SAND93-1430

Unlimited Release

Printed August, 1993

The Milling Assistant, Case-Based Reasoning, and Machining Strategy:

A Report on the Development of Automated Numerical Control Programming Systems at New Mexico State University

Authors:

William Burd	Distinguished Member of Technical Staff Sandia National Laboratories
David Culler	Research Specialist Industrial Engineering Department New Mexico State University
Tom Eskridge	Artificial Intelligence Programmer Computing Research Laboratory New Mexico State University
Leon Cox	Associate Professor Industrial Engineering Department New Mexico State University
Ted Siater	Artificial Intelligence Assistant Computing Research Laboratory New Mexico State University

Abstract

The Milling Assistant (MA) programming system demonstrates the automated development of tool paths for Numerical Control (NC) machine tools. By integrating a Case-Based Reasoning decision processor with a commercial CAD/CAM software, intelligent tool path files for milled and point-to-point features can be created. The operational system is capable of reducing the time required to program a variety of parts and improving product quality by collecting and utilizing "best of practice" machining strategies.

MASTER

EB

Acknowledgements

Thanks to the following people for supporting and contributing to the Milling Assistant project over the past three years.

Paul Plomp (Sandia National Laboratories)

Frank Whiston (Sandia National Laboratories)

Dr. Patrick Hanratty (Manufacturing and Consulting Services, Inc.)

William Hoverter (Manufacturing and Consulting Services, Inc.)

Eddie Rose (Manufacturing and Consulting Services, Inc.)

Steve Denkers (Manufacturing and Consulting Services, Inc.)

Stan Dornfeld (DORNFELD)

Craig Fischer (Texas Instruments)

Edwin Evans (Pantex Plant)

Satish Kamat (New Mexico State University)

George Mulholland (New Mexico State University)

Satish Kalaparthy (New Mexico State University)

Pamela Oldham (New Mexico State University)

Philipp Djang (New Mexico State University)

Charlie Parks (New Mexico State University)

Ed Romero (New Mexico State University)

Table of Contents

Introduction	1
Description of Current Implementation	3
The Milling Assistant's Revised Structure	3
Feature Descriptions	7
All Features	8
Milled Features	8
Pockets	8
Pockets with Islands	9
Slots (Open and Closed)	9
Notches	9
Slabs	9
Part Boundary (Open and Closed) and Cutouts	10
Bosses	11
Bottom Fillets:	11
Point-to-Point Features	11
Machining Strategy	15
Background	15
Automation of Shop-Floor Functions	16
The Milling Assistant	16
Knowledge Acquisition	17
Milling Strategies	19
1) SETUP	21
A) Material Machinability	21
B) Roughing and Finishing Operations	21
C) Tool Selection	22
D) Cutting Parameter Determination	24
2) Before and During Entry to Material	28
A) From Position	28
B) Approach Point	28
C) Entry Methods	28
D) Start Point	31
3) Cutting Methods	31
A) Conventional -vs- Climb Cutting	31
4) Feature by Feature Analysis	32
1) Pockets	32
2) Pockets with islands	33
3) Slots	33
4) Notches	33

5) Part Boundaries	34
6) Slabs	34
5) Special Cases	34
1) Sharp Corner Milling	35
2) Thin Walls	36
3) More on Speeds and Feeds	36
4) Slugs	37
Formalization of Milling Strategies Using IDEF3	39
Background on IDEF and Knowledge Based Systems	39
Description of IDEF3	41
Utilizing AI3 (KBSI) to Formalize Knowledge for the Milling Assistant	42
Case-Based Reasoning	47
Overview of Case-Base Reasoning	47
Secret of Expertise is Experience	47
Processing Flow	48
Why the Difference?	51
Implementation	54
Integration with ANVIL-5000	54
Knowledge Representation	56
Basics	56
Abstraction and Packaging Hierarchies	60
Calculation Functions	61
Learning from Mistakes	63
Learning through user interaction	63
Example	66
Adding new knowledge directly	66
Example	69
Where to from here?	70
Asking for help when siblings are adapted	70
Plan-oriented vs. feature-oriented	71
Knowledge base combination	71
Interaction with user	72
Beyond 2-1/2 axis milling	73
Knowledge Enhancement Examples	75
General Methods	75
Example: Knowledge for a Drill "Pecking" Cycle	75
Example: Cut Amounts Based on Material Hardness and Fixture Rigidity	78
Interface to Case-Based Reasoning Knowledge	81
Milling Assistant's User Interface Utilities	81
The Garnet User Interface Development Environment	83

Use of Garnet in the Implementation of the Milling Assistant's User Interface Utilities	85
Merit of the Use of Garnet in the Development of Milling Assistant's User Interface	87
Operating Guide	89
Start-Up and Initialization of MA, ANVIL-5000, and CBR	89
Case Based Reasoning: Two modes of operation for processing features	90
User Interface Utilities	91
A) Display Abstraction Hierarchy	91
B) Feature Information Display and Knowledge Modification Area	92
Conclusions	95
References	96

List of Illustrations

Figure 1: The Revised Milling Assistant's Structure	5
Figure 2: Milling Assistant Demonstration Part	7
Figure 3: Typical Rectangular Pocket (no islands)	8
Figure 4: Pocket with Islands	9
Figure 5: Narrow, Wide and Angled Slots	9
Figure 6: Typical 3-Sided Notches	9
Figure 7: Recessed Part Surface	10
Figure 8: Slabbing Area	10
Figure 9: Slab Volume With Tool Access Choices	10
Figure 10: Variations on Part Profiles	11
Figure 11: Part Boundary Cutout Examples	11
Figure 12: Drilling	12
Figure 13: Tapping	12
Figure 14: Reaming	12
Figure 15: Boring	12
Figure 16: Sample Scenario Used for Strategy Meetings	18
Figure 17: Milling Assistant Strategy Issues	19
Figure 18: Actual Distance Travelled for Various Tool Diameters	24
Figure 19: Features Requiring Circular Entries	30
Figure 20: Sample Part	30
Figure 21: Climb-vs-Conventional	31
Figure 22: Constant Angle and Collapse Pocketing	32
Figure 23: Influence of Small Fillets on Tool Selection.	35
Figure 24: Standard IDEF Block Diagram applied to material removal operations.	41
Figure 25: Model of Milled Features Developed for MA	43
Figure 26: Sequence of NC programming decisions associated with all features.	46
Figure 27: Data flow diagram of conventional case-based reasoning.	49
Figure 28: Data flow diagram for the case-based version of the Milling Assistant.	51
Figure 29: Decision tree to classify problems according to cases in case library.	53
Figure 30: File-Based Communication Controller Program	55
Figure 31: Syntax of slots->mop function.	58
Figure 32: Pseudo-code for the slots->mop function	59
Figure 33: Portion of Case Library before adaptation	60
Figure 34: Operation of install-abstraction.	61
Figure 35: Portion of the abstraction hierarchy.	62
Figure 36: Portion of Case Library before adaptation.	64
Figure 37: User interface for specifying adaptation rules.	67
Figure 38: Modified Case Library showing movement of the overlap slot.	68
Figure 39: Alternative Methods for Modifying CBR Knowledge	76
Figure 40: Various Pecking Depths Based on Material's Brinell Hardness	77

Figure 41: Boundary Tool Path Before Change.	78
Figure 42: Boundary Tool Path After Change	78
Figure 43: Notch Tool Path Before Change	78
Figure 44: Notch Tool Path After Change	78
Figure 45: MA-Learn Window for Part Boundary Case	79
Figure 46: Milling Assistant's Abstraction Hierarchy	82
Figure 47: MA-Learn Window (Slot Values and User Interface Area)	84

List of Text Boxes

Text Box 1:	Width of Cut Categories to Achieve Side Surface Finish	23
Text Box 2:	Depth of Cut Categories to Achieve Base Surface Finish	23
Text Box 1:	ICAM Definition (IDEF) Methodologies	40
Text Box 4:	Information in J3 for determining collapse or constant angle pattern	44
Text Box 5:	Index generation methods used to assign qualitative labels to numerically valued attributes.	52
Text Box 6:	MOP with slots and slot fillers.	57
Text Box 7:	DefMop and examples.	58
Text Box 8:	The M-Mill-Feature MOP.	69
Text Box 9:	Calculation Functions Added Directly by User.	70
Text Box 10:	CBR LISP Code for Implementing Pecking Cycle	77

Introduction

The Milling Assistant (MA) Numerical Control programming system combines the attributes of CADD/CAM/CAE systems and Artificial Intelligence computer software to automate the generation of machining plans. The development of MA technology since 1989 has lead to a more thorough understanding of how to gather and utilize machining strategies, the potential benefits and shortcomings of decision processors, and the importance of building a knowledge rich system. The transfer of technology from Sandia National Laboratories (SNL) to New Mexico State University and private industry has provided a means to demonstrate and expand on the original MA ideas. Important accomplishments during the past three years include:

- 1) Replaced EXSYS rule based decision processor with a non-proprietary Case-Based-Reasoning (CBR) implementation written in LISP. The CBR system demonstrated an artificial intelligence learning technique.
- 2) Developed methodologies for acquiring and documenting "industrial strength" machining strategies.
- 3) Designed and implemented a knowledge acquisition tool for the purpose of extending current machining rules and modifying CBR calculation functions. The result is that domain experts are able to modify the information within the CBR.
- 4) Achieved SANDIA Technology Maturation Program (TMP) missions by releasing MA programs and documentation to Manufacturing and Consulting Services (MCS, Scottsdale AZ) for future work.
- 5) Developed an "Art to Part" demonstration center at NMSU where an entire production cycle for machined parts can be observed, studied, and utilized for teaching students and raising the awareness of manufacturers.
- 6) Complemented the Advanced Manufacturing Center now under development at NMSU. MA demonstrations play an important role in achieving recognition and credibility for NMSU manufacturing initiatives.
- 7) Tested and evaluated various operating platforms including a PC based (DOS) implementation and a faster, more efficient, UNIX version. In this sense, portability and enhancement of the MA concept were demonstrated
- 8) Documented MA programs and recorded the results of machining strategy meetings.
- 9) Utilized IDEF3 to document and formalize knowledge about milled features.

What is The Milling Assistant?

Creating tool motions to machine a desired part correctly is what NC programming with a CAM system is all about. The generality of most CAM systems provide the flexibility to NC program for a variety of tools, materials, and part geometry. Unfortunately this is normally an involved process requiring the user to choose from a confusing array of options and parameters. Having to develop machining plans while also directing a CAM system makes modern NC programming very complicated and time consuming. The Milling Assistant, using the knowledge of what feature is being machined, simplifies the NC programming process by selecting a suitable set of machining options and parameters. MA uses the machining options and parameters to construct a set of machining tool motions for the feature.

Document Content

This document is a culmination of all aspects of the work done on the MA project at NMSU. The reader is encouraged to scan the table of contents to identify individual areas of interest. The topics covered in this report are as follows:

- Description of current MA implementation at NMSU
- MA feature descriptions
- Compilation of machining strategy information
- Formal representation of knowledge using IDEF3
- Description of MA's CBR decision processor
- Examples of CBR knowledge modification alternatives
- Description of MA's user-interface utilities
- MA operating procedures

Description of Current Implementation

Development System at NMSU

MA utilizes a Case-Based Reasoning (CBR) system to automate the decision process required for machining 2 1/2 axis milled and point-to-point features on CNC machine tools. MA creates NC tool paths by integrating CBR with a commercial Computer Aided Design and Drafting/Computer Aided Manufacturing/Computer Aided Engineering (CADD/CAM/CAE) software, ANVIL-5000 [1]. The user inputs feature geometries, design parameters, set-up information, and machine tool limitations. MA programs, written in an Application Interface language within ANVIL-5000, link the entire system by transferring feature attribute data to CBR and, after the CBR processor has devised a solution in the form of machining actions, by returning directives to ANVIL-5000. These directives consist of the operation(s) to be performed and the machining parameter settings chosen by the strategies embedded in the CBR. MA's application interface program directs ANVIL-5000 to construct the tool motions necessary to machine the feature under consideration. The user ultimately has the ability to inspect and modify all decisions before the final path is output.

The current version of the Milling Assistant was developed and advanced at NMSU through a series of Technology Maturation Project contracts with SNL. The Milling Assistant is implemented on a SUN SPARC station 2 workstation and should be portable to any UNIX platform. A previous version was implemented and demonstrated on a 386/33 PC system under DOS. Quarterdeck's DESQview was used to provide the windowing and multitasking functions. While the 386 provided adequate response, the added user interface complexity caused the project to concentrate on the UNIX workstation implementation until improved operating systems are available for PC's.

The Milling Assistant's Revised Structure

The initial implementation of the Milling Assistant is described in SAND89-1519 [3]. Project development has continued during the three years since the initial description. This section describes the current MA structure.

The revised MA still creates ANVIL-5000 tool paths by collecting a feature definition from the user and then simulating the machining strategy decisions and CAD/CAM interactions of the NC programmer. Several major changes were made to provide improved decision processing capabilities, use an improved ANVIL application interface, and simplify the implementation. As illustrated in the new flow diagram, the NC programmer still identifies part features which are processed by the MA to produce NC tool paths. However, the division of tasks between ANVIL-5000 and the external MA processes has shifted to utilize a more robust ANVIL application interface. This structure directly uses ANVIL's internal analysis functions and eliminates the previously duplicated analysis code.

The flow diagram on the facing page represents the revised MA structure. Each step in the feature processing cycle is now described briefly and the differences from the earlier MA system are noted. Steps 1, 2 and 3 are implemented in ANVIL's GRAPL-IV application interface language.

1. The NC programmer starts with the part geometry created in ANVIL or transferred from another system using IGES or DXF data exchange formats. The NC programmer answers several questions to define material, machine tool, fixture rigidity, part quantities, and various defaults to initialize MA.
- 2-3. MA's menus and icon options allow selection of features with associated geometry and parameters. The feature geometry is used by ANVIL's analysis functions to calculate feature attributes. This processing phase also defines point-to-point tools and selects milling tools from the tool collection available in the part. All of the selected and calculated values are written to a file for the knowledge processor.

Steps 4 and 5 are a separate UNIX process and are activated by a message from the ANVIL interface. The ANVIL process is suspended while the external processing is performed.

4. MA's Case-based Reasoning decision processor uses the feature's data and it's machining strategy rules to reach machining goals. Typical goals include:
 - a. Types of ANVIL-5000 tool paths.
 - b. Cutting conditions suited for the machine-material-tool combination.
 - c. Appropriate cutting tool motions

MA's learning cycle is a separate process described later.

5. The machining goals from step 4 are processed by a FORTRAN program to generate a file of action codes. MA's GRAPL-IV program will execute the action codes and direct ANVIL CAM to build the feature's tool paths. Step 5 is a table-driven one-to-many translator designed to map high level decisions into the appropriate (one or many) ANVIL modal settings. This program has been simplified from the earlier MA implementation by moving code specific to MA's knowledge domain to either GRAPL-IV or the CBR module.

Control is returned to ANVIL when the UNIX process for steps 4 and 5 completes. The GRAPL-IV application interface is used to process the MA's results.

Figure 1: The Revised Milling Assistant's Structure

6. Tool paths are generated by MA's GRAPL-IV program to machine the feature. The calculations at this final stage produce speeds, feeds and incremental cut values. The user is asked to verify (and change if desired) several of the parameters.
7. The generation of a machining process plan has not been implemented.
8. MA's associative tool paths are automatically combined with other tool paths for the complete job and ordered into a complete machining sequence. Optional modifications can be made to meet individual requirements. The completed job

is written to a NC CL-data output file.

