AUTOMATIC SETUP SELECTION AND OPERATION SEQUENCING FOR
COMPUTER-AIDED MANUFACTURING SOFTWARE

A thesis presented to
the faculty of
the Russ College of Engineering and Technology of Ohio University

In partial fulfillment
of the requirements for the degree
Master of Science

Narender Neerukonda
November 2005
This thesis entitled

AUTOMATIC SETUP SELECTION AND OPERATION SEQUENCING FOR
COMPUTER-AIDED MANUFACTURING SOFTWARE

by

NARENDER NEERUKONDA

has been approved for

the Department of Industrial and Manufacturing Systems Engineering

and the Russ College of Engineering and Technology by

Dušan N. Šormaz

Associate Professor of Industrial and Manufacturing Systems Engineering

Dennis Irwin

Dean, Russ College of Engineering and Technology
The thesis describes the procedure for manufacturing setup selection and sequencing with the goal to reduce the processing time of parts manufactured on a 3-axis milling machines. This procedure consists of two steps: operation clustering and operation sequencing. In the first step, geometry data for each operation is analyzed for spatial position and orientation. Based on the analysis, the tool direction for each operation is computed, and then all operations with the same tool direction are clustered into a single setup. In the second step, operations within a single setup are sequenced according to process precedence criteria to generate the final order of machining operations. The clustering and sequencing of operation is implemented as the setup selection application, which sends the tree structure of operations back to the CAM system. The procedure has been tested and verified on several mechanical designs.

Approved:

Dušan N. Šormaz
Associate Professor of Industrial and Manufacturing Systems Engineering
Dedication

To

My Family and Dr. Šormaz
Acknowledgements

I would like to take this opportunity to thank all those who contributed to the successful completion of this thesis.

First of all, I thank my parents Mrs. Rajya Lakshmi and Mr. Samba Siva Rao, my sister Mrs. Kalyani and brother-in-law Mr. Sudheer Babu for their continued love, affection and moral support. My sincere thanks to my advisor Dr. Dušan N. Šormaz for being a friend, mentor, guiding force and for sharing his ingenuity, insight and expertise in the area of manufacturing. I would also like to thank Dr. Dušan N. Šormaz for helping me, in every possible way, to overcome the obstacles I faced during my Masters at Ohio University. I would also like to take this opportunity to thank Dr. David Koonce and Dr. Constantinos Vassiliadis for their innovative ideas and valuable suggestions to enhance the content of this thesis document.

I would also like to thank Jaikumar Arumugam (my colleague in IMPlanner Group) for his important suggestions. I would also like to express my sincere thanks to Ajit Wadatkar, Harihara Sharma, Deepak Pisipati and Chandrasekhar Ganduri for supporting and motivating me during my difficult times. I would also like to thank my other friends at Ohio University who made my stay in Athens a pleasing and memorable one.
Table of Contents

Dedication ............................................................................................................... 4
Acknowledgements ................................................................................................. 5
List of Tables ........................................................................................................... 9
List of Figures ......................................................................................................... 10
1 Introduction ........................................................................................................ 13
   1.1 Technical Terms ........................................................................................... 14
       1.1.1 Computer-Aided Design (CAD) .............................................................. 14
       1.1.2 Computer-Aided Manufacturing (CAM) ............................................... 15
       1.1.3 Process Planning .................................................................................. 16
       1.1.4 Computer-Aided Process Planning (CAPP) .......................................... 17
   1.2 Problem Statement ....................................................................................... 17
   1.3 Objective ..................................................................................................... 18
   1.4 Assumptions ............................................................................................... 19
   1.5 Outline of Thesis Report ............................................................................. 20
2 Previous Work ..................................................................................................... 22
   2.1 Process Sequencing .................................................................................... 22
   2.2 Reducing Number of Tool Changes .......................................................... 25
   2.3 Data Representation and Exchange (XML and JNI) ................................ 27
3 Methodology ....................................................................................................... 29
   3.1 Tools Used .................................................................................................. 29
       3.1.1 IMPlanner System ................................................................................ 30
       3.1.2 Unigraphics .......................................................................................... 34
       3.1.3 Java Native Interface .......................................................................... 40
   3.2 Creating Manufacturing Operations ........................................................... 43
       3.2.1 Manufacturing Module ....................................................................... 44
       3.2.2 Creating Operation ............................................................................ 45
3.2.3 Operation Details ....................................................................................... 48
3.2.4 ToolPath Generation ................................................................................... 51
3.3 Retrieving Part Model and UG Operations .............................................. 53
3.4 Retrieving CAM Roots ............................................................................ 54
3.5 Creating MCS Groups .............................................................................. 55
  3.5.1 Tool Axis Vector for Volume Milling Operation ..................................... 57
  3.5.2 Tool Axis Vector for Face Milling and Planar Milling Operations .......... 60
  3.5.3 Tool Axis Vector for Drilling .................................................................... 62
3.6 Clustering of Operations .......................................................................... 63
3.7 Operation Sequencing ............................................................................ 64
4 Implementation ........................................................................................... 65
  4.1 IMPlanner Modules ................................................................................. 65
    4.1.1 Features Package ................................................................................... 65
    4.1.2 Geometry Package ................................................................................. 66
  4.2 CAM Architecture ................................................................................... 67
  4.3 CAM Package Classes ............................................................................. 68
    4.3.1 CAMOperation ..................................................................................... 68
    4.3.2 PartModel ............................................................................................. 72
    4.3.3 CAMPartModel ..................................................................................... 72
    4.3.4 NCGroup ............................................................................................. 72
  4.4 Flow of Control ........................................................................................ 78
    4.4.1 Retrieving Unigraphics Operations ......................................................... 78
    4.4.2 Retrieving CAM Roots ........................................................................... 83
    4.4.3 Creation of MCS Groups ........................................................................ 85
    4.4.4 Clustering of Operations ........................................................................ 88
    4.4.5 Sequencing of Operations ...................................................................... 90
5 Test Cases ..................................................................................................... 93
5.1 Bracket .....................................................................................................93
  5.1.1 Retrieving Unigraphics Operations.......................................................96
  5.1.2 Retrieving CAM Roots........................................................................97
  5.1.3 Creating MCS Groups .......................................................................98
  5.1.4 Clustering Operations .......................................................................100

5.2 ANC101 ...............................................................................................103

6 Conclusions ..........................................................................................106
  6.1 Contributions .......................................................................................106
  6.2 Limitations .........................................................................................106
  6.3 Further Research ................................................................................107

References .................................................................................................. 108

Appendix: List of Acronyms .....................................................................111
List of Tables

Table 3.1  UG/ Naming Conventions................................................................. 35
Table 3.2  UG/Open Argument Types .............................................................. 36
Table 3.3  UG/Open Function Prototype ......................................................... 36
Table 5.1  Operations, Corresponding Features Machined and Tools Used .......... 95
List of Figures

Figure 1.1  CAD Model of a Vehicle ................................................................. 15
Figure 1.2  Bracket with Generated Toolpaths .............................................. 16
Figure 3.1  IMPlanner System Architecture ..................................................... 31
Figure 3.2  CAM Architecture ....................................................................... 33
Figure 3.3  Header File Containing the Signature of Java Methods ................. 42
Figure 3.4  C Method that Calls the UG/Open Methods .............................. 43
Figure 3.5  Opening Manufacturing Module in Unigraphics ......................... 43
Figure 3.6  Various Setups during Initialization .............................................. 44
Figure 3.7  CreateOperation Toolbar ................................................................. 45
Figure 3.8  CreateOperation Dialog Box .......................................................... 46
Figure 3.9  Geometries Required for Planar and Contour Milling ................. 50
Figure 3.10 Parameters Required to Generate Toolpath ............................. 52
Figure 3.11 Generating ToolPath .................................................................... 52
Figure 3.12 Generated ToolPath ..................................................................... 53
Figure 3.13 Operation Types and Respective Subtypes ................................. 58
Figure 3.14 Feature Selected and Geometry Retrieved ............................... 59
Figure 3.15 Tool Axis Vector of the Feature Selected ................................. 60
Figure 3.16 FaceMilling of the Boss Top ......................................................... 61
Figure 3.17 Tool Axis Vector Retrieved for FaceMilling ............................... 62
Figure 3.18 Operation Precedence ................................................................. 64
Figure 4.1  IMPlanner Features Package Hierarchy ....................................... 66
Figure 4.2  IMPlanner Geometry Package Hierarchy ....................................................... 66
Figure 4.3  CAM Module Framework ........................................................................... 69
Figure 4.4  Tasks of CAMOperation Class ................................................................. 71
Figure 4.5  CAMOperation and Its Subclasses ............................................................ 72
Figure 4.6  NCGroup and Its Subclasses ..................................................................... 73
Figure 4.7  Data Transfer between IMPlanner System and Unigraphics ....................... 74
Figure 4.8  CAMSequencingPanel ................................................................................ 76
Figure 4.9  Tasks of CAMSequencingPanel .................................................................. 77
Figure 4.10  OpenUGFileButton Action Listener ......................................................... 79
Figure 4.11  Populate Operations Method .................................................................... 80
Figure 4.12  getOperationDetails Method .................................................................. 81
Figure 4.13  getOperationGroups Method ................................................................... 82
Figure 4.14  getNCGroups Method in CAMSystem ...................................................... 84
Figure 4.15  setAssociatedNormals Method in ZlevelMilling ...................................... 86
Figure 4.16  setAssociatedNormals Method in CAMOperation .................................. 87
Figure 4.17  Methods of camPartModel Used for Clustering Operations .................... 89
Figure 4.18  compareTo Method of CAMOperation ..................................................... 90
Figure 4.19  Precedence Information Stored in XML File ............................................ 91
Figure 4.20  Static block in CAMPartModel to Initialize the Operation Precedence ....... 92
Figure 4.21  moveOperation Method in CAMSystem .................................................. 92
Figure 5.1  Features Used to Model Bracket ................................................................ 94
Figure 5.2  Bracket with All Operations Added to the Default Group ........................... 94
Figure 5.3 Opening Bracket in CAMSequencingPanel ................................................... 96
Figure 5.4 Bracket Details Shown in CAMSequencingPanel....................................... 97
Figure 5.5 Retrieving Group Details in CAMSequencingPanel ..................................... 98
Figure 5.6 Geometry of the Operations Moved to New Group ................................. 99
Figure 5.7 Creating New Group in CAMSequencingPanel ........................................... 100
Figure 5.8 Operation Geometry in rightPanel........................................................... 101
Figure 5.9 Operations Moved to New Group.............................................................. 102
Figure 5.10 Final Result of Bracket ........................................................................... 102
Figure 5.11 ANC101 with All Operations Added to the None Group......................... 104
Figure 5.12 Final Result of ANC101 .......................................................................... 104
Figure 5.13 Final Result of ANC101 Showing the Four Different Access Directions.. 105
1 Introduction

“In today’s highly competitive environment, organizations that successfully predict and appropriately match consumer demand with products will be survivors. Product development is a complex process with many parties involved. It requires enormous resources both in manpower as well as in financial investments. Product development is a process starting with product innovation, then moving to the development of products, production and finally the market launch. The process involves various internal activities like product management, research & development, production and purchasing.”[1]

Product development that undergoes such a complex procedure cannot afford unnecessary production time delays. Such delays may hamper the overall product value drastically. Another important factor that should be taken into consideration in product development is the production cost. This factor creates additional constraint on the complete development process.

Manufacturing systems play a vital in the product development process. The success of the product largely depends on the performance of the manufacturing systems. The performance of the manufacturing systems can be increased by lowering the lead times, facilitating smooth production flows and decreasing the down time of the system.

In order to achieve the above mentioned goals, i.e. shorter production times and low manufacturing costs, effective communication and data exchange between various functions of the development process are required. This calls for Computer-Aided Integration. The product development involves integration of computer-aided design
(CAD), computer-aided engineering analysis (CAE), computer-aided manufacturing (CAM), and computer-aided process planning (CAPP) tools. There are many systems developed to achieve the integration. But the integration does not provide the desired results unless each module that participates in the integration performs to its full potential. The present work aims to improve the overall efficiency of one of the modules of the Computer Integrated Manufacturing Framework called the CAM.

