt : 8.0
Dwell(Sec): 1.0


Machining Process for Countersink3800I2

Opr No: 1; Opr Name: Finish CounterSinking
Tool For Opr: ('CSink' 25.0 90.0 35.0 6.0)
Machine For Opr: ('BoringMc' 'DrillingMc' 'MillingMc' 'MachiningCentre')
Finish CounterSinking Cycle Locations...
PositionPt: (100.0 75.0 0.0)
ApproachPt: 3.0
PlungePt  : -3.0
RetractPt : 8.0
Dwell(Sec): 1.0
```

The Shop Specific Resource Search for TestComp2

The CAPP Command: **Assign Resources**

Message in the Text Display Window:

```
('Bore' 10.0 10.0 10.0 18.0) Not available
('Reamer' 10.0 10.0 10.0 18.0) Not available
('TwistDrill' 29.25 29.5 29.75 67.0) Not available
('Bore' 30) HSS Tool material / grade not in the database
```

The Shop Specific knowledge based Process Plan for TestComp2

The CAPP Command: Show Component Process

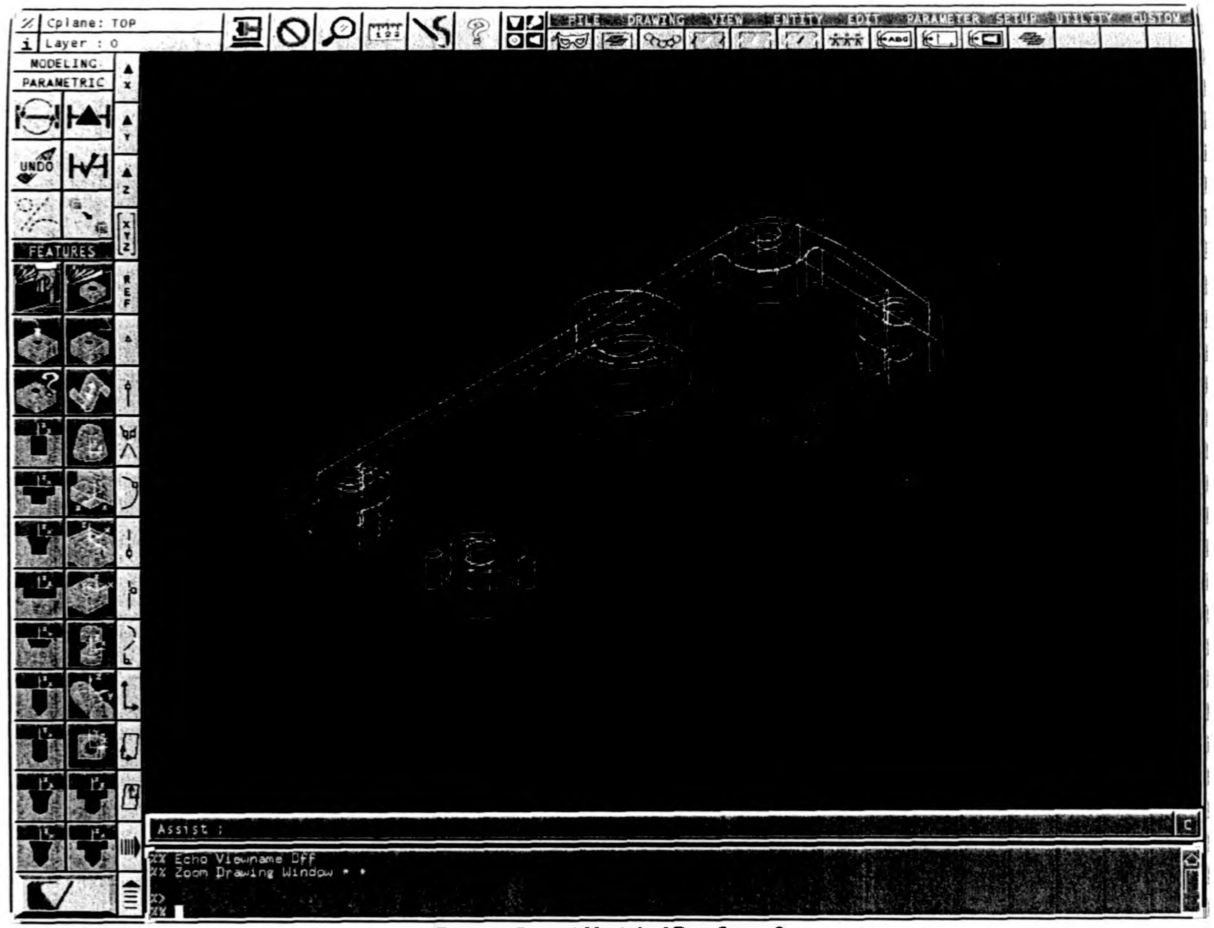
```
The process plan based on specific knowledge:
The CAPP Command: Show Component Process

Automated Process Plan
Component Name: TestComp2
Component Material: Free Machining Carbon Steel
Matl Description: Medium carbon Resulfurized Matl Condition: Hot rolled or
                                                              Cold drawn
Material Specifications: 1140 Material Hardness: (175 225)

SetUp No: 1
Fixture For SetUp: UniversalVice

Machining Process for PocketRectangular261I1

Opr No: 1; Opr Name: Rough TwistDrilling
Tool For Opr: ('TwistDrill' 5.0)
Recommended Cutting Speed: ('m/min' 24) Feed: ('mm/rev' 0.08)
Machine For Opr: 'MitsuSeikiHorizontalMachiningCentreI1'
Rough TwistDrilling Cycle Locations...
PositionPt: ((25.0 25.0 75.0) (175.0 25.0 75.0) (175.0 125.0 75.0) (25.0
             125.0 75.0))
ApproachPt: 78.0
PlungePt  : 55.0
RetractPt : 83.0
Dwell(Sec): 1.0


Opr No: 2; Opr Name: Rough EndMilling
Tool For Opr: ('EndMill' 25)
Recommended Cutting Speed: ('m/min' 120) Feed: ('mm/tooth' 0.2)
Machine For Opr: 'MitsuSeikiHorizontalMachiningCentreI1'
Rough EndMilling Cycle Locations...
PositionPt    : (87.5 62.5 75.0)
ApproachPlane: 95.0
ZWorkPlane    : 55.4
RetractPlane : 95.0
ClearPlane    : 145.0


Opr No: 3; Opr Name: Finish EndMilling
Tool For Opr: ('EndMill' 25)
Recommended Cutting Speed: ('m/min' 120) Feed: ('mm/tooth' 0.2)
```

```
Machine For Opr: 'MitsuSeikiHorizontalMachiningCentre11'
Finish EndMilling Cycle Locations...
PositionPt   : (87.5 62.5 75.0)
ApproachPlane: 95.0
ZWorkPlane   : 55.0
RetractPlane : 95.0
ClearPlane   : 145.0


Opr No: 4; Opr Name: Finish EndMilling
Tool For Opr: ('EndMill' 5.0)
Recommended Cutting Speed: ('m/min' 160) Feed: ('mm/tooth' 0.05)
Machine For Opr: 'MitsuSeikiHorizontalMachiningCentre11'
Finish EndMilling Cycle Locations...
PositionPt   : (87.5 62.5 75.0)
ApproachPlane: 95.0
ZWorkPlane   : 55.0
RetractPlane : 95.0
ClearPlane   : 145.0


Machining Process for Counterbore3356I1

Opr No: 1; Opr Name: Finish CounterBoring
Tool For Opr: ('CBore' 50.0)
Recommended Cutting Speed: ('m/min' 37) Feed: ('mm/rev' 0.2)
Machine For Opr: 'MitsuSeikiHorizontalMachiningCentre11'
Finish CounterBoring Cycle Locations...
PositionPt: (100.0 75.0 55.0)
ApproachPt: 75.0
PlungePt  : 45.0
RetractPt : 80.0
Dwell(Sec): 2.0


Machining Process for Countersink3800I1

Opr No: 1; Opr Name: Finish CounterSinking
Tool For Opr: ('CSink' 35)
Recommended Cutting Speed: ('m/min' 37) Feed: ('mm/rev' 0.15)
Machine For Opr: 'MitsuSeikiHorizontalMachiningCentre11'
Finish CounterSinking Cycle Locations...
PositionPt: (100.0 75.0 45.0)
ApproachPt: 48.0
PlungePt  : 42.5
RetractPt : 53.0
```

```
Dwell(Sec): 1.0


Machining Process for HoleThrough3673I1

Opr No: 1; Opr Name: Rough CentreDrilling
Rough CentreDrilling MUST PRECEDE Counterbore3356I1 Finishing
Tool For Opr: ('CentreDrill' 5.0)
Recommended Cutting Speed: ('m/min' 20) Feed: ('mm/rev' 0.07)
Machine For Opr: 'MitsuSeikiHorizontalMachiningCentreI1'
Rough CentreDrilling Cycle Locations...
PositionPt: (100.0 75.0 55.0)
ApproachPt: 58.0
PlungePt  : 47.0
RetractPt : 63.0
Dwell(Sec): 1.0


Opr No: 2; Opr Name: Rough TwistDrilling
Rough TwistDrilling MUST PRECEDE Counterbore3356I1 Finishing
Tool For Opr: ('TwistDrill' 29.5)
Recommended Cutting Speed: ('m/min' 24) Feed: ('mm/rev' 0.55)
Machine For Opr: 'MitsuSeikiHorizontalMachiningCentreI1'
Rough TwistDrilling Cycle Locations...
PositionPt: (100.0 75.0 55.0)
ApproachPt: 58.0
PlungePt  : -9.0
RetractPt : 63.0
Dwell(Sec): 0.0


Opr No: 3; Opr Name: Finish Boring
Tool For Opr: ('Bore' 30)
Recommended Cutting Speed: ('m/min' 150) Feed: ('mm/rev' 0.075)
Machine For Opr: 'MitsuSeikiHorizontalMachiningCentreI1'
Finish Boring Cycle Locations...
PositionPt: (100.0 75.0 45.0)
ApproachPt: 48.0
PlungePt  : -1.5
RetractPt : 53.0
Dwell(Sec): 0.0

SetUp No: 6
Fixture For SetUp: UniversalVice

3 Options Exist to machine HoleBlind7278I1
```

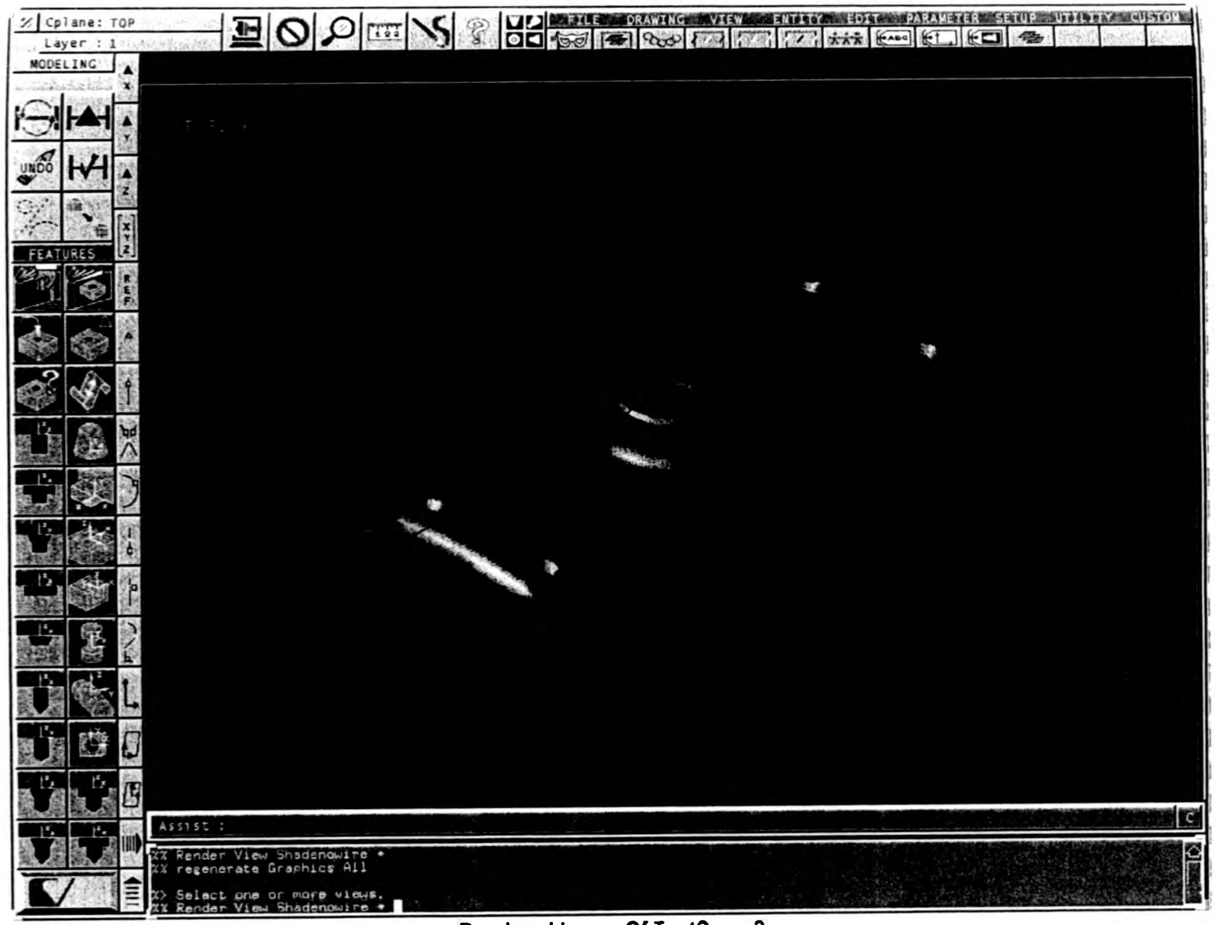
```
Option no: 3 is Selected

Opr No: 1; Opr Name: Finish TwistDrilling
Tool For Opr: ('TwistDrill' 10)
Recommended Cutting Speed: ('m/min' 24) Feed: ('mm/rev' 0.11)
Machine For Opr: 'MitsuSeikiHorizontalMachiningCentreI1'
Finish TwistDrilling Cycle Locations...
PositionPt: (175.0 125.0 0.0)
ApproachPt: 3.0
PlungePt  : -15.0
RetractPt : 8.0
Dwell(Sec): 1.0

Machining Process for Countersink3800I2

Opr No: 1; Opr Name: Finish CounterSinking
Tool For Opr: ('CSink' 35)
Recommended Cutting Speed: ('m/min' 37) Feed: ('mm/rev' 0.15)
Machine For Opr: 'MitsuSeikiHorizontalMachiningCentreI1'
Finish CounterSinking Cycle Locations...
PositionPt: (100.0 75.0 0.0)
ApproachPt: 3.0
PlungePt  : -3.0
RetractPt : 8.0
Dwell(Sec): 1.0
```

Component Name: TestComp3



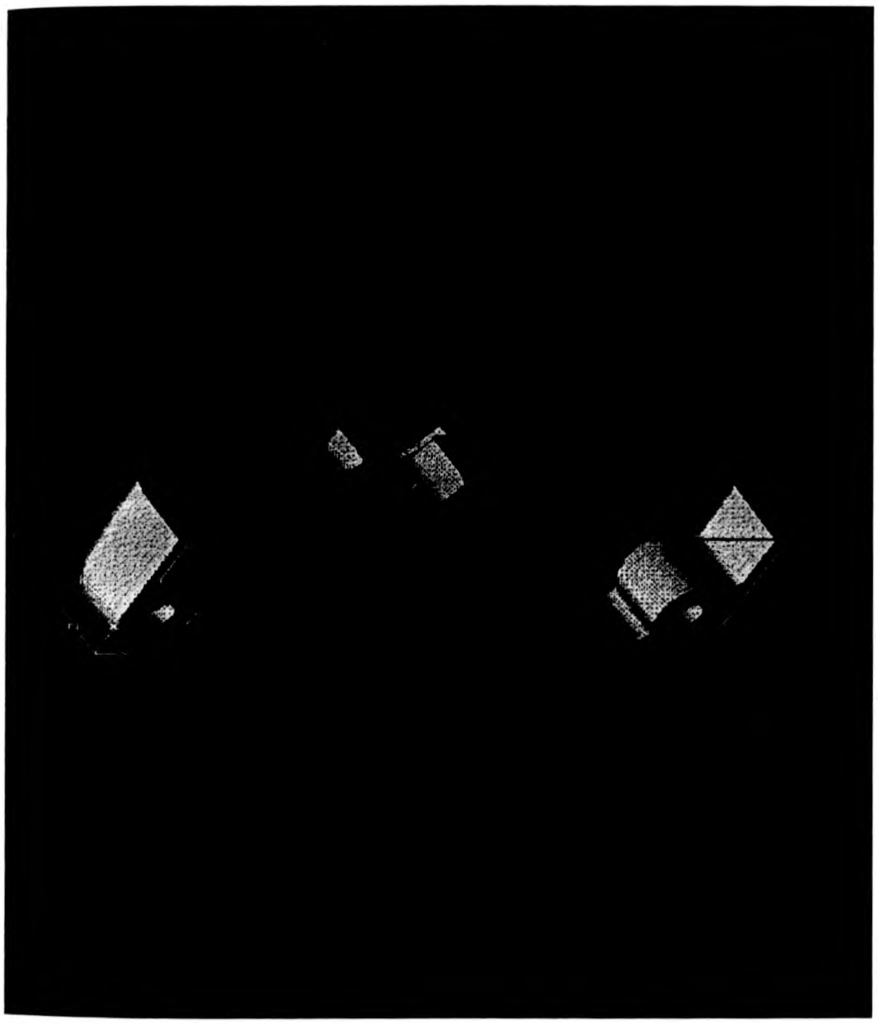
```
%% Echo Viewname Off
%% Zoom Drawing Window * *
%>
%%
```

Feature Based Model of TestComp3



```
%% Render View Shadenowire *
%% regenerate Graphics All

%> Select one or more views.
%% Render View Shadenowire *
```

Rendered Image Of TestComp3


Rendered Split Section Of TestComp3

CADDS5 Report file for TestComp3

```
*****     printing 13340 Countersink13337_I1     *****

class :
    class : Countersink                          library : CVOBJECTS
    classtype : Feature                          timestamp : 02-20-95 19:09
static attributes :
    BooleanOp  Operation                               = Subtract
attributes :
    Workpiece  Workpiece                               = 21894
    Name       Representation                          = rep
    CPlane     Orientation                          id = 6
                                                  name = TOP
                                                   i x = 1.000000e+00
                                                     y = 0.000000e+00
                                                     z = 0.000000e+00
                                                   j x = 0.000000e+00
                                                     y = 1.000000e+00
                                                     z = 0.000000e+00
                                                   k x = 0.000000e+00
                                                     y = 0.000000e+00
                                                     z = 1.000000e+00
    Location   Origin                                x = 2.950000e+01
                                                     y = 7.500000e+01
                                                     z = 2.650000e+01
    ApplyType  Application                             = Boolean
    String     rawMatlCondition                        = Solid
    Real       surfFinish                              = 2.000000e+00
    String     mcNotmc                                 = Y
    Face       EntryFace                               = 21864
    String     ChildOf                                 = Counterbore2764_I1
    Diameter   HoleDiameter                            = 1.250000e+01 = millimeters
    Length     CsinkDepth                              = 5.000000e-01 = millimeters
    Angle      CsinkAngle                              = 9.000000e+01 = degrees
    String     TolHoleDiameter                         = 0.1 / - 0.1
    String     TolCsinkAngle                           = 0.1 / - 0.1
    String     TolCsinkDepth                           = 0.1 / - 0.1
*****     printing 18276 Countersink13337_I2     *****

class :
    class : Countersink                          library : CVOBJECTS
    classtype : Feature                          timestamp : 02-20-95 19:09
static attributes :
    BooleanOp  Operation                               = Subtract
attributes :
    Workpiece  Workpiece                               = 21894
    Name       Representation                          = rep
    CPlane     Orientation                          id = 6
                                                  name = TOP
                                                   i x = 1.000000e+00
                                                     y = 0.000000e+00
                                                     z = 0.000000e+00
                                                   j x = 0.000000e+00
                                                     y = 1.000000e+00
                                                     z = 0.000000e+00
                                                   k x = 0.000000e+00
                                                     y = 0.000000e+00
                                                     z = 1.000000e+00
    Location   Origin                                x = 1.000000e+01
                                                     y = 1.400000e+02
                                                     z = 1.155494e+01
    ApplyType  Application                             = Boolean
    String     rawMatlCondition                        = Solid
    Real       surfFinish                              = 2.000000e+00
    String     mcNotmc                                 = Y
    Face        EntryFace                                   = 0
    String      ChildOf                                     = Bracket54_I1
    Diameter    HoleDiameter                                = 6.000000e+00 = millimeters
    Length      CsinkDepth                                  = 2.000000e+00 = millimeters
    Angle       CsinkAngle                                  = 9.000000e+01 = degrees
    String      TolHoleDiameter                             = 0.1 / - 0.1
    String      TolCsinkAngle                               = 0.1 / - 0.1
    String      TolCsinkDepth                               = 0.1 / - 0.1

*****     printing 18489 Countersink13337_I3     *****

class :
    class : Countersink                          library : CVOBJECTS
    classtype : Feature                          timestamp : 02-20-95 19:09
static attributes :
    BooleanOp   Operation                                   = Subtract
attributes :
    Workpiece   Workpiece                                   = 21894
    Name        Representation                              = rep
    CPlane      Orientation                              id = 6
                                                       name = TOP
                                                        i x = 1.000000e+00
                                                          y = 0.000000e+00
                                                          z = 0.000000e+00
                                                        j x = 0.000000e+00
                                                          y = 1.000000e+00
                                                          z = 0.000000e+00
                                                        k x = 0.000000e+00
                                                          y = 0.000000e+00
                                                          z = 1.000000e+00
    Location    Origin                                    x = 5.000000e+01
                                                          y = 1.400000e+02
                                                          z = 1.155494e+01
    ApplyType   Application                                 = Boolean
    String      rawMatlCondition                            = Solid
    Real        surfFinish                                  = 2.000000e+00
    String      mcNotmc                                     = Y
    Face        EntryFace                                   = 0
    String      ChildOf                                     = Bracket54_I1
    Diameter    HoleDiameter                                = 6.000000e+00 = millimeters
    Length      CsinkDepth                                  = 2.000000e+00 = millimeters
    Angle       CsinkAngle                                  = 9.000000e+01 = degrees
    String      TolHoleDiameter                             = 0.1 / - 0.1
    String      TolCsinkAngle                               = 0.1 / - 0.1
    String      TolCsinkDepth                               = 0.1 / - 0.1

*****     printing 18519 Countersink13337_I4     *****

class :
    class : Countersink                          library : CVOBJECTS
    classtype : Feature                          timestamp : 02-20-95 19:09
static attributes :
    BooleanOp   Operation                                   = Subtract
attributes :
    Workpiece   Workpiece                                   = 21894
    Name        Representation                              = rep
    CPlane      Orientation                              id = 6
                                                       name = TOP
                                                        i x = 1.000000e+00
                                                          y = 0.000000e+00
                                                          z = 0.000000e+00
                                                        j x = 0.000000e+00
                                                          y = 1.000000e+00
                                                          z = 0.000000e+00
                                                        k x = 0.000000e+00
                                                          y = 0.000000e+00
                                                          z = 1.000000e+00
    Location    Origin                                    x = 5.000000e+01
```

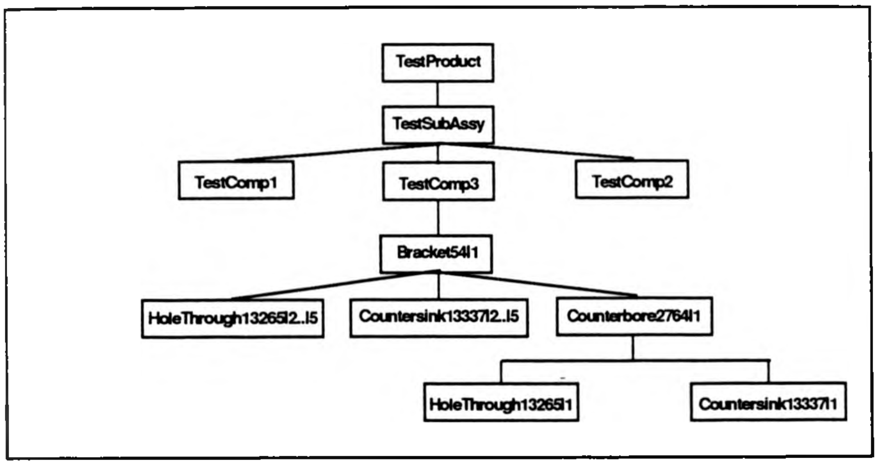
```
                                                          y = 1.000000e+01
                                                          z = 1.155494e+01
    ApplyType  Application                                  = Boolean
    String     rawMatlCondition                             = Solid
    Real       surfFinish                                   = 2.000000e+00
    String     mcNotmc                                      = Y
    Face       EntryFace                                    = 0
    String     ChildOf                                      = Bracket54_I1
    Diameter   HoleDiameter                                 = 6.000000e+00 = millimeters
    Length     CsinkDepth                                   = 2.000000e+00 = millimeters
    Angle      CsinkAngle                                   = 9.000000e+01 = degrees
    String     TolHoleDiameter                              = 0.1 / - 0.1
    String     TolCsinkAngle                                = 0.1 / - 0.1
    String     TolCsinkDepth                                = 0.1 / - 0.1