9. The complete machining sequence is verified using the VERICUT [16] solid modelling simulation system.
10. After verification, the NC data (tool motions) are postprocessed to produce instructions for the target machine tool and then downloaded to the machine tool for manufacturing the part.

Feature Descriptions

The concept of "features" plays an important role in linking design to manufacturing. CAM-I's Current Status of Features Technology [15] defines features used in process planning for manufacturing as 'shapes and technological attributes associated with manufacturing operations and tools'. From a machinist's viewpoint, the first step in developing a machining plan is to break the part into features. Each feature is characterized as a structural entity whose attributes specify lower-level geometry, topology, tolerance, surface finishes, etc..

Within the context of the Milling Assistant (MA), features are a collection of facts that add information to the CAD/CAM part model. MA relies on an experienced user to select and geometrically identify individual machinable features. Information about feature parameters is interrogated from the ANVIL databases and tolerance, material, machine tool, and specific factors that influence the machining plan are queried from the user. The definition of some features, such as slots or cylindrical holes, are similar in both design and manufacturing. Other features, such as slabs and cutouts, are related to specific machining operations. Each feature has characteristics which influence tool and machining parameter selection, tool access, and cutter path motions. The figure below illustrates a part composed of typical features. Following sections discuss each MA feature.

Figure 2: Milling Assistant Demonstration Part

All Features

Part descriptions consist of a wide variety of geometric and design specifications. Certain information applies to all features and other information to only particular features. All features have a top of stock, a final cut depth, dimensional tolerances, and surface finishes. Milled and Point-to-Point features contain implicit tool access positions, for example, a notch by MA's definition has an open side where the tool can plunge safely and feed into the material.

Most often, milling is divided into roughing and finishing operations. The roughing operations are required to remove the majority of the stock, and the finishing operations machine the feature to final dimensions and surface conditions. This division of operations is used in selecting the tools and the types of ANVIL-5000 tool paths for the particular feature.

When a feature's definition includes a bottom surface, the machining rules determine if a finish base pass is required. The bottom of stock variable is used in the rules to eliminate the finish base sequence whenever the feature's depth breaks through the part's bottom plane.

The Milling Assistant's interface with ANVIL-5000 permits variations in the actual feature geometry from the "generic feature" and is still able to produce valid machining tool paths. The NC programmer relies on experience to determine when the geometry is "close enough" to be processed as one of the standard MA features. If the user defines a feature that deviates considerably from the generic feature, some material may not be removed.

Milled Features

Pockets

The pocket feature is defined by a closed set of contiguous perimeter geometry, a top of stock, a bottom fillet, and the feature depth. Several shape properties are calculated for pockets to aid in tool size selection and in choosing either collapse or constant angle cutting patterns. The calculated properties include the pocket's area, smallest corner fillet radius, perimeter length, maximum DX and DY, concave and convex shape flags, and mass property data.

Figure 3: Typical Rectangular Pocket (no islands)

Pockets with Islands

Pockets with islands contain additional sets of geometry which define the internal areas to be avoided during machining. Constant angle or collapse pocketing may be used for this feature case. Multiple islands can be selected, with each one's geometric description being identified individually. An example of a pocket with islands is pictured in Figure 4. Properties calculated include the same as for pockets with the addition of a smallest gap distance (ie. gap - finish side stock on each side.)

Figure 4: Pocket with Islands

Slots (Open and Closed)

An open slot is bounded by two sets of side geometry and two open ends. The two slot openings are approximately equal in width. The simplest slot is bounded by two parallel straight sides, however, slot boundaries may vary from the simple form and still produce valid machining motions as long as a fairly uniform width is maintained. A closed slot (or blind slot) is bounded by one set of contiguous side geometry and does not contain an open end. This special form of the general slot is treated as a collapse pocket by MA. [see Fig. 5]

Figure 5: Narrow, Wide and Angled Slots

Notches

The notch is similar to a slot, but it has only one open end. The notch's boundary is a contiguous set of geometry that roughly defines three sides of a rectangle. Examples are shown in Figure 6.

Figure 6: Typical 3-Sided Notches

Slabs

The slab is a rectangular or oddly shaped region of stock which is to be removed to a specified depth. The slab's boundaries are currently established by two diagonal corner points. Either existing points, user entered coordinates, or screen positions may be used to identify the corners. The slab feature could be extended to use a polygon as the

boundary, however, this has not yet been implemented. Slab machining finishes a region's bottom surface, but does not finish the sides. If the sides require finishing, either a pocket feature or an additional profiling tool path are required. Figures 7 and 8 illustrate a slabbing operation and Figure 9 the access vectors and corner points for a slab.

Figure 7: Recessed Part Surface

Figure 8: Slabbing Area

Figure 9: Slab Volume With Tool Access Choices

Part Boundary (Open and Closed) and Cutouts

Open and closed part boundary features are used to machine contours that are not part of other features such as slots, notches, or pockets. Figure 10 illustrates both open and closed boundaries. The open part boundary is a shoulder which remains after the slab is machined, and the closed part boundary is a small shelf which is cut around the top of the pocket's exterior contour.

A part boundary feature is defined by either open or closed geometry, the width of side stock to be removed, the top of stock, and the feature's depth. The side stock to be removed is an

approximate value which is entered by the user when the feature's boundary geometry is selected. The side stock is assumed to be of equal thickness along the entire boundary.

Figure 10: Variations on Part Profiles

Figure 11: Part Boundary Cutout Examples

A special case is available when the part is being cut from a larger piece of material rather than machining all of the side stock to the final geometry. "Cutout" features may be either internal or external. Examples are shown in Figure 11.

Bosses

MA processes a boss as an instance of a closed part boundary.

Bottom Fillets:

Bottom fillets may be specified between the feature's vertical side walls and bottom plane. A number of cases must be considered to determine an efficient sequence of cuts. Flat bottom end mills are preferred to remove the majority of stock and to finish the feature's bottom surface. If the amount of side stock left by the roughing tool equals the fillet radius, then a ball end mill is selected to finish the feature's sides and bottom fillet.

Point-to-Point Features

Point-to-point features include drilled holes, tapped holes, counterbores, countersinks, etc. They are formed by machining with one or more tools at a single point in space along the feature's principle axis.

A typical point-to-point feature requires several machining steps. NC tool paths are created in

ANVIL-5000 for each machining step. For example, the NC program produced by the Milling Assistant for tapping would include centerdrilling, tap drilling, and tapping.

Additional point-to-point features can be "built up" by selections from the menu. The menu is repeated to allow several operations to be grouped and associated with each location. This permits features like "flat bottom drilled holes with milled counterbores" to be defined. MA is able to handle operations such as tapping (described above) automatically, while multiple point-to-point features can be placed at a single location by sequentially selecting the desired features from the MA menu. Only valid combinations are allowed, and the rules force the operations into the correct order.

Point-to-Point Operations:

1. Drill	5. Countersink
2. Tap	6. Counterbore
3. Ream	7. Install Inserts
4. Bore	

Figure 12: Drilling

Figure 13: Tapping

Figure 14: Reaming

Figure 15: Boring

Point-to-point features not yet implemented

1. Flat Bottom
2. Dowel Holes
3. Milled Holes
4. Milled Counterbores
5. Spotfaces

The Milling Assistant references several standard tool tables to complete point-to-point operations. These include:

1. Letter, number, and fractional drills.
2. UNC and UNF taps sizes up to 1.000 in.
3. Standard thread inserts.

Information in these tool tables can be extended as necessary. However, if information is not in the table for a required tool, the user is prompted to enter the necessary parameters.

Machining Strategy

Milling Assistant's knowledge base of machining strategies is used to automate decisions made during the creation of NC tool paths. Providing reasonable, safe solutions to part programming problems requires a skilled craftsman to consider and interpret many variables and working conditions. One goal of the MA project at NMSU has been to accumulate and formalize milling strategies for the purpose of creating robust rules in the decision processor. During the MA project, knowledge engineering techniques have been employed to acquire information from domain experts and the results represented in a form that could be used to develop AI computer programs. This section defines machining, identifies the processes being automated by MA and presents information obtained from the research efforts and strategy meetings conducted with industrial contacts.

Background

A large percentage of manufactured products require machining at some stage of their production. In addition to many direct applications for machined parts, castings, molds, prototypes, and assemblies are often created and developed using machined pieces. As discussed here, the term machining is defined as *the removal of unwanted material from a workpiece in chip form so as to obtain a completed product that meets size, shape and finish characteristics specified by the designer* [5]. Machining operations can be sub-divided into seven basic chip formation processes: turning, shaping, milling, drilling, sawing, broaching and grinding. Parts created using chip type manufacturing operations vary significantly in terms of their application, size, and tolerance and finish characteristics. The following factors contribute to the need for improved techniques to manage the manufacture of machined parts.

- * Advanced machine tool and computer technology
- * Higher quality requirements,
- * New materials such as plastics, composites, and exotic alloys,
- * Smaller batch sizes,
- * More complex part descriptions, and
- * More competitive marketplace

Modern machining processes rely on the expertise of skilled craftsmen to convert large amounts of operating data and design parameters into production strategies and subsequent NC part programs. Machining requires a vast experience with many machine, tool, and material combinations before one can become proficient. Even then, changing conditions in conjunction with unique new part features and fixturing orientations demand that old solutions be adapted to new conditions regularly. Methods for creating machined pieces have improved with advancing technology in the areas of computers, CAD/CAM software, and machine controls. Effective use of modern machining techniques requires an understanding of Computer Numerical Control (CNC) programming, technology implementation issues, and process integration.

Upgrading existing operations in small and medium sized shops is difficult to economically justify on a short term basis. The process of introducing a part to manufacture often involves a sizable dedication in capital equipment. The latest milling and turning centers can cost up to \$500,000 and from \$100 to \$300 per hour to operate. In prototype or small batch production, a large portion of the labor cost incurred can be attributed to NC programming and operations planning. A significant savings can be realized if the creation, verification, editing, and tuning of part programs was done prior to them being downloaded to the controller. The development of improved fabrication and resource management methods are also expensive and time consuming. Efficient management and engineering techniques must be developed so that small and medium sized machine shops can utilize advanced technology and remain competitive in the international marketplace.

Automation of Shop-Floor Functions

Artificial Intelligence (AI) is a valuable tool when used with software applications which address the design and manufacture of products. AI implies the automation of functions which would normally require some level of human intelligence and interaction. AI also deals with that part of computer science concerned with developing intelligent computer systems. Material removal operations provide a rich application environment for automated systems and decision processors. The efficient execution of milling, point-to-point, and turning operations requires a working knowledge of a wide variety of machining conditions and the machinable features which compose a part design.

Process planning for machined parts converts design specifications into detailed shop-floor work instructions on how to create and validate the work-piece. Existing methods for handling machined parts consists of drawings, route sheets, material requirements, tooling, fixturing, numerical control programs and job tracking data. Operations planning is the phase of process planning which determines the manufacturing sequences required to produce a part, identifies the machines capable of doing the job and specifies the strategies, tools and machining parameters for each machine. The number of decisions to be made and the complexity of parts have contributed to making the automation of machining functions difficult. Successful work has been demonstrated in areas where families of similar parts are produced.

The Milling Assistant

MA aids in the programming of CNC machine tools by automating the decision process used to create tool path files. MA combines a user interface to a mature commercial CADD/CAM system and a decision processor which contains the rules and strategies for machining features. The current version of MA utilizes ANVIL-5000 in conjunction with a LISP implementation of a Case Based Reasoning (CBR) decision processor. The user interacts to identify the feature and its design characteristics, access the available cutting tools, select the material to be machined, identify the machine tool to be used, and insert stock parameters. MA creates a tool path based

on the inputs and then interacts further with the user allowing for modifications to the proposed solution. The goal is to make the system learn from the user's changes and produce better initial solutions in the future.

Knowledge Acquisition

A major goal during NMSU's involvement in MA has been to obtain and document "industrial strength" machining strategies. Knowledge was acquired through informal meetings with machinists in which manufacturing scenarios were presented and analyzed. Pictures, working conditions, and machine tool limitations would be given with related questions posed to the participants. The initial scenario was best presented in hard copy form so that the machinist could write comments and answers to questions directly on the form. The sample scenario in Figure 16 illustrates the presentation of a part requiring a pocketing operation in the center. The forms were not designed to work as a questionnaire but rather as a means to spark conversation on related areas during the strategy meetings. Responses were documented and analyzed for the purpose of devising better rules for the CBR. Conditions causing the programmer to vary his approach were also recorded. The people (companies) below provided assistance:

Stan Dornfeld (DORNFELD, San Diego CA)
Eddie Rose (Manufacturing and Consulting Services, Scottsdale AZ)
Craig Fischer (Texas Instruments, Dallas TX)
Charlie Parks (New Mexico State University, Las Cruces NM)
Edwin Evans (Pantex Plant, Amarillo TX)

As modifications to CBR knowledge are reviewed and tested, the decision processor can be improved and "tuned" to work in a specific environment. The information required to create effective tool paths can be classified by ordering the general tasks accomplished by the programmer or machinist. Setup, Approach and Entry, Cutting Methods, Intermediate Motions, and Exit/Withdrawal issues are associated with all machined features. Figure 17 presents general strategy questions that machinists must answer in order to create NC part programs.

To determine effective machining plans, the part descriptions and operating conditions must be thoroughly analyzed. Coding of the part program in any CAM software or MDI interface requires programmers to make decisions relevant to each machinable feature on the part. The information presented in sections below assumes that the following data is available for determining cutting parameters and prescribing specific tool motions.

- Part descriptions (dimensions, tolerances, finishes, etc.)
- Fixture orientations and rigidity
- Available tooling
- Material type
- Machine tool parameters and limitations
- Part quantity

PART NAME: FINPRT3 DATE:

Page 1/1

NOTES ON PART SETUP:

- Rigid fixturing
- 2 1/2 Axis Features
- 6000 rpm = Max speed
- 600 ipm = Max feed
- No obstructions

FEATURES:

- Boundary
- Pocket
- 4 oval cutouts
- 4 drilled holes

Tools Available:

Flat End Mills: 1/16 1/8 3/16 1/4 5/16 3/8
7/16 1/2 9/16 5/8 3/4 1"

Other Tool? _____

FEATURE / INFORMATION

STRATEGIES AND REASONS FOR DECISIONS

Feature: Pocket

Material: Aluminum (6061)

Depth of Feature = 0.5"

Datum is Center of Part

Stock Thickness = 1.0"

Tool Used Tool Mat'l Cut Length # Flutes

ROUGH

FINISH

Speed (RPM)

Feed (IPM)

ROUGH

FINISH

Stepover

Base Cuts

Conv./Climb

Coolant

ROUGH

FINISH

Show the Approach point (A) and the Entry point (E) on the figure to the left. Why??

What Entry method to use for Rough and Finish cuts?? (Example: Ramp/Angle, Direct, Circular blend, etc.)

Exit location, (X), and Exit method:

If the material is changed to stainless (304), which of the above decisions would be effected?

If the depth of the pocket was changed to .75" what changes to the above decisions would be made?

Would constant angle or collapse pocketing be used ??