This research seeks to provide an application that is capable of eliminating the unnecessary setup changes that increase the production time.

1.1 Technical Terms

This thesis project deals with some technical terms associated with the Computer Integrated Manufacturing field. A brief explanation of these terms is given.

1.1.1 Computer-Aided Design (CAD)

“Computer-Aided Design is modeling of physical systems on computers, allowing both interactive and automatic analysis of design variants and the expression of designs in a form suitable for manufacturing.”[2]

Part designs can be produced more quickly and easily with the required precision. As a result, the time and cost involved in designing can be reduced to a great extent. Moreover CAD files are easy to duplicate and distribute. Parts under development can be subjected to various tests without being produced. Without having to reproduce the complete drawing, resizing can be done easily. All these advantages make CAD more economical and efficient when compared to the more tedious manual process. Figure 1.1 shows the CAD model of a vehicle in design phase [3].
1.1.2 Computer-Aided Manufacturing (CAM)

“The use of computers and computer technology to control, manage, operate and monitor manufacturing process.”[4]

Parts can be manufactured quickly with the required accuracy. Manufacturing can be done continuously for long time without any human interference. Machinery can be reprogrammed easily for every new order. Parts can be manufactured with high precision that is not possible with manual process. The simulation of manufacturing provided by CAM software can be used to avoid errors and make necessary changes. All these advantages make CAD more economical and efficient when compared to the more
tedious manual process. Figure 1.2 shows a part with generated toolpaths in CAM software.

![Bracket with Generated Toolpaths](image)

### 1.1.3 Process Planning

“Process planning encompasses the activities and functions to prepare a detailed set of plans and instructions to produce a part. The results of process planning are:

- Routings which specify operations, operation sequences, work centers, standards, tooling and fixtures. These routings become a major input to the manufacturing resources planning system to define operations for production activity, control purposes and required resources for capacity requirements planning purposes.
• Process plans which typically provide more detailed step-by-step work instructions, including dimensions related to individual operations, machining parameters, set-up instructions and quality assurance check points.

• Manufacturing and assembly drawings to support manufacture as opposed to engineering drawings to define the part.”[5]

1.1.4 **Computer-Aided Process Planning (CAPP)**

As mentioned above process planning lists the steps to transform raw material to finished products. These steps are generated manually by the process planner. But using computers to generate the process plans will increase the efficiency. This increase in efficiency can be attributed to decrease in time and cost to develop the plans, product consistency etc. So the use of computers to develop process plans is called “Computer-Aided Process Planning”.

1.2 **Problem Statement**

Unigraphics is modeling software that is used widely in various phases of the CIM framework in manufacturing industries. One such phase is manufacturing, i.e. Unigraphics is also used as CAM software. The module of Unigraphics used in industry for CAM is called the “manufacturing module”. The manufacturing module is used to generate toolpaths for various features in the part and these toolpaths are used to generate NC code.

The Unigraphics manufacturing module allows the user to interactively program and post-process milling, drilling, sheet metal and turning operations. Post-processing can be
used to generate NC code that can be given as input to CNC machines to manufacture the part. But the order of machining of features, developed in NC code, depends on the order of creation of toolpaths in the manufacturing module. During toolpath generation, if operations sequence and setup clustering is not taken into consideration, the resultant NC code generated may lead to unnecessary setup changes and increased process times. Increase in process times may also be caused by user inconsistency. So if clustering of operations, according to setups, and sequencing of operations, within each cluster, is done before NC code generation (post-processing) then the increased process times and unnecessary setup changes can be avoided. The present work aims at clustering the operations according to setups and sequencing the operations in each cluster before generating the NC code.

1.3  Objective

The present thesis aims to reduce the processing time of a part by eliminating unnecessary setup changes during manufacturing. This is achieved by transferring the manufacturing operations structure from CAM System to Process Planning Application, clustering them according to setup requirement, sequencing the operations within each cluster and sending them back to the CAM System. The result is a manufacturing operations tree-structure that is capable of manufacturing the part in less time than before using the software, also helping in “consistency” validation.

The general approach for transforming the operations added in a single group to different clusters according to various setups required to manufacture the part is:
• Retrieve CAM Operations to Process Planning System: In this step, manufacturing operations objects corresponding to CAM System operations are created in Process Planning System.

• Retrieve CAM roots of the Operations: In this phase, Group roots of operations are retrieved from the CAM System.

• Create setup groups for all operations that have different access directions than the ones that are already in the structure. The operations are later moved to these newly created groups. New clusters/groups are created using geometrical information about operations from the CAM System.

• Regrouped clusters and operations are transferred back to the CAM System.

The present application is developed as an extension to a very diverse yet effective, multifaceted yet integrated Computer-Aided Process Planning System called IMPlanner System [15], developed by the IMPlanner Research Group at Ohio University. IMPlanner System consists of several interdependent modules, each capable of performing a particular aspect of Computer-Aided Process Planning. Various aspects of the IMPlanner Module include feature modeling, feature recognition, process selection, machine selection, process data exchange, and process sequencing. The goal of this thesis is to increase the potential of manufacturing facet of the system.

1.4 Assumptions

The following assumptions are made in the present work:

• Parts analyzed in the present work are only prismatic in nature, ones that can be completely manufactured on 3-axis milling machines.
• All manufacturing operations created in Unigraphics are added to the “work-
piece” group under “MCS” geometric group. The same hierarchy is maintained
during the creation of new geometric groups.

• All manufacturing operations created in Unigraphics are added to the “program”
group under “NC_Program” program group. The same hierarchy is maintained
during the creation of a new program group.

• The Unigraphics part file given as input to the present software application should
have all the necessary operations required to completely manufacture the part.

• The goal of the present research is not to create CAM Operations for a given part.
The procedure only clusters and sequences the given set of operations to minimize
the setup changes.

• The final clusters produced are not guaranteed to be the best possible
combination, because the geometric interactions between features associated with
operations are not taken into consideration while sequencing the operations.
However, the clusters produced will reduce the production time.

1.5 Outline of Thesis Report

The thesis is organized into the following chapters:

Chapter 1 gives an introduction to the present thesis and also explains the problem
statement, objective and assumptions.

Chapter 2 explains about the various systems that are similar to the present system,
but the systems explained in the chapter use different methodologies in developing.
Chapter 3 explains the methodology used in developing the present system. This chapter also describes the various tools like IMPlanner System, Unigraphics, UG/Open JNI that are used in the present thesis.

Chapter 4 describes the implementation part of the thesis.

Chapter 5 lists the various cases used to test the implemented application for verification.

Chapter 6 states the conclusions drawn from the present work and the scope of future work.
This chapter presents a literature review of papers that explains the work done in similar fields. Computer-Aided Process Planning acts as the link between Computer-Aided Design and Computer-Aided Manufacturing. There are many steps in the process of computer-aided process planning. The present thesis deals only with few aspects like operation clustering and process sequencing. So the present chapter presents papers that are related only to those aspects of Computer-Aided Process Planning.

2.1 Process Sequencing

Sormaz and Khoshnevis [7] consider process sequencing instead of feature sequencing for process planning. The proposed algorithm consists of two parts: process clustering and process sequencing. Process sequencing is preferred instead of feature sequencing for process planning to consider constraints imposed by design and manufacturing practice. These constraints may be geometrical constraints, technology constraints and economical constraints.

The process starts with process selection for the features of the part. After the processes are selected for the features, feasible tool orientations are assigned to the processes. After assigning the feasible tool orientations to the processes, the first step of the algorithm, process clustering, is started. Process clustering is accomplished in two stages. In the first stage, all the processes with same tool orientations are grouped in the same cluster. Same tool orientations are obtained by considering the face normals of the planar faces and tool axis direction for cylindrical faces. Then in the second stage of
process clustering, all the tool orientation clusters are groups together for the same machine clusters.

Process sequencing follows the process clustering according to tool orientation and machine type. State Space search is used to sequence the operations. The present work also clusters the processes according to same tool orientation, but does not use Space Search to sequence the processes. Instead operation sequence relationship data stored in an XML file is used to sequence the processes.

Wu and Chang [8] propose the use of tolerance information for setup planning. They propose that the use of tool approach direction in setup planning may sometimes lead to incorrect parts. The focus of the paper is to generate the setup plans taking the tolerance information into consideration. So an automated tolerance analysis approach for selecting setups and datums is developed for prismatic parts. After developing the setup plans, the number of setup plans is reduced to identify the optimal setup plans. In order to do that, the setup plans are ranked according to the positional tolerance variable induced and number of setups in the plan. The tolerance zone and operation error induced in the features by all the setups are calculated. Comparing the setup plans with the calculated values of error, machinability of the features can be determined. Perpendicularity and Parallelism are also considered while comparing the setups. The ranked setup plans are then reported to the downstream applications. The present work does not take into consideration the aspect of reducing the total number of setups. Operations added to a single setup (cluster) are divided into different setups.
Lee, et al. [11] propose heuristics based on simulated annealing and tabu search to obtain good solutions for sequencing operations to minimize the sum of machine, setup and tool change costs. Machine, Tool or Setup changes are required when two adjacent operations use different machines, tools or setups respectively. The paper also provides the formulas that are used for calculating the machine, setup and tool change costs.

The precedence relation among operations, the constraint for the problem, is considered as a tree and the path from root to leaf is considered as a precedence sequence. Precedence directed graphs are used to eliminate the infeasible sequence in the tree. The initial solution is obtained by greedy algorithms by sequentially adding the best operation to the last operation of each partial sequence with respect to the sum of machine, setup and tool change costs.

Two different search techniques, namely simulated annealing and tabu search, are used to move away from local minimum to global minimum. Simulated Annealing tries to move from its current position to a neighborhood solution that guarantees a decrease in the objective function value. Like Simulated Annealing, Tabu search also starts with an initial solution and moves to the neighborhood solution, but tabu search moves to the next step even if the solution is worse than the present one. A termination condition is also described.

The authors state the tabu search algorithms are better than the simulated annealing algorithms on overall average. The present work does not use heuristics to cluster or sequence the operations. Instead geometric data and precedence information is used. Also, manufacturing costs are not taken into consideration; rather it is proposed that
manufacturing costs are reduced as a result of a reduction in manufacturing time. But there are many other factors that influence manufacturing costs. All these factors are not taken into consideration in the present work.

Oral and Cakir [14] describe a system that automatically selects the tools required to manufacture rotational parts. An optimization method based on “Rank Order Clustering” is used to sequence tools and operations for minimum number of tool changes to reduce the manufacturing time. First step in the process is selection of the tool. Selection of tools is done taking into consideration various factors like work-piece material, chemical and metallurgical state, machine tool characteristics, and cutting tool characteristics. After tool selection, optimal tool and operation sequence is done with “diameter matrix”. In diameter matrix, rows are tool codes and columns are for feature numbers on the work-piece. At the end, the tool order that provides a minimum number of tool changes is established. So an optimization method is used to sequence the operation and tool order to manufacture rotational parts. But in the present work, geometric information is used to cluster the operations and precedence information is used to sequence them. Moreover the present work is developed for prismatic parts.

2.2 Reducing Number of Tool Changes

Macchiaroli and Riemma [6] propose a system that reduces the number of tool changes for a part program used in a flexible manufacturing system. Reducing the number of tool changes in Flexible Manufacturing Systems managed by Tool Handling Systems provides the machining centers with appropriate tools when required. By using a Tool Handling System, tool duplication can be reduced, tools from the machining centers
can be removed when that tool is no longer necessary for production. But such systems face the synchronization problem when moving tools between different machining centers. The authors propose a system where, by reducing the number of tool changes during the part program, the overall cutting time can be reduced.

The clustering algorithm presented in the paper identifies the part programs that require similar tools, i.e. all the parts in a batch are divided into groups such that the common tools required by the groups are minimal. After partitioning the batch, a balancing procedure is introduced to reduce the imbalance among the machines. The final stage of clustering is an iterative re-clustering procedure applied to reduce the tools on clusters having too many tools in common. So the three-step process groups the parts into clusters to reduce the number of tool changes for the part programs. After clustering, a sequencing algorithm is applied to sequence the movement of the parts to complete the process plan.