*****     printing 18549 Countersink13337_I5     *****

class :
    class : Countersink                        library : CVOBJECTS
    classtype : Feature                        timestamp : 02-20-95 19:09
static attributes :
    BooleanOp  Operation                                    = Subtract
attributes :
    Workpiece  Workpiece                                    = 21894
    Name       Representation                               = rep
    CPlane     Orientation                               id = 6
                                                       name = TOP
                                                        i x = 1.000000e+00
                                                          y = 0.000000e+00
                                                          z = 0.000000e+00
                                                        j x = 0.000000e+00
                                                          y = 1.000000e+00
                                                          z = 0.000000e+00
                                                        k x = 0.000000e+00
                                                          y = 0.000000e+00
                                                          z = 1.000000e+00
    Location   Origin                                     x = 1.000000e+01
                                                          y = 1.000000e+01
                                                          z = 1.155494e+01
    ApplyType  Application                                  = Boolean
    String     rawMatlCondition                             = Solid
    Real       surfFinish                                   = 2.000000e+00
    String     mcNotmc                                      = Y
    Face       EntryFace                                    = 0
    String     ChildOf                                      = Bracket54_I1
    Diameter   HoleDiameter                                 = 6.000000e+00 = millimeters
    Length     CsinkDepth                                   = 2.000000e+00 = millimeters
    Angle      CsinkAngle                                   = 9.000000e+01 = degrees
    String     TolHoleDiameter                              = 0.1 / - 0.1
    String     TolCsinkAngle                                = 0.1 / - 0.1
    String     TolCsinkDepth                                = 0.1 / - 0.1
*****     printing 13268 Hole_Through13265_I1     *****

class :
    class : Hole_Through                       library : CVOBJECTS
    classtype : Feature                        timestamp : 02-20-95 19:20
static attributes :
    BooleanOp  Operation                                    = Subtract
attributes :
    Workpiece  Workpiece                                    = 21894
    Name       Representation                               = rep
    CPlane     Orientation                               id = 6
                                                       name = TOP
                                                        i x = 1.000000e+00
                                                          y = 0.000000e+00
                                                          z = 0.000000e+00
                                                        j x = 0.000000e+00
```

```
                                                          y = 1.000000e+00
                                                          z = 0.000000e+00
                                                        k x = 0.000000e+00
                                                          y = 0.000000e+00
                                                          z = 1.000000e+00
    Location   Origin                                     x = 2.950000e+01
                                                          y = 7.500000e+01
                                                          z = 2.650000e+01
    ApplyType  Application                                  = Merge
    String     rawMatlCondition                             = 1.25
    Real       surfFinish                                   = 1.000000e+00
    String     mcNotmc                                      = Y
    Face       EntryFace                                    = 21864
    Face       ExitFace                                     = 21782
    String     ChildOf                                      = Counterbore2764_I1
    Diameter   HoleDiameter                                 = 1.250000e+01 = millimeters
    String     TolHoleDiameter                              = 0.05 / -0.05

*****     printing 18070 Hole_Through13265_I2          *****

class :
    class : Hole_Through                        library : CVOBJECTS
    classtype : Feature                         timestamp : 02-20-95 19:20
static attributes :
    BooleanOp  Operation                                    = Subtract
attributes :
    Workpiece  Workpiece                                    = 21894
    Name       Representation                               = rep
    CPlane     Orientation                               id = 6
                                                       name = TOP
                                                        i x = 1.000000e+00
                                                          y = 0.000000e+00
                                                          z = 0.000000e+00
                                                        j x = 0.000000e+00
                                                          y = 1.000000e+00
                                                          z = 0.000000e+00
                                                        k x = 0.000000e+00
                                                          y = 0.000000e+00
                                                          z = 1.000000e+00
    Location   Origin                                     x = 1.000000e+01
                                                          y = 1.400000e+02
                                                          z = 1.155494e+01
    ApplyType  Application                                  = Merge
    String     rawMatlCondition                             = Solid
    Real       surfFinish                                   = 3.000000e+00
    String     mcNotmc                                      = Y
    Face       EntryFace                                    = 21035
    Face       ExitFace                                     = 21123
    String     ChildOf                                      = Bracket54_I1
    Diameter   HoleDiameter                                 = 6.000000e+00 = millimeters
    String     TolHoleDiameter                              = 0.1 / - 0.1

*****     printing 18150 Hole_Through13265_I3          *****

class :
    class : Hole_Through                        library : CVOBJECTS
    classtype : Feature                         timestamp : 02-20-95 19:20
static attributes :
    BooleanOp  Operation                                    = Subtract
attributes :
    Workpiece  Workpiece                                    = 21894
    Name       Representation                               = rep
    CPlane     Orientation                               id = 6
                                                       name = TOP
                                                        i x = 1.000000e+00
                                                          y = 0.000000e+00
                                                          z = 0.000000e+00
                                                        j x = 0.000000e+00
```

```
                                                                y = 1.000000e+00
                                                                z = 0.000000e+00
                                                              k x = 0.000000e+00
                                                                y = 0.000000e+00
                                                                z = 1.000000e+00
    Location   Origin                                           x = 5.000000e+01
                                                                y = 1.400000e+02
                                                                z = 1.155494e+01
    ApplyType  Application                                        = Merge
    String     rawMatlCondition                                   = Solid
    Real       surfFinish                                         = 3.000000e+00
    String     mcNotmc                                            = Y
    Face       EntryFace                                          = 21035
    Face       ExitFace                                           = 21123
    String     ChildOf                                            = Bracket54_I1
    Diameter   HoleDiameter                                       = 6.000000e+00 = millimeters
    String     TolHoleDiameter                                    = 0.1 / - 0.1

*****     printing 18182 Hole_Through13265_I4     *****

class :
    class : Hole_Through                       library : CVOBJECTS
    classtype : Feature                        timestamp : 02-20-95 19:20
static attributes :
    BooleanOp  Operation                                          = Subtract
attributes :
    Workpiece  Workpiece                                          = 21894
    Name       Representation                                     = rep
    CPlane     Orientation                                     id = 6
                                                             name = TOP
                                                              i x = 1.000000e+00
                                                                y = 0.000000e+00
                                                                z = 0.000000e+00
                                                              j x = 0.000000e+00
                                                                y = 1.000000e+00
                                                                z = 0.000000e+00
                                                              k x = 0.000000e+00
                                                                y = 0.000000e+00
                                                                z = 1.000000e+00
    Location   Origin                                           x = 5.000000e+01
                                                                y = 1.000000e+01
                                                                z = 1.155494e+01
    ApplyType  Application                                        = Merge
    String     rawMatlCondition                                   = Solid
    Real       surfFinish                                         = 3.000000e+00
    String     mcNotmc                                            = Y
    Face       EntryFace                                          = 21035
    Face       ExitFace                                           = 21123
    String     ChildOf                                            = Bracket54_I1
    Diameter   HoleDiameter                                       = 6.000000e+00 = millimeters
    String     TolHoleDiameter                                    = 0.1 / - 0.1

*****     printing 18214 Hole_Through13265_I5     *****

class :
    class : Hole_Through                       library : CVOBJECTS
    classtype : Feature                        timestamp : 02-20-95 19:20
static attributes :
    BooleanOp  Operation                                          = Subtract
attributes :
    Workpiece  Workpiece                                          = 21894
    Name       Representation                                     = rep
    CPlane     Orientation                                     id = 6
                                                             name = TOP
                                                              i x = 1.000000e+00
                                                                y = 0.000000e+00
                                                                z = 0.000000e+00
                                                              j x = 0.000000e+00
```

```
                                                          y = 1.000000e+00
                                                          z = 0.000000e+00
                                                        k x = 0.000000e+00
                                                          y = 0.000000e+00
                                                          z = 1.000000e+00
    Location   Origin                                     x = 1.000000e+01
                                                          y = 1.000000e+01
                                                          z = 1.155494e+01
    ApplyType  Application                                  = Merge
    String     rawMatlCondition                             = Solid
    Real       surfFinish                                   = 3.000000e+00
    String     mcNotmc                                      = Y
    Face       EntryFace                                    = 21035
    Face       ExitFace                                     = 21123
    String     ChildOf                                      = Bracket54_I1
    Diameter   HoleDiameter                                 = 6.000000e+00 = millimeters
    String     TolHoleDiameter                              = 0.1 / - 0.1
*****     printing 2767 Counterbore2764_I1      *****

class :
    class : Counterbore                      library : CVOBJECTS
    classtype : Feature                      timestamp : 02-20-95 19:09
static attributes :
    BooleanOp  Operation                                    = Subtract
attributes :
    Workpiece  Workpiece                                    = 21894
    Name       Representation                               = rep
    CPlane     Orientation                               id = 6
                                                       name = TOP
                                                        i x = 1.000000e+00
                                                          y = 0.000000e+00
                                                          z = 0.000000e+00
                                                        j x = 0.000000e+00
                                                          y = 1.000000e+00
                                                          z = 0.000000e+00
                                                        k x = 0.000000e+00
                                                          y = 0.000000e+00
                                                          z = 1.000000e+00
    Location   Origin                                     x = 2.950000e+01
                                                          y = 7.500000e+01
                                                          z = 2.950000e+01
    ApplyType  Application                                  = Boolean
    String     rawMatlCondition                             =  5
    Real       surfFinish                                   = 2.000000e+00
    String     mcNotmc                                      = Y
    Face       EntryFace                                    = 21784
    String     ChildOf                                      = Bracket54_I1
    Diameter   CboreDiameter                                = 2.000000e+01 = millimeters
    Length     CboreDepth                                   = 3.000000e+00 = millimeters
    String     TolCboreDiameter                             = 0.1 / - 0.1
    String     TolCboreDepth                                = 0.1 / - 0.1
*****     printing 57 Bracket54_I1      *****

class :
    class : Bracket                          library : CVOBJECTS
    classtype : Feature                      timestamp : 02-20-95 18:48
static attributes :
    BooleanOp  Operation                                    = Add
    Workpiece  Workpiece                                    = 0
    ApplyType  Application                                  = NoApply
attributes :
    Name       Representation                               = rep
    CPlane     Orientation                               id = 6
                                                       name = TOP
                                                        i x = 1.000000e+00
                                                          y = 0.000000e+00
```

```
                                                  z = 0.000000e+00
                                                j x = 0.000000e+00
                                                  y = 1.000000e+00
                                                  z = 0.000000e+00
                                                k x = 0.000000e+00
                                                  y = 0.000000e+00
                                                  z = 1.000000e+00
Location    Origin                                x = 0.000000e+00
                                                  y = 0.000000e+00
                                                  z = 0.000000e+00
String      rawMatlCondition                        = Solid
Real        surfFinish                              = 2.000000e+00
String      mcNotmc                                 = N
Length      BracketLength                           = 1.500000e+02 = millimeters
Length      BracketWidth                            = 6.000000e+01 = millimeters
Length      BracketBossHgt                          = 3.000000e+01 = millimeters
```



Feature Instance hierarchy of TestComp3

The CAPP System Process Plan Generation for TestComp3

The CAPP Command: **Generate Process**

Message in the Text Display Window:

```
Countersinking is included in CentreDrilling
Countersinking is included in CentreDrilling
Countersinking is included in CentreDrilling
Countersinking is included in CentreDrilling
HoleThrough132651 2 cannot be finished by Boring
HoleThrough132651 2 cannot be finished by Reaming
HoleThrough132651 2 can be finished by TwistDrilling
HoleThrough132651 2 CentreDrilling not reqµired
HoleThrough132651 2 Rough Drilling not necessary
HoleThrough132651 3 cannot be finished by Boring
HoleThrough132651 3 cannot be finished by Reaming
HoleThrough132651 3 can be finished by TwistDrilling
HoleThrough132651 3 CentreDrilling not required
HoleThrough132651 3 Rough Drilling not necessary
HoleThrough132651 4 cannot be finished by Boring
HoleThrough132651 4 cannot be finished by Reaming
HoleThrough132651 4 can be finished by TwistDrilling
HoleThrough132651 4 CentreDrilling not required
HoleThrough132651 4 Rough Drilling not necessary
HoleThrough132651 5 cannot be finished by Boring
HoleThrough132651 5 cannot be finished by Reaming
HoleThrough132651 5 can be finished by TwistDrilling
HoleThrough132651 5 CentreDrilling not required
HoleThrough132651 5 Rough Drilling not necessary
Counterbore27641 1 cannot be finished by CounterBoring
Countersink133371 1 CounterSinking required
HoleThrough132651 1 can be finished by Boring
HoleThrough132651 1 cannot be finished by Reaming
HoleThrough132651 1 cannot be finished by TwistDrilling
HoleThrough132651 1 Rough Boring is required
```

The Shop Specific Resource Search for TestComp3

The CAPP Command: **Assign Resources**

Message in the Text Display Window:

```
('TwistDrill' 6.0 6.0 6.0 16.35494) Not available
('Bore' 12) carbide Tool material / grade not in the database
('Bore' 12.5 12.5 12.5 31.0) Not available
```

The Shop Specific knowledge based Process Plan for TestComp3

The CAPP Command: **Show Component Process**

Automated Process Plan
Component Name: TestComp3
Component Material: Aluminum Alloy
Matl Description: Die Casting **Matl Condition:** Solution treated and Aged
Material Specifications: A384.0 **Material Hardness:** (70 125)

SetUp No: 1
Fixture For SetUp: BracketFixture

Machining Process for HoleThrough132651 2

Opr No: 1; Opr Name: Finish TwistDrilling
Tool For Opr: ('TwistDrill' 6.0)
Recommended Cutting Speed: ('m/min' 37) Feed: ('mm/rev' 0.18)
Machine For Opr: 'BokoVerticalMachiningCentre11'
Finish TwistDrilling Cycle Locations...
PositionPt: (10.0 140.0 11.55494)
ApproachPt: 14.55494
PlungePt : -1.8
RetractPt : 19.55494
Dwell(Sec): 0.0

Machining Process for HoleThrough1326513

Opr No: 1; Opr Name: Finish TwistDrilling
Tool For Opr: ('TwistDrill' 6.0)
Recommended Cutting Speed: ('m/min' 37) Feed: ('mm/rev' 0.18)
Machine For Opr: 'BokoVerticalMachiningCentre11'
Finish TwistDrilling Cycle Locations...
PositionPt: (50.0 140.0 11.55494)
ApproachPt: 14.55494
PlungePt : -1.8
RetractPt : 19.55494
Dwell(Sec): 0.0

Machining Process for HoleThrough1326514

Opr No: 1; Opr Name: Finish TwistDrilling
Tool For Opr: ('TwistDrill' 6.0)
Recommended Cutting Speed: ('m/min' 37) Feed: ('mm/rev' 0.18)

```
Machine For Opr: 'BokoVerticalMachiningCentre11'
Finish TwistDrilling Cycle Locations...
PositionPt: (50.0 10.0 11.55494)
ApproachPt: 14.55494
PlungePt  : -1.8
RetractPt : 19.55494
Dwell(Sec): 0.0


Machining Process for HoleThrough132615

Opr No: 1; Opr Name: Finish TwistDrilling
Tool For Opr: ('TwistDrill' 6.0)
Recommended Cutting Speed: ('m/min' 37) Feed: ('mm/rev' 0.18)
Machine For Opr: 'BokoVerticalMachiningCentre11'
Finish TwistDrilling Cycle Locations...
PositionPt: (10.0 10.0 11.55494)
ApproachPt: 14.55494
PlungePt  : -1.8
RetractPt : 19.55494
Dwell(Sec): 0.0


Machining Process for Countersink133371l

Opr No: 1; Opr Name: Finish CounterSinking
Tool For Opr: ('CSink' 15)
Recommended Cutting Speed: ('m/min' 120) Feed: ('mm/rev' 0.2)
Machine For Opr: 'BokoVerticalMachiningCentre11'
Finish CounterSinking Cycle Locations...
PositionPt: (29.5 75.0 26.5)
ApproachPt: 29.5
PlungePt  : 26.0
RetractPt : 34.5
Dwell(Sec): 1.0


Machining Process for HoleThrough132651l

Opr No: 1; Opr Name: Rough Boring
Tool For Opr: ('Bore' 12)
Recommended Cutting Speed: ('m/min' 335) Feed: ('mm/rev' 0.2)
Machine For Opr: 'BokoVerticalMachiningCentre11'
Rough Boring Cycle Locations...
PositionPt: (29.5 75.0 26.5)
ApproachPt: 29.5
```

```
PlungePt  : -1.5
RetractPt : 34.5
Dwell(Sec): 0.0


Opr No: 2; Opr Name: Finish Boring
Tool For Opr: ('Bore' 12.5)
Recommended Cutting Speed: ('m/min' 410) Feed: ('mm/rev' 0.15)
Machine For Opr: 'BokoVerticalMachiningCentre11'
Finish Boring Cycle Locations...
PositionPt: (29.5 75.0 26.5)
ApproachPt: 29.5
PlungePt  : -1.5
RetractPt : 34.5
Dwell(Sec): 0.0
```

APPENDIX G

IMPEMENTATION OF FEATURE INTERACTION METHODS
AND WMG CAPP SYSTEM TEST RESULTS

1. Interaction related default attribute assignment

class: a Feature
method: initialize

```
    "initialise proximityFeature attributes with default values."

    proximityFeature := Dictionary new.
    proximityFeature at:'intFeature'       put:nil;
                     at:'to'               put:nil;
                     at:'on'               put:nil;
                     at:'past'             put:nil;
                     at:'in'               put:nil;
                     at:'out'              put:nil;
                     at:'right'            put:nil;
                     at:'left'             put:nil;
                     at:'top'              put:nil;
                     at:'bottom'           put:nil;
                     at:'atAngle'          put:nil.

    super initialize
```

2. Importing Component Feature Data from CADDS5 Feature Report File

class: a ProductCom
method: importComponent

```
"converts Cadds5 report file into a Component feature instance hierarchy"

  (curProduct isMemberOf: Component)
     ifFalse: [self error: 'Sorry, Select a Component']
      ifTrue: [curProduct importFeatures]
```

class: a Component
method: importFeatures

```
   "converts Cadds5 report file into a Component feature instance hierarchy"

  | input reportFileName className libraryName operation workpiece representation orientation
   vector org origin datum aString printingPattern instanceName char classPattern oprPattern
   application word line line1 attribute attributeType attributeVar attributeCollection  lastCollection
   fileIn featureData  |

  (reportFileName := Prompter
     prompt: 'Component report file name ?'
     default: 'test') isNil
     ifTrue: [self].

  input := File pathName: '\smaltalk\',reportFileName.
  instanceName := ''.
  className := ''.
  libraryName := ''.
  orientation := OrderedCollection new.
  origin := OrderedCollection new.
```

```
attributeCollection := OrderedCollection new.
featureData := Dictionary new.

(printingPattern := Pattern new: #('printing'))
   matchBlock:[ input nextWord.
          [(aString := input nextWord) = 'class']
            whileFalse: [instanceName := instanceName,aString].
          input nextWord.

          [(aString := input nextWord) = 'library']
          whileFalse: [className := className, aString].

          [(aString := input nextWord) = 'static']
          whileFalse: [libraryName := libraryName, aString].

          [(aString := input nextWord) = 'Operation']
          whileFalse: [].
          operation := input nextWord.

          [(aString := input nextWord) = 'Workpiece']
          whileFalse: [].
          input nextWord.
          workpiece := (input nextWord) asInteger.

          [(aString := input nextWord) = 'Representation']
          whileFalse: [].
          representation := input nextWord.

          [(aString := input nextWord) = 'id']
          whileFalse: [].
          orientation add: (input nextWord) asInteger.
          input nextWord.
          datum := input nextWord.
          orientation add: datum.

          [(aString := input nextWord) = 'Location']
          whileFalse: [[(char := input next) = $=]
                whileFalse: [].
                oVector := (input nextLine).
                orientation add: oVector.
               ].

          input nextWord.
          3 timesRepeat:[input nextWord.
                input skip: 3.
                org := (input nextLine).
                origin add: (org asFloat).
               ].

                [(line := input nextLine asArrayOfSubstrings) isEmpty]
                whileFalse: [line1 := line asOrderedCollection.
                      (line1 size > 4 and:[ (line1 at:5) ~= '/'])
                       ifTrue: [line1 remove: (line1 at:3);
                              remove: (line1 at:5)]
                       ifFalse: [line1 remove: (line1 at:3)].

                      attributeCollection add: line1.
```

```
                        ].

                Test answer: 'instanceName ',instanceName.
                Test answer: 'className ', className.
                Test answer: 'libraryName ', libraryName.
                Test answer: 'operation ', operation.
                Test answer: 'workpiece ', workpiece printString.
                Test answer: 'representation ', representation.
                Test answer: 'datum id ', (orientation at:1) printString.
                Test answer: 'datum ', (orientation at:2) printString.
                Test answer: 'orientation size ', (orientation size) printString.
                3 to: orientation size do: [:index|
                  Test answer: (orientation at:index) printString].
                  Test answer: 'origin is '.
                   origin do:[:each | Test answer: each printString].
                attributeCollection do: [:each| Test answer: (each at:1),' ',(each at:2),' ',(each at:3)].

                featureData at: 'instanceName' put: instanceName;
                        at: 'className'    put: className;
                        at: 'libraryName'  put: libraryName;
                        at: 'workpiece'    put: workpiece;
                        at: 'operation'    put: operation;
                        at: 'representation' put: representation;
                        at: 'childOf'      put: self;
                        at: 'orientation'  put: orientation;
                        at: 'origin'       put: origin asArray;
                        at: 'attributeCollection' put: attributeCollection.

                "Check Feature Class, if exists, initialize"

                ((Smalltalk at: className asSymbol) subClassOf: Feature)
                 ifTrue: [className := Smalltalk at: className asSymbol.
                        className initialize: featureData]
                 ifFalse: [Test answer: 'Define Feature class ',className].

                instanceName := ''.
                className   := ''.
                libraryName := ''.
                orientation := OrderedCollection new.
                origin := OrderedCollection new.
                attributeCollection := OrderedCollection new.
                featureData := Dictionary new.
            ]. "end of match block"

    [input atEnd]
      whileFalse: [ [(word := input nextWord) isNil]
                whileFalse: [printingPattern match: word].
            ].

    self sortChildren
```

class: a Feature
method: transformOrigin

```
      "The feature report file provides the origin coordinates with respect to
       TOP cpl. Therefore, it is necessary to transform the origin w.r.to the
       feature cpl"
    | transformeeOrg x y z oldLocCollection oldOrg|
   self datum = 'TOP'
   ifFalse:[
   x := (self origin) at:1.
   y := (self origin) at:2.
   z := (self origin) at:3.
   oldLocCollection := OrderedCollection new.
   oldLocCollection add: x;
                        add: y;
                        add: z.
   oldOrg          := Vertex new initialise: oldLocCollection.
   transformeeOrg:= oldOrg transformTopTo: (self datum).
   origin          := OrderedCollection new.
   origin add: transformeeOrg x;
        add: transformeeOrg y;
        add: transformeeOrg z].
   origin := origin asArray
```

class: a Feature
method: getProximityAttributes: aFeatureData

```
     "Assigns feature proximity attributes from aFeatureData"

  (aFeatureData at: 'attributeCollection') do: [:each |
     (each at: 2) = 'intFeature'
        ifTrue: [proximityFeature at:'intFeature' put:((each at: 3) reject: [:c | c = $_])].

     (each at: 2) = 'to'
        ifTrue: [proximityFeature at:'to' put: (each at: 3) ].

     (each at: 2) = 'on'
        ifTrue: [proximityFeature at:'on' put: (each at: 3) ].

     (each at: 2) = 'past'
        ifTrue: [proximityFeature at:'past' put: (each at: 3) ].

     (each at: 2) = 'in'
        ifTrue: [proximityFeature at:'in' put: (each at: 3) ].

     (each at: 2) = 'out'
        ifTrue: [proximityFeature at:'out' put: (each at: 3) ].

     (each at: 2) = 'right'
        ifTrue: [proximityFeature at:'right' put: (each at: 3) ].

     (each at: 2) = 'left'
        ifTrue: [proximityFeature at:'left' put: (each at: 3) ].

     (each at: 2) = 'top'
        ifTrue: [proximityFeature at:'top' put: (each at: 3) ].

     (each at: 2) = 'bottom'
        ifTrue: [proximityFeature at:'bottom' put: (each at: 3) ].

