PROCESS SELECTION FOR HOLE OPERATIONS USING A RULE BASED APPROACH

A thesis presented to

the faculty of the

Fritz J. and Dolores H. Russ

College of Engineering and Technology of

Ohio University

In partial fulfillment

of the requirement for the degree

Master of Science

Ajit Wadatkar

November 2004
This thesis entitled

PROCESS SELECTION FOR HOLE OPERATIONS USING A RULE BASED APPROACH

BY

AJIT WADATKAR

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, Fritz J. and Dolores H. Russ
College of Engineering and Technology
This thesis deals with developing an architecture for rule based machining process selection for hole operations. The system developed in this thesis is a rule based intelligent process planning system which selects the necessary manufacturing processes for hole making operations for metal mechanical parts. This system consists of two modules: process selection module for hole making and user interface module. Process selection module performs rule base selection of alternative processes for manufacturing holes. User interface module provides user interaction and functional prototype monitoring for process selection. It includes functionalities that are necessary for the user to navigate the system. The research work can be applied to any feature and a process plan can be generated to satisfy the feature requirements.
Acknowledgements

I would like to sincerely thank all the people who made this thesis possible. My sincere gratitude to my thesis advisor, Dr. Dušan Šormaz, who helped and guided me towards implementing the ideas presented in this thesis. His dedication to research and his effort in the development of my thesis was an inspiration throughout this work.

The thesis would not be successful without the support of other committee members Dr. David Koonce and Dr. Constantinos Vassiliadis. I am grateful to Pravin Khurana and Serhiy Kovbasa from Delphi Research Labs for their valuable suggestions.

I would like to thank my friend Nikhil Pujari for his help and support during my thesis work. Thanks to my fellow colleague Deepak Pisipati from Implanner Group for his suggestions and support. I am thankful to Shachi, Mayuresh, Abhi, Sidd, Amitayu and Satyajit for their encouragement.

If there were any persons who have always backed me from the beginning to the end, they would be my parents: my father, Virochan and my mother, Nirmala.

I would like to dedicate my entire work to my parents.
# Table of Contents

Table of Contents .............................................................................................................. 5

List of Tables ..................................................................................................................... 9

List of Figures ................................................................................................................... 10

Chapter 1  Introduction ............................................................................................. 13

1.1 Need for CAPP ................................................................................................. 13

1.2 Advantages of CAPP systems ........................................................................... 13

1.3 Classification of CAPP ..................................................................................... 14

1.3.1 Variant Process Planning .............................................................................. 14

1.3.2 Generative Process Planning ........................................................................ 14

1.4 Thesis Overview ............................................................................................... 15

1.5 Summary of Modules ........................................................................................ 17

1.5.1 Representation and Database Model for Process Selection ...................... 17

1.5.2 Rule Based System for Selection of Hole Making Processes .................... 17

Chapter 2  Background ............................................................................................. 19

2.1 Process Selection .............................................................................................. 19

2.2 Process Knowledge ........................................................................................... 20

2.3 Process Capabilities .......................................................................................... 21

2.4 Basic Expert System ........................................................................................ 21

2.5 Classification of Expert Systems on Strategies ................................................ 22

2.5.1 Local Information: ........................................................................................ 22

2.5.2 Global Planning ............................................................................................ 23

2.5.3 Based on Input Data View ............................................................................ 24

2.5.4 Based on Method .......................................................................................... 24

2.5.5 Based on Planning Direction ........................................................................ 25
6.4 Future Work........................................................................................................................................ 89

Appendix A.................................................................................................................................................. 90

Appendix B................................................................................................................................................ 91

References.................................................................................................................................................. 93
List of Tables

Table 3-1: Process Capability in inches ................................................................. 40
Table 3-2: Process Capability in millimeters ....................................................... 41
Table 3-3: Capability check at level one ............................................................. 44
Table 3-4: Capability check at level two ............................................................. 45
Table 3-5: Capability check at level three ......................................................... 46
List of Figures

Figure 2-1: Algorithm for process selection ................................................................. 19
Figure 2-2: Precedence relation for processes .............................................................. 20
Figure 2-3: Basic expert process planning system structure ....................................... 21
Figure 2-4: Local planning ......................................................................................... 23
Figure 2-5: Planning with global consideration ............................................................ 23
Figure 2-6: Forward planning ..................................................................................... 25
Figure 2-7: Backward planning ................................................................................... 26
Figure 2-8: Network of frames .................................................................................... 27
Figure 2-9: Hierarchical planning knowledge .............................................................. 30
Figure 2-10: Framework of the IIAIA-CAPP planner .................................................. 32
Figure 2-11: Process selection using backward planning ............................................ 34
Figure 3-1: Use of Jess from Java and call to Java methods from Jess ...................... 36
Figure 3-2: Process selection algorithm ..................................................................... 42
Figure 4-1: Process package structure ......................................................................... 47
Figure 4-2: MfgProcesses to manufacture hole .......................................................... 48
Figure 4-3: Feature package structure ....................................................................... 49
Figure 4-4: XML part file ........................................................................................... 50
Figure 4-5: Schema for part file ................................................................................ 51
Figure 4-6: Functioning of XML handlers ................................................................... 52
Figure 4-7: XML writers for processes ...................................................................... 53
Figure 4-8: XML writers for features ........................................................................ 53
Figure 4-9: Core making rule............................................................................................ 54
Figure 4-10: Hole improving rule ..................................................................................... 55
Figure 4-11: Spot drilling rule .......................................................................................... 57
Figure 4-12: Rule for checking and assigning status ....................................................... 58
Figure 4-13: Rule for complete match ............................................................................. 59
Figure 4-14: Rule for partial/no match ............................................................................. 60
Figure 4-15: Tool parameter selection rule ..................................................................... 61
Figure 4-16: Introspector class in Jess converts Javabean to a deftemplate ..................... 62
Figure 4-17: Complete class file loaded into inference engine......................................... 63
Figure 4-18: Definstance code snippet from Java class .................................................. 64
Figure 4-19: Code snippet in DelphiHole to add PropertyChangeEvent support ............ 64
Figure 4-20: Jess Java interaction ................................................................................... 65
Figure 4-21: Execution diagram for process selection procedure .................................... 66
Figure 4-22: Process selection prototype GUI ................................................................. 68
Figure 4-23: Tolerance editing steps ............................................................................... 69
Figure 4-24: Process capability XML file ....................................................................... 70
Figure 4-25: Schema for process capability XML file .................................................... 71
Figure 4-26: Process precedence graph generated after parsing the XML file ............... 72
Figure 4-27: Rule execution trace example ..................................................................... 73
Figure 4-28: Color coding of part tree ............................................................................ 74
Figure 4-29: Process clustering GUI ............................................................................... 75
Figure 4-30: Resulting process plan for housing example ............................................. 76
Figure 4-31: Pop-up menu showing the options on selected feature ............................... 77
Figure 4-32: Feature process capability table ................................................................. 78
Figure 4-33: Capability matrix for selected processes with feature requirements ........ 79
Figure 5-1: Cast hole example XML file ......................................................................... 80
Figure 5-2: Process plan for cast hole example .............................................................. 81
Figure 5-3: Spot Drilling XML file. .................................................................................. 82
Figure 5-4: Process plan for spot drilling example ......................................................... 83
Figure 5-5: Core making XML example. ......................................................................... 84
Figure 5-6: Results for core making example .................................................................. 85
Figure 5-7: Part geometry for housing example .............................................................. 86
Figure 5-8: Process plan for housing example .................................................................. 87
Chapter 1 Introduction

1.1 Need for CAPP

Process Planning is the link which transforms the design information into process steps and instructions to effectively manufacture products. Process plan generation always begins by collecting data from engineering drawings, specifications, and the list of materials or parts. Process planning has routings specifying operations, their sequences and the tooling and fixtures required for manufacturing. These routings help develop the detailed instructions to manufacture product. Manual process planning is based on experience and knowledge of the process planner and hence it is time consuming and the results are dependant on the planner. CAPP (Computer Aided Process Planning) systems are viewed as an evolutionary path to improvise the function of process planning.

1.2 Advantages of CAPP systems

CAPP systems have distinct advantages which are listed below [7]:

1. Reduction in process planning effort
2. Savings in direct labor
3. Savings in scrap material
4. Savings in scrap
5. Savings in tooling
6. Reduction in WIP (work in progress)
7. Fewer calculation errors and improved cost estimation
8. Faster response to engineering changes
9. Reduced time in process planning
10. Greater process plan consistency

1.3 Classification of CAPP

Computer aided process plans are important for integrating design and manufacturing. CAPP is broadly classified in the following categories.

1. Variant process planners
2. Generative process planners

1.3.1 Variant Process Planning

Variant process planning involves retrieving a process plan similar to the one needed for a similar new part. Later this plan is modified to suit the current feature requirements. A large number of part attributes are identified based on group technology coding and classification approach in variant process planning. This process plan represents around ninety percent of planning. The planner adds the remaining effort to complete the process plan by modifying and adding some processes. The process plans retrieved are entered manually in the computer based on experience and knowledge of many planners and manufacturing engineers.

1.3.2 Generative Process Planning

Generative process plan involves generating a new process plan for a given part using logic and process knowledge. The advantage of generative process plan is that it does not depend upon any earlier process plan and creates alternative process plans for the same design. Currently process planning decision rules are built into the system.
These rules are specified using decision trees involving if-then type statements or artificial intelligence (AI) approach using object oriented design.