Mahavadevan, et al. [12] propose an approach to tool selection and sequencing for roughing stage of three-axis machining. The objective of the paper is to select a tool sequence that minimizes the total rough-machining time. Selecting appropriate tool and generating toolpaths are the two important steps of three-axis rough machining. The first step, tool selection, depends on tool geometry and material removal rate. In order to select the appropriate tool for machining, the part is divided into slabs. This reduces the problem into a series of two axis problems. Then an algorithm is used to identify the largest possible cylinder in the slab. Depending upon the size of the cylinder identified the tool is selected. While identifying the tool to machine the slab, tool change time,
cutting portion of tool are also taken into consideration. Then integer linear programming and network flow formulation are presented to solve the problem.

2.3 Data Representation and Exchange (XML and JNI)

Mervyn, et al. [13] proposes a system for flawless integration of the various products and process design systems or modules. The system uses a manufacturing application that acts as a middleware between a geometric modeling server and application clients. Software components like Java and XML are used to make sure that the middleware is portable across multiple platforms. These interfaces are not known during the development of an application. But using XML for data representation can help application developers to concentrate on domain related problems instead of integration problems.

The most important part of the manufacturing application is the software application that integrated the modeling kernel and other applications. The modeling kernel written in C programming language and other applications in the system is developed using Java. So a special feature of Java called JNI is used. JNI allows Java code that runs within a Java Virtual Machine to operate with applications and libraries written in other languages. In the system explained, data is transferred to the user via a HTTP server.

Due to the client-server approach used in the system, transferring data from the modeling kernel server to the application client may lead to some performance problems. So a neutral data format should be used for data exchange. So an XML file stored in HTTP Server is used to store the product data information. This file contains all the information about the part. This XML file works as an efficient means of data exchange.
The file can be extended to include any other information required at a later stage without affecting the system. So when an application client requests any geometric data the information is transferred from the server to the client in an XML file. In this way data exchange is achieved between the systems. This type of data exchange is required when product development is carried out by geographically distributed applications.
3 Methodology

This chapter describes the methodology used for clustering and sequencing of manufacturing operations from the CAM System. The approach includes CAM data transfer (operation tree structure) from the CAM System to the IMPlanner System, new setup creation, operations sequencing and data structure update in Unigraphics. This chapter also gives a detailed description of various tools that are used in the present implementation to successfully overcome the data transfer problem between two heterogeneous systems.

This chapter starts with a description of the IMPlanner system, explaining various modules of the system that are used in the present work. A description of the CAM module is also given in the section. Section 3.1 explains various tools used in the present work. Section 3.2 describes the Unigraphics operation creation. Section 3.3 and Section 3.4 explain the procedure followed to retrieve Unigraphics operations and CAM roots from Unigraphics to the IMPlanner Module. Section 3.5 explains the algorithm used to create new groups. Section 3.6 explains the process clustering and Section 3.7 describes the process of operation sequencing.

3.1 Tools Used

The present section explains about the tools like the IMPlanner System, Unigraphics, UG/Open, and JNI used in the present thesis.
3.1.1 IMPlanner System

The important modules of IMPlanner System are the process plan module, the feature mapping module, the feature module, the process selection module, the manufacturing module, the process visualization module and component interfaces.

“IMPlanner system relies on existing software tools for CAD/CAM and CAPP, provides distributed processing of process plans across enterprise and virtual enterprise in an intranet/internet environment, utilizes available technology and emerging standards for internet computing (namely Java and its tools JNI, Sockets, and RMI), and utilizes standards for data storage and exchange (relational databases and XML). Those properties enable evolutionary development of the system and incremental transfer of the technology into enterprises. Due to software development tools selected for the system (namely, Java, HTML, JNI and XML), all of the modules may exist on different computers with various operating systems and in geographically distant locations.”[15] IMPlanner system architecture is shown in Figure 3.1 along with the interaction of various modules of the system. The present implementation is also highlighted.

“3I-PP system is a knowledge-based application that provides information about process selection, machine and tool constraints, and cutting parameter selection. The 3I-PP system generates processes for the features which are represented in the 3I-PP schema. Data is exchanged between the 3I-PP system and the process plan module in a client-server environment where the 3I-PP knowledge-based system represents the server and the process plan module acts as a client that initiates a request to the server. The
process plan model interacts with the process plan network module to generate a process network.”[9][10]

“All the models are developed using object-oriented modeling. Characteristics of object modeling such as data encapsulation, inheritance and method overloading have been utilized. For process and feature, a hierarchy of classes has been implemented in order to provide a smooth link with the existing knowledge-based process planning system (3I-PP system). The Feature model has a link to the CAD system and provides reference to the part geometry. The IMPlanner system has been integrated with CAD and CAPP systems to utilize existing expertise in geometric modeling, part programming and
process planning explained above. CAD and CAPP systems used in the integration process provides specialized knowledge in a given domain and implementation in the desired functionality."[15]

Component interfaces are used to transfer data between different modules of the system. These interfaces act as the connectors between all the modules that are otherwise independent.

**CAM Architecture:** The CAM Module of the IMPlanner System is implemented as a part of the present research. The CAM Module needs to communicate with other modules of the system to get information that is needed in creating manufacturing objects. For example, the CAM Module needs to communicate with the feature module to get information about the geometric nature of the part on which manufacturing operations are generated.

Figure 3.2 shows the way in which the algorithm works to create a new setup, transfer operations to newly created setups and sequence operations within each setup. As shown in Figure 3.2, the user creates a feature-based model in the CAD System. This part file is given as input to the CAM System which is then transferred to the CAM Module of the IMPlanner System. The CAM Module then analyses the geometric information of all the operations in the part and creates new groups for all the setups that are not already addressed by the part. After creating the new setups, operations are moved to the newly created setups. While moving the operations, operation precedence is taken care of by the Process Sequencing Module. After sequencing the operations, the newly created data is again transferred to the CAM System. Figure 3.2 also shows a second way to create new
processes and setups, transfer operations to newly created setups and sequence operations in each setup. As shown in Figure 3.2 the user creates a part. This part is given as input to the feature mapping module of the IMPlanner System. The Feature Precedence Network is created from the features created by the Feature mapping module. After creating the feature precedence network, manufacturing processes are created in the 3I-PP Process Selection module for each feature in the feature precedence network. The newly created processes are used in the IMPlanner Process Module to analyze the geometry associated with them and create new geometric groups. The processes are then transferred to the newly created groups.

![Figure 3.2  CAM Architecture](image-url)
While transferring the processes, sequencing is taken care of by the Process Sequencing module. After sequencing the operations, the newly created data is transferred to the CAM System and part and operation data in the CAM System is updated.

3.1.2 Unigraphics

Unigraphics is a software application used in various stages of computer integrated manufacturing. It is used as a CAD tool to model the part, CAE tool to analyze the mechanical, thermal and structural properties of the part, CAM tool to generate toolpaths and NC code to manufacture the part.

3.1.2.1 UG/Open API

“UG/Open API is a collection of routines that allows programs to access and affect the Unigraphics Object Model. In addition to the routines, there are tools provided to compile and link programs in a manner consistent with what Unigraphics expects”.

Unigraphics allows the users to access its object model and make changes to it. Changes to the graphics interfaces of various modules of Unigraphics can also be done. Third party applications can also be integrated to Unigraphics. All these integration and customization is possible by an open architecture provided by Unigraphics. This open architecture is called UG/Open.

The reason to provide such an open architecture is to help different teams to cooperate with each other more effectively on designing, developing, and manufacturing
mechanical products. The next section gives a description about the syntax of UG/Open programs and functions.

**UG/Open API Functions:** The following description provides information on the UG/Open API function naming convention, arguments, and prototypes [17]. The content in this section is from [17] and is given here to provide user a better understanding of UG/Open API. The majority of function names are of the form UF_<area>_<name> where the following apply:

<table>
<thead>
<tr>
<th>UF_</th>
<th>Denotes UG/Open API Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;area&gt;</td>
<td>This part is usually an acronym for the application or area of functionality. For example, MODL and CAM are acronyms for the Modeling and Manufacturing functions respectively.</td>
</tr>
</tbody>
</table>

| <name> | This part is broken down into a verb and noun separated by an underscore. For example, UF_MODL_create_block1 is the UG/Open API Modeling routine to create a block. |

The functions conform to the ANSI/ISO C standard and are prototyped in header files in the following form.

<return data type> <function name> (argument list)
Table 3.2 UG/Open Argument Types

<table>
<thead>
<tr>
<th>I</th>
<th>Input arguments read by the function.</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>Output arguments written to by the function.</td>
</tr>
<tr>
<td>OF</td>
<td>Output Free arguments are written to by the function and the programmer must reallocate memory for the argument with a call to the specified function to free the memory (e.g. UF_free, UF_free_string, etc.)</td>
</tr>
</tbody>
</table>

The following is the prototype for a function:

\[
UF\_PART\_open
\]

\texttt{Internal \& External}

\[
\text{int } UF\_PART\_Open \text{ (char*part_name, tag_t*part, UF\_PART\_load\_status_t*error\_status)}
\]

Table 3.3 UG/Open Function Prototype

<table>
<thead>
<tr>
<th>Parameter</th>
<th>I/O</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>part_name</td>
<td>I</td>
<td>Char*</td>
<td>Name of part to retrieve</td>
</tr>
<tr>
<td>Part</td>
<td>O</td>
<td>tag_t*</td>
<td>Tag of the retrieved part or NULL_TAG if part retrieval fails.</td>
</tr>
<tr>
<td>error_status</td>
<td>OF</td>
<td>UF_PART_load_status_t*</td>
<td>Structure filled with names and associated error codes of any parts that did not load correctly.</td>
</tr>
<tr>
<td>Return</td>
<td>O</td>
<td>Int</td>
<td>0 = no error</td>
</tr>
</tbody>
</table>
The above function retrieves an existing UG part into the session and makes it the work and display part. For this function all of the arguments are pointers, a pointer to a character string, an object identifier (of the part), and a load status structure. The declarations that are required and the procedure to pass the arguments to the function are described below. This is a code fragment only. Before calling this function, the following declarations should be made:

```c
char partname[MAX_FSPEC_SIZE+1];
tag_t part;
UF_PART_load_status_t load_status;
int err;
err = UF_PART_open (partname, &part, &load_status);
```

### 3.1.2.2 UG/Open Files and Functions

This section describes a few functions that are defined in UG/Open APIs that are used in the present implementation for querying and modifying the manufacturing data of the part. These methods are used to communicate with the Unigraphics part file.

UG/Open functions are defined in C/C++ Source and Header files. The commonly used header files of UG/Open are uf.h, uf_cam.h, uf_camgeom.h, uf_oper.h, uf_obj.h, uf_part.h, uf_ncgroup.h, uf_group.h, uf_object_types.h, uf_cambnd.h, uf_defs.h, uf_part_types.h, uf_part_ugopenint. These files define various functions that can be used to query any Unigraphics part for its model data. Description of the files provided below is based on [17] and is given here to provide user a better understanding of UG/Open interface used in this thesis.