     (each at: 2) = 'atAngle'
        ifTrue: [proximityFeature at:'atAngle' put: (each at: 3) ]].
```

```
class:    a Feature
method: readData: aFeatureData

      "Assigns a feature data from aFeatureData collection"

   self   getinstanceName:          aFeatureData;
          attachToComponent:        aFeatureData;
            getChildName:           aFeatureData;
            getlibraryName:         aFeatureData;
            getApplication:         aFeatureData;
            getRepresentation:      aFeatureData;
            getOperation:           aFeatureData;
            getWorkpiece:           aFeatureData;
            getOrientation:         aFeatureData;
            getOrigin:              aFeatureData;
            getdatumPlane:          aFeatureData;
            getRawMatlCondition:    aFeatureData;
            getSurfFinish:          aFeatureData;
            getMcNotMc:             aFeatureData;
            getProximityAttributes: aFeatureData.
  self transformOrigin

class:    HoleBlind
method: initialize: aFeatureData
      "initialize imported feature instance within the CAPP system. Other feature classes
       will have similar method"
       |instance newName|

newName := (aFeatureData at:'instanceName') asSymbol.
(Smalltalk includesKey: newName)
      ifTrue: [^self error: newName, ' already exists'].
      instance := HoleBlind new.
      instance initialize.
      Smalltalk at: newName asSymbol put: instance.

   instance readData: aFeatureData;
         getHoleDiameter: aFeatureData;
         getHoleDepth: aFeatureData.
   Test changed: #components
```

3. Development of Feature topology and geometry environment

These methods have been implemented to define feature geometry and topology in the CAPP system.

3.1 Topology and geometry

3.1.1 Loop

```
Object variableSubclass: #Loop
 instanceVariableNames:
   'eLoop '
 classVariableNames: ''
 poolDictionaries: ''
```

```
methods for class Loop:

initialise
      "Intialise edge and vertex loops as ordered collection"

   eLoop := OrderedCollection new.

addEloop: edges
      "adds an edge loop to  a Loop"
       eLoop add: edges

getLoop: loopNumber
      "Returns the loop as OrderedCoollection of edges.
      1:TOP, 2:FRONT, 3:RIGHT, etc"
|
   ^(eLoop at: loopNumber)


transformVert: edgeLoop from: aDatum to: newDatum
      "transforms the edge vertices of a loop as per newDatum"
   |oldEdge oldVerts newEdge newVerts v1 v2 org radius transformedLoop |

   transformedLoop := OrderedCollection new.
   edgeLoop do:[:oldEdge | oldVerts := oldEdge vertices.
               ((oldEdge curve) isKindOf: Line)
               ifTrue:[v1 := (oldVerts at:1) transform: aDatum to: newDatum.
                      v2 := (oldVerts at:2) transform: aDatum to: newDatum.
                 newEdge := Edge new linInitialise: v1 endPt: v2.
               transformedLoop add: newEdge].
               ((oldEdge curve) isKindOf: Arc)
               ifTrue:[v1 := (oldVerts at:1) transform: aDatum to: newDatum.
                      v2 := (oldVerts at:2) transform: aDatum to: newDatum.
                     org := ((oldEdge curve) orgPoint) transform: aDatum to: newDatum.
                    radius:= (oldEdge curve) radius.
                 newEdge := Edge new arcInitialise: org radius: radius startPt: v1 endPt: v2.
                      transformedLoop add: newEdge]].
   ^transformedLoop
```

3.1.2 Edge

```
Object variableSubclass: #Edge
 instanceVariableNames:
   'startPoint endPoint curve '
 classVariableNames: ''
 poolDictionaries: ''

Object variableSubclass: #Line
 instanceVariableNames:
   'startPoint endPoint '
 classVariableNames: ''
 poolDictionaries: ''

Object variableSubclass: #Arc
 instanceVariableNames:
   'origin radius startPoint endPoint '
 classVariableNames: ''
 poolDictionaries: ''
```

Methods with class Line:

```
initialise: startPt endPt:endPt
"Initialise Line instance"
startPoint := startPt.
endPoint := endPt

sPoint
"returns the start point value"
^startPoint

ePoint
"returns the end point value"
^endPoint
```

Methods with class Arc

```
initialise: org radius: rad startPt: startPt endPt: endPt
"Initialise Arc"
 origin     := org.
 radius     := rad.
 startPoint := startPt.
 endPoint   := endPt

sPoint
"returns the start point value"|
 ^startPoint

ePoint
"returns the end point value"
 ^endPoint

orgPoint
"returns the origin point value"|
 ^origin

radius
"returns the radius value"
  ^radius
```

Methods with the class Edge:

```
linInitialise: startPt endPt: endPt
"Initialise line type edge instance"

 startPoint := startPt.
 endPoint  := endPt.
 curve     := Line new initialise: startPt endPt: endPt

arcInitialise: origin radius: rad startPt: startPt endPt: endPt
"Initialise arc type Edge instance"

 startPoint := startPt.
 endPoint := endPt.
 curve := Arc new initialise: origin radius: rad startPt: startPt endPt:endPt
```

3.2 Edge intersection methods

```
class:    an Edge
method: intersect: anEdge
      "This method computes intersection betweeen two planar edges
       (for curves of line or arc types) and returns the point of
       intersection/s or nil"

       |intersection|

      "The following is relevant for the intersection between
       two lines"

       (((self curve) isKindOf: Line) and:[(anEdge curve) isKindOf: Line])
       ifTrue:[intersection := self linIntersect: anEdge.
              ^intersection].

       "The following is relevant for the intersection between
       line and arc"

       ((self curve) isKindOf: Line)
         ifTrue:[ ((anEdge curve) isKindOf: Arc)
                  ifTrue:[intersection := self linArcIntersect: anEdge.
                         ^intersection]]

        ifFalse:[ ((anEdge curve) isKindOf: Line)
                  ifTrue:[intersection := anEdge linArcIntersect: self.
                         ^intersection]].

        "The following is relevant for the intersection between
        two arcs"

         (((self curve) isKindOf: Arc) and:[ ((anEdge curve) isKindOf: Arc)])

          ifTrue:[intersection := self arcIntersect: anEdge.
              ^intersection]

class:    an Edge
method: linIntersect: anEdge
      "This method computes intersection betweeen two planar
       Line edges and returns the point of intersection or nil"


   | a b c d t0 u0 denom intType intX intY intZ intCoord intPoint int|
       intType := nil.
       a := self startPoint.
       b := self endPoint.
       c := anEdge startPoint.
       d := anEdge endPoint.
       int := OrderedCollection new.


       " Compute denominator of intersection equation"
       denom := ((b x - a x) * (d y - c y)) - ((b y - a y) * (d x-c x)).
      (denom ~= 0)
```

```
        ifTrue:[t0 := (((c x-a x) * (d y-c y)) - ((c y-a y) * (d x-c x))) / denom.
              ((d x-c x) = 0)
               ifTrue:[u0 := ((a y - c y) + ((b y - a y)*t0))/(d y - c y)]
              ifFalse:[u0 := ((a x-c x) + ((b x - a x)*t0)) / (d x-c x)].
                 (t0 >= 0 and:[ t0 <= 1.0])
                  ifTrue:[(u0 >= 0 and:[u0 <= 1.0])
                  ifTrue:["Compute intersection point"
                        intX := a x + ((b x - a x) * t0).
                        intY := a y + ((b y - a y) * t0).
                        intZ := a z. " or b z or c z or d z"
                        intCoord := OrderedCollection new.
                        intCoord add: intX;
                               add: intY;
                               add: intZ.
                        intPoint := Vertex new initialise: intCoord.
                        int add:intPoint.
                        ]]].
           int isEmpty
           ifTrue:[^nil]
           ifFalse:[^int]
```

class: an Edge
method: linArcIntersect: anEdge

```
      "This method computes intersection betweeen line and arc type
       planar edges and returns the point/s of intersection or nil"

    |s b c r h w aa bb cc discrim  t01 t02 denom intx1 intx2 inty1 inty2
     intZ intCoord intPoint tempCoord intersections orgx orgy ex1 ex2 ey1 ey2
     edget0 edget1 int1t int2t tx0 ty0 tx1 ty1 intxt intyt t0 t1 t validInt
     p1 p2 finalInt|

     s   := self startPoint.
     b   := self endPoint.
     r   := (anEdge curve) radius.
     h   := (anEdge curve) orgPoint.
   orgx  := ((anEdge curve) orgPoint) x.
   orgy  := ((anEdge curve) orgPoint) y.
    ex1  := anEdge startPoint x.
    ey1  := anEdge startPoint y.
    ex2  := anEdge endPoint x.
    ey2  := anEdge endPoint y.
   intZ  := (self curve) sPoint z.
   intersections := OrderedCollection new.
       validInt := OrderedCollection new.
       finalInt := OrderedCollection new.

     "c = b-s"
     tempCoord := OrderedCollection new.
     tempCoord add: (b x - s x);
             add: (b y - s y);
             add: intZ.
     c := Vertex new initialise: tempCoord.

     "w = s-h"
     tempCoord := OrderedCollection new.
     tempCoord add: (s x - h x);
```

```
        add: (s y - h y);
        add: intZ.
w := Vertex new initialise: tempCoord.

"Compute aa = |c|squared "
aa := (c x * c x) + (c y * c y).

"Compute bb = w.c"

bb := (w x * c x) + (w y * c y).

"Compute cc = |w| squared - r squared "

cc := ((w x * w x) + (w y * w y)) - (r * r).

aa = 0
   ifTrue:[^nil].

discrim := (bb * bb) - (aa * cc).
discrim < 0
   ifTrue:[^nil].

discrim = 0
   ifTrue:[ t01:=((-1)*bb)/aa.
         intx1 := s x + (c x * t01).
         inty1 := s y + (c y * t01).

         tempCoord := OrderedCollection new.
         tempCoord add: intx1;
                add: inty1;
                add: intZ.
         intersections add: (Vertex new initialise: tempCoord)].

discrim > 0
   ifTrue:[ t01 := (((-1)*bb) - discrim sqrt)/aa.
         t02 := (((-1)*bb) + discrim sqrt)/aa.

         intx1 := s x + (c x * t01).
         inty1 := s y + (c y * t01).
         intx2 := s x + (c x * t02).
         inty2 := s y + (c y * t02).

         tempCoord := OrderedCollection new.
         tempCoord add: intx1;
                add: inty1;
                add: intZ.
         intersections add: (Vertex new initialise: tempCoord).

         tempCoord := OrderedCollection new.
         tempCoord add: intx2;
                add: inty2;
                add: intZ.
         intersections add: (Vertex new initialise: tempCoord)].

intersections isEmpty
  ifTrue:[^nil]
  ifFalse:["Check intersection validity"
        t0 := anEdge getEdgePara: (anEdge startPoint).
```

```
            t1 := anEdge getEdgePara: (anEdge endPoint).

            intersections do:[:each |
                        t := anEdge getEdgePara: each.
                         (t0 <= t and:[t <= t1])
                        ifTrue:[validInt add: each]]].


            validInt isEmpty
            ifTrue:[^nil]
            ifFalse:["Check valid arcEdge intersections forlineEdge int."
                  validInt do:[:each | t := self getEdgePara: each.
                              (t >= 0 and:[ t <= 1.0])
                              ifTrue:[finalInt add:each]].
                  finalInt isEmpty
                  ifTrue:[^nil]
                  ifFalse:[^finalInt]
                  ]
```

class: an Edge
method: arcIntersect: anEdge

```
     "This method computes intersection betweeen two arc type
      planar edges and returns the points of intersection or nil

      x1y1   - self origin, x2y2   - anEdge origin.
      r1     - self radius, r2     - anEdge radius.
      x3y3 & x4y4 - self start & end points.
      x5y5 & x6y6 - anEdge self start & end points."

      |x0 x01 x02 x1 x2 x3 x4 x5 x6 y0 y01 y02 y1 y2 y3 y4 y5 y6 z0
      r1 r2 aa bb a b c t0 t01 t02 u0 descrim ySum xSum
      intX intY intZ intCoord intPoint intersections ip1Coord ip2Coord
      int1t int2t tx0 ty0 tx1 ty1 intxt intyt t0 t1 t validInt finalInt|

      x1 := ((self curve) orgPoint) x.
      y1 := ((self curve) orgPoint) y.
      r1 := (self curve) radius.

      x2 := ((anEdge curve) orgPoint) x.
      y2 := ((anEdge curve) orgPoint) y.
      r2 := (anEdge curve) radius.

      x3 := (self curve sPoint) x.
      y3 := (self curve sPoint) y.

      x4 := (self curve ePoint) x.
      y4 := (self curve ePoint) y.

      x5 := (anEdge curve sPoint) x.
      y5 := (anEdge curve sPoint) y.

      x6 := (anEdge curve ePoint) x.
      y6 := (anEdge curve ePoint) y.

      z0 := (self startPoint) z.
     intersections := OrderedCollection new.
        validInt := OrderedCollection new.
        finalInt := OrderedCollection new.
```

```
    "Check if two circles intersect"
     (((x2-x1) squared + (y2-y1) squared)) <= (r1+r2) squared
     ifFalse:[^nil]. "No intersection"
     ((y2-y1)=0 and:[(x2-x1)=0])
     ifTrue:[^nil].


    "Compute sum of y end points, x symmetric"
    ySum := y3+y4-y5-y6.

    "Compute sum of x end points, y symmetric"
    xSum := x5+x6-x3-x4.

    ySum = 0
      ifTrue:[
            x0 := ((x5*x6) - (x3*x4))/(x6+x5-x4-x3).
            (self startPoint x - self endPoint x) > 0
            ifTrue:[y0 := y1 + (((r1 * r1) - (x0-x1) squared) sqrt)]
           ifFalse:[y0 := y1 - (((r1 * r1) - (x0-x1) squared) sqrt)].

            intCoord := OrderedCollection new.
            intCoord add: x0;
                  add: y0;
                  add: z0.
            intPoint := Vertex new initialise: intCoord.
            intersections add: intPoint]

  ifFalse:[ xSum = 0
     ifTrue:[ y0 := ((x3*x4) + (y3*y4) - (x5*x6) - (y5*y6))/ySum.
             x0 := x1 + (((r1 * r1) - (y0-y1) squared) sqrt).

            x01 := x0.
            y01 := y0.
            x02 := x1 - (((r1 * r1) - (y0-y1) squared) sqrt).
            y02 := y0.

            ip1Coord := OrderedCollection new.
            ip1Coord add: x01;
                  add: y01;
                  add: z0.
            intersections add: (Vertex new initialise: ip1Coord).

            ip2Coord := OrderedCollection new.
            ip2Coord add: x02;
                  add: y02;
                  add: z0.
            intersections add: (Vertex new initialise: ip2Coord)]

      ifFalse:[

    "Compute Constants"
    aa := xSum/ySum.
    bb := (((x3*x4)+(y3*y4)-(x5*x6))-(y5*y6))/ySum .
     a := 1+(bb squared).
     b := 2*((aa*bb) - (bb*x1) - y1).
     c := (x1 squared) + (y1 squared) + (aa squared) - (2*aa*x1) - (r1 squared).
descrim := (b squared) - (4 * a * c).
descrim < 0
   ifTrue:[^nil].
descrim = 0
   ifTrue:[y0 := (-1*b) / (2*a).
          t0 := ((y0-y1)/r1) arcSin *(7/22).
          x0 := x1 + (r1 * ((22/7) * t0) cos).

         intCoord := OrderedCollection new.
         intCoord add: x0;
                add: y0;
                add: z0.
         intPoint := Vertex new initialise: intCoord.
         intersections add: intPoint].

descrim > 0
   ifTrue:[ y01 := ((-1*b) / (2*a)) - (descrim/(2*a)).
          y02 := ((-1*b) / (2*a)) + (descrim/(2*a)).
          t01 := ((y01-y1)/r1) arcSin /(2*(22/7)).
          t02 := ((y02-y1)/r1) arcSin /(2*(22/7)).
          x01 := x1 + (r1 * ((22/7) * t01) cos).
          x02 := x1 + (r1 * ((22/7) * t02) cos).

          ip1Coord := OrderedCollection new.
          ip1Coord add: x01;
                 add: y01;
                 add: z0.
          intersections add: (Vertex new initialise: ip1Coord).

          ip2Coord := OrderedCollection new.
          ip2Coord add: x02;
                 add: y02;
                  add: z0.
          intersections add: (Vertex new initialise: ip2Coord)]].
          ].

intersections isEmpty
 ifTrue:[^nil]
 ifFalse:["Check intersection validity"
         t0 := self getEdgePara: (self startPoint).
         t1 := self getEdgePara: (self endPoint).

          intersections do:[:each |
                      t := self getEdgePara: each.
                       (t0 <= t and:[t <= t1])
                      ifTrue:[validInt add: each]]].

          validInt isEmpty
          ifTrue:[^nil]
          ifFalse:["Check int validity for anEdge"
                 t0 := anEdge getEdgePara: (anEdge startPoint).
                 t1 := anEdge getEdgePara: (anEdge endPoint).

          validInt do:[:each |
                      t := anEdge getEdgePara: each.
                       (t0 <= t and:[t <= t1])
                      ifTrue:[finalInt add: each]]].
```

```
            finalInt
             isEmpty
            ifTrue:[^nil]
            ifFalse:[^ finalInt]


class:    an Edge
method: getEdgePara: edgeLoc
      "Returns the parameter of a point on the edge"
   |p1 p2 r org t dx string num|

   ((self curve) isKindOf: Line)
      ifTrue:[p1 := self startPoint.
            p2 := self endPoint.

            (p2 y - p1 y) = 0
            ifTrue:[t := (edgeLoc x - p1 x)/(p2 x - p1 x)]
           ifFalse:[t := (edgeLoc y - p1 y)/(p2 y - p1 y)].

            (p2 x - p1 x) = 0
           ifTrue:[t := (edgeLoc y - p1 y)/(p2 y - p1 y)]
           ifFalse:[t := (edgeLoc x - p1 x)/(p2 x - p1 x)].
             ^t].

    ((self curve) isKindOf: Arc)
    ifTrue:[
          p1 := self curve sPoint.
          p2 := self curve ePoint.
         org := (self curve) orgPoint.
           r := (self curve) radius.
          dx := (edgeLoc x - org x).
       string := (dx/r) printString.
          num := string asFloat.
           t := num arcCos / (2*(22/7)). "Between 0 to pi"

          "Check if arc from 0-180 or 180-360"
          (p1 x - p2 x) > 0
           ifTrue:[(edgeLoc y -  org y) < 0
                  ifTrue:[^(1.0-t)]
                  ifFalse:[^t]].

          (p1 x - p2 x) < 0
           ifTrue:[(edgeLoc y -  org y) > 0
                    ifTrue:[^t]
                    ifFalse:[^(1.0-t)]].
           ]


class:    an Edge
method: featureIntersect: aFeature datum: aDatum
      "returns the edge feature intersection as
       an orderedCollection of aFeature edge, int Vertex/Vertices; or nil"
   | loop int intersection |
   int  := OrderedCollection new.
   loop := aFeature getEloop: aDatum.

  "Check if self is the edge of the same loop"
  loop do:[:each | (((each curve isKindOf: Line) and:[(self curve isKindOf: Line)]) or:[
              ((each curve isKindOf: Arc) and:[(self curve isKindOf: Arc)])])
                  ifTrue:[
              ((each startPoint = self startPoint) and:[
               (each endPoint   = self endPoint)])
               ifTrue:[^nil].
               ((each startPoint = self endPoint) or:[
                (each  endPoint = self startPoint)])
                ifTrue:[^nil]]].

   loop do:[:each  | (intersection := self intersect: each) isNil
                ifFalse:[int add: each.
                        intersection do:[:i | int add: i]]].

    int isEmpty
      ifTrue:[^nil]
      ifFalse:[ ^int]

class:    an Edge
method:intNearStart: int1 or: int2
      "Returns the point near edge start point"
   |t0 t1 ti1 ti2 |
    t0 := self getEdgePara: (self startPoint).
    t1 := self getEdgePara: (self endPoint).
    ti1:= self getEdgePara: int1.
    ti2:= self getEdgePara: int2.
    (ti1-t0) < (ti2-t0)
    ifTrue:[^1]
    ifFalse:[^2]

class:     a Feature
method: getEloop: aDatum
      "Return the edge loop of a feature by transforming its vertices to
       suit aDatum"

   | defaultEdgeLoop selfDatumLoop|

    self datum = aDatum
      ifTrue:[^(edgeTopology getLoop: 1)].



class:    an Edge
method: insideFeature: aFeature datum: aDatum
      "Returns 1 if the receiver is completely immersed
       inside aFeature or nil"
   | loop v1 v2 org rad px py p aVert stAng endAng status|

  v1   := self startPoint.
  v2   := self endPoint.
  status := 0. "not Inside"

  loop := aFeature getEloop: aDatum.

  "Check if self is the edge of the same loop"
  loop do:[:each | (((each curve isKindOf: Line) and:[(self curve isKindOf: Line)]) or:[
             ((each curve isKindOf: Arc) and:[(self curve isKindOf: Arc)])])
```

```
              ifTrue:[
            ((each startPoint = self startPoint) and:[
             (each endPoint   = self endPoint)])
             ifTrue:[^nil].
             ((each startPoint = self endPoint) or:[
              (each  endPoint = self startPoint)])
              ifTrue:[^nil]]].


"Check if self is a Line edge"
(self curve isKindOf: Line)
    ifTrue:[
          (((loop at:1) curve) isKindOf: Arc)
             ifTrue:[((v1 insideCircularProf: loop)notNil and:[(v2 insideCircularProf: loop)notNil])
                    ifTrue:[^1]
                    ifFalse:[^nil]].
          (((loop at:1) curve) isKindOf: Line)
             ifTrue:[(((v1 insideRectangularProf: loop) notNil) and:[(v2 insideRectangularProf: loop)
                            notNil])
             ifTrue:[1]
             ifFalse:[^nil]].
          ].

 "Check if self is an Arc edge"
 (self curve isKindOf: Arc)
   ifTrue:[
          org  := self curve orgPoint.
          rad  := self curve radius.
          "Generate points on the edge and test for all points"

           (v1 x - org x) > 0 "First Edge 0 - 180"
            ifTrue:[stAng := 0.
                   endAng:= 22/7].
           (v1 y - org y) > 0
            ifTrue:[stAng := 0.
                   endAng:= 22/7].
           (v1 y - org y) < 0
            ifTrue:[stAng := 22/7.
                   endAng:= (2* (22/7))].
           (v1 x - org x) < 0
            ifTrue:[ stAng := 22/7.
                   endAng:= (2 * (22/7))].

          stAng to: endAng by: ((22/7) * 0.05) do:[:ang |
                          px := org x + (rad * ((ang) cos)).
                          py := org y + (rad * ((ang) sin)).
                          p  := OrderedCollection new.
                          p add: px;
                            add: py;
                            add: 0.
                        aVert := Vertex new initialise: p.


                        (((loop at:1) curve) isKindOf: Arc)
                           ifTrue:[(aVert insideCircularProf: loop)isNil
                                  ifTrue:[status := 1]]. "a point not inside ft"
                        (((loop at:1) curve) isKindOf: Line)
                           ifTrue:[(aVert insideRectangularProf: loop)isNil
                    ifTrue:[status :=1]]].
              status = 1
               ifTrue:[ ^nil]
               ifFalse:[^1]]
```

3.3 Angle between Edge tangents

```
class:    an Edge
method: getAng: tan1  endTan:  tan2
      "Returns the angle between two tangents"
   |uax uay ubx uby aDotb ang aCrossbz |
    uax := (tan1 at:1).
    uay := (tan1 at:2).
    ubx := (tan2 at:1).
    uby := (tan2 at:2).
    aDotb := (uax*ubx)+(uay*uby).
    aDotb = -1
    ifTrue:[ang:= 22.0/7.0.
          ^ang].

    ang   := aDotb arcCos.
     "Decide the turn of the edge, k = 0 0 1"
    aCrossbz := (uax*uby) - (uay*ubx).
    aCrossbz > 0
    ifTrue:[^ang]
    ifFalse:[^((22/7)+ang)]


class:    an Edge
method: getStartTan
      "Returns the tangent at the start point of an edge."
   | x0 y0 x1 y1 x2 y2 modA startTan ua ub wa wb aCrossbz|
   startTan := OrderedCollection new.

   (self curve isKindOf: Line)
   ifTrue:[
   x1 := self startPoint x.
   y1 := self startPoint y.

   x2 := self endPoint x.
   y2 := self endPoint y.

   modA := ((x2-x1)squared + (y2-y1)squared)sqrt.
   startTan add: ((x2-x1)/modA);
          add: ((y2-y1)/modA).

   ^startTan].

    (self curve isKindOf: Arc)
    ifTrue:[
         x0 := (self curve orgPoint) x.
         y0 := (self curve orgPoint) y.
         x1 := self startPoint x.
         y1 := self startPoint y.
       modA := ((x1-x0)squared + (y1-y0)squared)sqrt.
```

```
        ua := (x1-x0)/modA.
        ub := (y1-y0)/modA.
        wa := ub.
        wb := (-1 * ua).
      "w must have a left turn for a start tangent"
      aCrossbz := (ua*wb) - (ub*wa).
      aCrossbz < 0
      ifTrue:[wa := (-1 * wa).
            wb := (-1 * wb)].

      startTan add: wa;
             add: wb.
   ^startTan].
```

class: an Edge
method: getEndTan

```
      "Returns the tangent at the end point of an edge."
   | x0 y0 x1 y1 x2 y2 modA endTan ua ub wa wb aCrossbz|
   endTan := OrderedCollection new.