Figure 16: Sample Scenario Used for Strategy Meetings

Figure 17: Milling Assistant Strategy Issues

Milling Strategies

Features requiring 2 1/2 axis milling can be generalized in the sense that certain machining sequences are common to all features. The machinist makes setup parameter decisions based on the information mentioned above. These parameters include a specific cutting tool and appropriate speeds, feedrates and depths of base and side cut for both roughing and finishing operations. Functional specifications such as dimensional tolerance and surface finish have an influence on the strategies selected during the preparation sequences.

Next, the machinist determines the best way to approach and enter the material for efficient and safe execution of material cutting motions. The tool may approach a feature from either a previously machined feature requiring the same tool, from the tool change position or from some other designated "home" location. The tool usually moves at rapid feedrate to a spot above the feature and plunges or ramps into the material in some manner. Once the tool is at the correct depth of cut, tool motions are executed in a pattern so as to remove the material efficiently, accurately and without causing significant tool wear. These motions may include intermediate "non-cutting" tool motions for re-entry as in the case of deep features or for utilizing solely climb or conventional type cuts as in an open part boundary situation.

After the volume of material has been removed from a feature, the tool retracts in some manner and moves to a location influenced by the next operation, if any, to be performed. It is also desirable to insert some operator instruction information into the CNC program during generation.

The sections below will cover strategies general to all milling features by using the process outlined above. The discussion remains independent of the CAM programming system and in no way presumes to present a "complete" set of machining practices. An attempt must be made to document basic strategies so that NC material removal operations can be better understood at a process planning level. The machining strategy discussion continues to include information specific to certain features and a section on special cases. The outline below describes the flow of information in the strategies section:

General Outline

Setup

- A) Material Machinability Data
- B) Roughing and Finishing Operations
- C) Tool Selection
- D) Cutting Parameter Determination

Before and During Entry to Material

- A) From Position
- B) Approach Point
- C) Entry Method
- D) Start Point

Cutting Methods

- A) Conventional -vs- Climb Cutting

Intermediate Tool Motions

- A) Re-entry point and method
- B) Entry, exit, and approach points

Retraction and Departure

- A) Exit/Withdrawal points and methods

Specific to Features

- Pockets
- Pockets with Islands
- Notches
- Slots
- Boundaries/Cutouts
- Slabs

Special Cases

Sharp Corners

Thin Walls

More on Speeds and Feeds

Slugs of Material

1) SETUP

A) Material Machinability

Materials differ significantly from one another in terms of their composition and ability to be machined. Machinability is defined by a material's surface integrity, reaction to being cut, effects on tooling, and force/power requirements for cutting. The microstructure, inclusion of any coldworking, tensile properties and strain hardenability have been determined to be the most important factors in anticipating the machinability of any particular material. Various hardness factors, such as Brinell, are easily accessible and thus are used to classify materials so that preliminary cutting rates can be determined.

The relationship between material characteristics and machining lies in the fact that an enormous amount of heat is generated at the tool-part interface. The cutting temperature must stay within a certain range for safe and efficient operations to be performed. Coolant considerations play a vital role in maintaining the critical surface temperatures. To avoid abnormal tool wear and early tool failure, cutting rates must be reduced for harder materials.

Hardness should not be solely relied upon to describe the machinability of various materials. Two dissimilar metals could have the same exact hardness rating and machine very differently. To develop more advanced machinability data than now exists in the MA system, the effects of other properties mentioned above should be investigated. The scenario is further complicated by the fact that materials may have inconsistencies within their internal structure. Surface scale, and pockets of hardened material makes each application unique.

Conditions often force shops to use trial and error techniques to set cutting parameters for materials commonly encountered in job requests. When this method is employed, chip color, chip form, surface finish, tool wear characteristics, torque applied at the spindle and sounds made during material removal tell the most about cutting conditions. Machinability tables, commercial softwares, indexes, mathematical formulas and databases can aid in the documentation and utilization of material properties and machinability data.

B) Roughing and Finishing Operations

Identification of machinable features and their associated design parameters is a big step toward closing the gap between the design function and the production floor.

Standardization of a set of machinable features is of extreme importance when it comes to utilizing CAD/CAM systems to generate NC part programs and is a necessity if the process is ever to be automated. A series of roughing and finishing operations are often necessary to achieve the tolerance and surface finish characteristics desired by a part's designer. Machining parameters must be chosen for both types of operations and the tool path files must be generated accordingly. Rough and finish cuts also require unique strategies in terms of entry locations, cutting methods, and the amount of material taken per pass. The operations may or may not require the same tool diameter and geometry.

Dimensional tolerance, positional tolerance, surface finish, surface integrity, and contour are a few of the common design specifications for milled features. Surface finish call-outs for both the base and sides of features are parameters often specified by the designers of parts used in a variety of engineering applications. MA currently has the ability to query the user about these finish characteristics.

Surface finish for milled features (specified in micro-inches or milli-meters) is a measure of how far a surface varies from the reference plane which defines a feature's side wall or base. It is measured by moving a very sensitive stylus across a surface and recording the results on a chart that shows the variation in distance from a designated reference plane. The machine utilized in this type inspection is called a surface analyzer. The resulting data basically measures the "roughness" that a surface exhibits. Typical values for the milling process are around 16-500 μ -inches (16 being called a very "high" surface finish and 500 a "low" finish).

Machinists achieve specified finishes by varying the amount of material that is removed on the final one or two passes across a surface. In addition, the speeds and feeds may be adjusted and the number of flutes on the cutter may be increased to avoid the "waves" that are inherently left by climb cutting operations. Rotational speed and feedrate adjustments vary greatly with respect to machine tool, cutter, and fixtures chosen for the workpiece. Dimensional tolerances are achieved in much the same manner but with more emphasis on continuously reducing the amount of cut as approaching the desired size and shape. Text Boxes 1 & 2 present ranges of side and base cut amounts for various material hardness values to account for surface finishes specified by the user.

C) Tool Selection

The selection of a cutting tool influences all other decisions made subsequently during the generation of an NC part program. Special tooling is very prevalent in modern industry. The task of automating tool selection is difficult but, can be easier if the set is reduced to those available in the particular shop or from a particular manufacturer/supplier. Benchmarking commonly used cutting tools on particular machine tools can drastically reduce tooling costs. Not only are the diameter and cut length critical but, also tool & part materials, machine tool and multiple tool characteristics. In addition, geometry, number of flutes, availability, cost and sharpness can heavily

Sample Category and Corresponding Rules for Side Surface Finishes:

65-125 (μ-inches): Medium Side Surface Finish

If (m-hard-material) ie. Brinell > 200

finish side stock = $0.030 * \phi$ (roughing tool)

spring pass required (ie. extra pass against feature geometry)

If (m-medium-material) ie. Brinell > 100.LE.BH.LE.200

finish side stock = $0.040 * \phi$ (roughing tool)

no spring pass required

If (m-soft-material) ie. Brinell < 100

finish side stock = $0.050 * \phi$ (roughing tool)

Text Box 1: Width of Cut Categories to Achieve Side Surface Finish

Sample Category and Corresponding Rules for Base Surface Finishes:

16-64 (μ-inches): High Base Surface Finish

If (m-hard-material) ie. Brinell > 200

finish base stock = $0.010 * \phi$ (roughing tool)

spring pass required

If (m-medium-material) ie. $100 < \text{Brinell} < 200$

finish base stock = $0.020 * \phi$ (roughing tool)

spring pass required

If (m-soft-material) ie. Brinell < 100

finish base stock = $0.030 * \phi$ (roughing tool)

Text Box 2: Depth of Cut Categories to Achieve Base Surface Finish

influence the actual results realized by a milling process.

There are numerous considerations in selecting appropriate tools for milling operations. The constraints of the feature geometry drive the initial selection of diameter and cut length. On most features, the smallest gap distance between two geometric entities determines the largest roughing tool diameter capable of doing the job and the smallest corner radius creates the upper bound on the finishing tool diameter. Other feature parameters, such as width of opening in notches or slots and smallest gap between an island and a boundary or two islands in a pocket limits the range of diameters for the milling operation.

Minimization of tool motion through the use of a smaller cutter is one documented strategy for tool selection. Figure 18 shows how a smaller cutter actually travels a shorter distance than does a large cutter. If both were travelling at the same feedrate, the smaller

tool would be done faster. On applications where a small amount of side stock is to be removed, smaller cutters can be utilized. Features requiring deeper cuts should be done with stiffer tools so that deflection is not a problem. When clearing an area of material, as with a pocketing operation, the larger tools can remove more material per pass and are thus preferred over a smaller tool. Tools used on previous part features, cutters available in a machine's tool changer and an operators "favorite tool" are often

considerations in the final decision. Depending on the feature geometry and specifications, roughing and finishing operations could be performed with the same tool. Traditionally tool selection has been done by following a simple approach; "use the stiffest, biggest and shortest cutter that will fit within the confines of the feature geometry". The decisions have been complicated by more complex parts, exotic part materials and advances in milling tool materials and geometries. For instance, if the rigidity of the part fixturing is in question and the material is hard, a small tool may be selected to reduce the cutting force exerted on the setup during milling.

Figure 18: Actual Distance Travelled for Various Tool Diameters [9]

The number of teeth on the cutter can influence the results achieved significantly. Theory says to maintain a consistent and particular chip load per tooth for each tool/material combination. The best number of teeth for the application is determined by the thickness of the chip desired, the power available, the finish required and the speed and feed. The use of carbide and ceramic tools has become prevalent for special applications. These tools can be beneficial if fixturing is rigid and the machine has the capability to generate high speeds and feeds as is the case with most modern CNC machining centers. Carbides last longer than High Speed Steel (HSS), are more rigid and can be economically advantageous due to the high hourly cost of running CNC machines. A rather tricky situation occurs in tool selection when machining small corner radii as can be the case with many milled features. To read more, see the section titled "Sharp Corner Milling" further down in the report.

D) Cutting Parameter Determination

Once a tool has been selected, cutting conditions or cutting parameters must be chosen. The cutting conditions affect the operation of the machine tool in addition to drastically affecting the life of the cutting tool and the quality of part produced. The Taylor tool life equation defines tool life as the total elapsed time in continuous cutting to reach a

specified amount of flank wear. The cutting speed and feed may be increased if shorter life is accepted. Tool life is influenced most by cutting speed, then by feed rate, and least by the depth of cut. Sounds made by the machine and tool often dictate the cutting conditions selected on the shop floor. This process is often carried out at the particular machine tool by trial and error.

Experience in production and scientific investigations conducted in machinability laboratories have shown that a change in the feed or in the depth of cut will require a compensating change in the cutting speed if the tool life is to remain unchanged. For this reason a modification in the cutting speed should be made if the feed or depth of cut is changed significantly.

Amount of Base and Side Cut

Definitions:

Side and Base Stock

The amount of finishing stock which is left after the roughing operations is a function of the feature's dimensions and material. Relatively deep features can distort due to material stress when the roughing cuts are made. If stress is a problem, extra material is often retained to allow stabilization by heat treatment steps before the finish cuts are completed.

Side and Base Cut Thickness

Side and base cut thickness calculations are made after the roughing and finishing tools have been selected. Maximum cutting load (cross-sectional areas) calculations are based upon the material being machined, the machine tool's spindle power, the peripheral speed of the cutter, and the number of cutter teeth. This cutting value is considered in conjunction with machinist-provided rules to establish side and/or base cut depths.

The amount of side and base cut taken are directly dependent on the cutting tool diameter and geometry, the fixture rigidity and the machine tool. The following cut amounts are common for materials with varying hardness. (ϕ = cutting tool diameter)

	Soft ($0 < BH < 65$)	Medium ($66 < BH < 130$)	Hard ($131 < BH < 200$)
BASE	$(1 - 1.5 * \phi)$	$(.5 - 1.0 * \phi)$	$(.1 - .5 * \phi)$
SIDE	$(.6 - 1.0 * \phi)$	$(.3 - .7 * \phi)$	$(.1 - .4 * \phi)$

The consideration of a fixture rigidity factor is applicable in this situation. Depth of cut on roughing operations as well as feedrates are influenced by how secure the workpiece is. The rigidity factor could easily be used to scale output machining parameters to attain satisfactory results in this type of situation.

Finishing operations take much less material per pass. Anywhere from 1 - 10% of the finish tool diameter are common finish side and finish base stock values; with 1 being for hard and 10 for soft materials. The values chosen here are also adjusted based on the surface finish and dimensional tolerance specified in the part description. Less material is taken on the final pass if high surface finish is called-out and the actual shape and size is "approached" using smaller and smaller cuts if dimensional tolerance is a consideration. Both operations may require a "spring" or "free" pass in which no new material should be removed but, if tool deflection occurred during previous passes, the feature edge could actually be tapered as usually happens with small diameter tools on deep features. In this case carbide tools may be used to gain tool rigidity.

The order of cuts is also an important factor in pocketing. Machinists say that base cuts should be executed first, especially in finishing operations, to avoid waves that can be left if a different tool is used for a final side finish pass.

Methods for Determining Milling Speeds and Feeds

A important consideration in the machining process plan is the assignment of appropriate spindle speeds and feedrates. A part programmer relies on tool, machine, fixture and material parameters as well as past experience to choose good values of speed and feed for any particular application. Even then, many times the values are ultimately adjusted by the machinist once the job has begun and the machining process is observed and evaluated.

Speeds and feeds are calculated after the rough and finish tools have been determined. The most common approach goes as follows: The spindle speed is calculated for the specific tool diameter using recommended surface feet per minute cutting speeds. After checking the spindle speed against the limits for the machine tool, the tool's feedrate is set to obtain the recommended chip thickness for the spindle speed. The value is ultimately compared to the limitations of the machine tool motors and adjusted accordingly

The cutting speed refers to the speed at which the cutting edge of the tool passes over the surface of the workpiece. The value is usually given in feet per minute and is often specified based on the Brinell Hardness (BH) of a material. The cutting speed can be converted to a spindle speed in rev/min. by using the following equation:

$$\text{Spindle Speed} = (\text{surface cutting speed} / (\pi * \text{tool diameter})) \\ (\text{rev./min.}) = (\text{inches/min.}) / (1/\text{rev} * \text{inches})$$

Feedrate refers to the speed at which the cutting tool is traversed across the surface of the workpiece. The maximum feedrate is limited by the machine tool capabilities, tool and part deflections, specified surface finishes, and tooling capabilities. The feed rate along with the depth of cut determine material removal rate on any pass. As a rule of thumb, it is more efficient to remove material with large depths of cut and slow feedrates rather than vise-versa and better to increase feedrate over cutting speed. Feedrates are usually given in units of IPR (inches per revolution) or IPM (inches per minute). In milling it is common to give feeds in terms of the chip thickness to account for varying numbers of flutes. There are many factors to consider in selecting the feed per tooth and no formula is available to resolve these factors. Among the factors to consider are:

- a) cutting tool material
- b) work material and its hardness
- c) width and depth of cut
- d) type and size of milling cutter
- e) surface finish requirements
- f) power available on machine tool
- g) rigidity of setup

- * milling machine
- * workpiece
- * fixturing
- * cutter
- * cutter mounting

The feedrate in IPM is found from multiplying the desired chip thickness in IPT (inches per tooth) by the R.P.M selected and by the # teeth on the cutter. When speed and feed combinations have been calculated, it is best to relate them back to the machine tool parameters so as to fully utilize the power of the machine. The higher end of feedrates possible should be used because lower feeds reduce tool life. Additional variables must be accounted for when machining features that are contoured or made up of contoured surfaces.