**uf.h**: This file defines functions that will be common to most UG/Open programs, as well as specialized functions that do not belong to any of the other interfaces.
uf_cam.h: This file contains the information required to use the UF_CAM module. This module allows the UG CAM User Function programmer to interface with the current CAM session object.

uf_camgeom.h: This file contains UG/Open interface to geometry in CAM. The CAM geometry routines in this file enable the user to append entities to CAM geometry, get CAM geometry items, delete CAM geometry, get the item entity, get the item application data, set the item application data, or delete a single item from CAM geometry.

uf_oper.h: This file enables the user to perform functions which are particular to operations as compared to the functions which can be performed on Operations when an operation is viewed as a UF_PARAM object.

uf_ncgroup.h: The UF_NCGROUP module allows access to CAM UF_NCGROUPs. A CAM UF_NCGROUP is a collection of objects. Each object is either a UF_OPER object or a UF_NCGROUP object. This definition is intentionally recursive. The functions for dealing with UF_NCGROUPs allow for operations like asking for the members of a UF_NCGROUP, adding a new object to a UF_NCGROUP and asking if an object may be added to a UF_NCGROUP.

uf_object_types.h: Define names of all UG object types and subtypes. For example:

```
UF_machining_operation_type                  100
UF_machining_geometry_grp_type              105
UF_machining_mach_tool_grp_type            106
UF_machining_geom_type                           111
```
UF Machining_null_grp_type

UF_MACH_POCKET_SUBTYPE = 110
UF_MACH_CAVITY_MILLING_SUBTYPE = 260;
UF_MACH_FACE_MILLING_SUBTYPE = 261;
UF_MACH_ZLEVEL_MILLING_SUBTYPE = 263;
UF_MACH_DRILL_SUBTYPE = 10;

**uf_cambnd.h**: This file contains the UG/Open interface to boundary in CAM. The CAM boundary routines in this file enable the user to append boundaries created from a planar face, append a single boundary created from a list of curves/edges, get boundaries, delete all boundaries of a geometry type, delete a single boundary, get boundary data, get boundary application data, set boundary application data, get boundary items, get item entity, get item application data, get boundary group application data (offset), set boundary group application data (offset), get item group application data (offset), set item group application data (offset), or determine whether boundary geometry is inherited.

**uf_defs.h**: This file contains UG/Open definitions. These are general typedefs and macros required by all UG/Open interfaces. Typedefs that are specific to a particular module will be placed in a header file for that specific module.

The above mentioned files and functions are used in this thesis to retrieve and modify the manufacturing data of the Unigraphics file.
3.1.3 Java Native Interface

“The Java Native Interface (JNI) is the native programming interface for Java that is part of the JDK. The JNI allows Java code that runs within a Java Virtual Machine (VM) to operate with applications and libraries written in other languages, such as C, C++, and assembly language.”[18]

JNI is used to write methods in native languages in the following situations:

- JNI is used to solve some problems that are better handled outside the Java environment.
- Applications may be written in other programming languages. But if a Java application tries to interact with it, then there should be a mechanism in either of the applications to make the interaction possible. This is achieved by JNI in Java.
- All the features in Java API may not have the required performance. In such a case, application can be written in a native code and can be used by Java application whenever required. For example, time-critical code can be written in low-level languages and Java application can use it whenever required.

3.1.3.1 Need to Use JNI

Unigraphics is developed in the C environment. The IMPlanner System is developed in Java. Unigraphics is a good modeling tool that allows the creation of toolpaths for operations and later post-processes the same toolpaths to generate NC code to manufacture the part. The IMPlanner System is a CAPP system that is built on Object Oriented Principles. So in order to use the advantages of both the systems, there should be a way in which the manufacturing data in Unigraphics should be brought to the
IMPlanner System to prevent “re-inventing the cycle” of performing the clustering and sequencing of operations in Unigraphics. Java Native Interface can be termed as the perfect solution to achieve the above.

3.1.3.2 Implementation of JNI

The first step in implementing the Native Methods Interface is generating the header file for the Java Native methods. This is done by running the `javah` utility on the Java class file that has Native methods declared in it.

For creating a header file with `javah` utility, class paths of all the classes that the present class uses should be given. In a batch file, all the commands can be saved and executed sequentially. Once the batch file is executed on the Java class file, a C header file (.h extension) containing the signatures of the Java methods declared as native in the Java class is created.

Figure 3.3 shows the header file created as a result of executing the batch file. The signatures declared in the header file are the signatures of the corresponding Java Native methods.

For example,

```c
JNIEXPORT jint JNICALL Java_edu_ohiou_implanner_cam_CAMSystem_createGeometriGroup(JNIEnv*, jobject,jint,jint)
```

is the header file methods declaration of the native method

```java
private native int createGeometricGroup(int geometricGroupTag,int workPieceTag);
```

where

`geometricGroupTag` is the unique tag used to identify the GeometryGroup in Unigraphics
`workPieceTag` is the unique tag used to identify the work-piece in Unigraphics
Figure 3.3 Header File Containing the Signature of Java Methods

This header file acts as the channel for the control to go from Java classes to the C method written as native code. Figure 3.4 shows the `createGeometricGroup` function written in C.

The native code that has Unigraphics methods is written in C. The C file shown in Figure 3.4 is used to generate a dll application. This application is loaded into the Java environment when the Java class is loaded into JVM.
Creating Manufacturing Operations

Operations in Unigraphics are created to generate toolpaths to manufacture the part. Figure 3.5 shows the method for accessing the manufacturing module in Unigraphics.
3.2.1 Manufacturing Module

Operations are created in the “Manufacturing Module” of Unigraphics. Any “Part” that is modeled in Unigraphics needs to be opened in the manufacturing module to be able to access various manufacturing functionalities available in Unigraphics. Figure 3.6 shows the various setups available during the initialization process.

![Figure 3.6 Various Setups during Initialization](image)

A default setup can be selected when the part is opened for the first time in the manufacturing module. The various setups that are available during the selection are mill_contour, mill_multi_axis, drill, hole_making, turning, and wire_edm. The result of
opening the manufacturing module can be seen in Figure 3.7. Various toolbars that are used in generating toolpaths can also be seen in the Figure 3.7.

![CreateOperation Toolbar](image)

**Figure 3.7 CreateOperation Toolbar**

### 3.2.2 Creating Operation

The first step in the process of creating operation to manufacture any part is selecting the “Type” of Operation and various groups. Figure 3.8 shows the dialog box that is used for selecting the operation type and groups. Each operation in Unigraphics is also
associated with a program, machine tool, method, and cutter group [17]. Depending on
the setup a pre-defined set of templates are loaded for operations.

![Figure 3.8 CreateOperation Dialog Box](image)

Each operation is also assigned to a set of parameters in Unigraphics Object Model. These parameters are OPERATION NAME, OPERATION TYPE, and OPERATION TAG.

Every object in Unigraphics is identified by a unique tag. So operation is Unigraphics
is identified by OPERATION TAG. Similarly operation type and operation name of the
operation are given by OPERATION TYPE and OPERATION NAME respectively. As
mentioned earlier each operation is also assigned a Program Group, Geometry Group,
Tool Group and Cutter Group. These groups are explained in the following sections.
3.2.2.1 Program Group Hierarchy

The view shows the order in which the operations will be executed on the machine tool. This order is used for output to the post processor. This view helps to group all the operations in the setup in the order they are created by the user.

3.2.2.2 Geometry Group Hierarchy

The operations in this view will be shown according to the common machining geometry or MCS each operation uses. Geometry Group View groups all the operations in the part according to geometry groups. Geometry groups are used to share geometry and orientation entities among operations. The contents of the groups are inherited by the groups and operations below them in the Navigator tree. Planning the NC program in Geometry view instead of Program Order View can be useful if the NC Program is based on geometry. The advantage of using the Geometry View is that the setup can be organized for geometric features. The order of Geometric Groups does not matter, but the relative order of operations in each geometric group matters because it affects the tool change sequence within each setup. The following types of Geometry Groups can be created depending on the specified CAM Setup [17]:

MCS: The Machine Coordinate System is the base location for all subsequent toolpath output points. Each Mill Orientation group defines the MCS and associated clearance plane necessary to machine a specific side of the Part.

Mill Geometry/Work-piece: The Mill Geometry and Work-piece allow defining Part, Blank, and Check geometry from selected bodies, faces, curves, or surface regions. In addition, they allow defining Part thickness, and Part Material.
Mill Boundary: The Mill Boundary allows defining Part, Blank, Check and Trim boundaries which when used in various combinations, define areas that constrain cutting moves. These areas may be defined by either a single boundary that contains the tool or by a combination of boundaries that both contain and exclude the tool. The behavior, usage, and availability of boundaries differ depending on the machining module in which they are used. All boundaries, however, have certain common characteristics regardless of their application.

Mill Area: The Mill Area allows specifying the surfaces to machine by defining Part Geometry, Check Geometry, Cut Area, and Trimming boundaries.

3.2.2.3 Tool Group Hierarchy

Tool Group hierarchy organizes operations according to the same cutting tools. New tool groups can be created by creating a tool from a template or by getting from a library.

3.2.2.4 Method Group Hierarchy

All the operations that have the same machining method in the setup are grouped together in this view.

Tool Group hierarchy and Method Group hierarchy for operations are created in Unigraphics, but Program Group hierarchy and Geometry Group is updated with the framework developed in the present work.

3.2.3 Operation Details

After selecting the type and groups of the operation, the next step is to enter needed operation parameters that are used for toolpath generation.
These parameters can be divided into two types:

3.2.3.1 Geometric Parameters

Geometric Parameters are Part geometry, Floor geometry, Check geometry and Blank Geometry that the operation is going to machine. These parameters also depend on the type of operation selected. For example, if contour milling is selected, then only part, blank and check geometries are required. But for planar milling in addition to part, blank and check geometry, trim and floor geometries are required. All the above-mentioned geometries, part, check, trim, floor, check geometries are not all required to generate toolpath for an operation. Any combination of options that completely define the geometry required to generate toolpath for the operation is enough. The geometries required for each type of operation is shown in Figure 3.9. For example, geometry for the ZLevelMilling operation can be specified with Part, Blank and/or Check geometries, while geometry for the Planar Milling operation can be specified with Part, Blank, Check, Trim and/or Floor Geometries.

Definition for each geometry is given below [17]:

- Part Geometry is used to specify geometry that will represent the finished part.
- Blank Geometry is used to specify geometry or faceted bodies that will represent the raw material to be cut. Blank geometry does not represent the final part and can be cut through or engaged into directly.
- Check Geometry is used to define geometry that should not be violated such as clamps that hold the part.
Figure 3.9 Geometries Required for Planar and Contour Milling
• Face Geometry is used to machine closed boundaries whose inside material indicates the areas to be machined.

• Floor Geometry defines the lowest (last) cut level. All cut levels are generated parallel to the Floor plane. Only one Floor can be defined per operation. Redefining the Floor automatically replaces the existing Floor.

3.2.3.2 Machining Parameters

After selecting the geometric parameters, the next important set of parameters that are vital in generating the toolpath are machining parameters. These parameters are important in developing NC code from toolpaths as they define the kinematics details of the operation. Figure 3.10 shows the machining parameters entered during operation creation.

3.2.4 ToolPath Generation

After entering all the details of the operation, the next step is to generate the operation. Generating the operation means a generation of toolpaths taking into consideration all the parameters entered in the previous steps. Figure 3.11 shows the toolpath generation interface.

The toolpath that is generated from the operation can be seen on the part in Figure 3.12. The visual display of the toolpath can be used to correct any errors and regenerate the toolpath by selecting different values for the parameters discussed earlier.

A Unigraphics part that has its operations generated as mentioned in the section is used as input to the system mentioned in this thesis.
Figure 3.10 Parameters Required to Generate Toolpath

Figure 3.11 Generating ToolPath
3.3 Retrieving Part Model and UG Operations

The first step in the process of clustering and sequencing is transferring all the details of manufacturing operations, created in the CAM System, to the IMPlanner System and creating corresponding Java objects. This process can also be seen as the data exchange between the CAM System and the IMPlanner system. In order to facilitate the data
exchange with other applications, Unigraphics provides UG/ Open API that consists of classes written in C environment. Java provides an API called the Java Native Interface (JNI) that allows any Java Application to interact with other applications in different environments. So both Unigraphics (CAM System) and IMPlanner have tools to interact with applications that are developed in different environments. UG/ Open API of Unigraphics (CAM System) consist of few predefined methods that can be used to access the underlying data structure of a part file. The methods can be used to retrieve information and make changes to the part file. As a part of this step, the corresponding reference of Unigraphics Part Geometry called Part Model is created in IMPlanner. For transfer of part geometry, the procedure described in [19] is used.

The unique tag of the operations in Unigraphics is used while creating objects of type CAMOperation in the CAM module of the IMPlanner System. Whenever any information about any “CAMOperation” is needed, the unique tag number of the operation is used to retrieve information from the Unigraphics data structure. In this way, unique tags of objects are used by the CAM module to retrieve information stored in Unigraphics data structures.