   (self curve isKindOf: Line)
   ifTrue:[
   x1 := self startPoint x.
   y1 := self startPoint y.

   x2 := self endPoint x.
   y2 := self endPoint y.

   modA := ((x1-x2)squared + (y1-y2)squared)sqrt.
    endTan add: ((x1-x2)/modA);
         add: ((y1-y2)/modA).

   ^endTan].

   (self curve isKindOf: Arc)
   ifTrue:[
        x0 := (self curve orgPoint) x.
        y0 := (self curve orgPoint) y.
        x2 := self endPoint x.
        y2 := self endPoint y.
      modA := ((x2-x0)squared + (y2-y0)squared)sqrt.
        ua := (x2-x0)/modA.
        ub := (y2-y0)/modA.
        wa := ub.
        wb := (-1 * ua).
      "w must always have a right turn for an end tangent"
      aCrossbz := (ua*wb) - (ub*wa).
      aCrossbz > 0
      ifTrue:[wa := (-1 * wa).
            wb := (-1 * wb)].

         endTan add: wa;
              add: wb.
   ^endTan].
```

3.4 Detecting concave or convex angle between profile edges

class: an Edge
method: angleConcaveEdge: aLoop

```
   "An angle is measured between the start and end tangents of an edge loop.
    If any of the angles is concave (i.e. <= 180 degrees), the method returns 1,
    otherwise nil."

    | concaveEdge index tanAng startTan endTan |


     concaveEdge := nil.
          index := 1.
    (aLoop size) timesRepeat:[  index = (aLoop size)
                             ifTrue:[endTan  := (aLoop at:index) getEndTan.
                                    startTan:= (aLoop at:1) getStartTan]
                            ifFalse:[endTan  := (aLoop at:index) getEndTan.
                                   startTan := (aLoop at:(index+1)) getStartTan].
                                     tanAng := ((aLoop at:index) getAng: startTan endTan: endTan).
                                     tanAng < 3.14
                                      ifTrue:[concaveEdge := 1].
                                     Test answer: 'Angle > ',tanAng printString.
                                     index := index + 1
                                    ].

     ^concaveEdge
```

3.5 Vertex class and methods

```
Object variableSubclass: #Vertex
  instanceVariableNames:
    'x y z '
  classVariableNames: ''
  poolDictionaries: ''
```

Methods with class Vertex:

initialise: aCoordinate
"Initialise new vertex instance for a given coordinates, aCoordinate"

```
   x := aCoordinate at: 1.
   y := aCoordinate at: 2.
   z := aCoordinate at: 3
```

coordinates
"returns the coordinates of a vertex"

```
   |coordinate|
   coordinate :=  OrderedCollection new.
   coordinate at: 1 put: x.
   coordinate at: 2 put: y.
   coordinate at: 3 put: z.
   ^coordinate
```

```
x
  "returns x value"
 ^x

y
  "returns y value"
 ^y

z
  "returns z value"
   ^z
```

3.6 A point (vertex) inside or outside of a feature profile

class: a Vertex
method: insideCircularProf: aCprofile

```
 "Checks if the vertex is inside a circle. Returns dotProduct if inside, or nil"
  | v1 v2 v3 aDotb|
     v1 := ((aCprofile at:1) curve) sPoint.
     v2 := ((aCprofile at:1) curve) ePoint.
     v3 := self.

     aDotb := ((v2 x - v3 x) * (v1 x - v3 x))+ ((v2 y - v3 y) * (v1 y - v3 y)).

     aDotb > 0
     ifTrue:[^nil]
     ifFalse:[^aDotb]
```

class: a Vertex
method:insideRectangularProf: aRprofile

```
"Checks if the vertex is inside a Rectangle. Returns 1 if inside or nil"
| v1 v2 v3 |

   v1 := ((aRprofile at:1) curve) sPoint.
   v2 := ((aRprofile at:2) curve) ePoint.
   v3 := self.
  ((v3 x >= v1 x) and:[v3 y >= v1 y])
      ifTrue:[ ((v3 x <= v2 x) and:[v3 y <= v2 y])
      ifTrue:[^1]].
   ^nil
```

class: a Vertex
method:insideFeature: aFeature **datum:** aDatum

```
"Returns 1 if the vertex is inside another feature or nil"

 | loop v1 v2 org rad px py p aVert stAng endAng status|

loop := aFeature getEloop: aDatum.
"Check if the vertex belongs to the same loop"

loop do:[:each | (self = each startPoint or:[self = each endPoint])
                 ifTrue:[^nil]].

   (((loop at:1) curve) isKindOf: Arc)
    ifTrue:[(self insideCircularProf: loop)notNil
                  ifTrue:[^1]
```

```
                ifFalse:[^nil]].
  (((loop at:1) curve) isKindOf: Line)
   ifTrue:[(self insideRectangularProf: loop)notNil
                ifTrue:[^1]
                ifFalse:[^nil]].
```

3.7 Transformation of vertices

class: a Vertex
method:transform: currentCpl **to:** newCpl

```
"transforms the Vertex coordinates from current
 coordinate system to the new coordinate system "
| i j k coord topCoord newCoord topVertex tranVertex|

                i := OrderedCollection new.
                j := OrderedCollection new.
                k := OrderedCollection new.
            coord := OrderedCollection new
         newCoord := OrderedCollection new.

    currentCpl = 'TOP'
        ifTrue:[topVertex := self.
            tranVertex := topVertex transformTopTo: newCpl.
            ^tranVertex]
        ifFalse:[
    "First convert coordinates into TOP"
    currentCpl = 'FRONT'
    ifTrue:[i add:1;
            add:0;
            add:0.
          j add:0;
            add:0;
            add:-1.
          k add:0;
            add:1;
            add:0].

    currentCpl = 'RIGHT'
    ifTrue:[i add:0;
            add:0;
            add:1.
          j add:1;
            add:0;
            add:0.
          k add:0;
            add:1;
            add:0].

    currentCpl = 'REAR'
    ifTrue:[i add:-1;
            add:0;
            add:0.
          j add:0;
            add:0;
            add:1.
          k add:0;
```

```
            add:1;
            add:0].

    currentCpl = 'LEFT'
    ifTrue:[i add:0;
            add:0;
            add:-1.
          j add:-1;
            add:0;
            add:0.
          k add:0;
            add:1;
            add:0].

    currentCpl = 'BOTTOM'
    ifTrue:[i add:1;
            add:0;
            add:0.
          j add:0;
            add:-1;
            add:0.
          k add:0;
            add:0;
            add:-1].

          coord   add: (self x);
                  add: (self y);
                  add: (self z).

          topCoord := OrderedCollection new.
          topCoord add: (i dot: coord);
                  add: (j dot: coord);
                  add: (k dot: coord).
          topVertex := Vertex new initialise: topCoord.
          tranVertex := topVertex transformTopTo: newCpl.

          ^tranVertex]
```

class: a Vertex
method:transformTopTo: newCpl

```
"transforms the Vertex  from TOP coordinate system to
 FRONT, REGHT, REAR, LEFT or BOTTOM coordinate system.
 Returns the transformed Vertex."
 | i j k coord newCoord tranVertex|

    i := OrderedCollection new.
    j := OrderedCollection new.
    k := OrderedCollection new.
        coord := OrderedCollection new.
    newCoord := OrderedCollection new.

    newCpl = 'TOP'
    ifTrue:[i add:1;
            add:0;
            add:0.
          j add:0;
            add:1;
            add:0.
        k add:0;
          add:0;
          add:1].

newCpl = 'FRONT'
ifTrue:[i add:1;
          add:0;
          add:0.
        j add:0;
          add:0;
          add:1.
        k add:0;
          add:-1;
          add:0].

newCpl = 'RIGHT'
ifTrue:[i add:0;
          add:1;
          add:0.
        j add:0;
          add:0;
          add:1.
        k add:1;
          add:0;
          add:0].

newCpl = 'REAR'
ifTrue:[i add:-1;
             add:0;
             add:0.
        j add:0;
          add:0;
          add:1.
        k add:0;
           add:1;
           add:0].

newCpl = 'LEFT'
ifTrue:[i add:0;
             add:-1;
             add:0.
        j add:0;
          add:0;
          add:1.
        k add:-1;
          add:0;
          add:0].

newCpl = 'BOTTOM'
ifTrue:[i add:1;
          add:0;
          add:0.
        j add:0;
          add:-1;
          add:0.
        k add:0;
          add:0;
          add:-1].
```

```
      coord  add: (self x);
             add: (self y);
             add: (self z).

     newCoord add: (i dot: coord);
               add: (j dot: coord);
               add: (k dot: coord).

     tranVertex := Vertex new initialise: newCoord.
     ^ tranVertex
```

4. Assignment of Edge topology attributes to Feature instances

class: a HoleBlind
method: initEdgeTopology

```
"Initialise edge topology for HoleBlind feature instance"
    | org botOrg p1 p2 p3 p4
     edgeLoop vTopOrg vBotOrg
     v1 v2 v3 v4
     e1 e2 e3 e4 e5 e6 |

   org := OrderedCollection new.
   org add: (self orgX);
      add: (self orgY);
      add: (self orgZ).

botOrg := OrderedCollection new.
   p1 := OrderedCollection new.
   p2 := OrderedCollection new.
   p3 := OrderedCollection new.
   p4 := OrderedCollection new.

          botOrg add: (org at:1);
                add: (org at:2);
                add: (org at:3) - self depth.

          p1 add: ((org at:1) + (self diameter/2));
             add: (org at:2);
             add: (org at:3).

          p2 add: ((org at:1) - (self diameter/2));
             add: (org at:2);
             add: (org at:3).

          p3 add: ((botOrg at:1) - (self diameter/2));
             add: (botOrg at:2);
             add: (botOrg at:3).

          p4 add: ((botOrg at:1) + (self diameter/2));
             add: (botOrg at:2);
             add: (botOrg at:3).

           "Define vertices"
          v1 := Vertex new initialise: p1.
          v2 := Vertex new initialise: p2.
```

```
        v3 := Vertex new initialise: p3.
        v4 := Vertex new initialise: p4.
        vTopOrg := Vertex new initialise: org.
        vBotOrg := Vertex new initialise: botOrg.
         "Define Edges"
          e1 :=Edge new arcInitialise: vTopOrg radius: (self diameter)/2 startPt: v1 endPt:v2.
          e2 :=Edge new arcInitialise: vTopOrg radius: (self diameter)/2 startPt: v2 endPt:v1.
          e3 :=Edge new arcInitialise: vBotOrg radius: (self diameter)/2 startPt: v4 endPt:v3.
          e4 :=Edge new arcInitialise: vBotOrg radius: (self diameter)/2 startPt: v3 endPt:v4.

          e5 :=Edge new linInitialise:v1 endPt:v4.
          e6 :=Edge new linInitialise:v2 endPt:v3.

    edgeTopology := Loop new initialise.

    "Hole Top Face"
    edgeLoop := OrderedCollection new.
    edgeLoop add:e1;
          add:e2.
    edgeTopology addEloop: edgeLoop.

    "Hole Front Face"
    edgeLoop := OrderedCollection new.
    edgeLoop add:e2;
          add:e6;
          add:e4;
          add:e5.
    edgeTopology addEloop: edgeLoop.

    "Hole Rear Face"
    edgeLoop := OrderedCollection new.
    edgeLoop add:e1;
          add:e5;
          add:e3;
          add:e6.
    edgeTopology addEloop: edgeLoop.

    "Hole Bottom Face"
    edgeLoop := OrderedCollection new.
    edgeLoop add:e3;
          add:e4.
    edgeTopology addEloop: edgeLoop.
```

class: a HoleThrough
method: initEdgeTopology

```
"Initialise edge topology for HoleThrough feature instance."
   | org botOrg p1 p2 p3 p4
    edgeLoop vTopOrg vBotOrg
    v1 v2 v3 v4
    e1 e2 e3 e4 e5 e6 |

   org := OrderedCollection new.
   org add: (self orgX);
      add: (self orgY);
      add: (self orgZ).

botOrg := OrderedCollection new.
   p1 := OrderedCollection new.
```

```
p2 := OrderedCollection new.
p3 := OrderedCollection new.
p4 := OrderedCollection new.

        botOrg add: (org at:1);
              add: (org at:2);
              add: (org at:3) - self depth.

        p1 add: ((org at:1) + (self diameter/2));
           add: (org at:2);
           add: (org at:3).

        p2 add: ((org at:1) - (self diameter/2));
           add: (org at:2);
           add: (org at:3).

        p3 add: ((botOrg at:1) - (self diameter/2));
           add: (botOrg at:2);
           add: (botOrg at:3).

        p4 add: ((botOrg at:1) + (self diameter/2));
           add: (botOrg at:2);
           add: (botOrg at:3).

         "Define vertices"
        v1 := Vertex new initialise: p1.
        v2 := Vertex new initialise: p2.
        v3 := Vertex new initialise: p3.
        v4 := Vertex new initialise: p4.
        vTopOrg := Vertex new initialise: org.
        vBotOrg := Vertex new initialise: botOrg.
         "Define Edges"
          e1 :=Edge new arcInitialise: vTopOrg radius: (self diameter)/2 startPt: v1 endPt:v2.
          e2 :=Edge new arcInitialise: vTopOrg radius: (self diameter)/2 startPt: v2 endPt:v1.
          e3 :=Edge new arcInitialise: vBotOrg radius: (self diameter)/2 startPt: v4 endPt:v3.
          e4 :=Edge new arcInitialise: vBotOrg radius: (self diameter)/2 startPt: v3 endPt:v4.

          e5 :=Edge new linInitialise:v1 endPt:v4.
          e6 :=Edge new linInitialise:v2 endPt:v3.

edgeTopology := Loop new initialise.

"Hole Top Face"
edgeLoop := OrderedCollection new.
edgeLoop add:e1;
        add:e2.
edgeTopology addEloop: edgeLoop.

"Hole Front Face"
edgeLoop := OrderedCollection new.
edgeLoop add:e2;
        add:e6;
        add:e4;
        add:e5.
edgeTopology addEloop: edgeLoop.

"Hole Rear Face"
edgeLoop := OrderedCollection new.
```

```
    edgeLoop add:e1;
            add:e5;
            add:e3;
            add:e6.
    edgeTopology addEloop: edgeLoop.

    "Hole Bottom Face"
    edgeLoop := OrderedCollection new.
    edgeLoop add:e3;
            add:e4.
    edgeTopology addEloop: edgeLoop.


class:    a PocketRectangular
method: initEdgeTopology
"Initialise edge topology for PocketRectangular feature instance."
   | org l w h p1 p2 p3 p4 p5 p6 p7 p8
    edgeLoop
    v1 v2 v3 v4 v5 v6 v7 v8
    e1 e2 e3 e4 e5 e6 e7 e8 e9 e10 e11 e12 |

  org := OrderedCollection new.
  org add: (self orgX);
     add: (self orgY);
     add: (self orgZ).
  l := self length.
  w := self width.
  h := self depth.

   p1 := OrderedCollection new.
   p2 := OrderedCollection new.
   p3 := OrderedCollection new.
   p4 := OrderedCollection new.
   p5 := OrderedCollection new.
   p6 := OrderedCollection new.
   p7 := OrderedCollection new.
   p8 := OrderedCollection new.
          "Calculate vertex points"
          p1 add: (org at:1);
            add: (org at:2);
            add: (org at:3)-h.

          p2 add: (org at:1)+l;
            add: (org at:2);
            add: (org at:3)-h.

          p3 add: (p2 at:1);
            add: ((p2 at:2)+w);
            add: (p2 at:3).

          p4 add: (p3 at:1)-l;
            add: (p3 at: 2);
            add: (p3 at:3).

          p5 add:(org at:1);
            add:(org at:2);
            add:(org at:3).
```

```
        p6 add:(p5 at:1)+l;
          add:(p5 at:2);
          add:(p5 at:3).

        p7 add: (p6 at:1);
          add: (p6 at:2)+w;
          add: (p6 at:3).

        p8 add: (p7 at:1)-l;
          add: (p7 at:2);
          add: (p7 at:3).

       "Define vertices"
       v1 := Vertex new initialise: p1.
       v2 := Vertex new initialise: p2.
       v3 := Vertex new initialise: p3.
       v4 := Vertex new initialise: p4.
       v5 := Vertex new initialise: p5.
       v6 := Vertex new initialise: p6.
       v7 := Vertex new initialise: p7.
       v8 := Vertex new initialise: p8.

       "Define Edges"
       e1 :=  Edge new linInitialise:v1 endPt:v2.
       e2 :=  Edge new linInitialise:v2 endPt:v3.
       e3 :=  Edge new linInitialise:v3 endPt:v4.
       e4 :=  Edge new linInitialise:v4 endPt:v1.

       e5 :=  Edge new linInitialise:v5 endPt:v6.
       e6 :=  Edge new linInitialise:v6 endPt:v7.
       e7 :=  Edge new linInitialise:v7 endPt:v8.
       e8 :=  Edge new linInitialise:v8 endPt:v5.

       e9  :=  Edge new linInitialise:v2 endPt:v6.
       e10 :=  Edge new linInitialise:v3 endPt:v7.
       e11 :=  Edge new linInitialise:v4 endPt:v8.
       e12 :=  Edge new linInitialise:v1 endPt:v5.

edgeTopology := Loop new initialise.

"Plate Top Face"
edgeLoop := OrderedCollection new.
edgeLoop add:e5;
        add:e6;
        add:e7;
        add:e8.
edgeTopology addEloop: edgeLoop.

"Plate Front Face"
edgeLoop := OrderedCollection new.
edgeLoop add:e1;
        add:e9;
        add:e5;
        add:e12.
edgeTopology addEloop: edgeLoop.

"Plate Right Face"
edgeLoop := OrderedCollection new.
```

```
   edgeLoop add:e2;
          add:e10;
          add:e6;
          add:e9.
   edgeTopology addEloop: edgeLoop.

   "Plate Rear Face"
   edgeLoop := OrderedCollection new.
   edgeLoop add:e3;
          add:e11;
          add:e7;
          add:e10.
   edgeTopology addEloop: edgeLoop.

   "Plate Left Face"
   edgeLoop := OrderedCollection new.
   edgeLoop add:e12;
          add:e8;
          add:e11;
          add:e4.
   edgeTopology addEloop: edgeLoop.

   "Plate Bottom Face"
   edgeLoop := OrderedCollection new.
   edgeLoop add:e4;
          add:e3;
          add:e2;
          add:e1.
   edgeTopology addEloop: edgeLoop
```

class: a Plate
method: initEdgeTopology

```
"Initialise edge topology for Plate feature instance."
   | org l w h p1 p2 p3 p4 p5 p6 p7 p8
    edgeLoop
    v1 v2 v3 v4 v5 v6 v7 v8
    e1 e2 e3 e4 e5 e6 e7 e8 e9 e10 e11 e12 |

   org := OrderedCollection new.
   org add: (self orgX);
      add: (self orgY);
      add: (self orgZ).

  l := self length.
  w := self width.
  h := self height.

   p1 := OrderedCollection new.
   p2 := OrderedCollection new.
   p3 := OrderedCollection new.
   p4 := OrderedCollection new.
   p5 := OrderedCollection new.
   p6 := OrderedCollection new.
   p7 := OrderedCollection new.
   p8 := OrderedCollection new.
          "Calculate vertex points"
          p1 add: (org at:1);
```

```
        add: (org at:2);
        add: (org at:3).

      p2 add: ((org at:1) + l);
        add: (org at:2);
        add: (org at:3).

      p3 add: (p2 at:1);
        add: ((p2 at:2)+w);
        add: (p2 at:3).

      p4 add: ((p3 at:1)-l);
        add: (p3 at: 2);
        add: (p3 at:3).

      p5 add:(org at:1);
        add:(org at:2);
        add:((org at:3)+h).

      p6 add:((p5 at:1)+l);
        add:(p5 at:2);
        add:(p5 at:3).

      p7 add: (p6 at:1);
        add: ((p6 at:2)+w);
        add: (p6 at:3).

      p8 add: ((p7 at:1)-l);
        add: (p7 at:2);
        add: (p7 at:3).

    "Define vertices"
    v1 := Vertex new initialise: p1.
    v2 := Vertex new initialise: p2.
    v3 := Vertex new initialise: p3.
    v4 := Vertex new initialise: p4.
    v5 := Vertex new initialise: p5.
    v6 := Vertex new initialise: p6.
    v7 := Vertex new initialise: p7.
    v8 := Vertex new initialise: p8.

    "Define Edges"
    e1 := Edge new linInitialise:v1 endPt:v2.
    e2 := Edge new linInitialise:v2 endPt:v3.
    e3 := Edge new linInitialise:v3 endPt:v4.
    e4 := Edge new linInitialise:v4 endPt:v1.

    e5 := Edge new linInitialise:v5 endPt:v6.
    e6 := Edge new linInitialise:v6 endPt:v7.
    e7 := Edge new linInitialise:v7 endPt:v8.
    e8 := Edge new linInitialise:v8 endPt:v5.

    e9  := Edge new linInitialise:v2 endPt:v6.
    e10 := Edge new linInitialise:v3 endPt:v7.
    e11 := Edge new linInitialise:v4 endPt:v8.
    e12 := Edge new linInitialise:v1 endPt:v5.

edgeTopology := Loop new initialise.
```

```
"Plate Top Face"
edgeLoop := OrderedCollection new.
edgeLoop add:e5;
        add:e6;
        add:e7;
        add:e8.
edgeTopology addEloop: edgeLoop.

"Plate Front Face"
edgeLoop := OrderedCollection new.
edgeLoop add:e1;
        add:e9;
        add:e5;
        add:e12.
edgeTopology addEloop: edgeLoop.

"Plate Right Face"
edgeLoop := OrderedCollection new.
edgeLoop add:e2;
        add:e10;
        add:e6;
        add:e9.
edgeTopology addEloop: edgeLoop.

"Plate Rear Face"
edgeLoop := OrderedCollection new.
edgeLoop add:e3;
        add:e11;
        add:e7;
        add:e10.
edgeTopology addEloop: edgeLoop.

"Plate Left Face"
edgeLoop := OrderedCollection new.
edgeLoop add:e12;
        add:e8;
        add:e11;
        add:e4.
edgeTopology addEloop: edgeLoop.

"Plate Bottom Face"
edgeLoop := OrderedCollection new.
edgeLoop add:e4;
        add:e3;
        add:e2;
        add:e1.
edgeTopology addEloop: edgeLoop
```

class: a BossCircular
method: initEdgeTopology

```
"Initialise edge topology for BossCircular feature instance."
   | org topOrg p1 p2 p3 p4
     edgeLoop vOrg vtopOrg
     v1 v2 v3 v4
     e1 e2 e3 e4 e5 e6 |
   org := OrderedCollection new.
   org add: (self orgX);
      add: (self orgY);
      add: (self orgZ).

topOrg := OrderedCollection new.
   p1 := OrderedCollection new.
   p2 := OrderedCollection new.
   p3 := OrderedCollection new.
   p4 := OrderedCollection new.

         topOrg add: (org at:1);
              add: (org at:2);
              add: (org at:3) + self height.

         p1 add: ((topOrg at:1) + (self diameter/2));
           add: (topOrg at:2);
           add: (topOrg at:3).

         p2 add: ((topOrg at:1) - (self diameter/2));
           add: (topOrg at:2);
           add: (topOrg at:3).

         p3 add: ((org at:1) - (self diameter/2));
           add: (org at:2);
           add: (org at:3).

         p4 add: ((org at:1) + (self diameter/2));
           add: (org at:2);
           add: (org at:3).

          "Define vertices"
        v1 := Vertex new initialise: p1.
        v2 := Vertex new initialise: p2.
        v3 := Vertex new initialise: p3.
        v4 := Vertex new initialise: p4.
     vOrg := Vertex new initialise: org.
   vtopOrg := Vertex new initialise: topOrg.

          "Define Edges"
          e1 :=Edge new arcInitialise: vtopOrg radius: (self diameter)/2 startPt: v1 endPt:v2.
          e2 :=Edge new arcInitialise: vtopOrg radius: (self diameter)/2 startPt: v2 endPt:v1.
          e3 :=Edge new arcInitialise: vOrg radius: (self diameter)/2 startPt: v4 endPt:v3.
          e4 :=Edge new arcInitialise: vOrg radius: (self diameter)/2 startPt: v3 endPt:v4.

          e5 :=Edge new linInitialise:v4 endPt:v1.
          e6 :=Edge new linInitialise:v3 endPt:v2.

   edgeTopology := Loop new initialise.

   "Boss Top Face"
   edgeLoop := OrderedCollection new.
   edgeLoop add:e1;
         add:e2.
   edgeTopology addEloop: edgeLoop.

   "Boss Front Face"
   edgeLoop := OrderedCollection new.
```

```
  edgeLoop add:e2;
        add:e6;
        add:e4;
        add:e5.
  edgeTopology addEloop: edgeLoop.

  "Boss Rear Face"
  edgeLoop := OrderedCollection new.
  edgeLoop add:e1;
        add:e5;
        add:e3;
        add:e6.
  edgeTopology addEloop: edgeLoop.