There are a multitude of different speeds and feedrates to be considered for each operation performed. Each of the following sequences should be carefully considered:

- 1) Rapid Traverse
- 2) Entry rates during plunges, ramps, or special motions
- 3) Initial slotting cut (ie 100% tool width cutting)
- 4) Subsequent passes (Some % < 75 on each pass)
- 5) Corners
- 6) Intermediate tool motions
- 7) Retract and Exit moves

2) Before and During Entry to Material

A) From Position

The from position refers to the location of a tool just before motion toward the feature currently under consideration. It may be the tool change location, a point above a previously machined feature which required the same tool or some other designated location above or to the side of a part. Sometimes operators want tools to move away from the part for inspection purposes between machined features. For more efficient machining, care should be taken to strategically plan exit points for certain features so that the best entry point for the present feature can be reached quickly. The from position should be located so that a clear path to the machining area can be reached with a rapid motion. Care should be taken to ensure that fixturing and other obstacles are cleared easily as the tool approaches the part.

B) Approach Point

The approach point is a point in space that can be directly above, adjacent to, or anywhere near a feature. The point is strategically placed so that the tool can enter the workpiece. If possible, with flat end mills, the approach should be at the same cut depth as the feature to be machined. This means that the cutter will be cutting solely with the side of the cutter rather than with the ends which reduces tool life quickly. As is the case with pocketing or internal and external cutouts, a ramping move, direct plunge, helical move, or predrilling are necessary moves. With outside boundaries, notches, slots and slabs, these operations can be avoided by moving the tool in from the side of the part or other open area. This method seems to be preferred. More will be discussed on this area under Entry Methods below and for individual features.

C) Entry Methods

Entries into the workpiece material are responsible for a large part of tool wear which takes place during milling operations. The high stresses which occur at the flute tips along with the heat deformation that takes place combine to shorten tool life drastically. If they can be programmed in efficiently, pre-drilling operations are preferred to most other entry methods.

If pre-drilling operations are utilized, other considerations are important. The mill should not be centered with a predrilled hole of the same size or slightly larger. If this is done, a continuous chip is formed around the tool and cutting is temporarily interrupted. Rather, the cutter should be offset by 10-15 % of the tool diameter so that a clear path for chip flow can be created in the remaining part of the drilled hole. Furthermore, the drilling leaves material in holes due to the drill tip angle. Subsequent feedrates for the milling cutter into these holes should account for the material still remaining at the bottom. Failure to do this can result in abnormal tool wear. A minor detail concerns the

break-through point for a pre-drilled hole. If the feed and speed are not reduced during break-through, work-hardening takes place in a local area around the surface of the part. When a mill subsequently moves in for cutting, the tool may be damaged by the hardened area.

a) From above part feature

Plunge:

Milling cutters generally are not designed for plunging directly into materials. The direct entry is used when a center cutting end mill is available and/or if the workpiece material is soft. Plunges should be avoided for deep cuts and harder materials. In these cases an alternative entry method should be utilized or a pre-drilled hole should serve as a starting location. Plunges have been proven to lead to decreased tool life due to the enormous stresses created at the flute tips.

Ramp:

Entering the part material at some angle is a preferred method over plunging because more of the cut takes place on the side of the tool than on the bottom. Often ramp moves create problems because feature constraints and fixturing are not thoroughly considered when calculating ramp angles and distances. Many times the constraints are very limiting in terms of the length of ramp available. Ramping should be avoided at angles steeper than 45 degrees. In the cases where conflicts occur, helical moves can be considered (since they require a shorter x & y travel distances than do ramp moves) or a pre-drilled hole should be used as an entry location. An extreme example of this situation would be the case of a deep small pocket in hard material. Feedrate is an important consideration when ramping. The longer and shallower the ramp move is, the faster the feedrate can be. On the other hand, longer ramp moves are more likely to encounter obstructions and other part features than are short moves.

Helical:

Helical entries are efficient in that they do not require a large operating space to take place. It requires a rather complex motion to be executed and can cause problems if the machine tool does not have this capability. Therefore, if the CNC programming is taking place in a monitored environment with documented machine tool capabilities, the move is good. On the other hand, programming helical moves in an environment where any number of machines might be used, should be avoided.

b) From Adjacent to Part Feature

Entering a feature from the part edge, a previously machined feature or some other open access location is preferred to any of the vertical entries mentioned above. If the tool can

be moved into the material from an adjacent spot, all the cutting can take place with the side of the cutting tool. This is how many milling cutters have been designed to operate. Tool life can be extended if tools are used appropriately and conservatively.

Normal to Edge

Moving directly into the part feature from a side is appropriate for many notch, slot, boundary, and slab features. The tool is moved down in air to a certain depth of base cut and then moved horizontally into the part material. On subsequent base cuts the motion is repeated.

At An Angle to Feature Entity:

Moving in at an angle to some geometric entity that defines a feature's perimeter allows for smooth and un-noticeable entry and exit locations. This, as well as the circular entry discussed below, are most common for finish cuts where dwell marks and tool entry/exit inconsistencies want to be avoided.

Circular Blend

Figure 19: Features Requiring Circular Entries

Figure 20: Sample Part

Circular blend entries are often executed when the feature edge is defined by an arc. This type entry, in conjunction with a circular exit can avoid tool marks at the respective locations. Milling Assistant relies on the first entity selected in a feature profile to use as the starting place. An enhancement to the current system would be to determine if the first entity in the chain was an arc, and, if so, take the bisector of that arc and use it as

the entry point, using a circular blend. The tool could then execute its motions and exit at a point one tool diameter past the bisector, with a circular motion out. These types of motions during tool entry and exit can drastically increase the quality of surface finishes achieved with milling operations. Circular blends for finish cuts apply to a variety of features; part boundaries, pockets, cutouts, and pockets with islands should be considered for blend entry moves. The tool gets to a designated location in the path and gradually moves into or away from the part at a specified radius. [Figures 19 and 20]

D) Start Point

The start point is usually defined in reference to the end point of some geometric entity that makes up the feature perimeter. With plunge, ramp and helical type entry moves, the start point is at the depth of the base cut to be taken on that particular pass. On the other type entries, the start point is at the same depth as the approach point. The move from the approach point to the start point should be at an appropriate feedrate. Many times during plunge, ramp, and helical moves, the tool life is in danger because the bottom edges of the tool flutes are exposed to high stresses and temperatures. The start point is chosen as the place where cutting patterns will begin to remove the material contained in the part feature under consideration. Entering feedrates should be 50% or less of the milling rate figured for cutting. More examples will be presented for individual features.

3) Cutting Methods

A) Conventional -vs- Climb Cutting

Milling tools can remove material in various fashions. In Conventional milling, also known as "up-milling", the cutter rotates so that the forces created push out against the workpiece and oppose the table movement. This is the traditional method of machining due to the fact that the force against the table avoided the "backlash" or free-play in the table lead screws. Modern ball type lead screws have since corrected this shortcoming of older milling machines. In this orientation, the cut is made by an upward movement of the cutter teeth as the workpiece is forced into the spinning cutter. An important benefit of conventional milling is that each tooth on the tool cuts underneath the surface, thus avoiding the surface scale and hardened layer

Figure 21: Climb-vs-Conventional [5]

that can be predominant in cast iron and forgings. It does not have to first break through surface scale because it cuts into the material left by the previous tooth. Materials which work-harden during machining are best cut using climb milling. Climb (down) milling produces forces which act in the same direction as the table thrust and thus requires less force to move through the material, however, rigid setups and machines are necessary to avoid chatter. Only material that is free from rough or hard surface scale is recommended for climb milling. In addition, a common problem related to chips sticking to the cutter is avoided with climb milling. With each pass through the material, the cutter tooth is rotated into the material which dislodges any chips attached. In conventional milling the chip is forced into the material after it has been cut away. Climb milling is also preferred in thin, narrow, or irregular pieces because it forces the work into the table and creates a more rigid setup for supporting the workpiece.

Feature by Feature Analysis

1) Pockets

Collapse pocketing is preferred for simpler pockets and constant angle for the pockets with complex boundaries. Collapse pocketing is the technique for machining in one direction (such as clockwise) starting at the center and finishing around the periphery. Both cutting patterns are shown in Figure 22.

Most tools used for machining pockets are not designed to plunge directly into the material. To solve this problem the tool can enter into the material at a pre-drilled approach point or it can ramp into the material at a small angle. Newer motions, such as helical entries, can be useful but may also create problems for certain machine tools. A drilled hole at the tool's approach or plunge point is often necessary. The hole diameter is often drilled slightly larger than the milling tool. Very tough materials always required plunge holes.

Figure 22: Constant Angle and Collapse Pocketing

The tool's approach point selected by MA depends upon the type of ANVIL-5000 pocketing which is selected. If collapse pocketing is selected, the pocket's center of area

is used as an approach point, which in most cases, is near the start of the cutting motions. If constant angle pocketing is selected, the approach point is calculated from the geometry to be in the pocket's lower left corner.

Constant angle pocketing requires a direction for the cutting motions. The total number of tool motions is smallest if the constant angle direction is selected along the pocket's major axis.

To conservatively account for rigidity and power considerations, a full cutter width must be assumed when machining pockets since at least some of the cuts will be at full width. Therefore, the cutting depths are calculated for that situation in order to stay conservative. A more ideal situation would allow for multiple feedrates to be specified depending on the type of cut being executed (ie slot, rough, %tool, finish).

2) Pockets with islands

Tool diameter selection for pockets with islands is limited by the smallest space the tool must fit between the boundary and all islands as well as between each island. Care must be taken to include material left on each face for finishing operations. The current implementation requests the user to either pre-select the tool or define a maximum tool diameter that will fit in the gap space described above.

3) Slots

Slots are machined using tool diameters that are less than the slot's width. This allows the slot's sides to be finished separately with appropriate tools and feedrates. Tool selection and cut depths are a function of the slot's width, depth, and bottom fillet. Open slots are frequently machined by first cutting a center channel down to the feature's full depth. The channel's width is then enlarged by rough cuts on both sides. These cuts are followed by the side finish passes. The rough side cuts may be omitted if the remaining material can be removed by the finish passes.

Implicit in an open slot's definition is the assumption that entry and exit for the tool is available at both open ends. The area included in the extension of the slot's boundary geometry by one tool diameter is used for all entry and exit motions. When intermediate withdraws are required, they are made to a position above the slot's top surface.

4) Notches

The three sided notch is machined using an initial channel cut which follows the notch's boundary down the feature's full depth. Roughing side cuts follow the channel cut to enlarge the notch's width. The finish cuts generate the final feature geometry and surface finishes.

The width at the notch's opening will determine the maximum allowable tool diameter for the three sided notch. Entry and exit from the notch is at the open end and is similar to the geometry extension method used for a slot. Any intermediate withdrawl motions are also made at the notch's open end.

Often, notches have a little opening which forces the use of a small diameter tool. If this notch opens into a larger area, say, that looks like a pocket, a small tool will take many motions to remove the material or leave a slug in the center. This situation could be solved by using extra geometry to close the bigger area off, machining it as a pocket, and take the remaining portion as a slot.

5) Part Boundaries

Entry and exiting locations for a part boundary are at the first and last geometry elements which were selected. For open boundaries, geometry extensions are used for access, similar to the method which is employed for slots and notches. Closed boundaries use direct entry for roughing cuts and circular blends for finish passes.

The slotting (full diameter) cuts that are unavoidable with cutout features are both hard on the tool and difficult to control (This concept also applies to pockets, notches and slots). Slotting cuts can cause so much tool deflection that 3-5% of the roughing tool diameter must be left for finish passes. The tool can deflect enough to cause a noticeable taper. In addition, climb cuts result in the tool being pulled into the material which can cause gouging. Safe clearance distances from critical geometry should be kept on all slotting type cuts.

6) Slabs

Tool access to the slab's material is defined by the NC programmer selecting up to four direction vectors. The access vectors allow the programmer to choose entry directions which are free of clamps and/or other part features. If access is not specified by selecting direction vectors, the slab is machined like a collapse pocket with the tool either plunging or ramping into the material from the top. However when side access is specified, the tool will enter and exit the material along the specified side(s). Slabbing operations are frequently used to remove inconsistencies and minor contour changes on the surface of a work-piece. The operation can also be used to comply with surface finish tolerances specified by the part designer. Finally, part designs that consist of various surface levels are good candidates for needing multiple slabbing operations. The reaction from the questioned machinists has been that a user defined polygon slabbing operation would be helpful and avoid the "tool cutting air" situations that exist in applications of a rectangular slabbing routine.

Special Cases

1) Sharp Corner Milling

Milling of sharp corners (ie. small fillets) was a popular topic of discussion among machinists. It not only heavily influences tool selection and roughing/finishing patterns but, it also introduces options on how best to achieve the desired results.

The selection of roughing and finishing tools is driven by geometric constraints related to the feature. In the case of small radii corners (say $< 1/4"$) as are common in pockets, notches, and inside and outside cutouts, a large roughing tool can be selected. But, if this tool is more than about 1.5 times the finishing tool required for the small corner radii, large amounts of material will be remaining for the finishing operation. Figure 23 illustrates this concept. This, in turn, creates alternatives with respect to how the best results can be achieved.

Figure 23: Influence of Small Fillets on Tool Selection.

Options for Milling small fillets:

- "Bite" away at the corners with numerous side cuts.
(difficult to program passes)
- Pre-drill the corners and then remove the remaining material with a milling path.
(requires more programming, tool change, mixed results)
- Drastically reduce feedrates as entering and exiting corner and include spring pass.
(more programming, increased tool wear, risky)
- Reduce size of roughing tool so that not so much material is remaining for finish cuts taken with small diameter tool.
(increases machining time, re-programming of feature)
- Select intermediate tool for removing corner material.
(one more tool change, more programming)

Sharp open corners, as on part boundaries and islands in pockets are generated by a quick change of direction by the cutter. When this motion occurs, a small burr is left on the

otherwise finished corner. The problem is related to the fact that there is no natural deceleration of the machine tool as it approaches the corner and changes direction. A proposed method would be to generate a very small radius circle record so that the pause needed to remove the burr could be achieved. Another possible solution to the problem would be to allow modified feedrates to be inserted for inside and outside corners involved in the feature under consideration.

2) Thin Walls

Thin walls could be encountered with any one of the milling features available in MA. This term refers to a layer of material left behind after all machining has taken place. The area is critical because the wall could be forced away, pulled in by the cutter or left out of tolerance due to bad machining strategies. The walls under concern must be approached with smaller and smaller cuts as was the case with addressing dimensional tolerance described earlier. Fixturing or a backing plate may be used to hold the thin wall in place while machining operations are performed on the area. Also, a combination of climb and conventional cutting may be used to take advantage of the lifting action that takes place with conventional or the pushing action produced from climb cutting. The downward pressure from climb cutting is preferred because the tool presses on the workpiece, which in turn, forces the part into the table, the table into the machine, and so on. The whole setup becomes more rigid and stable.

There are other alternatives which allow the material around the thin wall to be strengthened either before or after other machining operations have been performed. Sometimes an intermediate heat treating step is used to relieve stresses from the part. Final passes are then executed at the finish dimensions of the feature.