3.4 Retrieving CAM Roots

In Unigraphics, operations of a part are organized into four different hierarchical tree structures, namely Program Order, Geometry, Machine Tool and Machining Method. These four different perspectives each have a root that acts as the root-node of the tree shown. The operations along with their groups become the children of the root node. In
the present work, new group creation and operation sequencing in the roots is done only in Program and Geometry views. Tool group and Method group hierarchies are created in Unigraphics itself.

The output of this step is the creation of all the four corresponding hierarchies on the IMPlanner side. Each hierarchy is treated as an instance of JTree and the root of each view is considered as the root of the tree. In this step, four different tree structures are created as Java Objects that correspond to the four views that are present in Unigraphics. The tree structure has operations and their groups as its children. All information about the groups and operations is available in CAM Module at the end of this step.

3.5 Creating MCS Groups

In the first two steps, complete manufacturing data, operations and CAM Roots that are created in Unigraphics are retrieved. In this step, new geometric groups are created and added to the geometric group root that is retrieved from Unigraphics. The procedure consists of the following steps:

- Retrieval of geometry associated with each operation. The retrieved geometry may contain faces, edges or features.
- Computation of a tool axis vector from the referenced geometry associated with the operations.
- Creation of new geometry groups for each distinct tool axis vector. Each geometry group (setup) is associated with a normal vector that can be treated as the tool axis vector for it.
- Clustering of operations with the same tool axis vector to the same group.
The algorithm for calculating the tool axis vector for an operation depends on the type and subtypes of operation. Unigraphics operations that belong to mill_planar, mill_countour and drill types are the subject of this thesis. These modules have the following subtypes:

- **mill_planar**
  - FACE_MILLING
  - PLANAR_MILLING
  - PLANAR_PROFILE
  - ROUGH_FOLLOW

- **mill_countour**
  - CAVITY_MILLING
  - ZLEVEL_FOLLOW_CAVITY
  - ZLEVEL_FOLLOW_CORE
  - ZLEVEL_ZIGZAG
  - ZLEVEL_PROFILE

- **drill**
  - SPOT_DRILLING
  - DRILLING
  - PECK_DRILLING
  - BORING
  - REAMING
  - COUNTER_BORING
Figure 3.13 shows the three dialogs for three operation types that are the subject in the present thesis. The figure also highlights the subtypes of the respective operations. The subtypes that are highlighted in Figure 3.13 are in the same order as mentioned above i.e. subtypes of “mill_contour” operation are CAVITY_MILLING, ZLEVEL_FOLLOW_CAVITY, ZLEVEL_FOLLOW_CORE, ZLEVEL_ZIGZAG, ZLEVEL_PROFILE. Unigraphics stores operation machining geometry as edges for PlanarMilling and FaceMilling and as faces for ZLevelMilling.

3.5.1 Tool Axis Vector for Volume Milling Operation

For an operation of type “ZlevelMilling”, geometry that is machined by the operation is given as a set of faces associated with the milling volume. As a result, the above algorithm is executed as follows:

**Geometry Retrieval:** Faces are retrieved from Unigraphics and stored in CAMOperation.

**Computation of tool axis:** For each face, its corresponding Geometric Feature is identified. The procedure assumes that for a single operation, all faces belong to the same Geometric Feature. For each geometric feature, the corresponding manufacturing feature is identified. The mapping algorithm has been developed as a part of the Feature Mapping Module of the IMPlanner System. For that feature, a normal vector is established as a tool axis vector for operation.

**Geometry Group Creation:** A tool axis vector for operations is used to search existing geometry groups. If the group with the same MCS vector is identified, then the current
operation is added to the group. If the group with the new tool axis vector is not found, the new geometry group is created and the current operations added to it.

For example, if ZLevelMilling Operation for top block is considered, the geometry machined by the operation is given as a set of faces. Figure 3.14 shows the set of faces given by UG/Open as geometry for ZLevelMilling operations. From the set of faces retrieved, the feature associated with each face is retrieved. Figure 3.15 shows the tool axis vector that is assigned for ZLevelMilling operation that machines the four faces of the Top Block.

![Figure 3.13 Operation Types and Respective Subtypes](image)
Figure 3.14 Feature Selected and Geometry Retrieved
3.5.2 Tool Axis Vector for Face Milling and Planar Milling Operations

For PlanarMilling and FaceMilling operations, the geometry is stored as a list of edges in a UG part file. As a result, the above algorithm is executed as follows:

**Geometry Retrieval:** The edges that represent the geometry are retrieved to the CAM Operation data structure. Then, all the faces of the part are looped through and those faces whose edges are in the retrieved list are identified. Each edge can be associated with more than one face. But only those faces whose edges are in the retrieved edges list are retrieved.

**Computation of tool axis:** For each face, its corresponding normal vector is identified. Due to the definition of face and planar milling operations, all the retrieved vectors are in
the same direction. As a result, only one vector is assigned for any face milling operation. The normal vector that is identified is established as a tool axis vector for operation.

**Geometry Group Creation:** A tool axis vector for operations is used to search existing geometry groups. If the group with the same MCS vector is identified, then the current operation is added to the group. If the group with the new tool axis vector is not found, a new geometry group is created and the current operations added to it. Similar to “ZlevelMilling”, the references to a geometry group and program groups of the operations are updated while creating the new groups.

For example, for FaceMilling of Boss top, geometry is given as edges (in this case small and large circles). The face that contains both the edges is shown in Figure 3.16.

![Figure 3.16 FaceMilling of the Boss Top](image-url)
So by retrieving the normal vector of the face, the tool direction of the operation can be decided. Figure 3.17 shows the retrieved normal for FaceMilling that is used to machine the Boss Top face.

![Figure 3.17 Tool Axis Vector Retrieved for FaceMilling](image)

3.5.3 Tool Axis Vector for Drilling

Operations of the drilling type are considered in a special way. Whenever a drilling operation is encountered, the user is prompted to enter the hole-axis vector due to the limitation in receiving geometric data from UG/Open for drilling operations. The vector that the user enters is treated as the axis vector and the operation uses that for all the steps to follow.
So, for ZLevelMilling, features are retrieved from faces and feature normal vectors are treated as operation tool axis vectors. But for PlanarMilling and FaceMilling, faces are retrieved from edges and face normal vectors are directly treated as operation tool axis vectors. After calculating the tool axis vectors for each operation, geometric and program groups whose tool axis vectors are equal to one of the retrieved normal vectors are created. For every tool axis vector, a geometric group and program group exists at the end of this step.

3.6 Clustering of Operations

After creating the geometry groups in IMPlanner and arranging all operations in clusters, the next and final step in the process is to transfer this information for each operation from IMPlanner to Unigraphics. This is done by looping through all the members of the geometric root. The members may be other groups or operations. If the member is a group, then the group is considered as a root and all the members of the group are looped until a member of that type of operation is found. If the member is an operation, then the geometric group and program group of the operation are retrieved and transferred to Unigraphics. The operations are then moved from the default group to the group that is retrieved from the operation. This is possible because the references to the geometric group and program group of operations are updated in the Unigraphics part file at the end of previous step.
3.7 Operation Sequencing

The sequencing of the operations within each group is taken care of while adding the operation to the groups as a result of functionality of the operation in the CamSystem. A neutral data representation model is used to store operation precedence information that is used as the criteria for deciding operation sequence. The information is then used by the CAM Module at runtime. So potentially any criteria for operation sequencing can be used by the system.

An operation with a lower precedence value appears before any other operation with higher precedence in the same group. The operations in the precedence information file are CAM Module counterparts of operations in the CAM System. So, while moving an operation from the default group to its new group, each operation is checked with precedence information. If the operation being moved precedes any other operation of the same group, then it is added before the other operation. In this way, operation sequencing is taken care of while moving the operations to the new groups. Figure 3.18 shows the pictorial view of the data that is stored in the data representation model.

Figure 3.18 Operation Precedence
4 Implementation

This chapter describes the various steps of implementation. The first part of the chapter describes various Java classes of the IMPlanner System that are used in the present thesis. The second half of the chapter describes the Graphical User Interface that was developed to control the flow of the clustering and sequencing process.

The chapter starts with a description of other modules of the IMPlanner System that are used in the present thesis. Section 4.2 describes the CAM Architecture. Section 4.3 is the description of the Java classes of CAM module that are used in the development cycle. The final section explains the flow of control of the system in the clustering and sequencing process.

4.1 IMPlanner Modules

The major packages of the IMPlanner System used in the present work are

\texttt{edu.ohiou.implanner.features}, \texttt{edu.ohiou.implanner.geometry}.

4.1.1 Features Package

The \texttt{edu.ohiou.implanner.features} package consists of classes as shown in Figure 4.1 that represent the manufacturability features defined in Unigraphics. \texttt{MfgFeature} is the parent class that defines the common functionality of all the features. Various features that are represented in this package are \texttt{Slot}, \texttt{Pocket}, \texttt{Hole} and \texttt{FeatureSet}. Some of these classes also act as parent classes to define a few more classes that represent specific features in Unigraphics. For example, \texttt{FeatureSet} is a parent class of \texttt{CircularFeatureSet} that corresponds to the \texttt{circularFeature} instances created in Unigraphics.
4.1.2 Geometry Package

The `edu.ohiou.implanner.geometry` package is used to define classes that represent the geometry of a part. The Geometry Package in the IMPlanner System serves only as a reference to the part geometry within the Unigraphics CAD System. The main classes that are used in the present implementation are shown in Figure 4.2. Edge and Face classes are used to represent an Edge and Face of a part.
So, objects of type Edge and Face are created for every edge and face of the Unigraphics part. These classes extend \textit{GeomObject} which in turn extends \textit{CADObject}. Another class that is used in the present implementation is \textit{PartModel}. This class is used to store the details of the Unigraphics Part in the \textit{IMPlanner} Module.

4.2 CAM Architecture

The architecture, as shown in Figure 4.3, has been implemented in this thesis as an extension of the IMPlanner system.

The CAM module in IMPlanner, also known as the Manufacturing module, is responsible for developing machining data. This module is also implemented on Object Oriented principles using inheritance, polymorphism, and method overloading as basic ideas. In the module, \textit{CAMSystem} and \textit{CAMPartModel} are the classes that are used to communicate with other modules of the system. The CAM module needs to communicate with other modules to get information that is needed in creating manufacturing objects. For example, the CAM module has to communicate with the Feature module to get information about the geometric nature of the part on which manufacturing operations are generated. \textit{CAMSystem} and \textit{CAMPartModel} classes in turn use other classes from the CAM module to successfully carry on the manufacturing task. \textit{NCGroup} and \textit{CAMOperation} are the classes in the CAM module that are used to store information about manufacturing specific entities.

\textit{NCGroup} and \textit{CAMOperation} classes are further extended by classes to add functionality that differs from the parent class. The Program Group and Geometry Group represent NCGroups, to which the manufacturing operations are added for NC code
generation in the CAM system that is used in the IMPlanner System. These two classes extend \textit{NCGroup} classes of CAM module. The important classes of CAM module that are used in the present implementation are shown in the Figure 4.3. The relationship between the classes is also shown in Figure 4.3. A detailed description of all the classes is given later.

4.3 CAM Package Classes

The IMPlanner System is an Object Oriented System developed in Java. It consists of various classes that can be used in the clustering process.