Dynamic, generative process planning takes into consideration the plant size, machines available, tooling available and maintenance downtime while developing the process plan. Depending on the resources available and the product size this type of process plan would vary over time.

Although there are very few systems which are capable of generating a complete process plan the following steps are followed in any generative CAPP system [7][7]:

1. Process selection

2. Identifying precedence constraints

3. Fixturability and setup planning

Generative process plans use process knowledge which comprises of process capabilities for various processes and also the preference and precedence constraints among these processes. This knowledge can be represented in various forms. Quite a few approaches store it using rules and frames. This thesis deals with process selection using rules to select a set of processes to manufacture a given feature. The process capability knowledge representation in the system is in XML format.

1.4 Thesis Overview

This thesis deals with developing an architecture for rule based machining process selection for hole operations. The system developed in this thesis is a rule based intelligent process planning system which selects necessary manufacturing processes for hole making operations for metal mechanical parts.
The system developed in this thesis consists of two modules: process selection module for hole making and user interface module.

Process selection module performs rule base selection of alternative processes for manufacturing holes. This module has the following capabilities:

- Coverage of various hole making processes in metal cutting
- Separation of process selection rules and inference engine
- Consideration of dimensional and tolerance capabilities of hole making processes
- Inclusion of process precedence into selection algorithm
- Verification of process plan selected by the user
- Saving process selection results into XML file

User interface module provides user interaction and functional prototype monitoring for process selection. It includes functionality that is necessary for the user to navigate the system. Specific capabilities of this module include:

- Opening and saving features and processes in XML files
- User guided entry of tolerance requirements with selection of types and entry of values
- Selection of rule files for process selection
- Control of execution of rule engine
- User interface for inspection of process alternatives and user selection of process plan
1.5 Summary of Modules

This section deals with the two modules in detail.

1.5.1 Representation and Database Model for Process Selection

This part of the thesis deals with development of representation model for manufacturing features and processes in hole making. Feature parameters and tolerance data for holes are included. For processes cutting parameters, tools, machine, and other necessary data are included. The feature model is hierarchical object oriented. The common data is stored on higher levels, and data specific for individual features is represented on lower level. Tolerances are represented as attributes for manufacturing features but are not included in the object model. Object oriented model for representation of process plan data is implemented. Data representation is specified in XML format, and functions are implemented for writing the data into XML file and parsing the data from file into memory object model.

1.5.2 Rule Based System for Selection of Hole Making Processes

The system consists of several components: rule base, inference engine, and GUI. Manufacturing process selection rule base is implemented in Jess which is a rule language compatible with CLIPS. Candidate manufacturing processes are identified for features covered by feature mapping in the feature model mentioned earlier. A set of rules is developed for these manufacturing processes in order to perform a selection of appropriate process, tools and machines that depend on feature dimensions, feature interactions, and specified geometric and dimensional tolerances. The rule base is stored
independently from the core inference engine, so that rules are managed in a separate system. This enables run-time selection of rule base to be used for inference that may depend on particular feature data or on user’s choice. The rules include preferences of processes and machines with the user having the ability to select if they are going to be used in generating alternatives.

The system allows a selection of several necessary processes for a single hole (for example, drilling and reaming for a single hole) and their sequencing in order to machine the part according to all specified tolerances. The sequencing procedure is limited to sequence only those processes required for a single feature. However, the user has a GUI for selecting alternatives for different holes and specifying their order. Clustering of processes for different tool directions is not computed automatically. However, the user is able to select processes with the same tool and verify his/her choice.

These rules are loaded into an inference engine that uses an expert system shell, Jess, to perform inference and select appropriate processes for each feature. As a result Jess inference engine creates necessary data to store the results of process selection. This data is stored in an XML file. This expert system is controlled from a user interface that enables the selection of rule base, control of rule execution, and monitoring of execution.
Chapter 2 Background

This chapter gives some background knowledge about process planning in expert systems and also gives a literature review of the work done in this area.

2.1 Process Selection

Process selection is that part of the process-planning task in which a set of alternative processes is selected for a given feature [9]. The process selection module of an expert system verifies the process if it can satisfy the tolerance, dimensional and surface requirements for the considered feature, either completely or partly. This module selects multiple processes for generating a feature when no single process can completely manufacture the required feature. It also estimates the time and the machining cost involved for the process candidates.

Following steps are employed in a process selection module as shown in Figure 2-1.

![Algorithm for process selection](image-url)
2.2 Process Knowledge

Process Knowledge is mostly organized in a hierarchical manner with precedence constraints as shown in Figure 2-2.

There are three levels of process knowledge - universal level, shop level and machine level [6]. The universal level assumes that a certain process can produce a certain feature with certain accuracy. This kind of information is available in handbooks and textbooks and is used only if specific details of a process are not available. The specific machine or cutters are considered to predict the required accuracy in the shop level. It mostly represents the knowledge of the best performing machine available in the shop. The machine level takes into consideration only the capability applicable to a particular
machine. This knowledge is important for selecting the machine to perform a specific process. This thesis takes into consideration only the universal level for the system but is flexible enough to take any detailed information in a specific format.

2.3 Process Capabilities

Process Capabilities in dimension, tolerance and surface properties are decided due to various factors. The dimension capability is determined by the tool size. The machine travel limits define the X and Y axis limits in dimension. The depth of hole is related to the diameter of the tool for hole features. Normally a drill can make a hole with a depth of about three to eight times the diameter of the tool [6]. Factors like tool wear, tool deflection, control inaccuracies, thermal deformation and fixture error decide the tolerance capabilities of a process.

2.4 Basic Expert System

An expert system functions as shown below in Figure 2-3:

![Basic expert process planning system structure](image)

Figure 2-3: Basic expert process planning system structure
The knowledge acquisition module shown in Figure 2-3 is generally used by the user to update the knowledge base because a manufacturing system is dynamic and new information regarding the machines, tools and processes needs to be added. The knowledge acquisition module helps the external knowledge to be transformed into internal knowledge of the system. It also helps to make sure that the new rules entered in the system are consistent with the old rules.

An expert system may use various strategies to solve the planning problem. Strategies are classified on the planning level

### 2.5 Classification of Expert Systems on Strategies

An expert system may use various strategies to solve the planning problem [6]. Strategies are classified on the planning level as follows.

#### 2.5.1 Local Information:

In this case the planning is done feature-wise using the local information. The whole process planning is applied on each individual feature and an operation plan is created for each of them as shown in Figure 2-4.

The final process plan is obtained by linking all these individual operation plans. Most of the existing expert systems use this technique to form a process plan. The search tree in this type is limited to only one feature at a time and hence is only a few levels deep. Hence mostly pruning is not required in these cases. The process plans built by this technique are easier to generate but may not always be efficient or optimum. This thesis uses the mentioned technique for planning.
2.5.2 Global Planning

In this approach the feature interactions are taken into consideration as shown in Figure 2-5. These considerations are mostly geometric relationships between the features. The system needs to have logic to understand these relations and check each one of them depending on the criteria given.

Figure 2-5: Planning with global consideration
2.5.3 Based on Input Data View

2.5.3.1 Volume Feature based

Here the basic geometric entity is a volumetric feature such as a hole, pocket etc. A manufacturing process is associated with a volumetric feature for example, drilling is associated with holes. Drilling would be selected whenever a hole feature is encountered in the design. The drill size is chosen from the details of the hole diameter under consideration.

2.5.3.2 Surface Characteristic based

In this case, the surface is the basic geometric entity instead of a volumetric feature. The surface is associated with manufacturing processes and then operations are chosen when these surfaces are encountered in design. However sometimes extra surfaces may confuse the planner and elimination of secondary surfaces can prove to be a challenge.

2.5.4 Based on Method

2.5.4.1 Composite Component

A composite component is an imaginary part which has all features existing in a part family. Operation plans are written for all the features using certain parameters as variables. Rules are written for tool selection and cutter paths for each of these features when used in an expert process planning system. Even though this approach is easy to implement its use is restricted as the system is not very flexible.
2.5.4.2 *Generic Process Capability*

This approach works by using individual processes to model process capability. In this case, the expert system determines which feature can be produced by a certain process. Thus this new feature name is inserted in the capability clauses of processes. This approach is best suited to solve process selection problems.

2.5.5 *Based on Planning Direction*

2.5.5.1 *Forward Planning*

In forward planning, the processes are selected starting from the stock (raw workpiece) to manufacture the final product (finished part) as shown in Figure 2-6. Each arrow represents an individual process that satisfies some or all of the feature tolerance requirements. As this approach is very similar to normal machining, a lot of processes are applicable to begin the operation. Hence the search tree is very large and pruning is required in most of the cases. This thesis uses forward planning approach to generate the process plan.

![Figure 2-6: Forward planning](image)

Figure 2-6: **Forward planning**
2.5.5.2 Backward Planning

In this case the planning starts from the finished product to the stock. The current state of the work piece is compared with the finished part and a set of features results from this comparison (Figure 2-7). Out of this set, one of the features is selected based on sequencing and an appropriate process is applied for the selected operation. In this case, the antecedent rule is the finished state that a process would manufacture and the consequent is the initial stage.

![Backward planning diagram](image)

Figure 2-7: Backward planning

The features are filled back in a backward planning approach till the workpiece is in the raw state.