  "Boss Bottom Face"
  edgeLoop := OrderedCollection new.
  edgeLoop add:e3;
        add:e4.
  edgeTopology addEloop: edgeLoop.
```

class: a TabRectangular
method: initEdgeTopology

```
    "Initialise edge topology for TabRectangular feature instance."
  | org l w h p1 p2 p3 p4 p5 p6 p7 p8
   edgeLoop
   v1 v2 v3 v4 v5 v6 v7 v8
   e1 e2 e3 e4 e5 e6 e7 e8 e9 e10 e11 e12 |

  org := OrderedCollection new.
  org add: (self orgX);
     add: (self orgY);
     add: (self orgZ).

  l := self length.
  w := self width.
  h := self height.

  p1 := OrderedCollection new.
  p2 := OrderedCollection new.
  p3 := OrderedCollection new.
  p4 := OrderedCollection new.
  p5 := OrderedCollection new.
  p6 := OrderedCollection new.
  p7 := OrderedCollection new.
  p8 := OrderedCollection new.
        "Calculate vertex points"
        p1 add: (org at:1);
           add: (org at:2);
           add: (org at:3).

        p2 add: ((org at:1) + l);
           add: (org at:2);
           add: (org at:3).

        p3 add: (p2 at:1);
           add: ((p2 at:2)+w);
           add: (p2 at:3).
        p4 add: ((p3 at:1)-l);
          add: (p3 at: 2);
          add: (p3 at:3).

        p5 add:(org at:1);
          add:(org at:2);
          add:((org at:3)+h).

        p6 add:((p5 at:1)+l);
          add:(p5 at:2);
          add:(p5 at:3).

        p7 add: (p6 at:1);
          add: ((p6 at:2)+w);
          add: (p6 at:3).

        p8 add: ((p7 at:1)-l);
          add: (p7 at:2);
          add: (p7 at:3).

        "Define vertices"
        v1 := Vertex new initialise: p1.
        v2 := Vertex new initialise: p2.
        v3 := Vertex new initialise: p3.
        v4 := Vertex new initialise: p4.
        v5 := Vertex new initialise: p5.
        v6 := Vertex new initialise: p6.
        v7 := Vertex new initialise: p7.
        v8 := Vertex new initialise: p8.

        "Define Edges"
        e1 :=  Edge new linInitialise:v1 endPt:v2.
        e2 :=  Edge new linInitialise:v2 endPt:v3.
        e3 :=  Edge new linInitialise:v3 endPt:v4.
        e4 :=  Edge new linInitialise:v4 endPt:v1.

        e5 :=  Edge new linInitialise:v5 endPt:v6.
        e6 :=  Edge new linInitialise:v6 endPt:v7.
        e7 :=  Edge new linInitialise:v7 endPt:v8.
        e8 :=  Edge new linInitialise:v8 endPt:v5.

        e9  :=  Edge new linInitialise:v2 endPt:v6.
        e10 :=  Edge new linInitialise:v3 endPt:v7.
        e11 :=  Edge new linInitialise:v4 endPt:v8.
        e12 :=  Edge new linInitialise:v1 endPt:v5.

edgeTopology := Loop new initialise.

"Tab Top Face"
edgeLoop := OrderedCollection new.
edgeLoop add:e5;
        add:e6;
        add:e7;
        add:e8.
edgeTopology addEloop: edgeLoop.

"Tab Front Face"
```

```
 edgeLoop := OrderedCollection new.
 edgeLoop add:e1;
        add:e9;
        add:e5;
        add:e12.
 edgeTopology addEloop: edgeLoop.

 "Tab Right Face"
 edgeLoop := OrderedCollection new.
 edgeLoop add:e2;
        add:e10;
        add:e6;
        add:e9.
 edgeTopology addEloop: edgeLoop.

 "Tab Rear Face"
 edgeLoop := OrderedCollection new.
 edgeLoop add:e3;
        add:e11;
        add:e7;
        add:e10.
 edgeTopology addEloop: edgeLoop.

 "Tab Left Face"
 edgeLoop := OrderedCollection new.
 edgeLoop add:e12;
        add:e8;
        add:e11;
        add:e4.
 edgeTopology addEloop: edgeLoop.

 "Tab Bottom Face"
 edgeLoop := OrderedCollection new.
 edgeLoop add:e4;
        add:e3;
        add:e2;
        add:e1.
 edgeTopology addEloop: edgeLoop
```

class: a ProductCom
method: generateProcess

```
    "Generate the automated machining process plan for the selected Component."
  |addDatumId parentFeatures addFtrsSameDatum highestFt |

  (curProduct isMemberOf: Component)
     ifFalse: [self error: 'Select Component instance'].

  "1. Clear previously generated Auto process"
  curProduct resetAutoProcess.

  "1a. Initialise feature topology"
   curProduct withAllChildren do: [:each |

     ((each isKindOf: Plate) or:[(each isKindOf: TabRectangular)])
        ifTrue:[each initEdgeTopology].
      (each isKindOf: BossCircular)
        ifTrue:[each initEdgeTopology].
    ((each isKindOf: HoleBlind) or:[(each isKindOf: HoleThrough)])
       ifTrue:[each initEdgeTopology].
     (each isKindOf: PocketRectangular)
       ifTrue:[each initEdgeTopology]].

"2. Read the datum id of additive feature instances"
addDatumId := Set new.
curProduct withAllChildren do: [:each |
          (each isKindOf: AdditiveFeature)
             ifTrue:[addDatumId add: each datum]].

"3. Select same datum additive pearentFeature instances"
addFtrsSameDatum := OrderedCollection new.
addDatumId do: [:eachDatum |
               curProduct withAllChildren do: [:each |
               (each isKindOf: AdditiveFeature)
                   ifTrue:[(each datum = eachDatum)
                          ifTrue:[addFtrsSameDatum add: each]]].

"4. While addFtrsSameDatum is not empty,
 For each addFtrsSameDatum, generate a process"
[addFtrsSameDatum isEmpty]
   whileFalse:[
            highestFt := addFtrsSameDatum showHighZ.
            highestFt addMcProcess.
            addFtrsSameDatum remove: highestFt.
            ]]
```

5. Detecting Feature interaction

class: ¯a Feature
method: addMcProcess

```
      "generates a mc process for an additive parent feature
       and also requests a process for its subtractive child feature instances"

  |subChildren nextSubChildren allSubChildren aDatum datums
   subCluster|
     datums              := #('TOP' 'FRONT' 'RIGHT' 'BOTTOM' 'LEFT' 'RIGHT').
     nextSubChildren     := OrderedCollection new.
     subCluster          := OrderedCollection new.


     datums do:[:aDatum| self mfgMethod: aDatum.
                subChildren := self getSubChildrenSameDatum:aDatum.
              allSubChildren := self getAllSubChildrenSameDatum: aDatum.
              [allSubChildren isEmpty]
          whileFalse: [subChildren do:[:aFeature |
                               "Select a Cluster of features for side interaction"
                               (subCluster := aFeature sideInteraction) isNil
                                ifFalse:[aFeature mfgMethod: subCluster datum: aDatum.

                                       subCluster do:[:each1 | nextSubChildren addAll: (each1
                                                  getSubChildrenSameDatum:aDatum)].
                                       subChildren removeAll: subCluster.
                                       allSubChildren removeAll: subCluster]
```

```
                    ifTrue:[
                        aFeature mfgMethod.
                        (aFeature withAllChildren) isEmpty
                                ifFalse:[nextSubChildren addAll: (aFeature
                                        getSubChildrenSameDatum:aDatum)].
                                        subChildren remove: aFeature.
                                        allSubChildren remove: aFeature].
                        ].
            subChildren isEmpty
            ifTrue:[subChildren := nextSubChildren].
           ]]
```

5.1 Validation of the feature interaction

class: a Feature
method: mfgMethod: subCluster **datum:** aDatum
"Checks the validity of subtractive cluster and requests the EndMilling machining process."

```
 | cParent level proc autoStatus|

   "Find cluster parent"
    cParent := subCluster select:[:each | (each intParentFeature)notNil].
    cParent := cParent at:1.

   "Check the cluster side interacting hierarchy level"
    level := self sideHrchy: subCluster.

    level = 1
      ifTrue:[(self clusterValidity: subCluster) notNil

           ifTrue:["Generaete pocket machining process plan"

               (cParent mcNotmc at:1) = $Y
                ifTrue:[ proc := EndMilling clusterPocket: subCluster datum: aDatum.
                      proc isNil
                    ifFalse: [cParent addProcess: proc]]]]
```

class: a Feature
method: clusterValidity: subCluster
"Checks if the interaction attributes of various features within the cluster make it a valid cluster. Validation method enables control over the combination of features. Returns 1 for valid, otherwise nil"

```
| clParent clChildren status|

  clParent := subCluster at:1.
  clChildren := OrderedCollection new.

  clChildren addAll:subCluster.
  clChildren removeIndex: 1.
  status := 1.


  clChildren do:[:child | ((child on) notNil or:[(child past) notNil])
```

```
                          ifFalse:[(child atAngle) notNil
                                ifFalse:[status := nil]]}.
 ^status
```

class: a Feature
method: intChildFeature

```
"Returns self if self is side interacting child feature, or nil"
 | fAttribute |
  fAttribute := proximityFeature at: 'intFeature'.

   fAttribute = 'self'
     ifTrue:[^nil}.

   fAttribute notNil
    ifTrue:[^self]
    ifFalse:[^nil]
```

class: a Feature
method: intParentFeature

```
"Returns self if self is the root side interacting parent feature, or nil"
    (proximityFeature at:'intFeature') = 'self'
     ifTrue:[^self]
     ifFalse:[^nil]
```

class: a Feature
method: sideParent

```
"Returns the side interacting parent feature name"
   | sParentName |
    sParentName := proximityFeature at: 'intFeature'.
    sParentName = 'self'
      ifTrue:[^self]
      ifFalse:[ ^(Smalltalk at: (sParentName asSymbol))]
```

class: a Feature
method: sideInteraction

```
"This method checks if the feature causes side interaction with neighbouring features.
 Returns a complete interacting feature cluster as an OrderedCollection or nil"

 | cluster parentFeature selfDatumChildren|

   cluster              := OrderedCollection new.
   parentFeature        := self childOf.
   selfDatumChildren := parentFeature getAllSubChildrenSameDatum: (self datum).

   "Search for cluster side parent feature"
   selfDatumChildren do:[ :each| (each intParentFeature isNil)
                        ifFalse:[cluster add: each]].

   "Search for cluster side child features"

    selfDatumChildren do:[ :each| (each intChildFeature) notNil
                        ifTrue:[ cluster do:[ :aMember | aMember name = ((each intChildFeature)
                                                        sideParent) name
                                                              ifTrue:[cluster add: each]]]].

      cluster isEmpty
```

```
        ifTrue:[^nil]
        ifFalse:[^cluster]
```

class: a Feature
method: interaction

```
"Checks for intFeature attribute value. Returns nil, if nil"
 | fAttribute |
    fAttribute := proximityFeature at: 'intFeature'.

    fAttribute isNil
        ifTrue:[^nil]
        ifFalse:[^1]
```

6. Feature Mapping

class: EndMilling
method: mapClusterFeatures: subCluster **datum:** aDatum

```
        "Maps the feature cluster into machining features.
         For each machining feature the collection syntex is
         at:1  absolute depth
         at:2  reletive depth
         at:3  tool parameters
         at:4  features in a cluster"


    | ftClusterIntoMcFeatures depths depthRange minDepth aDepth index sameDpara
     proc depthFts tools maxToolDiaForDepth maxToolDiaForLowerDepth|


    ftClusterIntoMcFeatures := Dictionary new.
    proc := EndMilling new. "to access instance method for tools"

     "Collect varous depths of features"
     depths := ( subCluster collect:[:fT | fT depth]) asSet.
     depths := depths asOrderedCollection.

     "Rearrange depths in ascending order"
      depthRange := OrderedCollection new.
     [depths isEmpty]
     whileFalse:[minDepth := depths at:1.
             depths do:[ :each | each < minDepth
                           ifTrue:[minDepth := each]].
                              depthRange add: minDepth.
                              depths remove: minDepth].

      "Map machining features as per depth"
      index := 1.
      depthRange size timesRepeat:[ aDepth := depthRange at: index.
                 sameDpara  := OrderedCollection new.
                   depthFts := OrderedCollection new.
                 sameDpara add: aDepth. "Absolute depth"
                 (index - 1) = 0
                 ifTrue:[sameDpara add: aDepth] "Relative depth"
                 ifFalse:[sameDpara add: (aDepth - (depthRange at: (index - 1)))].
```

```
            subCluster do:[:each | each depth >= aDepth
                            ifTrue:[depthFts add: each]].

            tools := proc getSameDepthToolAndFtCluster: depthFts datum: aDatum.

            "Check higher depth features machining tool dia to that of lower depth.
             Ensure that the same feature instance at different depth is machined by the
             same tool dia"
             (index - 1) = 0
              ifTrue:[ maxToolDiaForLowerDepth := (tools at: (tools size))at:2]
             ifFalse:[ maxToolDiaForDepth      := (tools at: (tools size))at:2.
                    maxToolDiaForDepth > maxToolDiaForLowerDepth
                      ifTrue:[(tools at: (tools size))at:1 put:maxToolDiaForLowerDepth.
                            (tools at: (tools size))at:2 put:maxToolDiaForLowerDepth.
                            maxToolDiaForLowerDepth := maxToolDiaForDepth]].

            tools do:[:t | t add: aDepth].
            sameDpara add: tools.
            sameDpara add: depthFts.
            ftClusterIntoMcFeatures at: index put: sameDpara.
            index := index +1].

    ^ftClusterIntoMcFeatures
```

6.1 Cutting tool selection method

class: an EndMilling
method: getSameDepthToolAndFtCluster: mcFtgroup **datum:** aDatum

```
 "The same depth feature cluster needs to be grouped with a two tier
  strategy. Firstly, on the basis of tool parameter selection. The
  resultant group/s may then split into joint or disjoint clusters.
        This method analyses the above and forms the cluster elements
  which could be machined by the recommended tool parameters."

   |index  disJointClusters ftExclDepth cElements cElementParents
    toolDia maxDia minDia cDia toolDiaAndParea ftArea approxArea
    tempDiaAreaFtRange range diaAreaFtRange diaRange tempSet pocketTools
    areaFtCluster toolPara splitPercArea aCluster disjClusters pocketProfiles
    a b toolStat|

  "The following decides the tool parameters for operation. The pocket profiles
   are machined by endmill cutters. It is assumed that the maximum tool diameter
   should be 50 mm. The cutter depth is decided by the max depth of the feature
   within a cluster."


   "1. Select the range of tool diameters for individual features"
      tempSet := Set new.

   1 to: mcFtgroup size do:[:n | ((mcFtgroup at:n) isKindOf: Pocket)
                         ifTrue:[toolDia := self getToolDiaPolygonProf: (mcFtgroup at:n) ].

                        (((mcFtgroup at:n)  isKindOf: HoleBlind) or:[((mcFtgroup at:n)  isKindOf:
                            HoleThrough)])
```

```
                    ifTrue:[toolDia := self getToolDiaRoundProf: (mcFtgroup at:n)].

                    tempSet add: toolDia.
                ].
    tempSet := tempSet asOrderedCollection.

"2. Sort out tool diameters in ascending order"

    diaRange := OrderedCollection new.
    [tempSet isEmpty]
        whileFalse:[ minDia := (tempSet at:1).
                tempSet do:[:dia | dia < minDia
                        ifTrue:[minDia := dia]].
                diaRange add: minDia.
                tempSet remove: minDia.
                    ].

"3. Compute approximate cluster area as the sum of the areas of individual features."

     approxArea := 0.
     1 to: mcFtgroup size do:[:n | ((mcFtgroup at:n) isKindOf: Pocket)
                ifTrue:[approxArea := approxArea+((mcFtgroup at:n) length * (mcFtgroup at:n)
                                        width)].
                (((mcFtgroup at:n) isKindOf: HoleBlind) or:[((mcFtgroup at:n) isKindOf:
                        HoleThrough)])
                ifTrue:[approxArea := approxArea+(((mcFtgroup at:n) diameter) squared *
                                                                        (3.14/4))].
                    ].

"4. For each tool in diaRange, compute the approx. machining area & percentage of
    machining area"

    diaAreaFtRange := OrderedCollection new.
    diaRange do:[:dia | ftArea := 0.
                areaFtCluster := OrderedCollection new.
        1 to: mcFtgroup size do:[:i | ((mcFtgroup at:i) isKindOf: Pocket)
                            ifTrue:[dia = (self getToolDiaPolygonProf:(mcFtgroup at:i))
                                ifTrue:[ftArea := ftArea + ((mcFtgroup at:i) length * (mcFtgroup
                                                at:i) width).
                                    areaFtCluster add: (mcFtgroup at:i)]].

                        (((mcFtgroup at:i) isKindOf: HoleBlind) or:[((mcFtgroup at:i)
                            isKindOf: HoleThrough)])
                            ifTrue:[dia = (self getToolDiaRoundProf:(mcFtgroup at:i))
                                ifTrue:[ftArea := ftArea + (((mcFtgroup at:i) diameter) squared *
                                                                        (3.14/4)).
                                    areaFtCluster add: (mcFtgroup at:i)]].
                    ].

                        tempDiaAreaFtRange := OrderedCollection new.
                        tempDiaAreaFtRange add: dia;
                                add: ftArea;
                                add: (ftArea/approxArea);
                                add: areaFtCluster.
                          diaAreaFtRange add: tempDiaAreaFtRange.
                        Test answer: 'ToolDia       > ',dia printString.
                        Test answer: '% total McArea> ',(ftArea/approxArea) printString.
            ].
```

```
"5.The decision about selecting the number of tools for pocket finishing will
   depend upon diaRange variation and the percentage of total area to be
   machined by the tool. The following assumptions have been made in the
   tool selection procedure.
   (a) If the minimum tool diameter from the range is 20 or more, then it should be
      selected for finishing the entire cluster.
   (b) If the minimum tool diameter from the range is less than 10 and the difference,
      min and max is more than 20mm and the percentage of area to be machined by min
      tool is greater 25%, the next higher dia tool should also be recommended.
   (c) At the moment, the method recommends a maximum of two tools"

"Select tool as per the assumption (a)"

 pocketTools := OrderedCollection new.

 (diaAreaFtRange size = 1 or:[ ((diaAreaFtRange at:1) at:1) >= 20 ])
      ifTrue:[toolStat := 1].

(((diaAreaFtRange at:1) at:1) < 10 and:[
(((diaAreaFtRange at:1) at:1) - ((diaAreaFtRange at:2) at:1))abs < 20])
      ifTrue:[
            ((diaAreaFtRange at:2)at:3) < 0.25
            ifTrue:[toolStat := 1]].

 toolStat = 1
      ifTrue:[toolPara := OrderedCollection new.
            toolPara  add: ((diaAreaFtRange at:1)at:1).  "Tool Min Dia for Pocket m/c"
            toolPara  add: ((diaAreaFtRange at:1)at:1).  "Tool Dia = Min Dia for Pocket m/c"
            toolPara  add: mcFtgroup.                    "Mc Features"
            pocketTools add: toolPara.
            ]

      ifFalse:["Select the first tool for case (b)"
            toolPara := OrderedCollection new.
            toolPara  add: ((diaAreaFtRange at:1)at:1).  "Tool Min Dia for Pocket m/c"
            toolPara  add: ((diaAreaFtRange at:1)at:1).  "Tool Dia = Min Dia for Pocket m/c"
            toolPara  add: ((diaAreaFtRange at:1)at:4).  "Mc Features"
            pocketTools add: toolPara.

            "Select the second tool"
            aCluster := OrderedCollection new.
            toolPara := OrderedCollection new.
            2 to: diaAreaFtRange size do:[:i |
                              aCluster addAll: ((diaAreaFtRange at: i)at:4)].

            toolPara add: ((diaAreaFtRange at:1)at:1); "min dia tool"
                  add: ((diaAreaFtRange at:2)at:1); "tool dia"
                  add: aCluster.                "Features for higner dia"
                  pocketTools add: toolPara].


  "6. Check if the same depth feature aggrigation causes disjoint
     clusters. If yes, replace aCluster with disjoint Clusters"

  pocketTools do:[:each |
                  disjClusters := (self breakClusterElement:(each at:3)).
                  each removeIndex:3.
```

```
                    each add: disjClusters].

        "Get tool machining profile"
        (pocketTools size) = 1
            ifTrue:[pocketProfiles := OrderedCollection new.
                    ((pocketTools at:1)at:3) do:[:each | pocketProfiles add: (self addFtProf: each  datum:
                                                                                        aDatum)].
                    (pocketTools at:1) add: pocketProfiles.

                    ].
        (pocketTools size) = 2
            ifTrue:["Only disjClusters size = 1 case is considered at the moment"
                     a := ((pocketTools at:2) at:3) at:1.
                     b := ((pocketTools at:1) at:3) at:1.

                     pocketProfiles := OrderedCollection new.
                     pocketProfiles add: (self subFtProf: a from: b datum: aDatum).
                     (pocketTools at:1) add: pocketProfiles.

                     pocketProfiles := OrderedCollection new.
                     pocketProfiles add:(self subFtProf: b from: a datum: aDatum).
                     (pocketTools at:2) add: pocketProfiles.

                    ].

        ^pocketTools
```

class: an EndMilling
mehod: getToolDiaPolygonProf: aSubFeature

```
    "Returns a recommended tool diameter to machine a polygon
     profile (Rectangular at the moment)."
    | diaPara toolMaxDia |

    toolMaxDia := 50.
    aSubFeature length < aSubFeature width
        ifTrue:[diaPara := aSubFeature length]
       ifFalse:[diaPara := aSubFeature width].

    (diaPara >= 3.5 and:[diaPara <= 5])
        ifTrue:[toolMaxDia := 3].

    (diaPara >= 5 and:[diaPara <= 10])
        ifTrue:[toolMaxDia := 6].

    (diaPara >= 10 and:[diaPara <= 15])
        ifTrue:[toolMaxDia := 8].

    (diaPara >= 15 and:[diaPara <= 20])
        ifTrue:[toolMaxDia := 12].

    (diaPara >= 20 and:[diaPara <= 25])
        ifTrue:[toolMaxDia := 16].

    (diaPara >= 25 and:[diaPara <= 40])
        ifTrue:[toolMaxDia := 20].

    (diaPara >= 40 and:[diaPara <= 100])
        ifTrue:[toolMaxDia := 25].
```

```
  (diaPara >= 100 and:[diaPara <= 200])
     ifTrue:[toolMaxDia := 30].

   diaPara >= 200
     ifTrue:[toolMaxDia := 50].

  ^toolMaxDia
```

class: an EndMilling
method: getToolDiaRoundProf: aSubFeature

```
  "Returns a recommended tool diameter to machine a circular
   profile."
  | diaPara toolMaxDia |

   toolMaxDia := 50.
   diaPara := aSubFeature diameter.

  (diaPara >= 3.5 and:[diaPara <= 5])
     ifTrue:[(toolMaxDia := 3)].

  (diaPara >= 5.001 and:[diaPara <= 10])
     ifTrue:[(toolMaxDia := 6)].

  (diaPara >= 10.001 and:[diaPara <= 15])
     ifTrue:[(toolMaxDia := 8)].

  (diaPara >= 15.001 and:[diaPara <= 20])
     ifTrue:[(toolMaxDia := 12)].

  (diaPara >= 20.001 and:[diaPara <= 25])
     ifTrue:[(toolMaxDia := 16)].

  (diaPara >= 25.001 and:[diaPara <= 40])
     ifTrue:[(toolMaxDia := 20)].

  (diaPara >= 30.001 and:[diaPara <= 100])
     ifTrue:[(toolMaxDia := 25)].

  (diaPara >= 40.001 and:[diaPara <= 200])
     ifTrue:[(toolMaxDia := 30)].

  (diaPara >= 55.001 and:[diaPara <= 500])
     ifTrue:[(toolMaxDia := 50)].

  ^toolMaxDia
```

6.2 Detecting the same depth tool machining profiles

class: an EndMilling
method: breakClusterElement: mcFtgroup

```
   "This method analyses the same depth feature cluster and forms the cluster elements
    which may be joint or disjoint"

  |index  disJointClusters ftExclDepth cElements cElementParents |
     ftExclDepth := OrderedCollection new.
disJointClusters := Dictionary new.
          index  := 1.

1 to: mcFtgroup size do:[:each | ftExclDepth add: (mcFtgroup at:each)].
[ftExclDepth isEmpty]
 whileFalse:[
        cElements := OrderedCollection new.
        cElements add: (ftExclDepth at:1).
        ftExclDepth size >= 2
        ifTrue:[
        2 to: ftExclDepth size do:[:i | (((ftExclDepth at:i) sideParent == (cElements at:1))
                                         or:[((ftExclDepth at:i) == (cElements at:1) sideParent)])
                                         ifTrue:[cElements add:(ftExclDepth at:i)]]].
        disJointClusters at:index put: cElements.
        index := index +1.
        ftExclDepth removeAll: cElements].

        disJointClusters := disJointClusters asOrderedCollection.