4.3.1 CAMOperation

This is base class of all the Java objects created after transferring Unigraphics operations into IMPlanner module. For example, Java object of the type \texttt{edu.ohiou.implanner.cam.FaceMilling} is created corresponding to FaceMilling operation in Unigraphics. \texttt{CAMOperation} is the super-class of \texttt{edu.ohiou.implanner.cam.FaceMilling}. Similarly many other classes that correspond to manufacturing operations in Unigraphics inherit from \texttt{CAMOperation}. \texttt{CAMOperation} defines methods that are common to all the camOperations that inherit from it. Some of them include \texttt{setAssociatedNormals}, \texttt{showFaceEdgeRelation}, \texttt{getCAMGeometry}, \texttt{getCAMPartModel}, \texttt{setCAMPartModel}, \texttt{createOperation}, \texttt{getFaces}, \texttt{addFaces}, \texttt{addFace} etc. This class also implements the Comparable interface which defines the compareTo method to facilitate the ordering of the camOperation objects when they are added to a TreeSet. This functionality is used in the populateOperations method of camPartModel
Figure 4.3 CAM Module Framework
while adding camOperation as they are created for every operation tag that is retrieved from Unigraphics. \textit{CAMOperation} extends CADObject class in \textit{edu.ohiou.implanner.geometry}. \textit{CAMOperation} is inherited by three other classes that increase the functionality by adding a few more methods. These classes are \textit{FaceMilling}, \textit{PlanarMiling} and \textit{ZLevelMilling}.

4.3.1.1 \textbf{FaceMilling}

This class is used to store information about the \textit{FaceMilling} operations of Unigraphics. As mentioned in Section 3.5, the geometry of the \textit{FaceMilling} operation in Unigraphics is given as edges. So, a collection called \textit{tagsArray} is maintained to store all the edges that are associated with the operation. A method called \texttt{getCAMGeometry (CAMSystem)} is used to retrieve all the edges of the operation.

4.3.1.2 \textbf{PlanarMilling}

This class is used to store information about all operations except \textit{FaceMilling} that are defined in the “mill\_planar” type of operations in Unigraphics. Similar to \textit{FaceMilling}, Unigraphics gives the geometry of Planar Milling operation as edges.

So \textit{tagsArray} is used to store all the edges that are associated with the operation.

4.3.1.3 \textbf{ZlevelMilling}

This class is used to store information about all the classes that are defined in “mill\_contour” except the “Cavity Milling” operation. In the case of \textit{ZLevelMilling}, Unigraphics gives the geometry associated with the operation as faces. These faces are
stored in the `tagsArray` variable. The role of `CAMOperation` is shown in Figure 4.4 and `CAMOperation` and its subclasses are shown in Figure 4.5.

Figure 4.4 Tasks of CAMOperation Class
4.3.2 PartModel

This class is used to store the geometric information of the Unigraphics part file on the Java side. This class interacts with \textit{CADSystem} to get details of the part from the C environment. \textit{PartModel} has an instance of \textit{CAMSystem} called \textit{camSystem} that is used to call the native methods declared in \textit{CAMSystem}.

4.3.3 CAMPartModel

\textit{CAMPartModel} extends the functionality of \textit{PartModel} to store the manufacturing data of the part along with geometric data stored by the \textit{PartModel}. It defines many methods that call methods in the \textit{CAMSystem} to retrieve and modify manufacturing parameters of the Unigraphics part file.

4.3.4 NCGroup

\textit{NCGroup} is used to store information about Unigraphics NCGroups. Every operation created in Unigraphics is a member of an \textit{NCGroup}. NCGroups can be used to store
geometric and orientation information that is common to all operations, so that all the operations in that group can inherit it. \textit{NCGroup} class in the \texttt{edu.ohiou.implanner.cam} package has a variable of TreeSet instance called \texttt{members} that stores all the members of the \textit{NCGroup}. The members can be either \textit{NCGroup} or \textit{CAMOperation}. This class also defines a method called \texttt{getTree()} that returns all its members in a tree structure. The implementation of \textit{NCGroup} is inherited by four subclasses called \textit{GeometryGroup}, \textit{ProgramGroup}, \textit{CutterGroup} and \textit{MethodGroup}.

ProgramGroup: The subclass of \textit{NCGroup} that represents the program group of Unigraphics.

GeometryGroup: The subclass of \textit{NCGroup} used to represent the geometry group of Unigraphics. \textit{NCGroup} and its subclasses are shown in Figure 4.6.

![Figure 4.6 NCGroup and Its Subclasses](image)
4.3.5 CAMSystem

CAMSystem is used as the bridge between the Java module and the UG/Open dynamic link library. This class declares native methods that converse with the methods written in C. The native methods are declared in the CAMSystem class by using the “native” keyword. One more important point in the CAMSystem class is that all the native methods are private to the class. In order to use native methods in the CAMSystem, public non-native methods in the class should be used. The reason for such an implementation is to prevent sending any erroneous information to Unigraphics. The CAMSystem will invoke the native C methods only when it is sure that the object it is sending is valid, i.e. that a valid corresponding object exists in the Unigraphics data structure.

So the usual flow of commands from CAMSequencingPanel to CAMPartModel to CAMSystem as shown in Figure 4.7.

![Figure 4.7 Data Transfer between IMPlanner System and Unigraphics](image-url)
4.3.6　CAMSequencingPanel

This is the user interface, as shown in Figure 4.8, of the Setup Selection Module. This class extends a ViewObject that has a JPanel variable called panel that accepts any component to be showed in the frame.

The following section explains about the various buttons on the tool-bar and other graphic elements used in the CAMSequencingPanel:

*Open UGFile* button used to open a Unigraphics file and retrieve the manufacturing operations from it.

*Get CAMRoots* button used to retrieve NC roots from the Unigraphics part file.

*Create MCS Groups*: Button used to call the action Listener that creates new geometric groups depending upon the tool axis vectors from the geometry of the operations.

*Cluster Operations*: Button used to move operations to the newly created geometric groups.

*buttonPanel*: This is an instance of a JPanel that contains the above-mentioned buttons in a flow layout. The detailed description of the functionality of each button is given in the next section.

*treePanel*: An instance of JPanel to which a JTree called featureTree that shows the components of mfgPartModel is added.

*mainSplitPane*: A instance of JSplitPane that has treePanel as the left component and another splitPane as the right component.
panel: This is a variable defined in ViewObject and is an instance of JPanel. mainSplitPane that is added to it is in the center. buttonPanel is added to the panel in the North.

Whenever a Unigraphics part file is opened and mfgPartModel is created, the tree structure of the part along with its features is shown in the tree. Another splitPane is added as the right component.

Figure 4.9 shows the flow of control on clicking various buttons on the button Panel.
Figure 4.9 Tasks of CAMSequencingPanel
4.4 Flow of Control

This section describes the sequence of steps that are performed in order to convert the Unigraphics part file from its initial form to a clustered and sequenced part. The process starts with a $CAMSequencingPanel$ class. This class opens a GUI that is used as the interface for the whole process. The GUI consists of various buttons that are used to navigate from one step to the next step in the process.

4.4.1 Retrieving Unigraphics Operations

The first step in the process is opening a Unigraphics file and retrieving Part Geometry and manufacturing operations associated with it. This is done by extending the work in [16]. The first step that is performed on clicking the $openUGFile$ button is creating a $CAMPartModel$ instance that retrieves both geometric and manufacturing information from the Unigraphics file. The Unigraphics part file is passed to the $CAMPartModel$ by giving the file name as a string. Then the $startCAMSession$ method is called on the $camSystem$ object. This method calls the native C method and initializes the manufacturing module of Unigraphics. The session should be created to start a channel through which querying and modification of CAM data can be done. Figure 4.10 shows the code for the execution of steps described.

After the session is created, the $populateOperations$ method of $camPartModel$ shown in Figure 4.11 is called passing a $camSystem$ as a parameter. It is at this place that the $camPartModel$ gets a reference of a $camSystem$ for the first time. $camPartModel$ later uses the same reference to call the native methods that query or manipulate the manufacturing data.
```java
/**
 * ActionListener method that opens a Unigraphics file and
 * populates the Unigraphics operations into the CAMPartModel
 * @param e ActionEvent that is generated when the openUGFileButton
 * is clicked
 */

void openUGFileButton_ActionListener(ActionEvent e) {

    int selection = ugFileChooser.showOpenDialog(null);
    if (selection == JOptionPane.APPROVE_OPTION) {
        selectedFile = ugFileChooser.getSelectedFile();
        if (selectedFile.getName().endsWith(".prt")) {
            saveXMLFileButton.setEnable(true);
            CAMOperationButton.setEnable(true);
            camPartModel = new CAMPartModel(selectedFile.toString());
            mfgPartModel = new MfgPartModel(camPartModel);
            camSystem.startCAMSession();
            try {
                camPartModel.populateOperations(camSystem);
            }
            catch (Exception ex) {
                ex.printStackTrace();
            }
        }
        featureTree.setModel(new DefaultTreeModel(mfgPartModel.getTree(new MfgPartModel.F_PROCESS)));
        mainSplitPane.setDividerLocation(175);
        saveXMLFileButton.setEnable(true);
        CAMOperationButton.setEnable(true);
        rootDetailsButton.setEnable(true);
    }
}
```

Figure 4.10 OpenUGFileButton Action Listener

The control goes to the `populateOperations` method of `camPartModel`. 
The first step that is performed in the `populateOperations` is to retrieve the manufacturing operations from the Unigraphics file. In order to retrieve the operations, `populateOperations` calls the `getOperationTags` native method in the `camSystem`. `getOperationTags` method returns an int array that represents the tags of the operations. The int array is looped through to create a `camOperation` that corresponds to the tag and subtype returned by `getOperationTags`. In order to achieve this, a static method is defined in `camOperation` that takes the subtype of the operation as a parameter and creates a java object.
After creating the *camOperation* object corresponding to the tag and subtype, *populateOperations* calls the *getOperationDetails* method of the *camSystem* by passing the created *camOperation* as the parameter. *getOperationDetails* in turn calls the *getOperationDetails* native method that takes the *camOperation* and *camOperation* tag.

The native method retrieves the name, type and subtype of the operation and updates the object sent to it as shown in Figure 4.12.

```c
/**
 * The getOperationDetails Method has the the Operation object and the operation
 * tag as arguments. It retrieves the attributes of a particular operation
 * i.e., operation type and operation name.
 */

JNIEXPORT void JNICALL Java_edu_chicu_planner_can_CANSystem_getOperationDetails
  (JNIEnv *env, jobject obj, jobject camop, jint opTag) {

  int op_type;
  char opName[UF_OPER_MAX_NAME_LEN + 1];
  jclass cls = (*env)->GetObjectClass(env, camop);
  jfieldID fid;
  jstring jstr;
  tag_t opTag;
  const char *str;

  // Operation Type
  opTag = (tag_t) (*env)->GetIntField(env, cls, "operationType", "I");
  if (opTag == 0) {
    return;
  }
  (*env)->SetIntField(env, camop, fid, op_type);

  // Operation Name
  //UF_OPER_ask_oper_name fpRegTag, cpTag, cpName);
  fid = (*env)->GetFieldID(env, cls, "operationType", "I");
  if (fid == 0) {
    return;
  }
  jstr = (*env)->GetStringUTFChars(env, jstr, 0);
  (*env)->ReleaseStringUTFChars(env, jstr, 0);
  (*env)->SetObjectField(env, camop, fid, jstr);

  return;
}
```

**Figure 4.12** getOperationDetails Method
camOperation object is updated with its name and type by the getOperationDetails method in the camSystem. The next getNCGroups method of the camSystem is called by passing the camOperation as a parameter. This method updates the camOperation with the four NCGroups that a camOperation can be associated with in Unigraphics as shown in Figure 4.13.

```java
/**
 * The getOperationGroups method takes a camoperation as argument.
 * The method will return an integer array of groups for each operation.
 * * Note: Each "tag" is associated to a integer value. Also the operTag has to
 * be passed as an argument as we do not have to reference each field separately from
 * Java if we can give direct reference from "C" program itself.
 * "geomGrp", "progGrp", "methodGrp", "cutGrp" are the 4 Groups defined in Java Class
 */

JNIEXPORT void JNICALL Java_emu_chin_implender_cam_CAMSystem_getOperationGroups
(JNIEnv *env, jobject obj, jobject camoper, jint operTag)
{
    // To define group objects from java class
tag_t group_tag;
    jclass cls = (*env)->GetObjectClass(env, camoper);

    // Geometry Group
    UF_OPERATION_geom_group(operTag, &group_tag);
    fid = (*env)->GetFieldID(env, cls, "geomGrp", "I");
    if (fid == 0) {
        return;
    }
    (*env)->SetIntField(env, camoper, fid, group_tag);

    // Method Group
    UF_OPERATION_method_group(operTag, &group_tag);
    fid = (*env)->GetFieldID(env, cls, "methodGrp", "I");
    if (fid == 0) {
        return;
    }
    (*env)->SetIntField(env, camoper, fid, group_tag);

    // Cutter Group
    UF_OPERATION_cutter_group(operTag, &group_tag);
    fid = (*env)->GetFieldID(env, cls, "cutGrp", "I");
    if (fid == 0) {
        return;
    }
    (*env)->SetIntField(env, camoper, fid, group_tag);
}
```

Figure 4.13 getOperationGroups Method
A `camOperation` with an operationName, type, and NCGroup is added to the `operations` instance variable. The `operations` is a TreeSet that stores `CAMOperations` according to the order defined in the `compareTo` method.