2.6 Knowledge about Part

The part details are fed to the expert process planning system. This part representation must be very clear. The part design in a CAD system is defined as the external representation for a part. The data format that goes into the planner is called the internal representation. This conversion from external to internal representation is automated or manual. The part information contains details about geometric relationships, dimensions
and tolerances and some other manufacturing specifications. This thesis uses an XML file format to input this data to the planner.

2.7 Procedure Knowledge

Planning knowledge helps in process selection, tool selection, machine selection and sequencing operations. The usual approach is to store this knowledge in the form of rules. Some other approaches use frames.

2.7.1 Frames

A frame system is a hierarchical organization of a network of nodes and relationship between them. A single frame is a data structure representing an entity type. It can store values or pointers to another frame in several slots as shown in Figure 2-8.

![Network of frames](image)

Figure 2-8: Network of frames
A slot can have a default value. It can be restricted to a range of values or values from a set. The tolerance for a feature can have default values which do not require the user to fill in all the values manually. Similar to an object oriented approach the frames also allow inheritance in which both parent child link as well as a sibling link can be established. One frame can link to another frame through a pointer in a slot. Functions can also be attached to a frame which can be called when various events occur. Frame is an organized structure and saves time in search when the number of processes is large. Nau et. al. [15] used a frame in SIPP and SIPS representation. A feature hierarchy of frames was used representing a surface feature and a contained surface. These systems consider only one feature at a time and generate process plans for this one feature.

2.7.2 Rule

Rules are like traditional programs of IF-THEN statements. The IF part or the left hand side of the rule is called as predicate, antecedent or the premise, and the THEN part is the right hand side of the rule mostly termed as action or consequent [8]. A rule based system uses rules to draw conclusions from predicate [6]. Normally in a process selection module the antecedents contain feature type tolerance information. The consequents describe the action that needs to be taken once a rule is fired.

2.8 Literature Review

Sadiah et. al. [17] discusses a CAPP system for prismatic parts. This system is divided into three modules. The first module does the feature extraction from CAD system. The VB program extracts these features from Solid Works CAD system in the CAPP system. The relevant data of the feature is stored in a text file and this file is
further used to find appropriate processes for the feature. The authors further discuss the setup planning for the part. An algorithm has been developed in this system which searches through all the features and keeps count of the number of features that can be machined for each face. The part is finally set up in a direction in which the maximum number of features can be machined. The process selection part is done using a database which has information stored about the surface finish requirement for a feature and other accuracy details. The corresponding feature-process-route is also stored in the database. Thus for process selection of a particular feature, the surface finish and other details are looked up and the route is assigned to it. There are various constraints like precedence geometry, precedence constraints, cutting forces, tool direction approach, cost and minimum work handling which are taken into consideration during this assignment. Furthermore the system selects a machine for each process using an algorithm to eliminate the alternative machines and a process plan is generated.

Similarities and differences between techniques about a system IMACS (Interactive Manufacturing Analysis and Critiquing System) and AI Planning are discussed by Nau et. al. [14]. IMACS scheme includes feature extraction, generating incomplete plans, generating operations plans, their evaluation and providing feedback to improve the performance. The generation of operation plans is done by translating a feature based model (FBM) into an ordered sequence of operations by identifying and generating all orderings consistent with the precedence relations that are decided with the help of some precedence constraints. Goals are also modified in this step. The system adds finishing operations, determines the setups and process details, and finally an optimal operation plan is generated by a depth-first branch and bound search. The authors
plan to further extend this work wherein a collection of planning problems and its solutions would be provided as a test set or a benchmark set for AI planning researchers.

The necessity of modeling complex systems is explained in the issues covered by Hang-Wai and Hon [12]. The authors emphasize that an object oriented approach to create a CAPP system is justified as the resulting model can be directly mapped while implementing it. According to the paper, the planning knowledge, which is an important part of the system, can be divided into logical groups and organized in a hierarchical way using the object oriented analysis and modeling approach. The Figure 2-9 shows the proposed hierarchy.

![Hierarchical planning knowledge](image)

Figure 2-9: **Hierarchical planning knowledge** [12]

This approach helps in adding new resources to the system without redefining the system structure, thus making it more reusable and portable.
Process and production planning is discussed as a part of dynamic planning in production control by Larsen and Alting [11]. The authors explain the effects of alternative routings in process plans. Productivity is increased by 36% and could be even higher in dynamically generated process plans. The lead times are significantly reduced and the production system has no bottleneck and is more balanced due to the alternative process plans. This paper also discusses the possibility of addressing product quality and traceability in an integrated system as it could be manufactured in a different way.

Chang and Chen [2] present the problem faced in many CAPP systems, where the shop floor is considered static and with unlimited resources which makes the generated process plan ineffective and impractical. The authors further discuss a dynamic programming model to tackle this problem. This could also be an effective way to get only those processes available on the shop floor into the system. This would make the system both flexible and practical. There might be some cases in which the required feature would not be possible to manufacture in the system but could always be traced and found exactly what tolerance requirement stops it from being processed. Such new equipment could be procured.

A novel approach of generating a process plan is discussed by Chang and Chang [5], about an IAI-CAPP (Integrated Artificial Intelligence Computer-Aided Process Planning) system which combines the variant approach with generative approach. This approach mimics human-like evaluation and planning, does fuzzy evaluation of process plans and combines both variant and generative process approaches, as shown in Figure 2-10.
Rules could be further developed by including more experience from expert planners.

In other approaches for process planning, case based reasoning (CBR) is a widely used technique in AI. Chang et. al. [4] deal with CBR wherein a process plan is developed by retrieving a relevant case of process planning for a part which is similar to the new part and then this case is modified to meet the requirements of the new parts. The authors use a feature based representation of a part and cutting processes. The part is indexed, a hierarchy of feature is formed based on the cutting processes and a similarity matrix is used to measure the similarity between the new part and old part. This paper only deals with an axisymmetric part.
Marefat and Britanik [13] give another system which uses the CBR approach for process selection and sequencing. The model is object oriented and focuses on three dimensional prismatic parts. The CBR approach is used to develop the sub-plan for each feature which is later merged with the global plan by a hierarchical plan merging mechanism. The object oriented approach helps in keeping the knowledge of the system in a structured manner.

This model also represents the process knowledge in a hierarchical manner as it facilitates direct mapping of processes to objects in the system. This approach merges the variant and generative approaches but does not consider selection of alternative process plan during merging of sub plans.

Backward inference planner used to generate a sub-plan for individual features is explained by Chang and Wen [3]. These sub-plans are then merged into one final process plan. This backward inference planner is made up of process selection module and tool selection module. The set of rules in the process selection module start the process search from feature description and try to fill the material of feature back in the work piece. Figure 2-13 explains the way the process selection in backward planning works. The retrieval of a similar plan is done using the CBR approach.

Various CAPP systems are reviewed by Alting L. and Zhang H. [1]. The following paragraphs describe few most important CAPP systems as discussed in the review. The CAM-I’s Automated Process Planning (CAPP) system retrieves a process plan from a database which the user can edit interactively. The system is user dependant because both the complete coding scheme for part classification and the output format are
to be added by the user which makes it tedious. The thesis work discussed in this document has part data stored, and process plan generated in XML files.

![Diagram](image)

Figure 2-11: Process selection using backward planning [1]

The Computer Managed Process Planning (CMPP) system is a generative process planning system written in FORTRAN which takes into consideration both cylindrical and non-cylindrical features. Processes like electrical discharge machining are included into its knowledge base along with plating and other heat treatment operations.

Interactive Computer Aided Process Planning (ICAPP) system is a variant process planning system built for prismatic parts. This system performs the basic machining operations like milling, drilling, boring, etc. It seems to be a general system without detailed process knowledge for manufacturing the various features.

GARI is a huge system consisting of 50 rules. This system takes into consideration technological and economical preferences. These preferences have weights according to their importance. In case of conflicts, these weights are referred to by the system. This system has been successfully used in a real world example for a metal cutting industry.
The Know-how and Knowledge Assisted Production Planning System (KAPPS) uses frames to store production rules. The system has a CAD interface which recognizes the difference between a rough and a machined surface. The feature reads all feature data directly from CAD. It also recognizes the feature precedence relations. In this thesis, a XML file is used in which the feature tolerance data is manually typed. The precedence constraints for features are not considered here.

Technostructure of Machining (TOM) is a rule-based system flexible enough to take data from a user or to read it directly from a CAD system. It deals only with the hole feature because translating other feature types directly is a very complex process. It works on backward chaining and fires the rule on the agenda according to the alphabetical order of the rules. This thesis also uses a similar approach as it takes into consideration only the hole feature and has extensive rules for the hole operations to generate the process plan.

Nau and Chang [15] explain the Semi-intelligent Process Selector (SIPS) which considers a part to be a set of features and then for each feature generates a process plan for each feature. It tries to overcome the problem with rules in process selection by using hierarchical knowledge clustering for representation of knowledge. Here, each process in the hierarchy is represented by a frame. This thesis follows a very similar line as it also uses a hierarchical object oriented approach for process representation. The system also uses planning by abstraction, which means meaning abstract machining processes are used. Even this approach is incorporated in this thesis.
Chapter 3 Methodology

3.1 Java Expert System Shell (Jess)

This research uses a rule engine for process selection. Jess is a rule engine and a scripting language developed in 1990. It is developed in Java and so it is the ideal tool for adding rules to Java-based software systems. Jess is dynamic and Java-centric and hence gives access to all Java application programming interface (API). So Jess was a natural choice for this research work.

One of the most powerful features of Jess is not only the ability to access Java API, but also enables the Java code to access all Jess libraries (Figure 3-1).