 ^disJointClusters
```

6.3 Adding interacting feature profiles

class: an EndMilling
method: addFtProf: pElement **datum:** aDatum

```
 "Returns the interacting feature profile topology as a Union of feature profiles in aDatum plane"

  | clusterLoop int lostEdges allEdges doubleIntStat edgeStartPt edgeEndPt newEdge newEdges
    index sp ep intPts ival stat sInt eInt dx dy
    toolDia maxDia minDia cDia tlength startTan endTan tanAng toolDiaAndParea concaveEdge
    ftArea approxArea tempDiaAreaRange range diaAreaRange diaRange tempSet pocketTools
    mcTopoTool |

 lostEdges       := OrderedCollection new.
 clusterLoop     := OrderedCollection new. "for profile topology"
   allEdges      := OrderedCollection new.
  newEdges       := OrderedCollection new.  "Additional Edges due to multiple intersections"
doubleIntStat    := 0. "To check if the edge has double intersection with a Feature"

 pElement size = 1
    ifTrue:[ pElement do:[:eachF | (eachF getEloop:aDatum) do:[:eachEdge |clusterLoop add:
                (eachEdge reInitialise)]]]
    ifFalse:["Do the following"

 "1. Get all the edges of a cluster"
 pElement do:[:eachF | (eachF getEloop:aDatum) do:[:eachEdge |allEdges add: (eachEdge
                         reInitialise)]].

 "2. Compute lost edges in the union of pElements"
 allEdges do:[: eachEdge |
      1 to: pElement size do:[:i | stat := (eachEdge insideFeature: (pElement at:i) datum: aDatum).
                                       stat isNil
```

```
                                        ifFalse:[lostEdges add: eachEdge]]].

"3. Remove lostEdges form allEdges"
 Test answer: 'Lost edges no > ', lostEdges size printString.
 allEdges removeAll: lostEdges.

"4. Check if the start and end points of remaining edges are inside other features.
 If yes, modiify them to the feature intersection points."
 allEdges do:[:edge |
        1 to: pElement size do:[:i | ((edge startPoint) insideFeature: (pElement at:i)  datum:
                                        aDatum) notNil
                      ifTrue:["Compute intersection & modify edge end point"
                             (int := edge featureIntersect: (pElement at:i) datum: aDatum)
                                        notNil
                             ifTrue:[int size = 2
                                    ifTrue:[edge startPoint: (int at:2)].
                                    int size > 2
                                    ifTrue:[intPts := OrderedCollection new.
                                           int do:[:each |(each isKindOf: Edge)
                                              ifFalse:[intPts add: each]].
                                                  ival := edge intNearStart: (intPts at:1) or:
                                                         (intPts at:2).
                                                  ival = 1
                                                    ifTrue:[edge startPoint:(intPts at:1)]
                                                   ifFalse:[edge startPoint:(intPts at:2)]]]].

                      ((edge endPoint) insideFeature: (pElement at:i) datum: aDatum) notNil
                       ifTrue:["Compute intersection & modify edge end point"
                             (int := edge featureIntersect: (pElement at:i) datum: aDatum)
                                        notNil
                             ifTrue:[int size = 2
                                    ifTrue:[edge endPoint: (int at:2)].
                                    int size > 2
                                    ifTrue:[intPts := OrderedCollection new.
                                           int do:[:each |(each isKindOf: Edge)
                                              ifFalse:[intPts add: each]].
                                                  ival := edge intNearStart: (intPts at:1) or:
                                                         (intPts at:2).
                                                   ival = 1
                                                    ifTrue:[edge endPoint:(intPts at:2)]
                                                   ifFalse:[edge endPoint:(intPts at:1)]]]].

                 ]].

"5. Generate new edges if edge end points are outside a feature
 but the edge intersects it"
 allEdges do:[:edge |
        1 to: pElement size do:[:i |
                      (((edge startPoint) insideFeature: (pElement at:i)  datum: aDatum)
                             isNil and:[
                       ((edge endPoint) insideFeature: (pElement at:i) datum: aDatum)
                             isNil ])
                      ifTrue:[(int := edge featureIntersect: (pElement at:i) datum:
                                        aDatum)notNil
                             ifTrue:[ int size > 2
                                    ifTrue:[
                                           intPts := OrderedCollection new.
                                           int do:[:each |(each isKindOf: Edge)
                                        ifFalse:[intPts add: each]].
                                        ival := edge intNearStart: (intPts at:1) or: (intPts at:2).
                                        ival = 1
                                        ifTrue:[sInt := (intPts at:1).
                                        eInt := (intPts at:2)]
                                        ifFalse:[sInt := (intPts at:2).
                                        eInt := (intPts at:1)].

                            "Do the following if ints and edge endpts are diff"
                              (((edge startPoint x = sInt x) and:[
                              (edge startPoint y = sInt y)]) or:[
                              ((edge endPoint x = eInt x) and:[
                              (edge endPoint y = eInt y)])])

                              ifFalse:[
                                  newEdge       := edge reInitialise.
                                  edge endPoint:      sInt.
                                  newEdge startPoint: eInt.
                                  Test answer:'Oldedge StartPoint>', (edge startPoint) x
                                          printString,' ',(edge startPoint) y printString.
                                  Test answer:'Oldedge EndPoint  >', (edge endPoint) x
                                          printString,' ',  (edge endPoint) y printString.

                                  Test answer:'Newedge StartPoint>', (newEdge startPoint) x
                                          printString,' ',(newEdge startPoint) y printString.
                                  Test answer:'Newedge EndPoint  >', (newEdge endPoint) x
                                          printString,' ',  (newEdge endPoint) y printString.
                                  newEdges add: newEdge]]]].
                    ]].


newEdges isEmpty
ifFalse:[allEdges addAll: newEdges].
newEdges := OrderedCollection new.

"6. Print vertices in the cluster"
 Test  answer: ' ';
      answer: 'Cluster All Edges';
      answer: 'Size>>',(allEdges size) printString.
1 to: allEdges size do:[:i | Test answer: 'Start-',(allEdges at:i) startPoint x printString, ' ', (allEdges
                                        at:i) startPoint y printString.
                            Test answer: 'End  -',(allEdges at:i) endPoint x printString , ' ', (allEdges
                                        at:i) endPoint y printString].

"7. From allEdges check if any  edge still has single or double intersection
 with a Feature"
 allEdges do:[:edge |
        1 to: pElement size do:[:i | ((edge startPoint) insideFeature: (pElement at:i)  datum:
                                        aDatum) notNil
                            ifTrue:["Compute intersection & modify edge end point"
                                  (int := edge featureIntersect: (pElement at:i) datum: aDatum)
                                        notNil
                                  ifTrue:[int size = 2
                                        ifTrue:[edge startPoint: (int at:2)].
                                        int size > 2
                                        ifTrue:[intPts := OrderedCollection new.
                                             int do:[:each |(each isKindOf: Edge)
                                                ifFalse:[intPts add: each]].
                                             ival := edge intNearStart: (intPts at:1) or:
                                                    (intPts at:2).
                                             ival = 1
                                               ifTrue:[edge startPoint:(intPts at:1)]
                                              ifFalse:[edge startPoint:(intPts at:2)]]]].

                         ((edge endPoint) insideFeature: (pElement at:i) datum: aDatum) notNil
                          ifTrue:["Compute intersection & modify edge end point"
                                (int := edge featureIntersect: (pElement at:i) datum: aDatum)
                                       notNil
                                ifTrue:[int size = 2
                                     ifTrue:[edge endPoint: (int at:2)].
                                     int size > 2
                                     ifTrue:[intPts := OrderedCollection new.
                                          int do:[:each |(each isKindOf: Edge)
                                            ifFalse:[intPts add: each]].
                                                  ival := edge intNearStart: (intPts at:1) or:
                                                         (intPts at:2).
                                                  ival = 1
                                                   ifTrue:[edge endPoint:(intPts at:2)]
                                                  ifFalse:[edge endPoint:(intPts at:1)]]]].

       (int := edge featureIntersect: (pElement at:i) datum: aDatum)notNil
        ifTrue:[ int size > 2
              ifTrue:[
                   intPts := OrderedCollection new.
                   int do:[:each |(each isKindOf: Edge)
                  ifFalse:[intPts add: each]].
                        ival := edge intNearStart: (intPts at:1) or: (intPts at:2).
                        ival = 1
                        ifTrue:[sInt := (intPts at:1).
                              eInt := (intPts at:2)]
                       ifFalse:[sInt := (intPts at:2).
                              eInt := (intPts at:1)].

                       "Do the following if ints and edge endpts are diff"
                        (((edge startPoint x = sInt x) and:[
                        (edge startPoint y = sInt y)]) or:[
                        ((edge endPoint x = eInt x) and:[
                        (edge endPoint y = eInt y)])])

                        ifFalse:[ newEdge      := edge reInitialise.
                              edge endPoint:     sInt.
                              newEdge startPoint: eInt.
                              Test answer:'Oldedge StartPoint>', (edge startPoint) x
                                     printString,' ',(edge startPoint) y printString.
                              Test answer:'Oldedge EndPoint  >', (edge endPoint) x
                                     printString,' ', (edge endPoint) y printString.

                              Test answer:'Newedge StartPoint>', (newEdge startPoint) x
                                     printString,' ',(newEdge startPoint) y printString.
                              Test answer:'Newedge EndPoint  >', (newEdge endPoint) x
                                     printString,' ', (newEdge endPoint) y printString.
                              newEdges add: newEdge.
                              doubleIntStat := 1]]]]].


doubleIntStat = 1
```

```
    ifTrue:[allEdges addAll: newEdges].
    Test answer: 'doubleIntStat', doubleIntStat printString.

   "8. Reorder edges within the cluster"
    [allEdges isEmpty]
    whileFalse:[
    clusterLoop add: (allEdges at:1).
    allEdges remove: (allEdges at:1).
    index := 1.
    allEdges size timesRepeat:[
    allEdges do:[:edge | dx := ((clusterLoop at: index) endPoint) x -  (edge startPoint) x.
                         dy := ((clusterLoop at: index) endPoint) y -  (edge startPoint) y.
                         ((dx abs) <= 0.01 and:[(dy abs) <= 0.01])
                         ifTrue:[clusterLoop add: edge.
                                allEdges remove: edge.
                                index := index +1]]].
            ].

    "9. Print reordered vertices"
     Test  answer: ' ';
         answer: 'Reordered Edges';
         answer: 'Size>>',(clusterLoop size) printString.
   1 to: clusterLoop size do:[:i | Test answer: 'Start-',(clusterLoop at:i) startPoint x printString, ' ',
                                         (clusterLoop at:i) startPoint y printString.
                                   Test answer: 'End  -',(clusterLoop at:i) endPoint x printString , ' ',
                                         (clusterLoop at:i) endPoint y printString.
                     ].
                               ]. "For pElement > 1 end"

^clusterLoop
```

6.4 Subtracting interacting feature profiles

class: an EndMilling
method: subFtProfBak: profFt1 **from:** profFt2 **datum:** aDatum

```
   "Returns the profile topology which is a result of
    subtracting profFt1(set of features) from profFt2(set of features)
    feature topology"


   |prof1 prof2 prof3 allFts lostEdges int newEdges newEdge intEdge
    intPts ival sInt eInt intEdges clusterLoop index dx dy partialEdges
    immersedEdges temp sPt ePt intPts thruEdges|

  lostEdges            := OrderedCollection new.
  newEdges             := OrderedCollection new.
  immersedEdges        := OrderedCollection new.
  clusterLoop          := OrderedCollection new.
  partialEdges         := Set new.
  thruEdges            := OrderedCollection new.

 "1. Remove common features between profFt1 and profFt2 from profFt2"
    profFt1 do:[:ft1 |
           profFt2 do:[:ft2 | ft1 == ft2
                          ifTrue:[profFt2 remove: ft2]]].
```

```
"2. Get profiles of profFt1 and profFt2"
   prof1 := self addFtProf: profFt1 datum: aDatum.
   prof2 := self addFtProf: profFt2 datum: aDatum.

"3. Get additive profile of profFt1 profFt2."
   allFts := Set new.
   allFts  addAll: profFt1;
           addAll: profFt2.
   allFts := allFts asOrderedCollection.
   prof3 := self addFtProf: allFts datum: aDatum.

"4. Compare start and end points of prof2 and prof3.
    If both ends exist in either prof, the edge is part of subtraction topo.
    If one end exists in either prof, that edge of prof3 is a part of subtraction topo.
    If the edge in prof2 and prof3 belongs the same curve, then that edge is a part of subtraction
        topo. "
    prof2 do:[:edge |
           1 to: prof3 size do:[:i |
           ((((edge startPoint) x - ((prof3 at:i) startPoint) x) abs <= 0.01 and:[
            ((edge startPoint) y - ((prof3 at:i) startPoint) y) abs <= 0.01])
           or:[
           (((edge endPoint) x - ((prof3 at:i) endPoint) x) abs <= 0.01 and:[
            ((edge endPoint) y - ((prof3 at:i) endPoint) y) abs <= 0.01])])
               ifTrue:[newEdges add: (prof3 at:i)].

           ((((edge curve sPoint) x - ((prof3 at:i) curve sPoint) x) abs <= 0.01 and:[
            ((edge curve sPoint) y - ((prof3 at:i) curve sPoint) y) abs <= 0.01])
           and:[
           (((edge curve ePoint) x - ((prof3 at:i) curve ePoint) x) abs <= 0.01 and:[
            ((edge curve ePoint) y - ((prof3 at:i) curve ePoint) y) abs <= 0.01])])
               ifTrue:[newEdges add: (prof3 at:i)].
                 ]].


 "5. Get completely and partially immersed edges of prof1 in profFt2.
        Trim them and reverse their direction"

    prof1 do:[:edge |
           profFt2 do:[:aFt | (((edge startPoint) insideFeature: aFt datum: aDatum) notNil and:[
                          ((edge endPoint) insideFeature: aFt datum: aDatum) notNil])
                          ifTrue:[temp := (edge startPoint).
                                edge startPoint:(edge endPoint).
                                edge endPoint: temp.
                                immersedEdges add: edge]
                          ifFalse:[
                          ((edge startPoint) insideFeature: aFt datum: aDatum) notNil
                          ifTrue:[newEdges do:[:anEdge |
                                (int:= (edge intersect: anEdge)) notNil
                                ifTrue:[Test answer:'Partial int detected'.
                                      edge endPoint:(edge startPoint).
                                      edge startPoint: (int at:1)]].
                                      partialEdges add: edge].
                          ((edge endPoint) insideFeature: aFt datum: aDatum) notNil
                          ifTrue:[newEdges do:[:anEdge |
                                (int:= (edge intersect: anEdge)) notNil
                                ifTrue:[Test answer:'Partial int detected'.
                                      edge startPoint: (edge endPoint).
                                edge endPoint: (int at:1)]].
                                partialEdges add: edge]].
                  ]].
 partialEdges := partialEdges asOrderedCollection.


"6. Get immersed edges of profl in profFt2 and the ones whose start & endPoints
   are outside profFt2"
   profl do:[:edge |
         profFt2 do:[:aFt |
                   (((edge startPoint) insideFeature: aFt datum: aDatum) isNil and:[
                    ((edge endPoint) insideFeature: aFt datum: aDatum) isNil])
                    ifTrue:[
                         (int := edge featureIntersect:aFt datum: aDatum) notNil
                          ifTrue:[intPts := OrderedCollection new.
                                int do:[:each |(each isKindOf: Edge)
                                        ifFalse:[intPts add: each]].
                                     ival := edge intNearStart: (intPts at:1) or: (intPts at:2).
                                     ival = 1
                                     ifTrue:[sPt := (intPts at:2).
                                            ePt := (intPts at:1)]
                                    ifFalse:[sPt := (intPts at:1).
                                            ePt := (intPts at:2)].
                                     newEdge := edge reInitialise.
                                     newEdge startPoint: sPt;
                                            endPoint: ePt.
                                     thruEdges add: newEdge].

                              ]]].

 newEdges addAll: immersedEdges.
 newEdges addAll: partialEdges.
 newEdges addAll: thruEdges.


"7. Reorder newEdges"

[newEdges isEmpty]
whileFalse:[
clusterLoop add: (newEdges at:1).
newEdges remove: (newEdges at:1).
index := 1.
newEdges size timesRepeat:[
newEdges do:[:edge | dx := ((clusterLoop at: index) endPoint) x -  (edge startPoint) x.
                dy := ((clusterLoop at: index) endPoint) y -  (edge startPoint) y.
                ((dx abs) <= 0.01 and:[(dy abs) <= 0.01])
                ifTrue:[clusterLoop add: edge.
                      newEdges remove: edge.
                      index := index +1]]].
                 ].

"8. Print Vetiices of subtractive edges."

 Test  answer: ' ';
    answer: 'Subtractive edge vertices';
    answer: 'Size>>',(clusterLoop size) printString.
 1 to: clusterLoop size do:[:i | Test answer: 'Start-',(clusterLoop at:i) startPoint x printString, ' ',
                                (clusterLoop at:i) startPoint y printString.
```

```
1 to: clusterLoop size do:[:i | Test answer: 'Start-',(clusterLoop at:i) startPoint x printString, ' ',
                                    (clusterLoop at:i) startPoint y printString.
                    Test answer: 'End  -',(clusterLoop at:i) endPoint x printString , ' ', (clusterLoop
                                    at:i) endPoint  y printString.
          ].

^clusterLoop
```

7. Process Plan Generation for cluster pockets

class: EndMilling
method: clusterPocket: subCluster datum: aDatum

```
 "Generates cluster pocket machining process"
 | mcFeatures mcFtgroup index sameDproc aProcess finProcess |


     "1.Map Cluster Features for machining"
      mcFeatures := self mapClusterFeatures:subCluster datum: aDatum.

     "2.For each same depth mcFeature cluster, generate a pocket
      profile Finishing Process. The finishing process searches the roughing process."
      index := 1.
      mcFeatures size timesRepeat:[mcFtgroup := mcFeatures at:index.
                        sameDproc := (EndMilling new) cPocketFinishInitialise: mcFtgroup datum:
                                    aDatum.
                         index = 1
                          ifTrue:[aProcess := sameDproc].
                         index > 1
                          ifTrue:[aProcess parentOf: sameDproc.
                                sameDproc childOf: aProcess].
                         index := index + 1.
                         ].

     ^aProcess
```

class: an EndMilling
method: cPocketFinishInitialise: mcFtgroup datum: aDatum

```
     "Initialises finish pocket Milling operation for the same depth mapped  features"

  | stockForOpr stockLeft depth aFt finStk  index topologyAndTool
   aProc topology toolPara tools ftProfiles pocketProfiles cornerOp
   finishOp finshProc cornerStatus largestToolDiaOp|

       finishOp := OrderedCollection new. "to record pocket finishing operations"
         tools := mcFtgroup at:3.

       depth := mcFtgroup at:2. "Relative depth"
         aFt := (mcFtgroup at:4) at:1.

   stockForOpr := aFt polygonStockValue. "'StockType(solid)' ZStock(aFt depth)' 'SideStock(solid)'"
   stockForOpr at:2 put:depth.
   finStk := 0.4.
   stockLeft := OrderedCollection new.
```

```
stockLeft add: 0;
           add: 0;
           add: 0.
stockLeft at:1 put: (stockForOpr at:1);
           at:2 put: (stockForOpr at:2) - finStk;
           at:3 put: (stockForOpr at:3).

"Assign common attributes of a pocket operation"
 self oprType: 'Finish';
  preMcState: stockLeft;
      oprStock: finStk.
 self  assignDepthOfCut.

 self initialize: aFt. "For reading a datum to assign setUpNo"

  "Assign tool parameters and pocket profile/s to the operation"
  cornerStatus := 0.

 1 to: tools size do:[:i |
                 "Initialise operation instance of self with common attributes, except
                  tool and topology"

                  aProc := (self reInitialise) initialize: aFt.

                 "Assign tool and feature topology for the operation"

                  aProc    mcFeatureTopology: ((tools at:i) at:4);
                              toolFromToolpara:  (tools at:i);
                         cPocketMillCycleLocs: mcFtgroup datum: aDatum.

                  cornerStatus = 0
                  ifTrue:[ Test answer: 'Profile Concave Angle Test'.
                         "Check finishOp profile for corner finishing operation.
                          For each profile check the angle between tangents, if the
                          angle is concave, assume 1.5mm corner rad and initialise new operation."

                         (aProc mcFeatureTopology) do:[:each |  (aProc angleConcaveEdge: each)
                                                                                isNil
                                             ifFalse:[
                                                   cornerOp := (self reInitialise) initialize: aFt.
                                                   cornerOp getCornerFinishTool:(tools at:i).
                                                   cornerOp oprName: 'Corner Finishing'.
                                                   aProc parentOf: cornerOp.
                                                   cornerOp childOf: aProc.
                                                   cornerStatus := 1.
                                                   ]].
                         ].

                 finishOp add: aProc.
                 ].

"Check the highest same depth finishing tool dia from finishOp. Make it
 the operation parent node"

 (finishOp size) > 1
    ifTrue:[largestToolDiaOp := self getLargestToolDiaOp: finishOp.
           finishOp remove: largestToolDiaOp.
           finishOp do:[:each | each childOf: largestToolDiaOp.
```

```
                        largestToolDiaOp parentOf: each]]
        ifFalse:[largestToolDiaOp := finishOp at:1].

    "Check roughing operation requirements of a cluster in the context of
     recommended finishing operation/s."

    rProc := largestToolDiaOp cPocketRoughInitialise: mcFtgroup  datum: aDatum.

    "3. Assign roughing op as a child node of finishing"

            largestToolDiaOp parentOf: rProc.
            rProc childOf: largestToolDiaOp.

    ^largestToolDiaOp
```

```
class:    EndMilling
method: cPocketRoughInitialise: mcFtgroup  datum: aDatum

     "Generates the same depth cluster roughing operation in the context of
      recommended finishing operation."
   |rProc rStock disjointHoles dProc rTopology|

   rTopology := OrderedCollection new. "Pocket Roughing topology minus disjointHoles"

     "1. Check if the roughing operation is necessary"
      (self preMcState at:2) > 0
        ifFalse:[Test answer: (self oprName)printString, '> ','No Stock, Pocket Roughing not
                    required']

        ifTrue:["2. Check if there are any disjoint holes in the cluster"
              disjointHoles := self getDisjHoles:(mcFtgroup at:4). "The holes are also <= 50 mm dia"

              "3. Drill the disjoint holes"
                disjointHoles isEmpty
                     ifFalse:[dProc := SlotDrilling holeCluster: disjointHoles.
                             "Uppend the plungePt of dproc"
                              dProc plungePt: ((dProc positionPt at:1) at:3) - (mcFtgroup at:1)
                                             + 0.4. "Finish Stock"

                "3A.Compute the pocket roughing topology by subtracting disjointHoles"
                 (((mcFtgroup at: 3) at:1) at:3) do:[:eachGroup |
                            rTopology add: (self subFtProf: disjointHoles from: eachGroup
                            datum:  aDatum)]].

              "4. End mill the remaining area"

               disjointHoles isEmpty
                     ifTrue:[rProc := self cPocketRoughInitialise]
                     ifFalse:[rProc := self cPocketRoughInitialise.
                            rProc mcFeatureTopology: rTopology.
                            rProc parentOf: dProc.
                           dProc childOf: rProc].

               ^rProc]
```

```
class:     SlotDrilling
method: holeCluster: holes
        "Drills a flat bottom hole for a holePattern"
    | proc tool toolDia minDia tempSet diaRange holeTopology aDatum |
         tempSet := Set new.
        diaRange := OrderedCollection new.
            tool := OrderedCollection new.
          aDatum := (holes at:1) datum.

        "1. Get minimum toolDia for slot drilling.
            Assume tool dia = min Hole dia - 0.4"

         holes do:[:each | tempSet add: (each diameter printString) asFloat].
         tempSet := tempSet asOrderedCollection.

         [tempSet isEmpty]
               whileFalse:[ minDia := (tempSet at:1).
                          tempSet do:[:dia | dia < minDia
                                       ifTrue:[minDia := dia]].
                          diaRange add: minDia.
                          tempSet remove: minDia.
                          ].

         minDia :=  diaRange at:1.
         tool add: 'EndMill';
             add: (minDia - 0.8);
             add: (minDia - 0.8);
             add: (minDia - 0.8);
             add: (holes at:1) depth.
         tool := tool asArray.

        "2. Initialise slot drilling operation"
            proc:= SlotDrilling new.
            proc        initialize: (holes at:1).
                    proc oprType: 'Rough';
                       oprStock: (minDia - 0.4)/2;
                      toolForOpr: tool;
                  drillCycleLocs: holes.
          ^proc


class:    an EndMilling
method: cPocketRoughInitialise

        "This method initialises the same depth cluster roughing operation in the context of
         recommended finishing operation."
        |rProc rStock |

        "1. Check if the roughing operation is necessary"

          (self preMcState at:2) > 0
            ifFalse:[Test answer: (self oprName)printString, '> ','No Stock, Pocket Roughing not
                      required']
            ifTrue:["1. initialise a roughing"
                   rProc := self reInitialise.