3) More on Speeds and Feeds

Many times machinability data provides only starting point values for speeds & feeds for milling and drilling operations. Indexing schemes for calculating machining parameters are very useful when considering variables that effect the values chosen. Speeds, feedrates, depths of cut, and width of cut are all good candidates for the application of an indexing system. Each of the above parameters influences the vital "chip-load per tooth" factor which is many times used as the basis for evaluating material removal efficiency. It is a measure of the work being done by the tool. As an example, an index may be devised which calculates speeds and feeds for the most commonly used materials and tools within a particular shop. One method, the Dornfeld Index (Stan Dornfeld, San Diego CA) uses benchmarks for a single tool of a certain type to determine feeds for all other diameter tools of the same type based on the chipload factor. The basis for the philosophy is that a tool looks the same to material being cut regardless of its diameter. Mr. Dornfeld states the following reasons as to why milling is such a difficult puzzle to solve:

- The cut is intermittent (chip thickness changes)
- Cutter geometries and capabilities change with manufacturer
- Theories that work well in one material fail in others
- Full diameter cuts do not receive coolant as expected

Mr. Dornfeld believes that once an optimum chip load is established for a tool type, and is defined as a percentage of the tool diameter, it can become a constant that provides proper feed for any given speed, regardless of the specific tool size. Including an indexing system for the materials most commonly encountered at DORNFELD. has proven to be both useful and economical at his shop in San Diego.

On finish passes, especially on hard material, Dornfeld slows speeds and feeds down until a "carving" action is achieved. The smooth cutting of material leaves high surface finishes and results in low tool wear.

4) Slugs

Solid pieces of material remaining after a milling operation are both dangerous to the operator and potentially damaging to the machine and workpiece. If a slug of material lodges in the area being machined, the part could be damaged, the tool could break off, or the slug and or tool can fly dangerously through the air. It is best to take care of slugs by bolting them down or making them into chips.

Formalization of Milling Strategies Using IDEF3

The acquisition and documentation of machining strategies was discussed in the previous section. Much of the work in achieving a "knowledge rich" decision processor involves the formalization of information gathered from domain experts and the subsequent development of the associated AI rules. One of NMSU's goals during the MA project has been to enhance the capability of the system by introducing new (or improved) knowledge that better represents a "consensus" approach among machinists. This section of the report extends the knowledge presented in the previous section to include a formal model of the decision making process followed by machinists or NC programmers as plans for specific part features are developed. The IDEF3 (Process Description Capture, Knowledge Based Systems Inc.) [7] structured modeling technique is described as a method for capturing and classifying machining expertise in a graphical manner.

Background on IDEF and Knowledge Based Systems

In 1981, the U.S. Air Force Integrated Computer-Aided Manufacturing (ICAM) program developed structured methods for applying computer technology to manufacturing. These methods were designed to better understand how to best improve manufacturing productivity. There are three well-documented modeling methodologies defined and developed around the ICAM Definition (IDEF) approach to system study; IDEF0, IDEF1, and IDEF2.

The Air Force contractor responsible for the continued development and maintenance of the public domain IDEF system/software definition, design, and engineering methods is Knowledge Based Systems (KBS) Inc. (College Station TX). KBSI has expanded the original IDEF languages to include IDEF3-IDEF6. Text Box 3 describes the original IDEF modeling techniques along with the KBSI additions since being founded in 1988.

The IDEF methodology can be applied as a descriptive tool or an analysis tool. The descriptive tool uses heuristic models and is applied to complex problems such as manufacturing processes. The analysis tool requires a more in-depth evaluation and is applied to more narrowly defined problems such as business operations or practices. The idea is to establish a common language to enhance communications and evaluation and to stimulate organizational learning. It uses a graphical approach to describe the manufacturing functions of a system. A standard notation has been developed that includes boxes or blocks to represent activities or functions and various connecting arrows that have specific representational purposes.

The modified version of DeVries [6] diagram in Figure 24 uses the IDEF standard block diagram [10] to illustrate the considerations that go into material removal process planning. The inputs to the process are on the left, the outputs on the right, the constraints on the top, and the mechanisms for converting the inputs to outputs on the bottom. The diagram points out many important issues associated with machining and manufacturing.

IDEF0

The IDEF0 Function Modeling method is designed to model the decisions, actions, and activities of an organization or system. It provides a functional description of a manufacturing system in terms of activities and relationships among these activities. IDEF0 was derived from a well-established graphical language known as the Structured Analysis and Design Technique.

IDEF1

The IDEF1 Information Modeling technique is viewed as a method for both analysis and communication in establishing requirements for the process. It represents the information flow to support the manufacturing functions. In CIM applications, IDEF1 is used to:

- 1) identify what information is currently managed in the organization
- 2) identify which of the problems identified during the needs analysis are caused by a lack of managing appropriate information, and
- 3) specify what information will be managed in the improved system.

IDEF2

The IDEF2 Dynamics Modelling method is a simulation design tool that provides a language for representing models of the time varying behavior of resources in a manufacturing system, providing a framework for specification of math model based simulations. It was created to improve the simulation design process for predicting a systems actual behavior under certain conditions.

IDEF3

IDEF3 (Process Description Capture) is a scenario-driven process flow description capture method, and it's goal is to provide a structured method for expressing a domain experts knowledge about how a particular system of an organization works. * A more detailed description of IDEF3 and AI3 is given on the following page.

IDEF4

IDEF4 (Object-oriented Design Method) is intended to be used as a design method for automated systems implementation. There are two primary goals of the method.

- 1) provide support for creating object-oriented designs whose implementations will exhibit desirable life-cycle qualities and reduce total implementation development time, and
- 2) make it easy to evaluate object-oriented code to determine whether or not the delivered product both conforms to the design and exhibits the desired life-cycle qualities.

IDEF5

IDEF5 (Ontology Description) models the concepts and the conceptual relations for a domain. Conceptual modeling provides an abstract level of representation for describing a problem domain and / or system which closely reflects the human conceptualization of that domain including representation of real-world objects, attributes, and functions. An ontology represents the theory of what exists in a domain.

IDEF6

IDEF6 (Design Rationale Capture) is intended to be a method with the representational capability to capture information system design rationale and associate that rationale with the design models and documentation for the end system. Thus, IDEF6 attempts to capture the logic underlying the decisions

Text Box 3: ICAM Definition (IDEF) Methodologies

Design of a Machining Process (similar to DeVries, pg.3)

Figure 24: Standard IDEF Block Diagram applied to material removal operations.

Description of IDEF3

IDEF3 is a scenario driven process flow description capture method, and it's goal is to provide a structured method for expression of the domain experts knowledge about how a particular system of an organization works. The following section (Titled: Process Flow & Object State Descriptions With IDEF3) was taken from the IDEF Family of Methods for Concurrent Engineering and Business Re-engineering Applications document supplied by KBSI.

One of the most common communication mechanisms to describe a situation or process is a story told as an ordered sequence of events or activities. IDEF3 is a scenario-driven modeling method created specifically for these types of descriptive activities. IDEF3 is based upon the concept of direct capture of descriptions of the precedence and causality relations between situations and events in a form that is natural to experts in an environment. An IDEF3 description can be used to provide the data for many purposes including the following.

1. *To provide a systematic method for recording and analyzing the raw data that results from fact-finding interviews in a systems analysis project.*
2. *To determine the impact of an organization's information resource on the major operating scenarios of an enterprise.*

3. *To provide a mechanism for documenting the decision procedures affecting the states and life-cycle of critical shared data (particularly manufacturing, engineering, maintenance, and product definition data).*
4. *To define data configuration management and change control policy definition.*
5. *To support system design and design tradeoff analysis.*
6. *To provide powerful mechanisms to support the generation of simulation models.*
7. *To provide useful information for the creation of functional (IDEF0) models.*
8. *To facilitate process mapping for the design of software to achieve real-time control by providing a mechanism for clearly defining the facts, decision points, and job classifications.*
9. *To provide an analyst with a method to clearly define the data needed to develop needs and requirements from a user viewpoint.*
10. *To collect and express the views of domain experts required for the development of expert systems.*

IDEF3 is used as an intuitive knowledge acquisition device for describing a system by capturing processed knowledge and structuring in within the context of a scenario. There are two modelling modes that exist within IDEF3: process flow description and object state transition description. A process flow description models "how things work" in an organization, and an object state transition description summarizes the transitional states an object may undergo throughout a particular process.

Utilizing AI3 (KBSI) to Formalize Knowledge for the Milling Assistant

The AI3, meaning artificial intelligence, software implementation of IDEF3 is being used to store knowledge gathered from machinists during the Sandia National Laboratories Milling Assistant research project between 1990 and 1993. Strategies for milling 2 1/2 axis features (pockets, slots, notches, boundaries, etc.) are stored and analyzed through the use of an AI3 scenario driven model. The goal is to develop a knowledge structure that can drive the generation of decision processing rules for the automated MA system. Techniques learned can be used to develop models for other NC machining operations such as turning, punching and cutting.

Figure 25 represents the structure of the decision process necessary to plan NC programs for individual 2 1/2 axis milled features. The structure pertains to the type of ANVIL-5000 tool paths required to perform the operation. The entire model can include a description, purpose, context, and viewpoint. AI3 allows users to store information related to each Unit of Behavior

Figure 25: Model of Milled Features Developed for MA

described in the process diagram. A Unit of Behavior (UOB) is the basic unit of the IDEF3 graphical description when using a scenario driven model. UOB's are connected to one another through the use of junctions and links. Each UOB can be decomposed into sub units to allow thorough descriptions of each stage in the scenario.

The Milling Assistant works on the premise that geometric characteristics for individual part features can be interrogated from CAD/CAM databases and basic working conditions queried from the user. The model in figure 25 works like a decision tree in that accumulated data is passed into the 2.5 Axis Milling UOB and from that point on, NC programming strategies are chosen at each junction and UOB and inherited (along with the inputs to the model) by the next stage where further planning occurs. For example, based on the feature type input to the model, the decision process branches at J1 to either a pocketing or profiling path. Pocketing is used for pockets (with or without islands) and slabs. Profiles are used for part boundaries, cutouts (inside and outside), notches, and slots.

Constant Angle: Constant Angle pocketing defines a pattern by which material within a specified area is to be removed. The tool starts in a designated location, and then makes constant cutting motions at some angle relative to the pocket orientation. This method combines both climb and conventional cutting unless intermediate (non-cutting) motions are performed between cuts. The angle of cut is optional but should be selected along the pocket's major axis to minimize the number of cuts necessary to remove the material.

Collapse: In collapse pocketing, all machining takes place in one direction (such as clockwise). The tool can start at the center of the area and work out to the boundary or, cutting may begin at the boundary and move to the center. With this type of milling, the tool always cuts in either climb or conventional mode throughout the process.

Several shape properties are calculated for pockets to aid in the tool size selection and in choosing either collapse or constant angle cutting. The calculated properties include the pocket's area, smallest corner fillet radius, perimeter length, maximum delta X and Y, concave or convex shape flags, and the center of the area. Collapse pocketing is preferred for simpler pockets and constant angle for the pockets with complex geometries. Collapse pocketing on complex pockets can result in wasted time due to the tool moving over areas already cut.

Text Box 4: Information in J3 for determining collapse or constant angle pattern.

Subsequent branches and UOB's in the model identify a sequence of programming decisions made by a domain expert (ie NC programmer or Machinist). A pocket without islands will be used here as an example. Junction Box 3 (J3 in figure 25) would fire a rule to determine whether collapse or constant angle cutting patterns should be used. This determination is based on the complexity of the pocket boundary geometry, the part material, design specifications, or machine tool capabilities. Text Box 4 is representative of the information stored in text windows associated with each UOB. If collapse were chosen (to further the example), Then Junction Box 9 represents the decision to perform a series of roughing and/or finishing operations to efficiently remove material while conforming to pocket design characteristics. The UOB's for Rough, Finish, and Rough & Finish apply to all tool path types in the model. They are only shown in the diagram for collapse pocketing a pocket without islands.

After the rough and finish operations are chosen, a sequential series of decisions is made in order to build the tool path for any of the features in the model. This may include separate sets of decisions for roughing and finishing paths if both are required. The UOB's in figure 26 pertain to the setup, entry, cutting motions, intermediate (non-cutting) motions, and exiting stages in all tool paths. Elaborations on each UOB are available in AI3 so that objects and constraints can be described and separate areas are available for storing notes, sources, glossaries, and facts to be considered. Further work could be done to utilize the Object State Transition Network capability of AI3 to track the value added at each stage of the decision process.

The IDEF methodologies have proven to be useful for understanding the type and structure of knowledge required to produce NC programming plans. AI3 has also provided a way to formally document the information obtained during machining strategy sessions conducted during the MA project. Not all IDEF3 descriptions apply to machining and thus the user is forced into some organization patterns rather than choosing one. A more ideal system would allow strategies to be stored at each UOB which would in turn write out executable rules, making the knowledge easy to modify and test on actual parts. This capability could allow "on the job" tuning of the system so as to conform to individual working conditions and preferences.

Figure 26: Sequence of NC programming decisions associated with all features.

Case-Based Reasoning

Overview of Case-Based Reasoning

Case-based reasoning (CBR) describes a paradigm within the Artificial Intelligence (AI) community where new problems are solved by adapting solutions to previously solved problems. All CBR systems have a case library that is the sum of the system's experience within its domain. When new problems are presented to the CBR system, the cases in the case library are consulted, and the case most similar to the new problem is selected. The solution for selected case can then be adapted for application to the current problem.

CBR can be viewed as an alternative paradigm to rule-based reasoning (RBR). A RBR system replaces the case library with a library of "IF ... THEN ..." rules. The individual rules can be chained together to construct a solution to the problem. RBR is commonly viewed of as a "bottom-up" reasoning method: individual rules are executed and aggregated in order to construct a solution. CBR takes the opposite, or "top-down" approach: an entire solution is found, then minor changes are made to fit the conditions of the new problem.

CBR differs from RBR in other ways. A RBR system is generally built to produce optimal answers to problems. This leads to a large number of rules to cover a general class of situations. Because of the large number of rules, the systems are generally quite slow. On the other hand, CBR systems are built to produce approximate solutions to problems quickly. Because entire solutions can be transferred at once, many of the solution construction rules can be bypassed entirely. As the RBR chains more and more rules together to form solutions in complex domains, the accuracy and justification of the solution becomes correspondingly more complex. CBR systems have the advantage of maintaining a rather short linkage between problem descriptions and solutions; the only reasoning chain to follow may be adaptations or modifications to the case solution.

CBR has two significant advantages over RBR systems. First, many real-world domains are so complex that it is either impossible or impractical to specify all the rules necessary for satisfactory performance. Even if it were practical to specify the rules, it may not be possible to extract knowledge from the domain experts in such a way as to make a rule-based implementation of that knowledge possible. The second advantage can be summed up with the slogan "The secret of expertise is experience".

Secret of Expertise is Experience

A central fact of human cognition that is captured by the CBR paradigm is reflected in the Power Law of Practice (REF). This law says that the more a person performs a task, the better he/she gets at the task. The Power Law makes claims on performance improvement on a single task. However, it also applies to domains as well as individual tasks. Within a domain, the more

problems that are solved provides a larger base of knowledge on which to draw when faced with a puzzling problem. This is the difference between a novice machinist and expert one: although they may both have the same basic understanding of the machines and materials they are working with, it is the experience of the expert machinist that makes him an expert. In that experience, the expert machinist has faces hundreds or thousands of problems that required minor, or even major adjustments to the "standard" methods of machining parts. What this experience gives the expert machinist is not only a broad base with which to compare new problems, but also a large number of ways of adapting the solutions of similar cases to fit his/her needs for the current problem.

CBR provides a means of developing systems with the same characteristics of the expert machinist: having a broad case library with which to compare new problems, and having a large number of adaptation methods at its ready disposal. While RBR systems view expertise as creating a set of rules that completely cover the domain and produce optimal answers, the CBR view of expertise is based on direct analogy with human behavior.