![Figure 3-1: Use of Jess from Java and call to Java methods from Jess](image)

3.2 Rules in Process Selection

The process selection algorithm consists of rules which work in the inference engine and it plays an important part in expert system. The rules used in this research are explained in brief and their interactions are discussed later.
3.2.1 **Rule for Core-making.**

*If (cast diameter = 0) and (no process is assigned to feature)*

*Then*

*Assign core making candidate processes to feature.*

This rule checks for features which are not assigned any machining process and have no pre-cast diameter in the specifications. This rule adds the core making processes like twist drilling, end drilling, spade drilling and gun drilling in the *may-be-machined-by* slot of the feature fact (object) as referred in AI terminology.

3.2.2 **Rule for Hole Improving**

*If (cast diameter > 0) and (no process is assigned to feature)*

*Then*

*Assign hole making candidate processes to feature.*

This rule is triggered if a feature has a pre-cast diameter greater than zero. This cast diameter makes the core making processes no longer required. It directly assigns the hole improving processes like boring, precision boring and reaming to be considered for machining the feature.

3.2.3 **Rule for Spot Drilling**

*If (true position is higher) and (no process is assigned to feature)*

*Then*

*Assign spot drilling as candidate process to feature.*

To obtain a tighter true position in a feature, it is necessary that the hole be started using spot drilling. This condition is taken into account by the rule. It checks for the
feature true position requirement and accordingly takes a decision whether or not to assign spot drilling process.

### 3.2.4 Rule for Checking and Assigning Process Status.

*If (candidate processes are assigned to feature)*

*Then*

*Assign status for complete, partial and no match.*

The left hand side of this rule checks if a feature is assigned any candidate processes or not. The right hand side of the rule calls a function which checks for a match between feature tolerance requirements and process capabilities of the assigned process and returns a status value which represents partial, complete or no match. This value is remembered in the feature slot which helps to trigger some other rules in the rule engine. A new in-process feature fact is also created here to be used in some other rules later.

### 3.2.5 Rule to Handle Complete Match

*If (status assigned is for complete match)*

*Then*

*Delete in process feature and accept the candidate process*

This rule is fired if the status in the feature slot indicates that there is a complete match between the process considered and the feature tolerance requirements. A process instance is created and is associated with the feature fact. Here the in-process feature created in the earlier rule is retracted (deleted) as there are no more tolerance requirements to be met.
3.2.6 Rule to Handle Partial or No Match

*If (status assigned is for complete match)*

*Then*

*Accept candidate process and assign new candidate processes for in process feature.*

This rule is fired when there is a partial or no match between the feature and the considered process.

In case of a partial match, the process is associated with the feature and the new in-process feature created in the earlier rule is assigned the unsatisfied tolerances and the processes to be considered for this new feature are taken from the precedence graph of the features. These are fetched by a function in the inference engine which takes the current process as an argument and returns a set of next level of processes to be considered.

In case there is no match, the new in-process feature is retracted and the set of next level processes is assigned to the feature assuming that these processes would be able to achieve the higher tolerance requirement.

This rule checks whether the feature has reached a dead end i.e. there is no process in the knowledge base which can create such a feature. In practical terms, it means there is no machine on the shop floor capable of manufacturing the feature with higher tolerance requirements. This rule assigns the feature a dead end status so that it is not considered for further computation again in the rule engine.
3.2.7 Rule to Select Tool Parameters

This rule sets default cutting parameters and tools to the processes chosen. One rule selects the default tool and parameters but the rule base may be expanded to select tools and cutting parameters for each process.

3.3 Process Capability

Process capabilities are shop specific and carry information about shape and size of a feature that a process can produce, and dimensional and geometric tolerances that can be obtained. The information about processes is shown below in Table 3-1 [10].

<table>
<thead>
<tr>
<th>TABLE 3-1: Process Capability in inches</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARAMETER</td>
</tr>
<tr>
<td>---------------</td>
</tr>
<tr>
<td>Smallest T.D.</td>
</tr>
<tr>
<td>Largest T.D.</td>
</tr>
<tr>
<td>Negative Tol</td>
</tr>
<tr>
<td>Positive Tol</td>
</tr>
<tr>
<td>Straightness</td>
</tr>
<tr>
<td>Roundness</td>
</tr>
<tr>
<td>Parallelism</td>
</tr>
<tr>
<td>Depth Limit</td>
</tr>
<tr>
<td>True Position</td>
</tr>
<tr>
<td>Surface Finish</td>
</tr>
<tr>
<td>Perpendicular</td>
</tr>
<tr>
<td>PARAMETER</td>
</tr>
<tr>
<td>---------------</td>
</tr>
<tr>
<td>Smallest T.D.</td>
</tr>
<tr>
<td>Largest T.D.</td>
</tr>
<tr>
<td>Negative Tol</td>
</tr>
<tr>
<td>Positive Tol</td>
</tr>
<tr>
<td>Straightness</td>
</tr>
<tr>
<td>Roundness</td>
</tr>
<tr>
<td>Parallelism</td>
</tr>
<tr>
<td>Depth Limit</td>
</tr>
<tr>
<td>True Position</td>
</tr>
<tr>
<td>Surface Finish</td>
</tr>
<tr>
<td>Perpendicular</td>
</tr>
</tbody>
</table>

These values are in inches and are to be converted to millimeters as that is the unit in which the feature tolerance requirements are given. So after converting we get a new process capability matrix in millimeters as shown in Table 3-2.
3.4 Working of All Rules

The working of the rule engine is shown in the Figure 3-2. The feature initially is considered by three rules - core making, hole improving and spot drilling. If a hole is to be started from a full material, core making processes can be used. However if a hole is to be started from a pre-cast hole then hole improving processes are required. Also, in case of tighter true position requirements, it is advisable that the spot drilling process be used to start the hole. Any of the three rules according to the feature requirements is fired and some candidate processes are selected to check whether any of them satisfy the feature requirements so they could be assigned to the feature. These are stored in a may-be-machined-by slot in the feature.
Figure 3-2: Process selection algorithm
The block in the Figure 3-2 representing check process capability is actually done by the rule which checks for process capabilities and assigns a status value for partial, complete or no match. This rule also creates a new feature and assigns the unsatisfied attributes of the current feature to the new feature.

If there is a complete match, the rule for complete match is fired. It deletes the new feature which was created by the process capability rule mentioned earlier. The process for the feature is accepted and it is assigned to the feature. The rule to select tool parameters is then fired, which specifies the machine and tool constraints on the feature processing.

If there is a partial match or a no match between the process and the feature then another rule would be fired. This rule assigns new candidate processes from the precedence graph of processes to the may-be-machined-by slot of the new feature. This new feature is again considered for process capabilities and the whole cycle is repeated as shown in the Figure 3-2.

Here there is no distinction between partial and no match. The assumption made for a no match is that, it would be possible to manufacture the feature with a process having higher process capabilities and hence it is not deleted but the next set of processes is assigned to the may-be-machined-by slot of the feature. Also this rule checks if a feature has reached a dead-end, meaning there is no process in the knowledge base which can manufacture the current feature. In such a case, it flags this feature so that it is not considered further or selected in the rule engine.
We shall consider an example to understand the functioning of the process selection algorithm. Feature requirements to be considered are listed below.

Feature: Locating Hole

\[
\begin{align*}
RADIUS & \quad 5.5125 \\
DEPTH & \quad 9.77 \\
TRUE \ POSITION & \quad 0.20 \\
POSITIVE-TOL & \quad 0.025 \\
NEGATIVE-TOL & \quad 0.025
\end{align*}
\]

The three rules, rule for core making, rule for hole improving and rule for spot drilling check on this feature and the core making rule is fired. The feature is assigned the candidate process of twist drilling. The next rule checks for the process capability for the twist drilling process.

Table 3-3: Capability check for twist drilling

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>Twist-Drill Value</th>
<th>Feature Requirement</th>
<th>New Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smallest T.D.</td>
<td>1.5875</td>
<td>1.5875</td>
<td>Diameter =11.025</td>
</tr>
<tr>
<td>Largest T.D.</td>
<td>50.8</td>
<td>11.025</td>
<td>(Can be drilled)</td>
</tr>
<tr>
<td>Negative Tol</td>
<td>0.89608 * D^0.5</td>
<td>2.975</td>
<td>0.025</td>
</tr>
<tr>
<td>Positive Tol</td>
<td>0.89608 * D^0.5 + 0.0762</td>
<td>3.05</td>
<td>0.025</td>
</tr>
<tr>
<td>Straightness</td>
<td>0.64005 * (1/D)^3 + 0.0508</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Roundness</td>
<td>0.1016</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parallelism</td>
<td>0.12801 * (1/D)^3 + 0.0762</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Depth Limit</td>
<td>12D</td>
<td>132.23</td>
<td>9.77</td>
</tr>
<tr>
<td>True Position</td>
<td>0.2032</td>
<td>0.2032</td>
<td>0.20</td>
</tr>
<tr>
<td>Surface Finish</td>
<td>2.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Perpendicularity</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This rule also creates a new feature and assigns the unsatisfied attributes of the old feature to the new feature. This rule assigns the status value for partial match as is clear from the Table 3-3. This new feature is again assigned a new set of candidate
processes from the precedence graph shown in Figure 3-2. This feature is again checked for process capability for the new set of processes. For instance, boring is the first process checked.