After adding the `CAMOperations` to the `operations` collection list, the control again goes back to the `openUGFile` button listener. The listener also updates the tree view that is showed in the left panel of the GUI.

The result of clicking the `openUGFile` button is opening the specified Unigraphics file in Java, creating corresponding `CAMOperations` of the Unigraphics operations, updating the names and NC Groups of the created operations. The methods described in this section represent extended versions of corresponding methods described in [16].

### 4.4.2 Retrieving CAM Roots

The next step in the process is retrieving the operations tree structure shown in the operation navigator of Unigraphics. The process is started by clicking the `CAMRoots` button of the GUI.

When the CAMRoots button is clicked, the `camSequencingPanel` calls the `setNCGroupRoots` method of the `camPartModel`. The `setNCGroupRoots` method in turn calls the `getNCGroupRoots` method of the camSystem which is shown in Figure 4.14.

The `CAMSystem` defines `getProgramGroupRoot`, `getGeometryGroupRoot`, `getToolGroupRoot` native methods that call the native C code and retrieve the respective roots from Unigraphics. `NCGroup` Objects are created for each of the roots retrieved. These objects are used to update the `NCGroup` root instance variables of the `camPartModel` object.
public void getNCGroupRoots(CAMPartModel camPartModel, 
CAMOperation camOperation) {
    int programGroupRootTag = getProgramGroupRoot(camOperation.getObjectID());
    int geometryGroupRootTag = getGeometryGroupRoot(camOperation.getObjectID());
    int toolGroupRootTag = getToolGroupRoot(camOperation.getObjectID());

    ProgramGroup programGroupRoot = new ProgramGroup(programGroupRootTag);
    GeometryGroup geometryGroupRoot = new GeometryGroup(geometryGroupRootTag);
    ToolGroup toolGroupRoot = new ToolGroup(toolGroupRootTag);
    printName(programGroupRoot, programGroupRootTag);
    printName(geometryGroupRoot, geometryGroupRootTag);
    printName(toolGroupRoot, toolGroupRootTag);

    camPartModel.setProgramGroupRoot(programGroupRoot);
    camPartModel.setGeometryGroupRoot(geometryGroupRoot);
    camPartModel.setToolGroupRoot(toolGroupRoot);

    programGroupRoot.setMembers(getMemberList(camPartModel, programGroupRootTag));
    geometryGroupRoot.setMembers(getMemberList(camPartModel, 
                                      geometryGroupRootTag));
    toolGroupRoot.setMembers(getMemberList(camPartModel, 
                               toolGroupRootTag));
}

Figure 4.14 getNCGroups Method in CAMSystem

The `getMemberList` native method of the `camSystem` is called for each of the 
`NCGroup` roots to retrieve the members associated with them. The member may be an 
`NCGroup` or a `camOperation`. These retrieved members are added to the `members` variable of the roots.

The result of this step can be seen as a tree structure of operations along with the NC Groups in a Java Dialog. Thus until this step, all the Unigraphics operations, with 
manufacturing data and NC groups with members are created in Java.

The first phase that is described till now constitutes only retrieving manufacturing 
data from Unigraphics. The next and most important phase of thesis starts with the third 
step.
4.4.3 Creation of MCS Groups

This step starts with clicking the `createMCSGroups` button. When this button is clicked, the `camSequencingPanel` calls the `createMCSGroups` method of the `camPartModel`. The `createMCSGroups` method of the `camPartModel` loops through operations collection to retrieve the geometry associated with each operation. This is done by calling the `getCAMGeometry` method on each `camOperation`. The `getCAMGeometry` method calls the `getCAMOperationGeometry` or `getCAMOperationBoundary` method of the `camSystem`. One important point to note here is that the parameters passed to the above methods depend on the type of operation.

The UG/Open methods return different geometries depending upon the operations, i.e. the UG/Open methods return geometry as faces for `ZLevelMilling` and geometry as boundaries (edges) for `PlanarMilling` and `FaceMilling`. To deal with different types of operations and different types of geometries returned, two different methods are implemented in the `CAMSystem`. These methods are `getCAMOperationBoundary` and `getCAMOperationGeometry`. `getCAMOperationBoundary` is called to retrieve geometry related to `PlanarMilling` and `FaceMilling` while `getCAMOperationGeometry` is called to retrieve geometry related to `ZLevelMilling`. `getCAMOperationBoundary` returns edges associated with operations and `getCAMOperationGeometry` returns faces associated with operations. In any case the final aim is to get the tool axis vectors associated with the operations so that the operations can be grouped or clustered according to the same tool axis vectors (setup).
The process implements different algorithms depending upon the type of operation. Figure 4.15 and Figure 4.16 show the different versions of methods implemented to retrieve the normals. Since ZLevelMilling gives faces, faces are used to retrieve the features associated with them. Tool axis vectors are retrieved by querying for the normal vector of the feature. So the tool axis vector of the operation is the normal vector of the retrieved feature.

```java
public void setAssociatedNormals() {
  LinkedList associatedFeatures = new LinkedList();
  for (int i = 0; i < faceList.size(); i++) {
    Face face = (Face) faceList.get(i);
    associatedFeatures.addAll(face.getFeatureList());
  }
  for (Iterator iter = associatedFeatures.iterator(); iter.hasNext(); ) {
    GeomFeature item = (GeomFeature) iter.next();
    MfgFeature mfgFeature = item.getMfgFeature();
    Vector3d normal = mfgFeature.getAccessVector();
    System.out.println("camOperation "+ this.getObject());
    geometryGroup = camPartModel.addGeometricGroup(normal, (GeometryGroup)geometryGroup);
    normalVectorList.add(mfgFeature.getAccessVector());
  }
}
```

**Figure 4.15 setAssociatedNormals Method in ZlevelMilling**

For PlanarMilling and FaceMilling, edges are retrieved. These edges can be used to retrieve faces that are associated with operations. In order to achieve this, the face and edge list of camPartModel is used. A special method is implemented that takes a list of edges and returns the faces that can be formed from those edges. As a result, the faces that are associated with the PlanarMilling and FaceMilling are retrieved. In this case
instead of querying for the features associated with the faces, face normal vectors are
directly used to get the normal associated with the operations.

After retrieving the tool axis vectors associated with the operations, the
*camPartModel* is queried to check for the existence of a geometric group that is
associated with the normal retrieved.

```java
public void setAssociatedNormals() {
    LinkedList faces = camPartModel.getFaces();
    Collection edges = new LinkedList();
    for (int i = 0; i < faces.size(); i++) {
        Face partFace = (Face) faces.get(i);
        edges = partFace.getEdges();
        if (edgeList.containsAll(edges)) {
            faceList.add(partFace);
            Vector3d normal = partFace.getNormal();
            geometryGroup = camPartModel.addGeometricGroup(normal,
                (GeometryGroup) geometryGroup);
            normalVectorList.add(partFace.getNormal());
        }
    }
}
```

**Figure 4.16** setAssociatedNormals Method in CAMOperation

If the geometric group exists, the *geometricGroup* variable of the *camOperation* is
updated with the new value. If the geometric group doesn’t exist in the *camPartModel*,
then a new one is created and the new geometric group is added to the *camPartModel* list
and the *geometricGroup* of the *camOperation* is updated to the new value. While creating
the new geometric Group, the constructor of the *GeometryGroup* class calls the
createGeometricGroup method of the camSystem. The native method creates a new GeometricGroup and returns the tag that is associated with the newly created geometry group. This returned value is set as the ObjectID of the geometricGroup created in the Java module. In this a 2-way link is maintained between the geometric Groups in Unigraphics and IMPlanner.

The result of this step is the creation of new Geometric Groups that correspond to a vector direction that are not present in the part file. At the end of this step, new geometric groups are created in Unigraphics and CAMOperations in the Java module have their corresponding geometric groups updated to the new values. The next and final step in the process is moving the operations to the geometric groups they are associated with.

4.4.4 Clustering of Operations

The final step in the process of clustering and sequencing is moving the operations to their corresponding geometric groups in Unigraphics. This step is started by clicking the clusterOperations button. On clicking the button, the clusterOperations method of the camPartModel is called. The clusterOperations method loops through the members of the geometryGroupRoot member of camPartModel to retrieve the operations. When an object of CAMOperation needs to be moved, moveOperation is called giving the camOperation as the parameter. This method in turn calls the moveOperation of camSystem that takes the operation tag and the geometric group tag to which the operation should be moved and moves the operation to that geometric group. Figure 4.17 shows the methods that are used in the clustering and sequencing of operations.
Figure 4.17  Methods of camPartModel Used for Clustering Operations
4.4.5 Sequencing of Operations

As mentioned earlier, the \textit{CAMOperation} class implements the \textit{Comparable} interface which defines the \texttt{compareTo} method to facilitate the ordering of the \textit{camOperation} objects when they are added to a TreeSet. A TreeSet guarantees that the sorted set will be in ascending element order or sorted according to the \textit{natural order} of the elements. Whenever an element of a type Comparable is added to the TreeSet, it calls the \texttt{compareTo} method on the element to decide the position of the element in the collection.

In this way, ordering of the element, \textit{CAMOperation} in this case, is taken care of while adding it to the TreeSet. The \texttt{compareTo} method, as shown in Figure 4.18, uses \texttt{processCapability} collection to decide the sequence between various subclasses of \textit{CAMOperation}.

```java
public int compareTo(Object o) {
    if (o == null) {
        throw new NullPointerException();
    }
    if (o instanceof CAMOperation) {
        CAMOperation camOperation = (CAMOperation) o;
        for (Iterator iter = processCapabilityList.iterator(); iter.hasNext(); ) {
            ProcessCapability processCapability = (ProcessCapability) iter.next();
            if (processCapability.getProcess().equals(this.getClass()) &&
                processCapability.getNextProcessList().contains(camOperation.getClass()).getName())) {
                return 1;
            }
        }
    }
    return -1;
}
```

Figure 4.18  compareTo Method of CAMOperation
processCapability uses an XML file shown in Figure 4.19 to get the information about sequencing. An XML file is used to store the metadata related to the sequencing of operations. An XML parser as shown in Figure 4.20 reads the XML file at runtime and converts all the tags in the file to objects. These objects are added to a collection in the same order as they are parsed. So the collection has a group of objects that indicate the operation sequence. An object in the collection is part of the operation class that should be first in the operation sequence when compared to other operations in the precedes collection.

```xml
<?xml version="1.0" encoding="iso-8859-1" ?>
  <FEATURESFILE>
    <Operations>
      - <edu.ohiou.implannar.cam.PlannarMilling>
        <precedes>PlannarMilling</precedes>
        <precedes>ZLevelMilling</precedes>
        <precedes>Drilling</precedes>
      </edu.ohiou.implannar.cam.PlannarMilling>
      - <edu.ohiou.implannar.cam.ZLevelMilling>
        <precedes>Drilling</precedes>
      </edu.ohiou.implannar.cam.ZLevelMilling>
      - <edu.ohiou.implannar.cam.FacMilling>
        <precedes>Drilling</precedes>
      </edu.ohiou.implannar.cam.FacMilling>
    </Operations>
  </FEATURESFILE>
```

Figure 4.19  Precedence Information Stored in XML File
Figure 4.20 Static block in CAMPartModel to Initialize the Operation Precedence

The result of this step is that the operations are moved to their corresponding geometric groups. After moving the operations from the default group to newly created groups, the operation tree structure is sent back to Unigraphics and the part file is saved to reflect the changes when the part is again opened in the Unigraphics interface as shown in Figure 4.21.

```java
/*
 * Class:     edu.ohiou.implanner.cam_CAMSystem
 * Method:    moveOperation
 * Signature: (II)V
 */
JNIEXPORT void JNICALL Java_edu_ohiou_implanner_cam_CAMSystem_moveOperation
  (JNIEnv *env, jobject obj, jint geometricGroupTag, jint operTag)
{
    UF_NGROUP_accept_member(geometricGroupTag, operTag);
    UF_PART_save();
}
```

Figure 4.21 moveOperation Method in CAMSystem
5 Test Cases

This chapter presents testing of the implementation described in the previous chapter. A variety of cases that include various manufacturing operations and setups are tested. Two of them are explained in this chapter.