Processing Flow

The processing flow of a CBR system is shown in Figure 27. The basic problem solving cycle in a CBR system is:

1. Input a description of the problem,
2. find a relevant case in the case library,
3. adapt the case's solution to fit the new problem.

The first problem faced by the CBR system is how to describe the new problem in such a way as to make relevant past experiences easy to retrieve, while reducing the amount of search required to find relevant past experiences as much as possible. This problem is often called the *index assignment* problem. CBR systems do not generally describe problems in terms of the attribute-value pairs of the problem itself, but rather assign indices that correspond to a high-level concept incarnate by those attribute-values pairs. The main idea in doing this is that high-level conceptual relationships that are shared by a relatively small portion of the case library are better discriminator than are attributes-value pairs of the problem.

The second problem is how to retrieve a relevant case from the case library. This is generally done by matching the augmented problem description against the cases in the case library. The matching process has the goals of rejecting cases that are too dissimilar to the problem at hand, and to determine which of the remaining cases is the most similar, or "best match" to the problem. The task of determining the similarity metric to be used is always one of the major obstacles faced when developing a CBR system. Some systems use a strict, exact feature match to measure similarity. Here, the most similar case will be the one that has the highest number of exact matches. On the other extreme are methods in which the similarity between cases is defined by the user. Ashley (1990) present a good example of the efficiency of this approach.

Figure 27: Data flow diagram of conventional case-based reasoning.

In his legal reasoning system, similarity is defined by how “on point” a case is to a particular line of legal reasoning. How “on point” a case is to a legal concept, such as “negligence”, is defined by the user. This similarity can then be used to retrieve cases that either support or refute a line of argument being proposed or being offered by the opposing counsel.

After the best match has been determined, its solution must be adapted to fit the needs of the current problem. This consists of two parts: identifying what is different between the problem and the case, and adapting the solution of the case to reflect the differences found. There are numerous approaches to both of these components. To identify what differences are present can be as simple as a syntactic survey of the problem and the case to determine a set of mappings

between objects and relations in each. These mappings are then used to adapt the solution of the case by replacing each object or relation in the solution for the case with the corresponding object or relation sanctioned by the mapping.

A more complex situation arises when syntactic information alone will not suffice to distinguish similarity and differences between problem and best match case. Semantic information is used many times to add discriminatory power to the CBR system. The advantages of using semantic information are that more complex syntactic structures can be used to represent cases, a larger number of distinctions can be made between cases with similar structure, and the equality of non-identical structures can be determined. However, the disadvantage of this approach is that the added information comes at the price of requiring additional computation for evaluating the case structures.

Once the case solution has been adapted to the current problem, it must be evaluated for its efficacy as a solution to the problem. If the adapted solution is satisfactory, then the problem and its solution are stored in memory in roughly the same spot as the best match case. However, if the adapted solution is not satisfactory, then the reasons why the adapted solution do not apply to the current problem need to be explained, and repaired if possible. Indexes must then be added to the case library so that the failure may be avoided in the future.

Milling Assistant Processing Flow

The processing diagram for the CBR version of the Milling Assistant is shown in Fig. 28. The processing done for the Milling Assistant is slightly different than many CBR systems. It can be viewed less as a planning system, and more of a case-based classification system.

The first step after the input of the problem from the ANVIL-5000 system is to generate a number of indices based on the numerical attribute-value pairs of the input problem. Examples of these index generation methods is shown in Text Box 5. During the course of developing the knowledge base for the original rule-based version of the Milling Assistant, it was found that it was natural to describe problems and cases with qualitative terms for numerical attribute values such as side stock (small, medium, large) or material type (soft, medium, hard).

The next step is to make the most specific classification of the problem as possible given the instances already stored in the case library. The most specific classification is desired, because that classification will impart the most information on the input problem. To facilitate this classification, the case library is stored in the form of a decision tree, where each node in the tree distinguished between cases having particular parameters (see Fig. 29). The CBR system compares each attribute-value pair of the input problem with the nodes of the decision tree. In the course of classifying the input problem, the CBR system computes values for attributes whose values are unknown. These values can then be used to further specialize the input problem. At the time the problem is classified at its most specific category, all attributes of the problem will be calculated as a side effect of the classification process.

Figure 28: Data flow diagram for the case-based version of the Milling Assistant.

When the system computes values to fill unknown attributes, it is using default computational methods called “CALC-FNs”, or calculation functions. These functions are defined by the system developer to provide a “default” behavior the system can fall back on when no more specific knowledge applies. The user can override this information by adding his own calculation functions. By doing so, the user can tune the system to the particular manufacturing style he/she is comfortable with.

Why the Difference?

There are a few differences between the conventional CBR flowchart shown in Fig. 27 and the Milling Assistant flowchart shown in Fig. 28. There are a number of reasons for this.

```

;; All materials have a brinell number
(defmop m-material (m-root)
  (brinell nil))

;; Soft materials have a brinell number below 100
(defmop m-soft-material (m-material)
  (brinell m-range (below 100)))

;; Medium materials have a brinell number between 100 and 199
(defmop m-medium-material (m-material)
  (brinell m-range (above 99) (below 200)))

;; Hard materials have a brinell number of 200 or above
(defmop m-hard-material (m-material)
  (brinell m-range (above 199)))

```

Text Box 5: Index generation methods used to assign qualitative labels to numerically valued attributes.

One of the most immediately obvious aspects of the CBR version of the Milling Assistant that is different from conventional CBR systems is the fact that the Milling Assistant deals almost exclusively with quantitative data. This is very different from conventional CBR systems that rely of being able to have reliable measure of equality for attribute values. This may seem odd at first, since numerical parameters have traditional algebraic equality to rely on. However, in different situations, a 0.01 difference between attribute values may or may not be a significant difference. In other words, algebraic equality does not give a general enough similarity metric with which to compare attribute values. Thus, where conventional CBR systems retrieve a solution to a similar case then adapt it, the Milling Assistant must compute some qualitative labels for numerical attribute values and functionally compute the remaining values.

A second major difference is that the Milling Assistant has no general adaptation rules. Conventional CBR systems often have a number of general adaptation rules that allow the system to autonomously adapt solutions to previous cases to the current problem. An example of a general adaptation rule is "If an attribute that is computed in the retrieved solution is not needed in the input problem, then do not compute it for the problem." Because of the level at which the Milling Assistant works, rules such as these do not apply.

This last point also highlights a difference between the Milling Assistant and conventional CBR systems. The scope of the Milling Assistant tasks are simply to compute a milling plan for one feature at a time. Thus, the task of the Milling Assistant can be viewed as one of finding the correct set of calculation functions to apply to the input problem. Because all of this processing is done at "one level", it is difficult to find a domain theory for the task that will allow the formation of a set of general adaptation rules. More adaptation rules would be possible if the

Figure 29: Decision tree used to classify problems according to the cases in the case library.

tasks of the Milling Assistant were extended to include creating milling plans for more than one feature at time, or to classify features directly from the CAD geometry.

The Milling Assistant may be better called a case-based *classification* system. This better reflects the type of reasoning that the Milling Assistant does - it takes its input case and tries to find a similar case from the case library that functions as an exemplar or prototype of a particular category or concept. Its output is the category of the new case along with missing features computed using methods specified by the stored case. Following Barrios (1990) we define four aspects of the milling domain about which the CBR version of the Milling Assistant can learn and reason about: features, categories, exemplars, and relations. Features are the terms used to describe the case. An exemplar is an instance of a concept known to the system that has been processed. For example, the exemplar “milling the wide slot in part DEMO1” is an instance of milling a slot (among others). An exemplar represents the most typical instance of a category, which is simply the collection of instances of a concept. Relations between concepts are learned through interactions with the user, collecting rules for recovering from inferences that are not satisfactory to the user.

By behaving as a classification system, the Milling Assistant avoids the problems associated with reasoning from “first principles”, or forward-chaining rules in rule-based reasoning systems. This notorious computational explosion associated with the very large number of possible actions that could be taken from any given state of a problem solving system. Classification systems avoid this problem by pruning the problem solving space using a vocabulary based on categories rather than features. In doing so, the system is continually reducing the size of the search space each time the problem instance is found to be included within a category known to the system.

Implementation

The CBR version of the Milling Assistant (CBR-MA) is implemented in Common Lisp, using the GARNET [12] user interface development environment as the basis for its graphical user interface. The basic components and graphical user interface of CBR-MA consists of roughly 24,740 lines of Lisp code, with another 4,716 lines of code defining the knowledge base. Common Lisp was chosen as the implementation language for several reasons. First, tools such as the GARNET interface development environment were available for use in the Lisp environment, saving a considerable amount of development time. Second, portions of a Case-Based Reasoning system had already been developed for another project and were used as the starting point for this project. Finally, Common Lisp is a dynamic language ideally suited to the memory intensive task of CBR. As a prototype system, it was important not to artificially constraint the size of memory or data structures until the point where an accurate predication could be made of how large the knowledge base would grow with experience. Our experiments have shown that the knowledge base does grow, but that the growth quickly levels off (at the point where one of each feature type has been run) and grows only slightly after.

Integration with ANVIL-5000

This section presents some of the implementation issues involved with the integration of an external reasoning system with the ANVIL-5000 system.

File-based communication

The most significant aspect of the integration of the Milling Assistant and ANVIL-5000 programs is the method used to pass information about the feature being milled from the CAM system to the reasoning system. The method chosen for this project was file-based communication. This was chosen for a number of reasons. First, the communication requirements between the two programs involves a relatively large amount of information to be transferred a very few number of times. Thus, some overhead can be tolerated in the communication method. Second, the GRAPL programming language interface does not internally provide hooks to the Remote Procedure Call (RPC) libraries, or allow the execution of compiled blocks of code (i.e. “.o” files). It does, however, allow the transfer of control over to a C-language program. This feature of GRAPL was chosen to implement file based communication between the two programs. Finally,

the simplicity of file-based communication over RPC or socket-based communication was the swing vote in favor of file-based communication. Because this project was ambitious to start with, any time spent developing a communication system took time away from developing the reasoning capabilities of the Milling Assistant.

```
#include <stdio.h>

main()
{
    FILE *fp, *fopen();

    printf("\n\nCalling LISP to run Milling Assistant...");
    /* system("echo '(read-ma-data-file)' > cases/ma-cbr-input");*/
    system("touch $HOME/mastart");

    while ((fp = fopen( "$HOME/madone", "r" )) == NULL )
    {
        sleep(2);
    }

    printf("\n\nReturning from Milling Assistant, with a job well
done.");
    system("rm $HOME/madone");
}
```

Figure 30: File-Based Communication Controller Program. This C-language controller program implements the file-based communication between the Milling Assistant and the ANVIL-5000 program.

A C-language controller program was developed that is called by GRAPL after the "cbrdata.dat" file is written to disk. The program is shown in Fig. 30. Because the program is called by GRAPL after the "cbrdata.dat" file is already written out, it does not check to see if the file is actually present, but instead assumes that it would not have been called if it were not present. The program first "touches" a file called "mastart". This creates an empty file that is used as a semaphore to the Milling Assistant program, telling it to start reading input. The Milling Assistant program, running as a separate process, is in a loop that only looks for the existence of the "mastart" file. When it is found, the Milling Assistant erases the file, reads in the "cbrdata.dat" data file and begins processing. Meanwhile, the C-program is in a loop checking the existence of the file "madone". This file is created by the Milling Assistant when it has finished writing out the "testmill.dat" file. If the file is not found, the C process goes into an idle wait for 2 seconds, then tries again. When it is found, the waiting loop is exited, the "madone" file is deleted, and the C-program returns control back to GRAPL.

Knowledge Representation

One of the keys to successful programs of any kind is to have the right data structure for the intended task of the program. This is also true for AI systems: a successful AI program must have the right knowledge representation to support the reasoning required by the program. For CBR systems, there are several characteristics of the knowledge representation that are desirable:

- The knowledge structures must be recursive. This is to say that the knowledge structures must be able to support embedding of similar knowledge structures to any level required. This automatically discards array-based or even resource-based implementations, unless arbitrary restrictions on the amount of work produced is acceptable in the domain. This also implies that the knowledge structures must be dynamically allocated. This is because the ability to embed structures to arbitrary levels requires an arbitrary number of knowledge structures to be available for use, which can only be done dynamically.
- The knowledge structures must support both structured and attribute-value pair representations of cases. This requirement is necessary for many CBR implementations because the problem is often input to the system as a set of attribute-values pairs, then augmented by the system with further descriptions and added structure. The structured aspects of the knowledge representation is best illustrated by the ubiquitous use of abstraction and packaging hierarchies in CBR systems.
- The knowledge structures must promote efficient matching and indexing. To do so, many implementations make the indices for the case a special part of the case representation itself, eliminating the need for a number of redundant slots.

Basics

The main knowledge representation structure in the Milling Assistant is the Memory Organization Packet, or MOP (Riesbeck and Schank, 1989; Schank, 1982) [14]. MOPs are used to represent knowledge about classes of events or concepts. For example, “milling a pocket” is represented by a MOP containing a set of “norms” or defaults that specify what basic features of the MOP must be present. Some defaults for “milling a pocket” would include the depth of the pocket, the smallest radii, finish requirements, etc. In the computer representation, the norms of a MOP are stored in slots. Slots are a particular aspect of the MOP on which they are stored. Slots have values that fulfill the function

slot-name(MOP) = value.

Thus, in the previous example, the slot names would include the depth of the pocket, the smallest radii, and the finish requirements for the pocket being milled. The values taken on by the slots would indicate the various constraints for the particular problem instance. Thus, the value (or

slot filler) of the depth of pocket may be a real value such as “1.25 inches”, or a qualitative value such as “deep”. These values can then be used by the system to make distinguishing inferences between the current problem and the cases stored in the case library. An example MOP with a number of slots and slot fillers is shown in Text Box 6.

```
(defmop m-mill-part-notch-pocket (m-mill-feature)
  (max-rough-side-cut m-pattern (calc-fn calc-max-rough-side cut))
  (max-rough-base-cut m-pattern (calc-fn calc-max-rough-base-cut))
  (max-finish-side-cut m-pattern (calc-fn calc-max-finish-side-cut))
  (max-finish-base-cut m-pattern (calc-fn calc-max-finish-base-cut))
  (overlap 35)
  (profile-approach-x m-pattern (calc-fn calc-notch-profile-x))
  (profile-approach-x m-pattern (calc-fn calc-notch-profile-x))
  (
```

Text Box 6: MOP with slots and slot fillers.

MOPs are networked together by links. Links may take various forms, such as specialization, abstraction, or part-of links. These will be discussed further in the following subsection. Other types of links used in the system are index links and failure links. Index links are used to identify particular classes of specializations. Failure links categorize methods for filling slots in a MOP that have failed to produce adequate solutions to previously encountered problems. MOPs that have been categorized in this way have new slots created attached to them by the system that indicate how that failure was overcome, or that warn the system to avoid making the same mistake in future situations.

Links are implemented the same as slots of the MOPs, but their semantics are different. In the CBR implementation of the Milling Assistant, particular slots are distinguished by the processing code to act as indices. These distinguished slots are used during the construction of abstraction and packaging hierarchies and for indexing sibling subclasses of a MOP.