Table 3-4: Capability check for boring

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>Boring</th>
<th>Feature Requirement</th>
<th>New Feature</th>
<th>Verdict</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smallest T.D.</td>
<td>9.525</td>
<td></td>
<td></td>
<td>Partial match</td>
</tr>
<tr>
<td>Largest T.D.</td>
<td>254</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Negative Tol</td>
<td>0.0762</td>
<td>0.025</td>
<td>Not satisfied</td>
<td>0.025</td>
</tr>
<tr>
<td>Positive Tol</td>
<td>0.0762</td>
<td>0.025</td>
<td>Not satisfied</td>
<td>0.025</td>
</tr>
<tr>
<td>Straightness</td>
<td>0.0127</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Roundness</td>
<td>0.0127</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parallelism</td>
<td>0.0254</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Depth Limit</td>
<td>9D</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>True Position</td>
<td>0.00254</td>
<td>0.2</td>
<td>Satisfied</td>
<td></td>
</tr>
<tr>
<td>Surface Finish</td>
<td>0.2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Perpendicularity</td>
<td>0.0254</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This new feature is sequenced after the earlier feature. Thus, there is a processing condition on the feature. This new feature is again assigned the next set of processes from the cutting taxonomy graph from Figure 2-2. The next process considered is reaming. Its process capabilities are analyzed below.

The complete match rule is fired, and the process is selected and the new feature is deleted which has no attributes assigned to it. The processing sequence for the feature is twist drilling, boring and reaming. Finally machine and tool constraints are assigned after the rule is fired.
Table 3-5: Capability check for reaming

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>Reaming</th>
<th>Feature Requirement</th>
<th>New Feature</th>
<th>Verdict</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smallest T.D.</td>
<td>1.5875</td>
<td></td>
<td></td>
<td>Complete Match</td>
</tr>
<tr>
<td>Largest T.D.</td>
<td>101.6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Negative Tol</td>
<td>0.01016</td>
<td>0.025</td>
<td>Satisfied</td>
<td></td>
</tr>
<tr>
<td>Positive Tol</td>
<td>0.01016</td>
<td>0.025</td>
<td>Satisfied</td>
<td></td>
</tr>
<tr>
<td>Straightness</td>
<td>0.00254</td>
<td>0.025</td>
<td>Satisfied</td>
<td></td>
</tr>
<tr>
<td>Roundness</td>
<td>0.0127</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parallelism</td>
<td>0.254</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Depth Limit</td>
<td>16D</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>True Position</td>
<td>0.254</td>
<td></td>
<td>Already satisfied</td>
<td></td>
</tr>
<tr>
<td>Surface Finish</td>
<td>0.4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Perpendicularity</td>
<td>0.254</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Chapter 4 Implementation

We will see in details how each unit of this model is implemented to make the module work.

4.1 Languages / API

Java language is used to develop build this research. This research has an object oriented approach; hence java is the ideal choice. It has various inbuilt libraries which help portability over the internet. Java can be run on any platform requiring only java-run time machine installed on it. Also, the rule based system is coded using Jess. Java has a Jess API that helps interaction.

The main packages in the java architecture are `edu.ohiou.implanner.processes`, `edu.ohiou.implanner.features` and `edu.ohiou.implanner.delphi` for this research.

4.2 Process Package Structure

The `MfgProcess` class is the main class and other classes extend it. The hierarchy of the process package is shown in Figure 4-1.

![Process package structure](image)

Figure 4-1: **Process package structure**
The class representing Coremaking and Holemaking directly extends MfgProcess as shown. HoleImproving and HoleStarting are child classes of the Holemaking class. The MfgProcess class stores the cutting tool details and cutting parameters. It can store the feature reference to be processed. These methods are inherited by other subclasses. Processes used to manufacture the hole feature are shown in Figure 4-2.

![Diagram of MfgProcesses](image)

Figure 4-2: MfgProcesses to manufacture hole

### 4.3 Feature Package

The package edu.ohiou.implanner.features deals with the various features to be manufactured by process planning. The important class here is MfgFeature which is the topmost class in the hierarchy of manufacturing features. This MfgFeature object stores tolerance requirements and all feature details. All other features extend the MfgFeature class and specify the dimensional tolerance variables which hold the values. As we only
deal with holes in this research, it is the only class of interest. Other features can be ignored. The package structure is shown in Figure 4-3.

![Diagram of package structure](image)

**Figure 4-3: Feature package structure**

### 4.4 Delphi Package

This package reads all the XML files from the knowledge base. It reads the feature file, populates the feature object with data and passes these features to the rule engine for process plan generation.

*DelphiHole* class defines the feature object. It extends the *Hole* class from the features object and inherits all feature attributes. The default value for the cast diameter attribute is zero. While parsing, the parser sets it to a different value if the feature begins with a pre-cast hole. The process object which is selected to manufacture this feature is stored in the feature object. Process candidates to be considered for the feature are set using setters in the class. The propertyChangeListeners enable this to be set from the Jess working environment. These propertyChangeListeners are a part of the Jess API in java.
The XMLPartHandler class in the Delphi package parses the part file which has a feature list with tolerance and dimensional requirements. This handler has a mechanism where the handlers are switched while the file is being parsed. The corresponding handlers are called for, as the tags are being read. The XML file is as shown in Figure 4-4. This file is validated by an XML schema which ensures that that structure of all part files is maintained as per the requirements. It validates the data entering the system. The schema is shown in Figure 4-5.

```
<xml version="1.0" encoding="iso-8859-1">
<FEATURESFILe>
<FEATURELIST>
<FEATURE Id="4" Name="Locating_Hole(4)" ClassId="SPHOLE">
<Parameter AxisPoint="55.63.22.45 -0.420000000000027"/>
<Parameter Axis="0.0 0.0 0.0"/>
<Parameter Radius="5.5"/>
<Parameter Depth="9.77"/>
<Tolerances>
<Parameter truePosition="0.012"/>
<Parameter roundness="0.1016"/>
<Parameter positiveTolerance="0.00762"/>
<Parameter surfaceFinish="0.1"/>
</Tolerances>
</FEATURE>
</FEATURELIST>
</FEATURESFILe>
```

Figure 4-4: XML part file

Figure 4-5 explains the working of the handlers in this system in detail. The XML PartHandler starts parsing the XML file. As soon as it encounters the <FEATURESLIST> tag it switches the handler and sets it to the FeatureHandler class which starts parsing the file further. The feature handler instantiates the feature object and
populates it with data using setters mentioned earlier. The feature handler, as it parses and reads the `<TOLERANCE>` tag, shifts and sets the handler to `PropertyTableHandler` class. This class sets all the tolerances on the current feature.

Figure 4-5: Schema for part file

On parsing the closing tolerance tag `</TOLERANCES>` the handler is switched back to the feature handler.
The feature is also added to MfgPart object. Finally when all features are done and the </FEATURELIST> tag is encountered; the handler is set to PartHandler.

The XML writers in the Delphi package write out XML files after the processes are selected for features and the selection data is to be stored in file. The DelphiPartProcessWriter writes part material and tool activities. It gives a call to the process writer to write the tool material details. It also writes about the depth of cut and the cutter details. The working is shown in Figure 4-6.
The feature writers also work in the same manner. The *DelphiFeatureWriter* writes the dimensional details like radius, depth, axis and axispoint about the feature. It gives a call to the *PropertyTableWriter* which writes the tolerance details like surface finish, roundness, true position and straightness. The working is shown in Figure 4-7.

4.5 Rules

The rules have been implemented in Jess which is compatible with CLIPS. There are seven rules which have been discussed briefly in the earlier chapter. These are
developed completely from scratch for this system. Here we will take a look at each one of them in detail.

4.5.1 Core Making Rule

```
(defrule AssignCoreMaking
  ?jh <- (Hole (mayBeMachinedBy ?process&:(eq (call ?process toArray) (create$ )))
           (processes ?listOfProcesses&:(eq (call ?listOfProcesses toArray) (create$ )))
           (OBJECT ?o) (partModel ?part&:(neq ?part nil))
           (castDiameter ?castDia&:(= ?castDia 0.0)))
  (not (FeatureRelation (Feature ?jh)))
 =>
  (addToMaybeMachinedBy ?o "edu.ohiou.implanner.processes.TwistDrilling")
  (addToMaybeMachinedBy ?o "edu.ohiou.implanner.processes.EndDrilling")
  (addToMaybeMachinedBy ?o "edu.ohiou.implanner.processes.SpadeDrilling")
  (assert (FeatureRelation (Feature ?jh)
                      (status MBMBAssigned)))
```

Figure 4-9: Core making rule

The core making rule shown in Figure 4-9 can be interpreted as follows- “For hole ?jh, which has an empty process list for mayBeMachinedBy list which stores the candidate processes, has no process assigned to it, is already assigned to a partModel, and has a cast diameter zero. It means that the hole feature is not pre-cast and does not exist in a fact for FeatureRelation. If all these requirements are met, Twist Drilling, End Drilling and Spade Drilling are assigned as candidate processes in the mayBeMachinedBy list. Also assert a new fact of type FeatureRelation using the hole ?jh and status as MBMBAssigned (mayBemachinedBy) which would be used in other rule further.”
4.5.2 Hole Improving Rule