                  "2. Assign relevant roughing attributes"
                   rStock := self preMcState at:2.
```

```
                (rProc preMcState) at:2 put: 0.
                rProc  oprType:'Rough';
                     oprStock: rStock;
                   zWorkPlane: (self zWorkPlane)+ (self oprStock).
                rProc assignDepthOfCut
                ^rProc].

class:    an EndMilling
method: reInitialise
      "Returns the instance of self with standard attributes"
   |opr  |
    opr := EndMilling new.
    opr parentInitialise.

    opr    oprStock:          self oprStock;
         processMode:         self processMode;
             setUpNo:         self setUpNo;
            oprType:          self oprType;
           oprName:           self oprName;
         preMcState:          self preMcState;
       reqdPosAccy:           self reqdPosAccy;
         toolForOpr:          self toolForOpr;
          mcForOpr:           self mcForOpr;
       depthOfCut:            self depthOfCut;
mcFeatureTopology:            self mcFeatureTopology;
          positionPt:         self positionPt;
         clearPlane:          self clearPlane;
     approachPlane:           self approachPlane;
       retractPlane:          self retractPlane;
           refPlane:          self refPlane;
       zWorkPlane:            self zWorkPlane.
   ^opr

class:    an EndMilling
method: getCornerFinishTool: aToolpara
"returns tool parameters for  Pocket corner finishing"
| toolValues toolHgt |

    "Format : ('CornerEndMill' 'MinDia' 'Dia' 'MaxDia' 'MinHgt')"
     toolHgt    := (aToolpara at:5) + 10. "10 mm clearance Assumed"

    toolValues := OrderedCollection new.
    toolValues add: 'EndMill';
            add: 2.0;
            add: 3.0;
            add: 3.0;
            add: toolHgt.

    toolForOpr := toolValues asArray
    toolFromToolpara: aToolpara

      "Assigns the tool for operation to machine a cluster pocket"
   | toolValues toolHgt |
   "Format: ('EndMill' 'MinDia' 'Dia' 'MaxDia' 'MinHgt')"

   toolHgt          := (aToolpara at:5)+10. "10 mm clearance Assumed"
   toolValues       := OrderedCollection new.
   toolValues add: 'EndMill';
```

```
            add: (aToolpara at:1);
            add: (aToolpara at:2);
            add: (aToolpara at:2);
            add: toolHgt.

    toolForOpr := toolValues asArray


class:    a Milling
method: cPocketMillCycleLocs: clusterFeatures datum: aDatum
"Returns the cluster pocketMill cycle machining planes"

        (aClusterParent origin at:1) -> Absolute depth of a feature from entry face
        (aClusterParent origin at:2) -> Relative depth of a feature within the cluster"

    | aClusterParent faceDepth  |

     positionPt := nil.
    "Select a feature from the cluster to decide pocketMill cycle machining planes"

     aClusterParent := nil.
    (clusterFeatures at:4) size = 1
       ifTrue:[aClusterParent := (clusterFeatures at:4) at:1]
       ifFalse:[(clusterFeatures at:4) do:[:each |
             (each intParentFeature)notNil
               ifTrue:[aClusterParent := each]]].
    aClusterParent isNil
    ifTrue:[aClusterParent := (clusterFeatures at:4) at:1].


    positionPt := OrderedCollection new.
    (aClusterParent isKindOf: Pocket)
       ifTrue:[
    positionPt      add:(aClusterParent origin at:1) + ((aClusterParent length)/2); "pocket centre"
                    add:(aClusterParent origin at:2) + ((aClusterParent width)/2);
                    add:(aClusterParent origin at:3)].

    ((aClusterParent isKindOf: HoleBlind) or:[(aClusterParent isKindOf: HoleThrough)])
       ifTrue:[
    positionPt      add:(aClusterParent origin at:1);
                    add:(aClusterParent origin at:2);
                    add:(aClusterParent origin at:3)].

    positionPt      := positionPt asArray.
    refPlane        := (positionPt at:3).
    refPlane        := refPlane - ((clusterFeatures at:1)-(clusterFeatures at:2)).
    approachPlane   := refPlane + 20.0.
    zWorkPlane      := refPlane - ((self preMcState at:2) + (self oprStock)).
    retractPlane    := approachPlane.
    clearPlane      := retractPlane + 50.0.
```

8. Process Plan Report Generation

class: a ProductCom
method: showComponentProcess

```
"Displays a Component machining process"

   (curProduct isKindOf: Component)
      ifTrue:[curProduct process isEmpty
              ifTrue:[ (curProduct withAllChildren isEmpty)
                     ifFalse:[curProduct printAutoProcess]
                     ]

             ifFalse:[curProduct process do: [:each |
                        (each isMemberOf: MfgProcess)
                          ifTrue:[Test answer: curProduct name, ' Machining Process Plan';
                                   answer: 'Batch Quantity : ',each batchQty printString;
                                   answer: 'SetUp No: ', each  setUpNo printString, ';  Fixture For SetUp:
                                                  ',each fixtureForOpr;
                                   answer: ' '.
                                 ]
                        ifFalse:[Test answer: 'Opr No: ',each ftOprNo printString, '; Opr Name: ',each
                                                  oprName.
                              each toolForOpr isNil ifFalse:[
                              Test answer: 'Tool For Opr: ',each toolClassName, '; Tool Dia: ',each
                                                  toolDiameter printString,'; Tool Min Hgt: ',each
                                                  toolMinHgt printString].
                               Test answer: ''.
                              ]]]]
```

class: a Component
method: printAutoProcess

```
"prints an auto process of a Component instance"
   |setUpNums addFtrs procSetUp subChildren allSubChildren nextSubChildren workMatlInst|
   setUpNums := #(1 2 3 4 5 6).
   nextSubChildren := OrderedCollection new.

   "1. Get all the additive features of a component"
   addFtrs := self showAllAddFeatures.
   addFtrs isEmpty
       ifTrue:[Test answer: self name, 'No additive features in the model!']
      ifFalse:[Test answer: 'Automated Process Plan';
                answer: 'Component Name: ',self name.

            (workMatlInst := Material getMatlInst: (self showComponentInst) material) isNil

               ifFalse:[Test answer: 'Component Material: ',workMatlInst name;
                        answer: 'Matl Description: ',workMatlInst description, ' ','Matl Condition:
                                  ',workMatlInst conditionDescr;
                        answer: 'Material Specifications: ',workMatlInst spec,' ','Material Hardness:
                                  ',workMatlInst hardness printString].
                      Test answer: ' '.
            setUpNums do:[:setUpNumber|
                    addFtrs do:[:addFtr |
                    (addFtr matchSetUp:setUpNumber)isNil
                            ifFalse:[Test answer: 'SetUp No: ', setUpNumber printString;
                                 answer: 'Fixture For SetUp: ', (self getFixture: setUpNumber) ;
                                  answer: ' '.
                      addFtr printFtProcess: setUpNumber.
                      subChildren := addFtr getSubChildrenSameDatum: (self getDatum:
                                      setUpNumber).
                      allSubChildren := addFtr getAllSubChildrenSameDatum: (self
                                      getDatum: setUpNumber).
                      [allSubChildren isEmpty]
                           whileFalse: [subChildren do:[:aFeature |
                                              aFeature printFtProcess: setUpNumber.
                                              (aFeature withAllChildren) isEmpty
                                                ifFalse:[nextSubChildren addAll: (aFeature
                            getSubChildrenSameDatum:(self getDatum: setUpNumber))].
                                         ].
                                  allSubChildren removeAll: subChildren.
                                  subChildren := nextSubChildren.
                              ]]]]]
```

```
class: a Feature
method: printFtProcess: setUpNum
"Prints the feature process/es in a setUpNo"
 |option intCluster|
    option := 1.
   (self process isEmpty)
      ifTrue:[^nil]
      ifFalse:[((self process at:1) setUpNo = setUpNum)
               ifFalse:[^nil]].

   (intCluster := self sideInteraction) isNil "Side interacting set of features"
   ifTrue:[
   (self process) size > 1
      ifTrue:[Test answer:(self process) size printString,' ','Options Exist to machine ',self name.
            self process do:[:each|
                         Test answer:'Option no: ',option printString.
                         each printProcess: self.
                         option := option + 1]]
     ifFalse:[Test answer: 'Machining Process for ',self name.
                     (self process at:1) printProcess: self]]

    ifFalse:[
           Test answer: (intCluster collect:[:i | i name]) asArray printString.
           Test answer: 'The above features form a complex pocket due to feature interaction'.
           Test answer: 'Machining process:'.
           (self process at:1) printClusterProcess:self.
          ]
```

```
class:    a mfgProcess
method: getClusterProcSeq
"returns the reordered collection of interacting feature cluster process
 operation sequence including self"
| opNo reorderOps roughOprs finishOprs |
       opNo := 1.
   reorderOps := OrderedCollection new.
   roughOprs  := OrderedCollection new.
   finishOprs := OrderedCollection new.

   self withAllChildren do:[:each |((each oprType = 'Finish') or:[ (each oprName = 'Corner Finish')])
                        ifTrue:[finishOprs add:each]
                        ifFalse:[roughOprs add:each]].

   finishOprs addFirst: self.
   reorderOps addAll: roughOprs;
           addAll: finishOprs.

   reorderOps do:[:each | each ftOprNo: opNo.
                       opNo := opNo + 1].
   ^reorderOps

class:     a MfgProcess
method: printClusterProcess: aFeature
 "prints the cluster process attributes on the text pane."
   |i |


        self getClusterProcSeq do: [:each |
                       Test answer: ' '.
                       Test answer: 'Opr No: ',each ftOprNo printString, '; Opr Name: ',(each
                             oprType),' ',(each oprName) printString.
                      (each precedsOpr) isNil
                        ifFalse:[each precedsOpr = 'Parent Finish'
                                 ifTrue:[Test answer: (each oprType),' ',(each oprName) printString,' ',
                                       'MUST PRECEDE ', (aFeature childOf) name, ' Finishing'].
                               each precedsOpr ='Parent Operation'
                                 ifTrue:[Test answer: (each oprType),' ',(each oprName) printString,' ',
                                       'MUST PRECEDE ', (aFeature childOf) name, ' Operation'].
                              ].
                      (each toolForOpr) isNil
                        ifFalse:[Test answer: 'Tool For Opr: ',(each toolForOpr) asArray printString].

                      (each speed) isNil
                        ifFalse:[Test answer: 'Recommended Cutting Speed: ',(each speed)
                                       printString,' ','Feed: ',(each feed) printString].
                      (each mcForOpr) isNil
                        ifFalse:[Test answer: 'Machine For Opr: ',(each mcForOpr) printString].

                      "Output Approach, Retract, Clear and Zwork plane information"
                           Test answer:'PositionPt   : ',each positionPt printString;
                                 answer:'ApproachPlane: ',each approachPlane printString;
                                 answer:'ZWorkPlane   : ',each zWorkPlane printString;
                                 answer:'RetractPlane : ',each retractPlane printString;
                                 answer:'ClearPlane   : ',each clearPlane printString;
                                 answer:''.

                      (each mcFeatureTopology) isNil
                        ifFalse:[Test answer: ' '.
                              Test answer: 'Edge profile for the operation:'.
                              (each mcFeatureTopology) do:[:aTopo |
                                          Test answer: (aTopo collect:[:edge |edge curve]) asArray
                                                          printString.
                                          Test answer: 'Edge profile details:'.
```

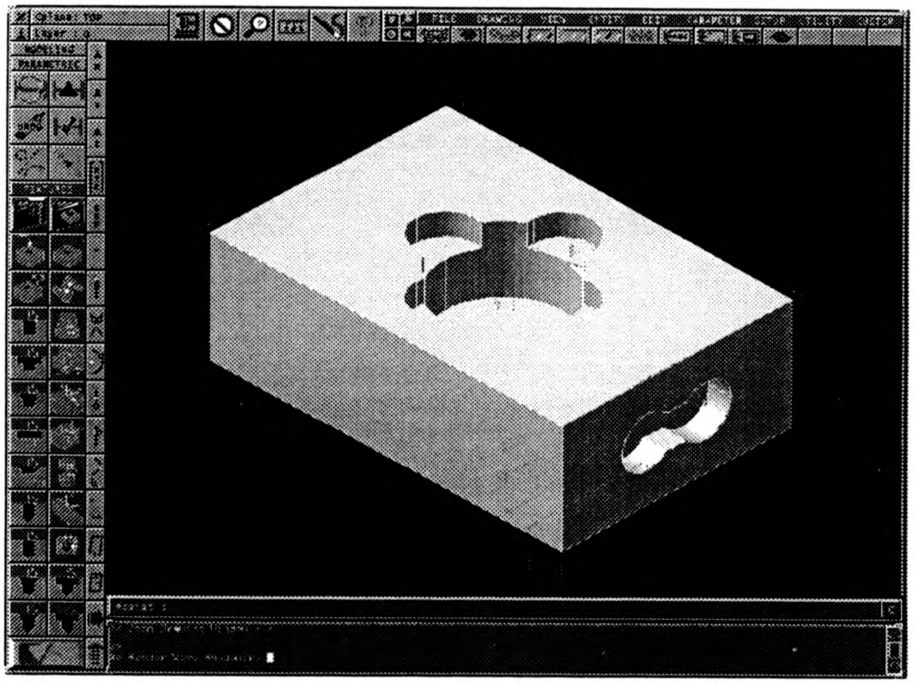
```
                                   1 to: aTopo size do:[ :i | edge := (aTopo at:i).

                                    Test answer: 'Edge ',i printString,': ',
                                                 (edge curve) printString.
                                    Test answer: 'Start  Point: ',((edge startPoint x)
                                                  printFraction:3),', ',((edge startPoint y)
                                                  printFraction:3),', ',((edge startPoint z)
                                                  printFraction:3).
                                    Test answer: 'End    Point: ',((edge endPoint) x
                                                  printFraction:3),', ',((edge endPoint) y
                                                  printFraction:3),', ',((edge endPoint) z
                                                  printFraction:3).
                                 (edge curve isKindOf: Arc)
                             ifTrue:[Test answer: 'Arc Origin  : ',((edge curve orgPoint) x
                                              printFraction:3),', ',((edge curve orgPoint) y
                                              printFraction:3),', ',((edge curve orgPoint) z
                                              printFraction:3).
                                    Test answer: 'Arc Radius  : ',((edge curve
                                                   radius)printFraction:3)].
                                    Test answer: ' ']]].
```

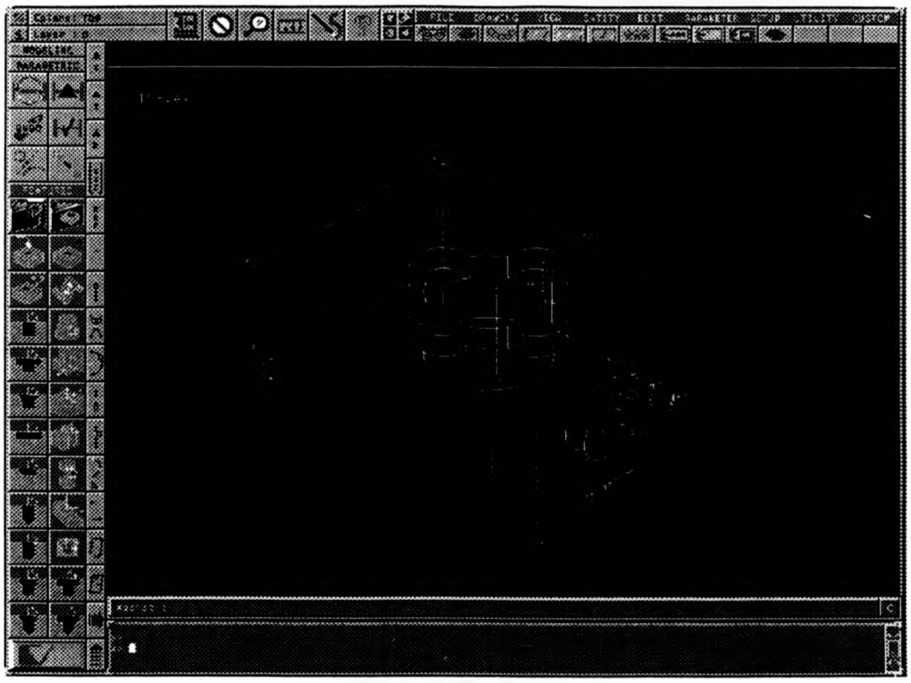
class: a MfgProcess
method: getClusterProcSeq

```
   "returns the reordered collection of interacting feature cluster process
    operation sequence"
   | opNo reorderOps drillOprs roughOprs finishOprs|
       opNo := 1.
   reorderOps := OrderedCollection new.
   drillOprs  := OrderedCollection new.
   roughOprs  := OrderedCollection new.
   finishOprs := OrderedCollection new.

   self withAllChildren do:[:each |((each oprType = 'Finish') or:[ (each oprName = 'Corner Finish')])
                          ifTrue:[finishOprs add: each]].
   finishOprs addFirst: self.

   self withAllChildren do:[:each | each oprType = 'Rough'
                          ifTrue:[(each isKindOf: SlotDrilling)
                                    ifTrue:[drillOprs add: each]
                                   ifFalse:[roughOprs add: each]]].

   drillOprs  := drillOprs  asArray.
   roughOprs  := roughOprs  asArray.
   finishOprs := finishOprs asArray.

   drillOprs isEmpty
      ifFalse:[reorderOps addAll: drillOprs].

   reorderOps addAll: roughOprs;
           addAll: finishOprs.

   reorderOps do:[:each | each ftOprNo: opNo.
                        opNo := opNo + 1].
   ^reorderOps
```

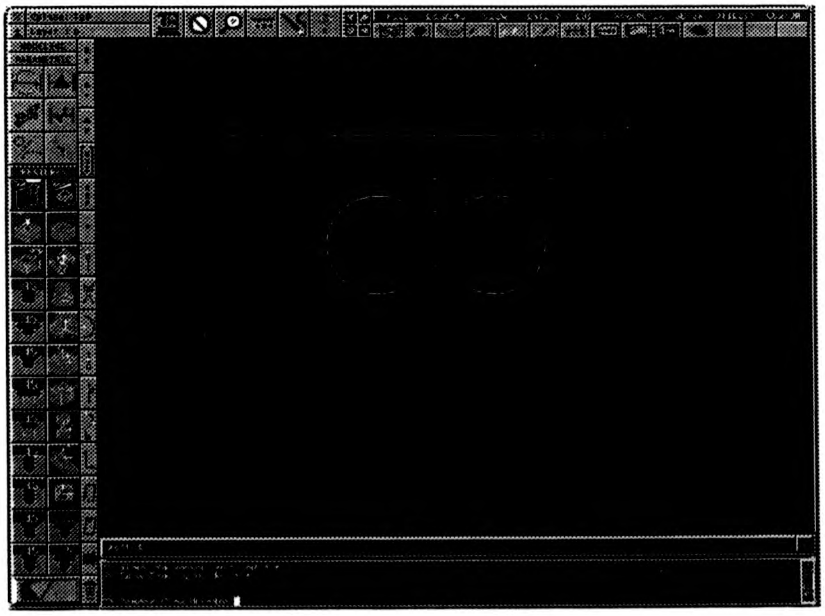
9. CAPP system Test Results for sample component

9.1 Feature Based Model of TestComp4


Rendered Image of TestComp4


ISO View of TestComp4

TOP View of TestComp4


RIGHT View of TestComp4

9.2 Feature Report file for TestComp4

```
*****     printing 1706 Pocket_Rectangular1703_I1      *****

class :
    class : Pocket_Rectangular                   library : CVOBJECTS
    classtype : Feature                          timestamp : 07-26-96
12:40
static attributes :
    BooleanOp  Operation                                  = Subtract
attributes :
    Workpiece  Workpiece                                  = 2801
    Name       Representation                             = rep
    CPlane     Orientation                             id = 15
                                                     name = RIGHT
                                                      i x = 0.000000e+00
                                                        y = 1.000000e+00
                                                        z = 0.000000e+00
                                                      j x = 0.000000e+00
                                                        y = 0.000000e+00
                                                        z = 1.000000e+00
                                                      k x = 1.000000e+00
                                                        y = 0.000000e+00
                                                        z = 0.000000e+00
    Location   Origin                                   x = 3.000000e+02
                                                        y = 7.500000e+01
                                                        z = 3.500000e+01

    ApplyType  Application                                = Extend
    String     rawMatlCondition                           = Solid
    Real       surfFinish                                 = 2.000000e+00
    String     mcNotmc                                    = Y
    String     atAngle                                    = nil
    String     bottom                                     = nil
    String     in                                         = nil
    String     intFeature                                 = self
    String     left                                       = nil
    String     on                                         = nil
    String     out                                        = nil
    String     past                                       = nil
    String     right                                      = nil
    String     to                                         = nil
    String     top                                        = nil
    Face       EntryFace                                  = 2545
    String     childOf                                    = Plate53_I1
    Length     PocketLength                               = 5.000000e+01 =
millimeters
    Length     PocketWidth                                = 3.000000e+01 =
millimeters
    Length     PocketDepth                                = 2.000000e+01 =
millimeters
    String     TolPocketLength                            = 0.1 / - 0.1
    String     TolPocketWidth                             = 0.1 / - 0.1
    String     TolPocketDepth                             = 0.1 / - 0.1

*****     printing 487 Hole_Blind484_I1      *****

class :
    class : Hole_Blind                           library : CVOBJECTS
    classtype : Feature                          timestamp : 07-26-96
12:42
```

```
static attributes :                                          = Subtract
    BooleanOp  Operation
attributes :                                                 = 2801
    Workpiece  Workpiece                                     = rep
    Name       Representation                             id = 6
    CPlane     Orientation                              name = TOP
                                                         i x = 1.000000e+00
                                                           y = 0.000000e+00
                                                           z = 0.000000e+00
                                                         j x = 0.000000e+00
                                                           y = 1.000000e+00
                                                           z = 0.000000e+00
                                                         k x = 0.000000e+00
                                                           y = 0.000000e+00
                                                           z = 1.000000e+00
    Location   Origin                                      x = 1.500000e+02
                                                           y = 1.000000e+02
                                                           z = 1.000000e+02
    ApplyType  Application                                   = Extend
    String     rawMatlCondition                              = Solid
    Real       surfFinish                                    = 2.000000e+00
    String     mcNotmc                                       = Y
    String     atAngle                                       = nil
    String     bottom                                        = nil
    String     in                                            = nil
    String     intFeature                                    = self
    String     left                                          = nil
    String     on                                            = nil
    String     out                                           = nil
    String     past                                          = nil
    String     right                                         = nil
    String     to                                            = nil
    String     top                                           = nil
    Face       EntryFace                                     = 2569
    String     childOf                                       = Plate53_I1
    Diameter   HoleDiameter                                  = 1.000000e+02 =
millimeters
    Length     Depth                                         = 6.000000e+01 =
millimeters
    String     TolHoleDiameter                               = 0.1 / - 0.1
    String     TolDepth                                      = 0.1 / - 0.1

*****      printing 594 Hole_Blind484_I2         *****

class :                                          library : CVOBJECTS
    class : Hole_Blind                           timestamp : 07-26-96
    classtype : Feature
12:42
static attributes :                                          = Subtract
    BooleanOp  Operation
attributes :                                                 = 2801
    Workpiece  Workpiece                                     = rep
    Name       Representation                             id = 6
    CPlane     Orientation                              name = TOP
                                                         i x = 1.000000e+00
                                                           y = 0.000000e+00
                                                           z = 0.000000e+00
                                                         j x = 0.000000e+00
                                                           y = 1.000000e+00
                                                           z = 0.000000e+00
                                                         k x = 0.000000e+00
                                                           y = 0.000000e+00
```

```
                                                        z = 1.000000e+00
    Location   Origin                                   x = 1.500000e+02
                                                        y = 1.500000e+02
                                                        z = 1.000000e+02

    ApplyType  Application                                = Extend
    String     rawMatlCondition                           = Solid
    Real       surfFinish                                 = 2.000000e+00
    String     mcNotmc                                    = Y
    String     atAngle                                    = nil
    String     bottom                                     = nil
    String     in                                         = nil
    String     intFeature                                 =
Hole_Blind484_I1
    String     left                                       = nil
    String     on                                         = 1
    String     out                                        = nil
    String     past                                       = nil
    String     right                                      = nil
    String     to                                         = nil
    String     top                                        = nil
    Face       EntryFace                                  = 2569
    String     childOf                                    = Plate53_I1
    Diameter   HoleDiameter                               = 5.000000e+01 =
millimeters
    Length     Depth                                      = 2.000000e+01 =
millimeters
    String     TolHoleDiameter                            = 0.1 / - 0.1
    String     TolDepth                                   = 0.1 / - 0.1

*****     printing 866 Hole_Blind484_I6     *****

class :
    class : Hole_Blind                              library : CVOBJECTS
    classtype : Feature                             timestamp : 07-26-96
12:42
static attributes :
    BooleanOp  Operation                                  = Subtract
attributes :
    Workpiece  Workpiece                                  = 2801
    Name       Representation                             = rep
    CPlane     Orientation                             id = 6
                                                     name = TOP
                                                      i x = 1.000000e+00
                                                        y = 0.000000e+00
                                                        z = 0.000000e+00
                                                      j x = 0.000000e+00
                                                        y = 1.000000e+00
                                                        z = 0.000000e+00
                                                      k x = 0.000000e+00
                                                        y = 0.000000e+00
                                                        z = 1.000000e+00
    Location   Origin                                   x = 2.000000e+02
                                                        y = 1.000000e+02
                                                        z = 1.000000e+02