MOPs are defined by the user through the DefMop function of the Milling Assistant. DefMop is an abbreviation for “Define MOP”. The syntax for the function is shown in Text Box 6, along with a few examples of concepts defined using it. In the current version of the Milling Assistant, the core knowledge base is defined in advance by the system programmer in order to provide a basic, default behavior for the system. This allows the system to be immediately useful to a machinist on the shop floor. In defining this core functionality, the system programmer creates a file of DefMop definitions and associated slot-filler methods.

The adaptability and creativity displayed by CBR systems stems from their ability to reorganize and restructure their case libraries (knowledge bases) based on the types of similarity found in the problems on which the system has worked, and also on the basis of user defined repairs to faulty knowledge. In some CBR systems, MOP classes that are not used to interpret incoming

```
Function defmop mopname
      {instance}
      slots
```

mopname: the name of the new mop.

instance: an optional flag that if present forces the new mop to be an instance mop.

slots: a list of slot-name, slot-value pairs that describe the new MOP.

Text Box 7: DefMop and examples.

problems over a long period of time may be erased, or “forgotten”, from the case library. The function that enables this the system to create and reorganize it's own knowledge base is called `slots->mop`. As is indicated by its name, the function takes a number of slots and creates a mop using them. However, it is not enough to simply create a new mop structure, but the mop must be placed in the correct position in the abstraction and packaging hierarchies used by the system to classify incoming problems.

```
Function slots->mop slots
      absts
      must-work
```

slots: a list of slot-name, slot-value pairs that describe the new MOP.

absts: a list of one or more abstract MOPs under which the new MOP may be placed.

must-work: a flag that, if true, causes an error to be raised if the new MOP cannot be placed under the given abstractions.

Figure 31: Syntax of `slots->mop` function.

The syntax of the `slots->mop` function is shown in Fig. 31. The function is given a number of slots with which to create a new MOP, and a specification of a MOP in the case library that the system thinks is a good place to start looking for the correct location of the newly created MOP.

The function inserts a new MOP, possibly created by the system itself, into memory under the MOP(s) given in the *absts* argument. This function is the basis of learning and reorganization in the case-based reasoning system.

This function is the basis of learning and reorganization in the case-based reasoning system. Fig. 32 shows the pseudo-code for the method used for placing the new MOP in memory. The heart of this function lies in the decision to call one of two other functions based on the type of the new MOP, either *install-instance* or *install-abstraction*. In the former function, the abstraction given is used as a starting point to find the most specific MOP under which the new MOP can be indexed (see Fig. 33, no. 1). If the given abstraction is an abstraction of the new MOP, then its children are then inspected to see if they too are abstractions of the new MOP (Fig. 33, no. 2). When the most specific MOP is found (Fig. 33, no. 3), its children are searched for MOPs that are equivalent to the new MOP (Fig. 33, no. 4). If an identical MOP is found, then the new MOP is made to point to it. If an identical MOP is not found, then the new MOP is indexed under the most specific MOP found.

```

function slots->mop (slots absts must-work)
  if null(slots) and (length absts) = 1 then
    return (absts)
  else
    create a new MOP with description = slots
    if new MOP is an instance
      result = install-instance ( MOP )
    else
      result = install-abstraction ( MOP )
    if must-work and null( result ) then
      signal error
    else
      return ( result )

```

Figure 32: Pseudo-code for the *slots->mop* function. The heart of the function is to decide whether to install the new MOP as an abstraction or as an instance.

In the install-abstraction function in Fig. 34, the new MOP is meant to be placed somewhere below one of the abstractions of the MOP given as the abstraction to the function. Each of the siblings of the abstraction given to the function are searched for MOPs that are equivalent to the new MOP. If one is found, the new instance is deleted, and a pointer is made to the existing MOP. If one is not found, then any instances of the new MOPs abstractions that are subsumed by the new MOP are placed under it.

Figure 33: Operation of the install-instance function. The function inserts the new MOP into the subtrees rooted at the MOPs given as the abstraction argument to the slots->mop function.

Abstraction and Packaging Hierarchies

Abstraction and packaging hierarchies play a major role in traditional CBR processing and in the CBR version of the Milling Assistant as well. The abstraction hierarchy can be viewed by following only the specialization links from the root node of the case library. This forms a tree that partitions the case library into a number of different classes that can be used to identify input problems. As an example of a step in the abstraction hierarchy, consider the previous example “milling a pocket”. There are numerous strategies for milling pockets, and some are based on the hardness of the material used for the part. Thus, specializations of the “milling a pocket” might include “milling a soft pocket”, “milling a medium pocket”, and “milling a hard pocket” (where soft, medium, and hard refer to the hardness of the material used to make the part).

Figure 34: Operation of install-abstraction. This function installs a new abstract MOP into memory, sublimating sibling MOPs if possible.

These may be further specialized by distinguishing on other features of the problem, such as the amount of side stock. Specializations of “milling a medium pocket” might include “milling a medium pocket with small side stock”, “milling a medium pocket with medium side stock”, and “milling a medium pocket with large side stock”. This portion of a CBR system's abstraction hierarchy is shown in Fig. 35. The classes of the abstraction hierarchy may share computational methods for computing slot-fillers for their required slots, or they may have separate methods for doing so.

Calculation Functions

The second way knowledge is programmed into the system besides MOPs is in the form of calculation functions. Calculation functions are used to specify a method of computing values using the slot-fillers of different slots of the MOP being computed. Fig. 36 shows an example calculation function for determining the amount of overlap of tool paths. The first expression defines the concept of the particular calculation function, calc-overlap, to the system. The next function is a LISP definition that does the actual computation of the amount of overlap needed. In this function, to supporting functions are used to access and inspect the values of a slot of the

Figure 35: Portion of the abstraction hierarchy.

MOP for which the overlap is being calculated. The first function, `get-filler`, is used to get the value of a slot in a MOP. In this case, the expression `(get-filler 'material mop)` retrieves the material hardness from the MOP describing the feature being milled. The value returned from this function will be an instance MOP that qualitatively describes the hardness of the material. The predicate `abstp` returns true if the first argument is an abstraction of the second argument. Thus, the conditional statement incorporating the `abstp` predicate evaluations bases its judgment solely on the hardness of the material - if the material is soft, then overlap is 20% else if the material is medium, then overlap is 40% else if the material is hard, then overlap is 60%.

The final portion of Fig. 36 shows how the calculation function is called from within a MOP. By defining the slot filler as a pattern MOP, the system is forced to evaluate the new MOP against the patterns given. In this case, there is only one pattern - a `calc-fn`. The `calc-fn`'s are evaluated using the name given to the `calc-fn`, then finding a LISP definition with the same name. If this is found, that function is applied to an argument list consisting of the `calc-fn` pattern and the new MOP itself. The result of the LISP function applied to these arguments is then stored as the value of the slot.

Learning from Mistakes

A key difference between the traditional rule-based reasoning and the case-based reasoning versions of the Milling Assistant is the ability to learn from its mistakes. In the context of the Milling Assistant, a mistake is a value for a slot in the `Mill-a-Feature MOP` that is either incorrect, or that the user does not agree with. Recall that the Milling Assistant is initially

programmed with a set of default knowledge that is capable of producing adequate results in most circumstances. However, this knowledge cannot represent the specialized knowledge of the end-users, or reflect a user's individual preferences for milling operations. Thus, the ability to learn from mistakes enables the Milling Assistant to tune itself to the preferences and expertise of its users.

Learning in the CBR paradigm is slightly different than other learning methods within AI. This is due mainly to the CBR philosophy of reusing the results of past experiences in the construction of a solution to a new problem. Thus, the CBR processing methodology is to “anticipate and avoid” mistakes rather than to “create and debug” them. Anticipating mistakes means that the conditions of a problem that lead to a mistake being made must be identified and highlighted in the case library so that future problems will recognize these conditions. Once the conditions for a potential mistake are identified, they may then be avoided by attaching failure MOPs to the slot in question so that the information the slot suggests may be avoided. Since the failure MOP will also contain information about how the mistake was overcome, that information may be applied directly when the conditions for the potential mistake are identified.

The philosophy of “anticipate and avoid” in the CBR paradigm also highlights another aspect of CBR that the Milling Assistant employs in its learning mechanisms. This is the notion of repairing not only the faulty plan for milling a feature, but to repair the case library that produced that faulty plan as well. This is another distinguishing characteristic of the CBR paradigm - that it is memory based. The term “memory-based” refers to source of the program's knowledge - in CBR systems, that knowledge is almost entirely embodied in declarative statements in the case library. This differs from other programs where much of the knowledge is locked in procedural representations that cannot be inspected by the program itself. The benefit of developing memory-based programs is exactly that the knowledge used to produce solutions can be inspected by the program and modified to improve its performance. Thus, in memory-based CBR systems the knowledge used to produce an unsatisfactory solution can be modified so that in the same, or similar, circumstances the unsatisfactory solution will not be produced again.

Learning through user interaction

The approach to learning from mistakes that is taken in the Milling Assistant project is to allow the user to define what are called “adaptation rules”. Adaptation rules define a method for modifying the unsatisfactory solution by specifying which aspects of the problem are making the solution unacceptable, and what steps must be taken to overcome them. Many CBR systems

Figure 36: Portion of Case Library before adaptation.

have a general set of adaptation rules that can be applied autonomously by the system to modify solutions that it finds unsatisfactory. For example, a general adaptation rule from a planning domain may be:

“If the preconditions for a step in the retrieved plan are not satisfied in the problem, then substitute an action with the same effect as the step considered, but whose preconditions are valid in the problem.”

There are a large number of such general purpose adaptation rules that can be implemented for a general purpose planning domain. As was mentioned before, there are a number of reasons why the Milling Assistant cannot rely on general adaptation rules to correct unsatisfactory

problem solutions. First, many of the types of repairs that will be made to the solutions produced by the Milling Assistant will reflect the individual tastes and programming styles of the job-shop NC programmers and machinists. The Milling Assistant was designed with this in mind, so that repairs such as using a “favorite” tool, or always drilling an insertion hole for an end mill entry method can easily be specified and acted on.

A second reason why general adaptation rules cannot be used is the fundamental lack of knowledge about the “science” of milling and manufacturing. There is no fundamental, scientific reason why one tool should be preferred over another if they both satisfy the constraints of the feature being milled. Because of this, there are no milling or manufacturing “laws” that can be counted on to evaluate autonomous corrections that have not previously been accepted by the user.

Finally, the level at which the Milling Assistant works prevents some general adaptation rules from being applicable. The Milling Assistant was designed to produce milling plans for one user-defined feature at a time. Thus, the information that the Milling Assistant gets from the Anvil-5000 CAD/CAM system relates entirely to the feature in question. There is no information presented to the Milling Assistant that is variable or that is an interpretation of some lower-level data: the data is low-level, hard facts about the feature being milled. Thus, the types of general planning adaptations that might be applicable if the Milling Assistant were scheduling a series of features to be milled (such as adaptations to make the sequence of features milled more optimal with respect to tool changes) cannot be applied.

Focusing the adaptation of faulty solutions on user defined corrections has several advantages that make it ideal for the Milling Assistant domain. Besides the above mentioned difficulty with general adaptation rules, the principal advantage of relying on user-defined adaptation rules is that they are tied to the specific types of features involved in the faulty solution. This makes the problems of maintaining case library consistency and over-generalization much more manageable.

Case library consistency is a problem that must be addressed whenever the program has the responsibility for propagating the changes specified by the adaptation rules throughout the case library. A significant problem when dealing with general adaptation rules is constraining the extent of the changes made by introducing the adaptation into the case library so that all previous cases will not need to be re-examined to place the adaptation into the abstraction and packaging hierarchies. By focusing user-defined adaptations on specific features involved in the faulty solution, the consistency problems are significantly reduced. This is because the user-defined adaptation rules are more specific to a particular slot in the faulty solution, and therefore only the consistency of that slot must be managed throughout the part of the abstraction and packaging hierarchies below the general MOP representing the feature being milled (e.g. Mill-A-Pocket).

The same aspects of the user-defined adaptation rules that reduce the amount of consistency checking also reduce the risk of over-generalizing the repair. Many general adaptation rules are over-generalized when they are placed into the abstraction and packaging hierarchies. This means that they are placed at a level where they will unnecessarily encompass too much of the

case library. This is a major problem for a domain such as the Milling Assistant, since so many of the repairs made will be repairs based on user preference. The user-defined adaptation rules do not suffer this problem, since they are tied so closely to the specific solution that is being adapted. This will be demonstrated through an example.

Example

Figure 36 shows a portion of the abstraction hierarchy for the Milling Assistant case library. The user has run the Milling Assistant on a notch that has been classified under the M-Mill-Medium-Notch as I-M-Mill-Feature.11. The solution of the milling problem as defined by I-M-Mill-Feature.11 has inherited a value for the overlap slot of 35 from the M-Mill-Part-Notch-Pocket MOP (shown in the central box in Fig. 36). For the purposes of this example, we assume that the user has decided that since the material is medium that a larger overlap is justified.

The user brings up the user interface for specifying rule changes by typing “(adapt)” in the LISP window (eventually, this will be made part of the overall CAD/CAM interface as a button). The user interface shown in Fig. 37 is brought up on the screen. This user interface shows the user all of the slots defined in the MOP representing the current problem. The lower portion of the screen has several lines where an adaptation rule can be specified. The condition lines (labeled `CONDITION_1` through `CONDITION_5`) are where the user specifies what aspects of the problem are responsible for the needed adaptation. In this case, the user types “(material m-medium-material)”. In other words, the adaptation is needed because the material is of medium hardness. The action line allows the user to type in a slot name and an equation for computing the value of the slot in a manner that will be acceptable to the user. In this case, the user specifies a simple equation, that the overlap should be set to 80 percent. Thus, the complete adaptation rule for this example is “If the material is medium, and the problem is similar to I-M-Mill-Feature.11, then the overlap should be 80 percent”.

The second condition to the rule stated above was not explicitly mentioned by the user in the user interface. However, the mechanism used to integrate this adaptation into memory is predicated on applying the adaptation to the most specific MOPs that subsume the faulty solution. In this example, the most specific MOP that subsumes I-M-Mill-Feature.11 is the M-Mill-Medium-Notch MOP. The adaptation mechanism treats the repair as a special case of this MOP and creates a new MOP subordinate to it. This MOP is shown in Fig. 38 and is called

M-Mill-Medium-Notch.5.