The rule shown in Figure 4-10 takes into consideration hole ?jh for which the mayBemachinedBy slot is empty, i.e. no candidate processes are assigned. Also the castDiameter should be greater than zero which means that the hole is pre-cast.

```lisp
(defun rule AssignHoleImproving
  (?jh <- (Hole (mayBeMachinedBy ?process&(= (call ?process size) 0) )
  (castDiameter ?castDia&:(> ?castDia 0))
  (processes ?listOfProcesses&(= (call ?listOfProcesses size) 0))
  (OBJECT ?featObj))
  (not (FeatureRelation (Feature ?jh)))
=>
  (addToMaybeMachinedBy ?featObj
  "edu.ohiou.implanner.processes.Boring" )

  (addToMaybeMachinedBy ?featObj
  "edu.ohiou.implanner.processes.PrecisionBoring" )

  (addToMaybeMachinedBy ?featObj
  "edu.ohiou.implanner.processes.Reaming" )

  (assert (FeatureRelation (Feature ?jh)
  (status MBMBAssigned)) )
```

Figure 4-10: Hole improving rule

There is no manufacturing process selected for the feature which is checked from the listOfProcesses slot. It is also checked whether this feature fact already exists in any of the FeatureRelation fact in the rule engine. When all these conditions are met, this rule is fired. The right hand side of the rule assigns hole improving processes like Boring, Precision Boring and Reaming for the mayBeMachinedBy slot as candidate processes and also asserts (creates) a feature relation fact which will fire other rules.
4.5.3 Rule for Spot Drilling

The rule shown in Figure 4-11 is for firing and assigning spot drilling process for features having higher true position requirement. The feature ?jh is checked for having no candidate processes assigned. The process capability fact ?pc should have the name SpotDrilling. The first test condition showed in the rule checks whether the feature tolerance value for true position is a non-nil value.

The second test condition checks whether the process capability tolerance value for true position is a non-nil number. The final test condition checks if the tolerance requirement on the feature is less than that of the capability true position value. The rule is fired on all these conditions being met and assigns Spot Drilling process as the candidate process in the MBMB slot and asserts a FeatureRelation fact using the feature fact ?fh.

4.5.4 Rule for Checking and Assigning Process Status.

The rule shown in Figure 4-12 checks and assigns status between the feature and the candidate process assigned in the MBMB slot. To begin, the rule checks for a feature relation fact ?relation such that the feature ?jh has status MBMBAssigned.
Figure 4-11: Spot drilling rule
Figure 4-12: Rule for checking and assigning status

This feature relation fact is asserted in one of the earlier three rules. The rule checks for a feature ?jh such that the MBMB slot is not empty and contains processName which matches the process Capability name of ?pc. It also checks for the part model to be non-nil. Once the rule is fired, the process is removed (deleted) from the mayBemachinedBy slot of the feature object. A new in-process feature fact ?nf is created using the old fact and the processName by calling the function createFact. The status (complete, partial or no match) is checked by passing the arguments ?jh ?pc and ?nf to the ProcessCapability function. This function matches each tolerance value for feature requirement with the process capability value for the process and if the process is not capable to achieve the required tolerance, it assigns the feature tolerance to the new feature ?nf. It finally returns a value -1 (no match), 0 (partial match) or 1 (complete
match) which is stored in ?retunValue. A new fact FeatureProcessRelation is asserted at the end with the status value in one slot to help fire the other rules in the rule engine.

4.5.5 Rule for Complete Match

```
(defrule SelectHMProcessCompleteMatch
    (?jh <- (Hole (mayBeMachinedBy ?mbmb)(processes ?processList)
          (OBJECT ?o) (featureName ?fName) (partModel ?part))
    (?pc <- (ProcessCapability (name ?processName)(OBJECT ?pcObj))
    (?nf <- (Hole (OBJECT ?nfObj))
    (?do <- (ProcessFeatureRelation (oldFeature ?jh)(newFeature ?nf)
          (processCap ?pc)(status 1) )
    =>
          (modify ?do (status 5))
          (bind ?processInstance (createProcessInstance ?jh ?processName))
          (addProcessToPart ?part ?processInstance)
          (retract ?nf))
)
```

Figure 4-13: Rule for complete match

This rule for process complete match shown in Figure 4-13 may be interpreted as:

“For hole ?jh, process capability ?pc for process ?processName, and process/feature relation with status of 1 (COMPLETE MATCH), modify process/feature relation status to 5 (SOLVED), create new process instance of the ?processName class on feature ?jh, add this instance to part ?part, and delete (retract) in-process feature ?n as it is no longer needed”.

4.5.6 Rule for Partial/No Match Condition

The rule (Figure 4-14) checks for hole ?jh , processCapability ?pc , and new in process feature ?nf if the status is 0 (PARTIAL MATCH) or -1 (no match). The rule when fired modifies the process feature relation status to 5 (solved).it creates a new
process instance of the ?processName class on the feature ?jh and adds this instance to part ?part. The new in-process feature is added to the process instance created. The in-process feature ?nf is assigned the next level of machining processes from the precedence graph for processes.

(defrule SelectHMProcessPartialMatch
  ?nf <- (Hole (mayBeMachinedBy ?newMbMb) (OBJECT ?nfObj) )
=>
  (modify ?do (status 5))
  (bind ?processInstance (createProcessInstance ?jh ?processName) )
  (if (= (call ?pList size) 0) then (assert (FeatureRelation (Feature ?nf)(status Deadend) )) (modify ?nf (color (new java.awt.Color 255 0 0) ) ) else (assert (FeatureRelation (Feature ?nf)(status MBMBAssigned) )))
)
)

Figure 4-14: Rule for partial/no match

This rule also checks whether all the processes have been exhausted trying to match with the feature requirements. In such a case it modifies the feature process status to dead-end meaning no further search is necessary as there are no more processes to be assigned to the mayBemachinedBy slot in the feature. It finally creates a FeatureRelation fact to be used by other rules in the rule engine.
4.5.7 Rule for Tool and Parameter Selection

It checks the feature object to match with the Holemaking process \( ?\text{process} \) and then fires to assign the default tool and cutting parameters. This rule can be expanded to work for each process (Figure 4-15).

```lisp
(defrule selectToolAndParameters
    ?process <- (Holemaking (OBJECT ?pObject) (feature ?fObject))
    ?feature <- (Hole (OBJECT ?fObject) (radius ?radius))
=>
    (call ?pObject setTool (new Tool (* 2.0 ?radius) 2.5 "diamond"))
    (call ?pObject setCuttingParameter (new CuttingParameter))
)
```

Figure 4-15: Tool parameter selection rule

4.6 Jess to Java Interaction

The Jess facts and java objects interact in the application by using shadow facts and `propertyChangeListener` property.

4.6.1 Shadow Fact

The shadow facts are unordered facts and are similar to JavaBeans. Shadow facts serve as a connection between the working memory and Java application in which Jess is running. The function `defclass` is used to create a `deftemplate` which is used to create shadow facts in Jess. The `definstance` command creates individual shadow fact. In real time applications it is required that the rule engine responds to events outside Jess. Jess allows regular java objects in its working memory which are instances of classes, where the only prerequisite is that, they be javabeans.
4.6.2 Javabeans

There is a similarity between a javabeans and unordered facts as both have a list of slots which can hold values and these values can change at run time. For a javabeans, these slots are called properties. A javabeans property is normally a pair of methods to get and set the property value. There is a class `Introspector` in the java.beans API which examines a javabeans and finds properties defined by the get and set methods as shown in the Figure 4-16 and creates a `deftemplate`.

```java
public class JavaBean {
    public String getPropertyOne()
    public void setPropertyOne(String)
    public int getPropertyTwo()
    public void setPropertyTwo(int)
}
```

Figure 4-16: **Introspector class in Jess converts JavaBean to a deftemplate**

A fact created from such a `deftemplate` serves as an adapter to store a javabeans in working memory as a shadow fact. The shadow fact created has one slot for each property of the javabeans. These slots are populated with the bean property values.
4.6.3 How Shadow Facts Work in the System

We shall consider the feature class to understand the steps involved in using the shadow fact to interact between Java and Jess languages. Creating a deftemplate for feature is the first step involved. In the system, a defclass file is loaded into the rule engine. This file looks as shown in the code snippet below in Figure 4-17.

(defclass Holemaking edu.ohiou.implanner.processes.Holemaking)
(defclass Hole edu.ohiou.implanner.delphi.DelphiHole)

Figure 4-17: Complete class file loaded into inference engine.

This defclass when loaded creates a deftemplate by name Hole with slots corresponding to the DelphiHole bean properties. The slot OBJECT contains a reference to the javabean for a shadow fact.

This hole is then put into working memory for which definstance is used. This is a code snippet (Figure 4-18) from the java code which creates this definstance in the rule engine.
As soon as this command is executed, a shadow fact representing this DelphiHole bean appears in the working memory. This fact is similar to other facts in the rule engine and responds to the rules in a similar manner as other facts. A shadow fact continuously tracks changes in a javabean by notifying Jess whenever there is a changed event in the bean. Javabean uses the javabeans. PropertyChangeEvent API to notify Jess about the changes. For this, the DelphiHole class should support the PropertyChangeListener as shown in Figure 4-19 which notifies Jess on any change in the bean and thus the shadow fact is in sync with the javabean.

```java
for (Iterator itr = partModel.getFeatureList().iterator(); itr.hasNext();)
{
    DelphiHole hole = (DelphiHole) itr.next();
    engine.definstance("Hole", hole, true, engine.getGlobalContext());
}
```

Figure 4-18: **Definstance code snippet from Java class.**

The way all the interactions take place between objects in Jess and Java code is shown in Figure 4-20.