In this chapter, the four steps that are explained in previous chapters are used to demonstrate the testing. The execution of the algorithm is explained step-by-step for Bracket. For another part, ANC101, the final result is shown. The examples used in the present work are obtained from the NIST design repository [20].

5.1 Bracket

The part has 15 features in total with 6 RECTANGULAR_SLOTS, 1 RECTANGULAR_POCKET and 8 SIMPLE_HOLES that are used to model the part. Figure 5.1 shows the features used to model the part. The part has 13 operations in total with 6 drilling operations, 2 FaceMilling Operations, 1 ZLevelMilling Operations, 2 Planar Milling Operation, 1 Rough Follow Operation and 1 Cavity Milling Operation. As shown in Figure 5.2, all operations are added to the default “Program Group” in Unigraphics. The Unigraphics part as shown in Figure 5.2 is the input for the system. The result of the algorithm is moving the operations from the default group to different groups that correspond to the setup.
Figure 5.1 Features Used to Model Bracket

Figure 5.2 Bracket with All Operations Added to the Default Group
Table 5.1 shows the operations on Bracket, corresponding features that are machined by the operations and tools used in machining. For example, the PLANAR_PROFILE operation is used to machine RECTANGULAR SLOT_5 with 1.00 inch in diameter Milling Tool.

<table>
<thead>
<tr>
<th>No</th>
<th>Operations</th>
<th>Features Machined</th>
<th>Tools Used (Diameter)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CAVITY_MILLING</td>
<td>All Features except Holes</td>
<td>Mill (1.00)</td>
</tr>
<tr>
<td>2</td>
<td>FACE_MILLING</td>
<td>RECTANGULAR SLOT_4,</td>
<td>Mill (1.00)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RECTANGULAR SLOT_5</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>ZLEVEL_PROFILE</td>
<td>RECTANGULAR SLOT_4,</td>
<td>Mill (1.00)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RECTANGULAR SLOT_5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>PLANAR_PROFILE</td>
<td>RECTANGULAR SLOT_5</td>
<td>Mill (1.00)</td>
</tr>
<tr>
<td>5</td>
<td>PLANAR_PROFILE_Y_AXIS</td>
<td>RECTANGULAR POCKET_9</td>
<td>MILL_1 (0.5)</td>
</tr>
<tr>
<td>6</td>
<td>FACE_MILLING_Y_AXIS</td>
<td>RECTANGULAR POCKET_9</td>
<td>MILL_1 (0.5)</td>
</tr>
<tr>
<td>7</td>
<td>ROUGH_FOLLOW</td>
<td>RECTANGULAR SLOT_6</td>
<td>DRILL (1.0)</td>
</tr>
<tr>
<td>8</td>
<td>DRILLING_HOLE_Y_1</td>
<td>SIMPLE_HOLE_11</td>
<td>DRILL (1.0)</td>
</tr>
<tr>
<td>9</td>
<td>DRILLING_HOLE_Y_2</td>
<td>SIMPLE_HOLE_12</td>
<td>DRILL (1.0)</td>
</tr>
<tr>
<td>10</td>
<td>DRILLING_HOLE_1</td>
<td>SIMPLE_HOLE_15</td>
<td>DRILL (1.0)</td>
</tr>
<tr>
<td>11</td>
<td>DRILLING_HOLE_2</td>
<td>SIMPLE_HOLE_16</td>
<td>DRILL (1.0)</td>
</tr>
<tr>
<td>12</td>
<td>DRILLING_MAINHOLE</td>
<td>SIMPLE_HOLE_10</td>
<td>DRILL (1.0)</td>
</tr>
<tr>
<td>13</td>
<td>DRILLING_HOLE_Y_3_4</td>
<td>SIMPLE_HOLE_13</td>
<td>DRILL (1.0)</td>
</tr>
</tbody>
</table>
5.1.1 Retrieving Unigraphics Operations

The first step of the procedure is retrieving Unigraphics operations to the CAM module of the IMPlanner System. Operations from Unigraphics are retrieved by the CAMSequencingPanel shown in Figure 5.3.

![Figure 5.3 Opening Bracket in CAMSequencingPanel](image)

On selecting a Unigraphics file from the file chooser, the CAMSequencingPanel displays all the details of the part in a tree structure. On selecting any member of the tree, information about the member is shown in the right panel. For example in Figure 5.4, the RECTANGULAR SLOT (3) is selected and information like width of the slot, Slot
normal Vector direction, Slot sweep vector, and Positive and Negative sweep lengths can be seen.

At the end of this step, all the information about all the Unigraphics operations and features is retrieved.

5.1.2 Retrieving CAM Roots

In the previous step, only the operations of the Unigraphics part are retrieved. In this step, the group details of all the operations are retrieved. As shown in Figure 5.5, tree structures from Unigraphics are retrieved.
The groups that are retrieved in this step are Geometry, Program and Tool groups. Each view is treated as an instance of JTree and the root of each view is considered as the root of the tree.

![Figure 5.5 Retrieving Group Details in CAMSequencingPanel](image)

On selecting each tab corresponding to each group, the tree structure of the entire group can be seen. So, in this step, four different tree structures are created as Java Objects that correspond to the four views that are present in Unigraphics.

5.1.3 Creating MCS Groups

In this step, new Groups are created for every new tool axis vector that is retrieved from operations. As shown in Figure 5.7, a new MCS group is created indicating that a new setup is required for `PLANAR_PROFILE_Y_AXIS`, `DRILLING_Y_1`, `DRILLING_Y_2`, `PLANAR_PROFILE_Y_AXIS`. 
\textit{DRILLING\_Y\_2 DRILLING\_Y\_3\_4} operations, in addition to the already existing default setup. These operations require a new setup, because the vector direction for these operations is different from other operations. So a new MCS group is created to move the four operations from the default group. These new groups are created, taking into consideration the tool axis vectors that are retrieved from Unigraphics operations. Figure 5.6 shows the geometry of the operations that are moved to the newly created groups.

![Figure 5.6 Geometry of the Operations Moved to New Group](image)

Figure 5.7 explains that the operations had geometry with normals that are different from the default direction.
5.1.4 Clustering Operations

After creating new MCS Groups, the next step is moving operations from the default group to new groups according to the tool axis vectors. As shown in Figure 5.8, the operations from the default group are moved to the newly created groups. As Figure 5.8 shows, two setups are required to manufacture the part.

When any operation in the tree structure is selected, the geometry that is machined by it is drawn in the rightPanel of the CAMSequencingPanel.
For example, in Figure 5.8 the ROUGH_FOLLOW operation is selected and the face that is machined by the operation is shown in the rightPanel. The final result can be seen in Figure 5.9 and Figure 5.10. Four operations are moved from the default group to the newly created group, because the tool axis vectors of the operations are different from the default setup (group) vector. When Figure 5.10 is compared to Figure 5.2, it can be seen that few operations added to the default group are moved to a newly created group. Now the operations in a single group share the same tool axis vector direction, i.e. the tool axis direction. Each group created corresponds to a separate setup.
Figure 5.9  Operations Moved to New Group

Figure 5.10  Final Result of Bracket
5.2 ANC101

The following example has a greater variety of operations and setups than the previous example. Figure 5.11 shows the initial state of the part. All the operations are added to the NONE group. The part has 13 operations in total with 4 drilling operations, 5 FaceMilling Operations, 2 ZLevelMilling Operations, 1 Planar Milling Operation and 1 Cavity Milling Operation.

One other important point of the implementation is that the operations can be added to the temporary Group called the NONE and the present system is capable of moving such operations from the NONE group to the group whose tool axis vector is equal to the vector retrieved from operation. For example, in Figure 5.11 all the operations are added to the NONE group.

At the end of the procedure, it can be seen that the operations are moved to one of the newly created groups. The destination group may be one of the newly created groups or an already existing default group. The criteria for moving the operation, similar to other operations, is the tool axis vector retrieved from the operation and the access vector represented by the setup (group).

Figure 5.12 shows that total four different setups are required to manufacture the part. The first group (PROGRAM) of Figure 5.13 is accessing the part from the top i.e. from the Z axis. The second group (PROGRAM_1) is for machining the side face of the part indicated by “B”. The third group (PROGRAM_2) of operation is for machining the front face of the part indicated by “C”. The last group (PROGRAM_3) is for machining the slant face of the part indicated by “D” in Figure 5.13.
Figure 5.11  ANC101 with All Operations Added to the None Group

Figure 5.12  Final Result of ANC101
Figure 5.13 Final Result of ANC101 Showing the Four Different Access Directions
6 Conclusions

This chapter summarizes the research performed on reducing the setup changes and sequencing of operations within each setup for a CAM System. The contributions as well as future extensions to this research work are discussed.

6.1 Contributions

This research has made the following contributions towards the effort made in the field of Computer Integrated Manufacturing to reduce the manufacturing time:

• Development of a mechanism for interacting with the design part for querying geometric and manufacturing details.

• Development of an integration mechanism to exchange manufacturing-related data between heterogeneous systems. This helps in reducing the complications and limitations caused by data transfer between different modules of the CIM

• Development of a mechanism for clustering the manufacturing operations with the same access directions to machine them together. This results in decreasing the setup changes.

• Development of a user interface that provides a visual confirmation of the geometry of the part and a mechanism to perform step-by-step execution of developed procedures.

6.2 Limitations

The following are identified as the limitations of the present thesis:
The present implementation is limited only to prismatic parts.

The present implementation is limited only to parts developed using Feature based modeling. If the part is not a feature based model, ZLevelMilling does not return any normal vector because the operation returns the feature normal that it is associated with.

The present implementation does not handle the retrieval of geometry of drilling operations. Instead the user is prompted to enter the hole-axis vector coordinates. These axis vector coordinates are used to decide the setup to which the drilling operations should be moved.

6.3 Further Research

This thesis can be extended by considering the following issues which may lead to a more comprehensive solution to the problem:

- Extend the system to take feature interactions into consideration while sequencing the operations within each setup.
- Extend the algorithm to handle an increased scope of features, geometry and manufacturing operations.
- Increase the number of operations considered for operation sequencing.
- Extend the system to interface with a broader variety of CAD/CAM systems.
- Extend the system to cluster the drilling operations based on geometry retrieved from the UG without the user having to enter the vector details.
References


http://composite.about.com/library/glossary/c/bldef-c1222.htm?terms=computer+aided+
manufacturing.


Hierarchical System Management in an FMS”, The International Journal of Flexible

in Process Planning using State Space Search”, Journal of Intelligent Manufacturing,


http://java.sun.com/docs/books/tutorial/native1.1/.


Appendix: List of Acronyms

CAD  --- Computer-Aided Design
CAM  --- Computer-Aided Manufacturing
CAPP --- Computer-Aided Process Planning
CIM  --- Computer Integrated Manufacturing
CNC  --- Computerized Numerical Control
XML  --- eXtensible Markup Language
HTML --- Hyper Text Markup Language
HTTP --- Hyper Text Transfer Protocol
JNI  --- Java Native Interface
RMI  --- Remote Method Invocation
3I-PP --- Intelligent Integrated Incremental Process Planning
GUI  --- Graphical User Interface