    ApplyType  Application                                = Extend
    String     rawMatlCondition                           = Solid
    Real       surfFinish                                 = 2.000000e+00
    String     mcNotmc                                    = Y
    String     atAngle                                    = nil
    String     bottom                                     = nil
    String     in                                         = nil
    String     intFeature                                 =
Hole_Blind484_I1
    String     left                                       = nil
    String     on                                          = 1
    String     out                                         = nil
    String     past                                        = nil
    String     right                                       = nil
    String     to                                          = nil
    String     top                                         = nil
    Face       EntryFace                                   = 2569
    String     childOf                                     = Plate53_I1
    Diameter   HoleDiameter                                = 5.000000e+01 =
millimeters
    Length     Depth                                       = 2.000000e+01 =
millimeters
    String     TolHoleDiameter                             = 0.1 / - 0.1
    String     TolDepth                                    = 0.1 / - 0.1

*****     printing 897 Hole_Blind484_I7     *****

class :
    class : Hole_Blind                             library : CVOBJECTS
    classtype : Feature                            timestamp : 07-26-96
12:42
static attributes :
    BooleanOp  Operation                                   = Subtract
attributes :
    Workpiece  Workpiece                                   = 2801
    Name       Representation                              = rep
    CPlane     Orientation                              id = 6
                                                      name = TOP
                                                       i x = 1.000000e+00
                                                         y = 0.000000e+00
                                                         z = 0.000000e+00
                                                       j x = 0.000000e+00
                                                         y = 1.000000e+00
                                                         z = 0.000000e+00
                                                       k x = 0.000000e+00
                                                         y = 0.000000e+00
                                                         z = 1.000000e+00
    Location   Origin                                    x = 1.500000e+02
                                                         y = 5.000000e+01
                                                         z = 1.000000e+02
    ApplyType  Application                                 = Extend
    String     rawMatlCondition                            = Solid
    Real       surfFinish                                  = 2.000000e+00
    String     mcNotmc                                     = Y
    String     atAngle                                     = nil
    String     bottom                                      = nil
    String     in                                          = nil
    String     intFeature                                  =
Hole_Blind484_I1
    String     left                                        = nil
    String     on                                          = 1
    String     out                                         = nil
    String     past                                        = nil
    String     right                                       = nil
    String     to                                          = nil
    String     top                                         = nil
    Face       EntryFace                                   = 2569
    String     childOf                                     = Plate53_I1
    Diameter   HoleDiameter                                = 5.000000e+01 =
millimeters
    Length     Depth                                       = 2.000000e+01 =
millimeters
    String     TolHoleDiameter                             = 0.1 / - 0.1
    String     TolDepth                                    = 0.1 / - 0.1
```

```
*****      printing 928 Hole_Blind484_I8      *****

class :
    class : Hole_Blind                         library : CVOBJECTS
    classtype : Feature                        timestamp : 07-26-96
12:42
static attributes :
    BooleanOp  Operation                              = Subtract
attributes :
    Workpiece  Workpiece                              = 2801
    Name       Representation                         = rep
    CPlane     Orientation                         id = 6
                                                 name = TOP
                                                  i x = 1.000000e+00
                                                    y = 0.000000e+00
                                                    z = 0.000000e+00
                                                  j x = 0.000000e+00
                                                    y = 1.000000e+00
                                                    z = 0.000000e+00
                                                  k x = 0.000000e+00
                                                    y = 0.000000e+00
                                                    z = 1.000000e+00
    Location   Origin                               x = 1.000000e+02
                                                    y = 1.000000e+02
                                                    z = 1.000000e+02

    ApplyType  Application                            = Extend
    String     rawMatlCondition                       = Solid
    Real       surfFinish                             = 2.000000e+00
    String     mcNotmc                                = Y
    String     atAngle                                = nil
    String     bottom                                 = nil
    String     in                                     = nil
    String     intFeature                             =
Hole_Blind484_I1
    String     left                                   = nil
    String     on                                     = 1
    String     out                                    = nil
    String     past                                   = nil
    String     right                                  = nil
    String     to                                     = nil
    String     top                                    = nil
    Face       EntryFace                              = 2569
    String     childOf                                = Plate53_I1
    Diameter   HoleDiameter                           = 5.000000e+01 =
millimeters
    Length     Depth                                  = 2.000000e+01 =
millimeters
    String     TolHoleDiameter                        = 0.1 / - 0.1
    String     TolDepth                               = 0.1 / - 0.1

*****      printing 1834 Hole_Blind484_I9      *****

class :
    class : Hole_Blind                         library : CVOBJECTS
    classtype : Feature                        timestamp : 07-26-96
12:42
static attributes :
    BooleanOp  Operation                              = Subtract
attributes :
    Workpiece  Workpiece                              = 2801
    Name       Representation                         = rep
    CPlane     Orientation                         id = 15
                                                 name = RIGHT
```

```
                                                            i x = 0.000000e+00
                                                              y = 1.000000e+00
                                                              z = 0.000000e+00
                                                            j x = 0.000000e+00
                                                              y = 0.000000e+00
                                                              z = 1.000000e+00
                                                            k x = 1.000000e+00
                                                              y = 0.000000e+00
                                                              z = 0.000000e+00
    Location    Origin                                        x = 3.000000e+02
                                                              y = 1.250000e+02
                                                              z = 5.000000e+01
    ApplyType   Application                                     = Extend
    String      rawMatlCondition                                = Solid
    Real        surfFinish                                      = 2.000000e+00
    String      mcNotmc                                         = Y
    String      atAngle                                         = nil
    String      bottom                                          = nil
    String      in                                              = nil
    String      intFeature                                      =
Pocket_Rectangular1703_I1
    String      left                                            = nil
    String      on                                              = 1
    String      out                                             = nil
    String      past                                            = nil
    String      right                                           = nil
    String      to                                              = nil
    String      top                                             = nil
    Face        EntryFace                                       = 2545
    String      childOf                                         = Plate53_I1
    Diameter    HoleDiameter                                    = 4.500000e+01 =
millimeters
    Length      Depth                                           = 2.000000e+01 =
millimeters
    String      TolHoleDiameter                                 = 0.1 / - 0.1
    String      TolDepth                                        = 0.1 / - 0.1

*****     printing 1926 Hole_Blind484_I10      *****

class :
    class : Hole_Blind                          library : CVOBJECTS
    classtype : Feature                         timestamp : 07-26-96
12:42
static attributes :
    BooleanOp   Operation                                       = Subtract
attributes :
    Workpiece   Workpiece                                       = 2801
    Name        Representation                                  = rep
    CPlane      Orientation                                  id = 15
                                                           name = RIGHT
                                                            i x = 0.000000e+00
                                                              y = 1.000000e+00
                                                              z = 0.000000e+00
                                                            j x = 0.000000e+00
                                                              y = 0.000000e+00
                                                              z = 1.000000e+00
                                                            k x = 1.000000e+00
                                                              y = 0.000000e+00
                                                              z = 0.000000e+00
    Location    Origin                                        x = 3.000000e+02
                                                              y = 7.500000e+01
                                                              z = 5.000000e+01
    ApplyType   Application                                     = Extend
    String      rawMatlCondition                                = Solid
    Real       surfFinish                                = 2.000000e+00
    String     mcNotmc                                   = Y
    String     atAngle                                   = nil
    String     bottom                                    = nil
    String     in                                        = nil
    String     intFeature                                =
Pocket_Rectangular1703_I1
    String     left                                      = nil
    String     on                                        = 1
    String     out                                       = nil
    String     past                                      = nil
    String     right                                     = nil
    String     to                                        = nil
    String     top                                       = nil
    Face       EntryFace                                 = 2545
    String     childOf                                   = Plate53_I1
    Diameter   HoleDiameter                              = 4.500000e+01 =
millimeters
    Length     Depth                                     = 2.000000e+01 =
millimeters
    String     TolHoleDiameter                           = 0.1 / - 0.1
    String     TolDepth                                  = 0.1 / - 0.1

*****      printing 56 Plate53_I1      *****

class :
    class : Plate                               library : CVOBJECTS
    classtype : Feature                         timestamp : 07-26-96
12:32
static attributes :
    BooleanOp  Operation                                 = Add
    Workpiece  Workpiece                                 = 0
    ApplyType  Application                               = NoApply
attributes :
    Name       Representation                            = rep
    CPlane     Orientation                            id = 6
                                                    name = TOP
                                                     i x = 1.000000e+00
                                                       y = 0.000000e+00
                                                       z = 0.000000e+00
                                                     j x = 0.000000e+00
                                                       y = 1.000000e+00
                                                       z = 0.000000e+00
                                                     k x = 0.000000e+00
                                                       y = 0.000000e+00
                                                       z = 1.000000e+00
    Location   Origin                                  x = 0.000000e+00
                                                       y = 0.000000e+00
                                                       z = 0.000000e+00
    String     rawMatlCondition                          = Solid
    Real       surfFinish                                = 2.000000e+00
    String     mcNotmc                                   = Y
    String     atAngle                                   = nil
    String     bottom                                    = nil
    String     in                                        = nil
    String     intFeature                                = nil
    String     left                                      = nil
    String     on                                        = nil
    String     out                                       = nil
    String     past                                      = nil
    String     right                                     = nil
    String     to                                        = nil
    String     top                                       = nil
    Length     PlateLength                         = 3.000000e+02 =
millimeters
    String     TolPlateLength                      = 0.1 / - 0.1
    Length     PlateWidth                          = 2.000000e+02 =
millimeters
    String     TolPlateWidth                       = 0.1 / - 0.1
    Length     PlateHeight                         = 1.000000e+02 =
millimeters
    String     TolPlateHeight                      = 0.1 / - 0.1
```

9.3 Process Plan For TestComp4

Automated Process Plan
Component Name: TestComp4
Component Material: Free Machining Carbon Steel
Matl Description: Medium carbon Resulfurized Matl Condition: Hot rolled or Cold drawn
Material Specifications: 1140 Material Hardness: (175 225)

SetUp No: 1
Fixture For SetUp: MachineVice

('HoleBlind484I1' 'HoleBlind484I2' 'HoleBlind484I6' 'HoleBlind484I7' 'HoleBlind484I8')
The above features form a complex pocket due to feature interaction
Machining process:

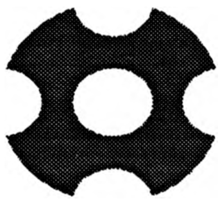
Opr No: 1; Opr Name: Rough SlotDrilling
Tool For Opr: ('EndMill' 49.2)
Recommended Cutting Speed: ('m/min' 100) Feed: ('mm/min' 150)
Machine For Opr: 'BokoVerticalMachiningCentreI1'
PositionPoint : ((150.0 150.0 100.0) (200.0 100.0 100.0) (150.0 50.0 100.0) (100.0 100.0 100.0))
ApproachPoint: 103.0
PlungePoint : 80.4
ReturnPoint : 108.0
ClearPoint : 175.0

Opr No: 2; Opr Name: Rough SlotDrilling
Tool For Opr: ('EndMill' 49.2)
Recommended Cutting Speed: ('m/min' 100) Feed: ('mm/min' 150)
Machine For Opr: 'BokoVerticalMachiningCentreI1'
PositionPoint : ((150.0 100.0 100.0))
ApproachPoint: 103.0
PlungePoint : 40.4
ReturnPoint : 108.0
ClearPoint : 175.0

Opr No: 3; Opr Name: Rough EndMilling
Tool For Opr: ('EndMill' 30.0)
Recommended Cutting Speed: ('m/min' 130) Feed: ('mm/min' 120)
Machine For Opr: 'BokoVerticalMachiningCentreI1'
PositionPt : (150.0 100.0 100.0)
ApproachPlane: 120.0
ZWorkPlane : 80.4
RetractPlane : 120.0
ClearPlane : 170.0

Stock Left after operation > 0.4 mm

Edge profile for the operation:



(an Arc an Arc an Arc an Arc an Arc an Arc an Arc an Arc an Arc an Arc an Arc an Arc)
Edge profile details:
Edge 1: an Arc
Start Point: 125.793, 143.750, 100.000
End Point: 106.250, 124.206, 100.000
Arc Origin : 150.000, 100.000, 100.000
Arc Radius : 50.000

Edge 2: an Arc
Start Point: 106.250, 124.206, 100.000
End Point: 125.000, 100.000, 100.000
Arc Origin : 100.000, 100.000, 100.000
Arc Radius : 25.000

Edge 3: an Arc
Start Point: 125.000, 100.000, 100.000
End Point: 175.000, 100.000, 100.000
Arc Origin : 150.000, 100.000, 100.000
Arc Radius : 25.000

Edge 4: an Arc
Start Point: 175.000, 100.000, 100.000
End Point: 193.750, 124.206, 100.000
Arc Origin : 200.000, 100.000, 100.000
Arc Radius : 25.000

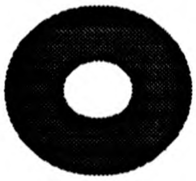
Edge 5: an Arc
Start Point: 193.750, 124.206, 100.000
End Point: 174.206, 143.750, 100.000
Arc Origin : 150.000, 100.000, 100.000
Arc Radius : 50.000

Edge 6: an Arc
Start Point: 174.206, 143.750, 100.000
End Point: 125.793, 143.750, 100.000
Arc Origin : 150.000, 150.000, 100.000
Arc Radius : 25.000

Edge 7: an Arc
Start Point: 106.250, 75.793, 100.000
End Point: 125.793, 56.249, 100.000
Arc Origin : 150.000, 100.000, 100.000
Arc Radius : 50.000

Edge 8: an Arc

```
Start  Point: 174.206, 56.249, 100.000
End    Point: 193.750, 75.793, 100.000
Arc Origin  : 150.000, 100.000, 100.000
Arc Radius  : 50.000

Edge 9: an Arc
Start  Point: 175.000, 100.000, 100.000
End    Point: 125.000, 100.000, 100.000
Arc Origin  : 150.000, 100.000, 100.000
Arc Radius  : 25.000

Edge 10: an Arc
Start  Point: 125.000, 100.000, 100.000
End    Point: 106.250, 75.793, 100.000
Arc Origin  : 100.000, 100.000, 100.000
Arc Radius  : 25.000

Edge 11: an Arc
Start  Point: 193.750, 75.793, 100.000
End    Point: 175.000, 100.000, 100.000
Arc Origin  : 200.000, 100.000, 100.000
Arc Radius  : 25.000

Edge 12: an Arc
Start  Point: 125.793, 56.250, 100.000
End    Point: 174.206, 56.250, 100.000
Arc Origin  : 150.000, 50.000, 100.000
Arc Radius  : 25.000


Opr No: 4; Opr Name: Rough EndMilling
Tool For Opr: ('EndMill' 30.0)
Recommended Cutting Speed: ('m/min' 130) Feed: ('mm/min' 120)
Machine For Opr: 'BokoVerticalMachiningCentre11'
PositionPt   : (150.0 100.0 100.0)
ApproachPlane: 100.0
ZWorkPlane   : 40.4
RetractPlane : 100.0
ClearPlane   : 150.0

Stock Left after operation > 0.4 mm

Edge profile for the operation:
(an Arc an Arc an Arc an Arc)
Edge profile details:
```



```
Edge 1: an Arc
Start  Point: 200.000, 100.000, 100.000
End    Point: 100.000, 100.000, 100.000
Arc Origin  : 150.000, 100.000, 100.000
```

Arc Radius : 50.000

Edge 2: an Arc
Start Point: 100.000, 100.000, 100.000
End Point: 200.000, 100.000, 100.000
Arc Origin : 150.000, 100.000, 100.000
Arc Radius : 50.000

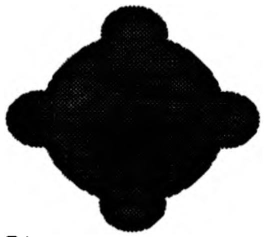
Edge 3: an Arc
Start Point: 125.000, 100.000, 100.000
End Point: 175.000, 100.000, 100.000
Arc Origin : 150.000, 100.000, 100.000
Arc Radius : 25.000

Edge 4: an Arc
Start Point: 175.000, 100.000, 100.000
End Point: 125.000, 100.000, 100.000
Arc Origin : 150.000, 100.000, 100.000
Arc Radius : 25.000

Opr No: 5; Opr Name: Finish EndMilling
Tool For Opr: ('EndMill' 30.0)
Recommended Cutting Speed: ('m/min' 175) Feed: ('mm/min' 125)
Machine For Opr: 'BokoVerticalMachiningCentre11'
PositionPt : (150.0 100.0 100.0)
ApproachPlane: 120.0
ZWorkPlane : 80.0
RetractPlane : 120.0
ClearPlane : 170.0

Stock Left after operation > 0.0 mm

Edge profile for the operation:
(an Arc an Arc an Arc an Arc an Arc an Arc an Arc an Arc an Arc an Arc an Arc an Arc an Arc an Arc an Arc)
Edge profile details:

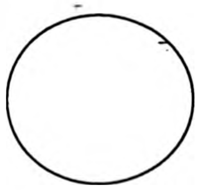


Edge 1: an Arc
Start Point: 193.750, 124.206, 100.000
End Point: 174.206, 143.750, 100.000
Arc Origin : 150.000, 100.000, 100.000
Arc Radius : 50.000

Edge 2: an Arc
Start Point: 174.206, 143.750, 100.000
End Point: 175.000, 150.000, 100.000
Arc Origin : 150.000, 150.000, 100.000
Arc Radius : 25.000

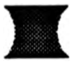
```
Edge 3: an Arc
Start  Point: 175.000, 150.000, 100.000
End    Point: 125.000, 150.000, 100.000
Arc Origin  : 150.000, 150.000, 100.000
Arc Radius  : 25.000

Edge 4: an Arc
Start  Point: 125.000, 150.000, 100.000
End    Point: 125.793, 143.750, 100.000
Arc Origin  : 150.000, 150.000, 100.000
Arc Radius  : 25.000

Edge 5: an Arc
Start  Point: 125.793, 143.750, 100.000
End    Point: 106.250, 124.206, 100.000
Arc Origin  : 150.000, 100.000, 100.000
Arc Radius  : 50.000

Edge 6: an Arc
Start  Point: 106.250, 124.206, 100.000
End    Point: 75.000, 100.000, 100.000
Arc Origin  : 100.000, 100.000, 100.000
Arc Radius  : 25.000

Edge 7: an Arc
Start  Point: 75.000, 100.000, 100.000
End    Point: 106.250, 75.793, 100.000
Arc Origin  : 100.000, 100.000, 100.000
Arc Radius  : 25.000

Edge 8: an Arc
Start  Point: 106.250, 75.793, 100.000
End    Point: 125.793, 56.249, 100.000
Arc Origin  : 150.000, 100.000, 100.000
Arc Radius  : 50.000

Edge 9: an Arc
Start  Point: 125.793, 56.250, 100.000
End    Point: 125.000, 50.000, 100.000
Arc Origin  : 150.000, 50.000, 100.000
Arc Radius  : 25.000

Edge 10: an Arc
Start  Point: 125.000, 50.000, 100.000
End    Point: 175.000, 50.000, 100.000
Arc Origin  : 150.000, 50.000, 100.000
Arc Radius  : 25.000

Edge 11: an Arc
Start  Point: 175.000, 50.000, 100.000
End    Point: 174.206, 56.250, 100.000
Arc Origin  : 150.000, 50.000, 100.000
Arc Radius  : 25.000

Edge 12: an Arc
Start  Point: 174.206, 56.249, 100.000
End    Point: 193.750, 75.793, 100.000
```

```
Arc Origin  : 150.000, 100.000, 100.000
Arc Radius  : 50.000

Edge 13: an Arc
Start  Point: 193.750, 75.793, 100.000
End    Point: 225.000, 100.000, 100.000
Arc Origin  : 200.000, 100.000, 100.000
Arc Radius  : 25.000

Edge 14: an Arc
Start  Point: 225.000, 100.000, 100.000
End    Point: 193.750, 124.206, 100.000
Arc Origin  : 200.000, 100.000, 100.000
Arc Radius  : 25.000


Opr No: 6; Opr Name: Finish EndMilling
Tool For Opr: ('EndMill' 30.0)
Recommended Cutting Speed: ('m/min' 175) Feed: ('mm/min' 125)
Machine For Opr: 'BokoVerticalMachiningCentre11'
PositionPt   : (150.0 100.0 100.0)
ApproachPlane: 100.0
ZWorkPlane   : 40.0
RetractPlane : 100.0
ClearPlane   : 150.0

Stock Left after operation > 0.0 mm

Edge profile for the operation:
(an Arc an Arc)
Edge profile details:
```



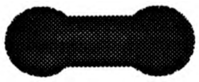
```
Edge 1: an Arc
Start  Point: 200.000, 100.000, 100.000
End    Point: 100.000, 100.000, 100.000
Arc Origin  : 150.000, 100.000, 100.000
Arc Radius  : 50.000

Edge 2: an Arc
Start  Point: 100.000, 100.000, 100.000
End    Point: 200.000, 100.000, 100.000
Arc Origin  : 150.000, 100.000, 100.000
Arc Radius  : 50.000


SetUp No: 3
Fixture For SetUp:

('PocketRectangular170311' 'HoleBlind48419' 'HoleBlind484110')
The above features form a complex pocket due to feature interaction
```

```
Machining process:

Opr No: 1; Opr Name: Rough SlotDrilling
Tool For Opr: ('EndMill' 44.2)
Recommended Cutting Speed: ('m/min' 100) Feed: ('mm/min' 150)
PositionPoint   : ((125.0 50.0 300.0) (75.0 50.0 300.0))
ApproachPoint: 303.0
PlungePoint   : 280.4
ReturnPoint : 308.0
ClearPoint   : 375.0


Opr No: 2; Opr Name: Rough EndMilling
Tool For Opr: ('EndMill' 20.0)
Recommended Cutting Speed: ('m/min' 130) Feed: ('mm/min' 120)
PositionPt   : (100.0 50.0 300.0)
ApproachPlane: 320.0
ZWorkPlane   : 280.4
RetractPlane : 320.0
ClearPlane   : 370.0

Stock Left after operation > 0.4 mm

Edge profile for the operation:
(a Line an Arc an Arc a Line an Arc an Arc)
Edge profile details:
```



```
Edge 1: a Line
Start  Point: 91.770, 35.000, 300.000
End    Point: 108.229, 35.000, 300.000

Edge 2: an Arc
Start  Point: 108.229, 35.000, 300.000
End    Point: 102.500, 50.000, 300.000
Arc Origin  : 125.000, 50.000, 300.000
Arc Radius  : 22.500

Edge 3: an Arc
Start  Point: 102.500, 50.000, 300.000
End    Point: 108.229, 65.000, 300.000
Arc Origin  : 125.000, 50.000, 300.000
Arc Radius  : 22.500

Edge 4: a Line
Start  Point: 108.229, 65.000, 300.000
End    Point: 91.770, 65.000, 300.000

Edge 5: an Arc
Start  Point: 91.770, 65.000, 300.000
End    Point: 97.500, 50.000, 300.000
Arc Origin  : 75.000, 50.000, 300.000
Arc Radius  : 22.500

Edge 6: an Arc
Start  Point: 97.500, 50.000, 300.000
End    Point: 91.770, 35.000, 300.000
```

```
Arc Origin  : 75.000, 50.000, 300.000
Arc Radius  : 22.500


Opr No: 3; Opr Name: Finish EndMilling
Tool For Opr: ('EndMill' 20.0)
Recommended Cutting Speed: ('m/min' 175) Feed: ('mm/min' 125)
PositionPt   : (100.0 50.0 300.0)
ApproachPlane: 320.0
ZWorkPlane   : 280.0
RetractPlane : 320.0
ClearPlane   : 370.0

Stock Left after operation > 0.0 mm

Edge profile for the operation:
(a Line an Arc an Arc a Line an Arc an Arc)
Edge profile details:
```



```
Edge 1: a Line
Start  Point: 91.770, 35.000, 300.000
End    Point: 108.229, 35.000, 300.000

Edge 2: an Arc
Start  Point: 108.229, 35.000, 300.000
End    Point: 147.500, 50.000, 300.000
Arc Origin  : 125.000, 50.000, 300.000
Arc Radius  : 22.500

Edge 3: an Arc
Start  Point: 147.500, 50.000, 300.000
End    Point: 108.229, 65.000, 300.000
Arc Origin  : 125.000, 50.000, 300.000
Arc Radius  : 22.500

Edge 4: a Line
Start  Point: 108.229, 65.000, 300.000
End    Point: 91.770, 65.000, 300.000

Edge 5: an Arc
Start  Point: 91.770, 65.000, 300.000
End    Point: 52.500, 50.000, 300.000
Arc Origin  : 75.000, 50.000, 300.000
Arc Radius  : 22.500

Edge 6: an Arc
Start  Point: 52.500, 50.000, 300.000
End    Point: 91.770, 35.000, 300.000
Arc Origin  : 75.000, 50.000, 300.000
Arc Radius  : 22.500
```