The shadow fact created calls a method on the OBJECT slot which is a reference to the Java object. The method in the Java code modifies the object and the

```java
public void addPropertyChangeListener (PropertyChangeListener p)
{
    pcs.addPropertyChangeListener(p);
}
```

Figure 4-19: **Code snippet in DelphiHole to add PropertyChangeListener support**
firePropertyChanged method updates the shadow fact with these modifications done with the bean. This way the synchronization between the bean and shadow fact is maintained.

Figure 4-20: Jess Java interaction

4.7 Files Loaded into Rule Engine

Various files are loaded into the rule engine. These are addressed below:

4.7.1 Rule File

This file loads all the rules discussed in section 4.4 into the inference engine.

4.7.2 Defclass File

class. Jess creates *deftemplates* for these classes when they are loaded into the inference engine.

### 4.7.3 Template File

This file loads two *deftemplates* FeatureRelation and *ProcessFeatureRelation* which are used to create instances in rule engine.

### 4.7.4 Functions File

This file loads the functions used by the rules.

### 4.8 Working of the Module

![Execution diagram for process selection procedure](image)

Figure 4-21: *Execution diagram for process selection procedure*

Interactions between the rule inference engine, the knowledge base, the data in XML format; and execution of the process selection procedure is shown in Figure 4-21. The execution starts by importing feature XML file and creating corresponding Java
objects in the memory space. The process capability knowledge base is then loaded into the inference engine. Next step is the loading of the rule base into inference engine and creation of a Rete network. It is followed by the creation of rule facts from feature Java objects. After that, the inference engine is run and cycle shown earlier in Figure 3-2 is executed as long as there are rules to be fired. During this procedure, process facts and their corresponding process Java objects are created, which is the result of the inference. During the inference, necessary in-process features are created in order to govern the needs for multiple processes on a single design feature.

4.9 User Interface for Process Selection

The user interface needs to satisfy the following requirements:

- Guide the user through process selection procedure
- Provide GUI for entering tolerance information
- Provide selection and control of rule loading and process capabilities
- Control process selection procedure
- Report results at different steps
- Save and retrieve feature and process model to/from XML file (database)

Process selection GUI components that provide these services are shown in Figure 4-22. These include:

- Control toolbar which allows the user to execute the process selection procedure in individual steps and monitor execution
- Part tree panel which displays the content of the part model and includes the trees with information about features, processes and process clustering
- Feature/process data panel, which shows data about the selected object from the part tree panel
- Tolerance panel which shows the tolerance information for a selected feature and allows the user to enter or modify tolerance requirements
- Termination buttons which allow the transfer of process selection results back to the XML file

Figure 4-22: Process selection prototype GUI

Tolerance editor shown in Figure 4-23 enables the user to enter tolerance requirements for each feature. A feature is selected within the part tree panel and becomes the active feature for tolerance editing. Tolerance type is selected from a drop down list and the value is entered into a text field on the right, and the entered value is tested for validity (needs to be a real number). Tolerances are saved in feature object and
cab be written back to XML files at the end of process selection procedure. GUI also provides the button for deletion of tolerance values.

![Figure 4-23: Tolerance editing steps](image)

After the features are loaded from XML file, the process capability knowledge is loaded into the system. The process capability data is stored in a XML file. This file is parsed by Java and corresponding process objects are created. The file is shown in Figure 4-24. The capability is parsed and stored in Java. The “<precedes>” tag gives information about the precedence constraints between the processes. For example, as shown in Figure 4-24 the twist drilling precedes boring, precision-boring and reaming. This file is validated using a XML schema as shown in Figure 4-25.
The graph package in the application is capable of taking an array of objects as its argument and representing a graph of these objects. This hierarchy is decided by the cell generator class which is also passed as an argument to the constructor of graph class. This class generator gives the relation of precedence between the objects. This graph class is generic and works for any set of objects supplied with the appropriate cell generator for those objects. The graph of processes is shown in Figure 4-26.
<xml version="1.0" encoding="UTF-8"/>
<!--VOC Schema generated by XMLSpy v2004 rel. 2 U (http://www.xmlspy.com) -->
- <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  - <xs:element name="AngularPositions" type="xs:double"/>
  - <xs:element name="DepthOfCut" type="xs:double"/>
  - <xs:element name="DraftX" type="xs:double"/>
  - <xs:element name="DraftY" type="xs:double"/>
  - <xs:element name="EntryDiameter" type="xs:double"/>
  - <xs:element name="ExitDiameter" type="xs:double"/>
  - <xs:element name="Feature">
    - <xs:complexType>
      <xs:attribute name="ItemName" type="xs:string" use="required"/>
      <xs:attribute name="ItemIndex" type="xs:integer" use="required"/>
    </xs:complexType>
  </xs:element>
  - <xs:element name="FileSeq">
    - <xs:complexType>
      - <xs:sequence>
        <xs:element ref="Operation"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  - <xs:element name="SolutionFileName" type="xs:string" use="required"/>
  - <xs:element name="Material" type="xs:string" use="required"/>
  - <xs:element name="MaterialLib" type="xs:string" use="required"/>
</xs:schema>

Figure 4-25: Schema for process capability XML file
Rule execution may be monitored in rule engine trace panel, which shows Jess trace of rule engine execution with reports on the changes in Jess memory during execution. Sample trace of such execution is shown in Figure 4-27.

The result of process selection is shown visually to the user in part tree panel in which the tree of design features is shown (Figure 4-28). For each feature, its alternative processes are shown as child nodes in the tree. In turn, for each hole making process, its in-process feature is shown as its child node, and then the process is repeated until the bottom of the tree. The part tree is color-coded for easier understanding, so that the
design features are shown in gray color, mfg processes are shown in green color, and in-process features are shown in blue color.

Figure 4-27: Rule execution trace example

If an in-process feature does not have any processes for its manufacturing it is shown in red color.
The final stage in the process selection module is process clustering. In this step, the processes generated in the previous procedure are clustered together. This is the required stage for the module because of the XML format requirements for process file. This stage is performed by the user. The prototype provides visual GUI tools for this stage. The following steps performed by the user in this stage:

- User visually orders and groups processes into the process plan
- Right-click on process node provides actions for adding the process to a new or an existing activity
- Results of action are visually shown in process plan tree

Popup menus for this stage are shown in Figure 4-29 and the resulting process plan is shown in Figure 4-30.
The summary of Java classes implemented for the prototype is given in Appendix A. Previously implemented Java classes which were used for this system are summarized in Appendix B. Additional documentation about these classes is given in the source code with javadoc utility which created the html documentation for the prototype.
Figure 4-30: Resulting process plan for housing example

4.10 Feature Process Capability Table

This table is a part of the user interface to display the results. The user can display this table for any feature whose details are to be known. Figure 4-31 shows this option
highlighted. This table displays the feature tolerance comparison with the capability values from the complete process knowledge.

The *Renderer* class colors the string value while displaying it. The green text color indicates that a particular feature tolerance requirement can be met by the process and red text color shows the inability of the process to obtain the desired tolerance value (Figure 4-32).

![Pop-up menu showing the options on selected feature](image)

**Figure 4-31: Pop-up menu showing the options on selected feature**
As shown in Figure 4-32 the feature Locating_Hole(4) is compared with the process capability of the system. It is clear that processes like spade drilling, end drilling and twist drilling do not satisfy any of the tolerance requirements for positive tolerance, true position and negative tolerance and hence form a no-match for the feature. Reaming and spot drilling form a partial match as seen while the processes honing, boring, precision boring and grinding form a complete match and can be selected. This visual tool helps understand why the processes are selected or rejected by a particular feature.

### 4.11 Capability with Selected Processes by Feature

This table is quite similar to the earlier table. The only difference is that only those processes which have been selected by the rule engine to manufacture the feature are displayed with the capability data against the feature tolerance requirement as shown in Figure 4-33. The feature Locating_Hole (4) selects the three processes boring, precision boring and reaming. Only these processes are displayed in the second case as opposed to displaying the complete process knowledge.
### Figure 4-33: Capability matrix for selected processes with feature requirements

<table>
<thead>
<tr>
<th>Name</th>
<th>Locating_Hole(I)#4</th>
<th>Boring_Locating_Hole(#1)</th>
<th>PrecisionBoring_Lo...</th>
<th>Reaming_Locating_Hole(#1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>positiveTolerance</td>
<td>0.025</td>
<td>0.00702</td>
<td>0.00254</td>
<td>0.01016</td>
</tr>
<tr>
<td>truePosition</td>
<td>0.2</td>
<td>0.00254</td>
<td>0.00254</td>
<td>0.254</td>
</tr>
<tr>
<td>negativeTolerance</td>
<td>0.025</td>
<td>0.00762</td>
<td>0.00254</td>
<td>0.01016</td>
</tr>
</tbody>
</table>

Row: negativeTolerance; Column: Locating_Hole(I)#4
Chapter 5 Testing

The rule based system is tested for test cases to ensure that all the rules fire as designed and the results obtained are as expected.

5.1 Cast Hole Test Case

The feature file in this case is shown in Figure 5-1. The cast diameter parameter is more then 0.0. This should trigger the Assign Hole Improving rule as mentioned in chapter 3. The steps are followed in the same way as explained in that chapter (Figure 3-2).

```xml
<?xml version="1.0" encoding="iso-8859-1" ?>
<FEATURESFILE>
  <FEATURELIST>
    <FEATURE Id="4" Name='Locating_Hole(4)"
      ClassId="SIMPLE_HOLE">
      <Parameter AxisPoint="55.63 22.48 -0.420000000000027" />
      <Parameter castDiameter="2.0" />
      <Parameter Axis="0.0 0.0 -1.0" />
      <Parameter Radius="5.5" />
      <Parameter Depth="9.77" />
      <Tolerances>
        <Parameter roundness="0.1016" />
        <Parameter positiveTolerance="0.00762" />
        <Parameter surfaceFinish="0.1" />
      </Tolerances>
    </FEATURE>
  </FEATURELIST>
</FEATURESFILE>
```

Figure 5-1: Cast hole example XML file

The process plan generated for the feature after rules are fired is shown in Figure 5-2. The hole improving rule assigns precision boring, boring and reaming as candidate
processes to be considered. The in-process features are created and the process selection algorithm is followed as shown in Figure 3-2. The process plan has alternatives and the user can select the optimal plan.

Figure 5-2: Process plan for cast hole example

The file for spot drilling example is shown in Figure 5-3.
<xml version="1.0" encoding="iso-8859-1" ?>
  <FEATURELIST>
    <FEATURE Id="10" Name="Mounting_Hole(10)" ClassId="SIMPLE_HOLE">
      <Parameter AxissPoint="55.63 22.48 -0.4200000000000027" />
      <Parameter Axis="0.0 0.0 -1.0" />
      <Parameter Radius="5.5" />
      <Parameter Depth="9.77" />
      <Tolerances>
        <Parameter truePosition="0.012" />
        <Parameter roundness="0.1016" />
        <Parameter positiveTolerance="0.00762" />
        <Parameter surfaceFinish="0.1" />
      </Tolerances>
    </FEATURE>
  </FEATURELIST>
</FEATURESFIL e>

Figure 5-3: **Spot Drilling XML file.**

The true position requirement for the feature triggers the spot drilling rule. The consequent rules are fired in a similar manner and rest of the processes is selected for other in-process features generated. The complete process plan generated for this feature is shown in Figure 5-4.
5.2 Core Making Example

The test on feature file shown in Figure 5-5 is to trigger the assign core making processes rule from Figure 3-2. The cast diameter is zero and the tolerance requirement is also not very high. So the core making processes are assigned as candidate processes in the `maybeMachined` slot of the feature. The results are shown in figure 5-6.
Figure 5-5: Core making XML example.
5.3 Housing Example Testing

A housing part with various features is tested in the rule base. The part geometry is shown in Figure 5-7. The other features, except the hole feature, are neglected and the nine hole features are tested in the rule base simultaneously. The results for this test example are shown in Figure 5-7.
Figure 5-7: Part geometry for housing example
Figure 5-8: Process plan for housing example
Chapter 6 Conclusion

6.1 Conclusion

This thesis has successfully developed the process selection module for hole making operations. The module is built using a rule based approach with separation of data knowledge and inference, thus enabling easy extensions and/or modifications in data representation and process capability knowledge. Rules for the selection can also be added or modified to accommodate further steps in process planning. The prototype of the module has been implemented in Java. So that it can be run in two modes: stand-alone and from other systems. In stand-alone mode the prototype interacts with XML files, while the second mode it could interact with other systems by passing XML strings. The module prototype has been tested on several examples to verify rule logic and on Housing example to demonstrate passing data from XML files.

6.2 Contributions

This is a comprehensive system for process selection in a CAPP system. It has extensive rules for hole features which also allow optional precedence among processes selected. The process knowledge is represented in the XML format which allows for easy update of knowledge for the system. Here the in-process features are remembered in the process object which is the right place to store this pointer as these in-process features are not a part of the original part.
6.3 Limitations

This system is only limited to the hole feature and cannot be used for other feature types like slot, pocket, etc. Also, the process selection details for the process selected are default for all the processes, which means all processes will have same default data. The process plan is to be manually selected by selecting processes for a set of features. The system also does not have any algorithm which would suggest the optimal process plan.

6.4 Future Work

The thesis work can be extended to work for other features like slot, pockets and slabs by adding new rules. The process knowledge for this process needs to be enriched with this new process data to machine these slots.

Also, at the end in this process planning the user just clusters the process plan manually. An algorithm could be developed to find the optimum process plan and guide the user to use it. Developing this algorithm and entering data for the cost of each machining operation would enable the selection of the best possible process plan for a given part.

Moreover, the tool selection rule could be extended in such a manner that each process would have a separate rule for assigning it specific machining and tool parameters and also the associated cost.
# Appendix A

Summary of all Java classes in Process Selection (all classes except ProcessCapability are in `edu.ohiou.implanner.delphi` package).

<table>
<thead>
<tr>
<th>Java class</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>DelphiFeatureWriter</td>
<td>Writer for generation of XML string for individual feature</td>
</tr>
<tr>
<td>DelphiHole</td>
<td>Extension of <code>edu.ohiou.implanner.features.Hole</code> to provide interface for rule engine (Java bean properties)</td>
</tr>
<tr>
<td>DelphiPartProcessWriter</td>
<td>Writer for generation of XML string for process XML file</td>
</tr>
<tr>
<td>DelphiPartWriter</td>
<td>Writer for generation of XML string for part feature file</td>
</tr>
<tr>
<td>DelphiProcessWriter</td>
<td>Writer for generation of XML string for individual hole making process</td>
</tr>
<tr>
<td>DelphiXMLFeatureHandler</td>
<td>XML handler for parsing individual feature from MBMS</td>
</tr>
<tr>
<td>DelphiXMLPartHandler</td>
<td>XML handler for parsing feature XML file from MBMS</td>
</tr>
<tr>
<td>PartDialog</td>
<td>Class that provides visual control for process selection module and interaction with MBMS</td>
</tr>
<tr>
<td>PartPanel</td>
<td>Class that provides GUI components for displaying and controlling all part, feature and process data</td>
</tr>
<tr>
<td>ProcessCapability</td>
<td>Class that implements process capability information for individual hole making process</td>
</tr>
<tr>
<td>PropertyTableHandler</td>
<td>Class for parsing XML string of tolerance and capability data</td>
</tr>
<tr>
<td>XMLFileFilter</td>
<td>Utility class that limits file selection in open dialog only to XML file</td>
</tr>
</tbody>
</table>
Appendix B

Summary of IMPlanner classes that were used (called from process selection classes).

<table>
<thead>
<tr>
<th>Java class</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>edu.ohiou.implanner.parts.MfgPartModel</td>
<td>Used to store and retrieve part data, feature list, and process plan list</td>
</tr>
<tr>
<td>edu.ohiou.implanner.features.MfgFeature</td>
<td>Used to store generic relations between feature, part and processes</td>
</tr>
<tr>
<td>edu.ohiou.implanner.features.Hole</td>
<td>All methods from Hole class are available in DelphiHole class, used to store data about each hole</td>
</tr>
<tr>
<td>edu.ohiou.implanner.processes.MfgProcess</td>
<td>Used to store generic relation between features and processes (including in-process feature)</td>
</tr>
<tr>
<td>edu.ohiou.implanner.processes.Holemaking and its subclasses</td>
<td>Used implicitly to create process instances during inference</td>
</tr>
<tr>
<td>edu.ohiou.implanner.processes.CuttingParameter</td>
<td>Encapsulates cutting parameter data</td>
</tr>
<tr>
<td>edu.ohiou.implanner.processes.Tool</td>
<td>Represents the tool in relation to process</td>
</tr>
<tr>
<td>edu.ohiou.implanner.processes.ProcessCapability</td>
<td>Class that implements process capability information for individual hole making process</td>
</tr>
<tr>
<td>edu.ohiou.implanner.processes.TokenizeName</td>
<td>Used to separate class names</td>
</tr>
<tr>
<td>edu.ohiou.implanner.resources.Machine</td>
<td>Encapsulates the machine in relation to process</td>
</tr>
<tr>
<td>edu.ohiou.implanner.activity.PartActivity</td>
<td>Used to store information about process clustering</td>
</tr>
<tr>
<td>edu.ohiou.implanner.activity.MachineActivity</td>
<td>Used to store information about process clustering</td>
</tr>
<tr>
<td>edu.ohiou.implanner.activity.ToolDirActivity</td>
<td>Stores information about process clustering</td>
</tr>
<tr>
<td>Package</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>edu.ohiou.labimp.basis.ImpXMLReader</code></td>
<td>Used to provide the basic mechanism for reading XML file with multiple handlers</td>
</tr>
<tr>
<td><code>edu.ohiou.labimp.basis.ImpXMLHandler</code></td>
<td>Used as a base class for various XML handlers to provide common methods</td>
</tr>
<tr>
<td><code>edu.ohiou.labimp.basis.ImpXMLWriter</code></td>
<td>Used to provide basic mechanism for writing XML strings</td>
</tr>
<tr>
<td><code>edu.ohiou.labimp.gtk3d.Tuple3dParser</code></td>
<td>Used to parse string into 3-d point and vector in XML handlers</td>
</tr>
<tr>
<td><code>edu.ohiou.labimp.table.RectangularTableModel</code></td>
<td>Utility class for representing table data</td>
</tr>
<tr>
<td><code>edu.ohiou.labimp.table.PropertyTable</code></td>
<td>Class that represents property/value data</td>
</tr>
<tr>
<td><code>edu.ohiou.labimp.table.ModelTable</code></td>
<td>Class for visual representation of table model</td>
</tr>
</tbody>
</table>
References