Adding new knowledge directly

The second method of adding new knowledge to the Milling Assistant is by directly programming the knowledge into the case library. Although this approach bypasses the benefits of uniformity and self-organization, it is necessary to provide the initial structure needed to solve problems quickly and efficiently. Because there is currently no generalization capabilities in the

Feature: I-M-PART-BOUNDARY			
SIZE	0.0	CLEARANCE-PLANE*	2.0
PROFILE-APPROACH-X	-0.38992602	SURFACE-FINISH-SIDE*	125
PROFILE-APPROACH-Y	3.8209674	SURFACE-FINISH-BOTTOM*	125
MAX-ROUGH-BASE-CUT	0.75	SIDE-TOLERANCE*	0.005
BASE-STOCK	0.5	BOTTOM-TOLERANCE*	0.005
PLAN	I-M-PLAN.4	BOTTOM-FILLET*	0.0
EXTEND-DISTANCE	0	CLEAN-UP-STOCK*	0.01
ROUGH-SIDE-STOCK	0.48	MAX-TOOL-DIA*	10.0
SIDE-THICK	0.01	FEATURE-WIDTH*	0.0
MAX-ROUGH-SIDE-CUT	0.3	START-X*	0.25
FINISH-SIDE-STOCK	0.01	START-Y*	3.6495
FINISH-SIDE-THICK	0.0	VEC-I*	-0.965926
MAX-FINISH-BASE-CUT	2.25	VEC-J*	0.258819
MAX-FINISH-SIDE-CUT	0.075	PERIMETER-LENGTH*	26.82796
BOUNDARY*	I-M-CLOSED-BOUNDARY	ROUGH-TOOL-NO*	5
SIDE-STOCK*	I-M-SIDE-STOCK.10	ROUGHING-TOOL-DIA*	0.75
MATERIAL*	I-M-MATERIAL.99	FINISH-TOOL-NO*	5
MILL-SPINDLE*	I-M-HORIZ-MILL-SPINDLE	FINISHING-TOOL-DIA*	0.75
MATERIAL-NAME*	STAINLESS STEEL	DX*	9.25
MACHINE-TOOL*	C/M T-10	DY*	5.75
FIXTURE-RIGIDITY*	5	SMALLEST-FILLET*	0.375
PART-QUANTITY*	1	XMIN*	-0.125
TOP-OF-STOCK*	0.0	YMIN*	0.125
FINAL-CUT-DEPTH*	-0.5		

CONDITION_1:	(material = m-hard-material)	Accept Quit Help PostScript
CONDITION_2:	(fixture-rigidity < 7)	
CONDITION_3:		
CONDITION_4:		
CONDITION_5:		
ACTION:	(max-rough-side-cut = roughing-tool-dia * 0.1)	

Figure 37: User interface for specifying adaptation rules.

Milling Assistant, only refinements of existing knowledge can be made through the user-defined adaptation rules. Knowledge that requires the introduction of new MOPs or slots with calculation functions must have those MOPs defined in the case library by direct programming.

There are three steps taken when directly programming knowledge in the Milling Assistant. The first step is to find the correct MOP in the case library in which the new slot is to be added, or subordinate the new MOP being defined. The second step is to add the slot or MOP into the text file that composes the initial case library. The final step is to write the calculation functions or provide linkages that will enable the system to act on its new knowledge.

In finding the appropriate MOP under which to insert a new MOP entails much the same procedures as the program uses to do this autonomously. The principal constraint under which this must be done is maintaining the consistency of the abstraction and packaging hierarchies.

Figure 38: Modified Case Library showing movement of the overlap slot.

The Milling Assistant, to a large extent, assumes the integrity of the initial case library loaded.

Thus, if the user inserts a MOP in a position that is not subsumed by its parent, the initial case library will not load. There are virtually no constraints on adding new slots to a MOP. If a new slot is added to a relatively abstract MOP, such as M-Mill-Feature, then that slot will be inherited by large portions of the case library. If the new slot is added to a more specific MOP, then it will have more local effects. The only decision that needs to be made when adding a new slot is the extent of the case library that needs to inherit the new slot. This decision effects at what level MOP the new slot should be added.

The only other constraint is that MOPs must be defined before they are referenced. This simplifies the loading process and adds only a small inconvenience to the user. There are currently no consistency checking methods other than for the integrity of the abstraction and packaging hierarchies. Future versions of the Milling Assistant will need more consistency and error checking to be implemented to relieve the user of any need for maintaining the consistency of the knowledge base himself.

Example

One aspect of error checking that was not implemented in the Milling Assistant at first was checking for constraint violations of the milling process itself. For example, if the user specifies a surface finish of eight for a milled feature, the milling process cannot achieve that surface finish, and a warning to the user should be issued. Another example of a process error to monitor is drilling a 1/16" diameter hole 5 inches deep. Because the hole is so small, it is doubtful that the drill bit will have the rigidity necessary to maintain accuracy on such a deep hole. This too should be brought to the attention of the user through a warning message.

To implement the error checking feature, it was decided that a new slot called Warning should be added into the system whenever it computes a plan to mill a feature. Because errors such as these should be computed for each feature milled, the appropriate MOP in the case library into which the new slot should be added is the most specific slot common to all milled features, which is the M-Mill-Feature MOP, shown in Text Box 8.

```
(defmop m-mill-feature (m-case));;  
;; the m-mill-feature mop is descended from m-case  
;; m-case is the mop that supports case-base adaptation.  
;; the definition of m-case is given at the top of this file as:  
;;  
;; (defmop m-case (m-root)(olf m-pattern (calc-fn get-sibling)))
```

Text Box 8: The M-Mill-Feature MOP.

To add the new slot to the MOP, a text editor was used to edit the text of the case library. The M-Mill-Feature MOP was located in the text and the slot “(warning m-pattern (calc-fn calc-warning))” was added in the body of the MOP. The exact position of the slot within the body of the MOP is immaterial. This slot instructs the system that to determine what warnings may apply to the current feature, execute the calc-warnings calculation function. This was the next section to add to the case library text.

To add the calculation function, first the MOP defining the calculation function was specified as

is shown at the top of Text Box 9. Immediately following that MOP definition was entered the LISP function that is called to generate the warning messages. This function simply accesses relevant parts of the M-Mill-Feature MOP and checks them against background knowledge entered by the user into the function. The calculation function shown in the lower portion of Text Box 9 shows only the surface finish constraint warning message mentioned above. When this constraint is violated, a warning is issued that appears on the ANVIL-5000 screen before the tool paths are displayed.

```
(defmop calc-check-finish (m-function))

(defun calc-check-finish (pattern mop)
  (declare (ignore pattern))
  (let ((msgs nil))
    (cond ((< (get-filler 'surface-finish-side mop) 16)
           (push "WARNING Less then 16" msgs))
          (t
           (push "WARNING GE 16 SURFACE FINISH" msgs)
           (push "WARNING No other warnings given" msgs)))
    msgs)
  )
```

Text Box 9: Calculation Functions Added Directly by User.

To inspect the results of this change immediately without restarting the Milling Assistant, the user can type "(reload-kb)" into the LISP window and the new knowledge will be loaded. This promote interactive development and acquisition of knowledge for the system.

Where to from here?

Asking for help when siblings are adapted

It is a weakness of the system that the adaptation of one particular aspect of a MOP is not carried over to that MOPs siblings. This is impossible to do in the milling domain for the general case, but it would be possible to have the system ask whether a change should be made in the computation of a particular slot, when it is known that the slot was modified in the computation of a siblings MOP. For example, if the computation of `overlap` is modified for the `mill-a-hard-notch` MOP, then the system should ask if the computation of `overlap` should be modified for the `mill-a-hard-notch` case, too. The asking could be controlled to include only direct siblings of a modified MOP, as in the example where only siblings of `mill-a-notch` would cause the querying to take place.

A second option would be to create an explicit similarity hierarchy within the case library that identifies for each slot type the slots that should cause the querying behavior. This would be a much more extensive modification to the existing system, but ultimately would allow the system to exhibit much more flexible behavior. This is because changes in overlap for notches could be reflected in the behavior for pockets as well, by causing the system to query if modifications should be made. An even more complex option would be to examine the modification made, and use it to index the situations under which it should be applied again. Thus, if the modification made to the overlap slot mentioned only feature depth and material hardness, any other feature with the same feature depth and material hardness would have the modification automatically applied.

Plan-oriented vs. feature-oriented

The current system is feature-oriented, using feature in the CBR sense, meaning the terms used to describe the cases. One of the final products of classifying the case is to create an explicit milling plan that includes methods for entering, removing, maneuvering, and exiting the feature. A new hierarchy could be build around the milling plan that would allow for more cross-feature interaction. Indeed, the main benefit of this type of reorganization would be that the case library would more likely resemble the knowledge organization of the experienced machinist, and therefore, would be easier to use and modify by the user.

Knowledge base combination

A key problem associated with any AI program is the acquisition of knowledge. The Milling Assistant acquires knowledge through interaction with the user and through the processing of milling problems. By putting a number of Milling Assistants (one to a user) on the job shop floor, some portion of the cumulative expertise of the job shop can be captured in the case library and in the repairs suggested by the users.

By combining the case libraries of all Milling Assistants in use, the rate of knowledge acquisition - and therefore, usefulness - of the individual Milling Assistants will be significantly increased. One simple way to combine the knowledge bases would be to store a time tag along with each case presented to the individual Milling Assistants, then present all the cases in order of these time tags to a "clean" version of the Milling Assistant, which would ingest each case in turn. After all cases are presented, the final case library will represent the cumulative experiences of all the other Milling Assistant programs. This knowledge base can then be passed out to the individual Milling Assistant programs, overwriting their private case libraries. Thus, the entire user community can benefit from the experiences of their fellow users.

However, combining the case libraries is a non-trivial problem in several respects. In many cases, the decision of a user to enter a repair rule to correct the default rules present in the system is a matter of personal preference. For example, one user may prefer to take more of an overlap when milling soft materials than another more conservative user. When two users both provide a repair based on their own personal preferences, a conflict arises as to which repair

should be suggested when the system is presented with a new case. This could be solved at the time of the case library integration by asking the system administrator to choose which repair should be used, or at runtime by having the user select from a list of possible repairs to apply to the particular situation.

Interaction with user

A key area of research that must be investigated is the dynamics of the interaction with the user. We have found in the past that many machinists are not eager to engage in drawn out knowledge acquisition sessions with either the computer or the knowledge engineer. Many machinists prefer not to give much, if any, control over to the computer to allow it to make milling decisions for them. We believe that this is due in large part to the unnatural way in which previous systems have forced users to interact with their systems. There are a vast number of possibilities to investigate when interacting with the user, and here we mention two of them: "looking over the shoulder" learning, and explanation.

Human apprentice machinists spend a large portion of their time watching the actions of the experienced machinists, then duplicating those actions in similar situations. Many times the experienced machinist will not take the time to instruct the apprentice on why a particular action was made, but will instead rely on the apprentice's ability to reason about the action to understand why it was done. This type of behavior is what is meant by "looking over the shoulder" learning. This is learning by observing the actions of an experienced user, and relating that information to what the system already knows about the milling domain.

In order for the Milling Assistant to approach the "looking over the shoulder" learning behavior, several modifications will need to be made. When the Milling Assistant is processing a milling problem, it knows the objectives to be accomplished, or relies on the user to enter a repair rule in order to correct faulty inferences. In order for the Milling Assistant to use the actions of the experienced machinist as a solution to a milling problem, it must first interpret the goal behind the observed actions, infer relations between actions, and keep track of how the actions affect the current problem solving episode. This will involve the introduction of more detailed knowledge about how milling plans succeed and fail, and new mechanisms for storing cases as groups of small "caselets" instead of one monolithic structure.

The second area to explore that will increase the usefulness of the Milling Assistant is an explanation facility that reflects the inferential decisions made by the system in a natural, easy-to-understand fashion. This capability is necessary for the system to be able to gain the confidence of the user, and for the user to better understand the reasoning being performed by the Milling Assistant without understanding the reasoning mechanisms explicitly.

The key to providing this facility in the Milling Assistant lies in explaining decisions based on the decisions made in previous cases. So the decision to enter the feature in a particular way could be explained by referring to a similar previous case where the user instructed the system to enter the feature in that way. This facility could be augmented by presenting the CAD

drawing and tool paths of the instructional case and highlighting the specific decision being explained. An additional explanation capability that would be useful is to explain the decisions made at a very abstract level, where the explanation focuses on the milling plans thematic or goal structure, while hiding the lower level details of the plan.

Beyond 2-1/2 axis milling

The current version of the Milling Assistant is limited to 2-1/2 axis milling. This is not due to any constraints of the reasoning or interfacing systems, but is a design choice made to expedite the development and delivery of the system. It is only the knowledge in the system that limits it to 2-1/2 axis milling. The addition of new knowledge would allow the Milling Assistant to perform 3,4, or 5 axis milling in the same manner as the 2-1/2 axis milling is currently handled. It is a goal for the future versions of the Milling Assistant to be able to provide capabilities in all types of milling. This capability could be especially useful in milling complex surfaces, which take a comparatively long amount of time to produce milling plans. However, the basic principles of the Milling Assistant, as well as its entire structure, could also be applied to more specific areas such as turning, bending, or forging. Like the modifications for 3 or more axis milling, the changes required to apply the Milling Assistant to these areas simply lie in acquiring the knowledge necessary to operate in those domains. The entire structure of the Milling Assistant could be brought to bear on these problems as is, leaving all the development time for knowledge acquisition and validation.

Knowledge Enhancement Examples

General Methods

Allowing domain experts to modify existing strategies and develop new knowledge are very important aspects of AI applications in manufacturing. The MA CBR decision processor interacts with users to review and modify proposed strategies for machining part features. In cases where methods suggested by CBR are rejected, immediate feedback to the system can result in better initial solutions for similar operations in the future. The CBR system can be developed to perform best on families of part features and with the operator/machine combination targeted for the job.

There are two techniques available for modifying CBR rules and calculation functions utilized in MA. The first method involves editing the LISP code and the second requires the user to run an interface tool (MA-Learn) so that existing knowledge can be changed or enhanced. New cases, or the addition of parameters to existing cases, requires the introduction of new variables and supporting LISP statements. If the rules exist but need to be "tuned", the MA-Learn utility is very useful. The tool provides a method for the expert to develop a knowledge rich system through a process of reviewing and correcting the strategies proposed by the CBR.

The MA user interface utility (MA-Learn) allows an operator to inspect feature conditions and machining parameter settings chosen by CBR. Problems encountered can be addressed by a set of new conditions and an action specified by the user. The input information, in turn, may be used to re-program the current feature under consideration. If changes are deemed appropriate, the added strategy is optionally made part of the core knowledge base. The CBR exhibits a learning capability by storing and utilizing the adaptation rules input by the user.

The difference between the two knowledge enhancement methods is illustrated in Figure 39 (following page). The left side diagrams the operation of the MA-Learn utility and the right side is a more abstract knowledge development process which requires the creation of new LISP variables and structures. Examples are given on adding LISP code for a new parameter in drilling and enhancing an existing strategy for calculating cut widths with the MA-Learn interface.

Example: Knowledge for a Drill "Pecking" Cycle

Deep drilling cycles involve the calculation of a depth of cut increment based on the tool, machine and material used in the operation. Materials within specified ranges of hardness require a different incremental depth, called "pecking", so that chips can be cleared and tool wear minimized. The current version of MA uses a default increment. The new knowledge utilizes the following categories for calculating the pecking increment:

Figure 39: Alternative Methods for Modifying CBR Knowledge

Soft material ($0 < BH < 100$)
 $\text{peck} = 0.50 * \text{Drill Dia.}$

Medium material ($100 < BH < 200$)
 $\text{peck} = 0.33 * \text{Drill Dia.}$

Hard material ($200 < BH < \text{MAX}$)
 $\text{peck} = 0.25 * \text{Drill Dia.}$

Figure 40: Various Pecking Depths Based on Material's Brinell Hardness

The LISP structures (mops) were developed by modifying existing code used to calculate cut depths for milling operations. Creating new code from scratch would be a very difficult task

without an extensive background in AI programming and LISP in particular. This factor limits the users ability to add desired capabilities to the system. The mop and calculation function along with the calling statement are presented in Text Box 10.

New LISP code to calculate the pecking increment
 for a DEEP drilling cycle.

```
(defmop calc-peck-drill-amt (m-function))
(defun calc-peck-drill-amt (pattern mop)
  (declare (ignore pattern))

  (let* ((mat (get-filler 'material mop))

         (val (cond ((abstp 'm-soft-material mat)
                     .5)
                    ((abstp 'm-medium-material mat)
                     .33)
                    ((abstp 'm-hard-material mat)
                     .25))))

    )

  (print mat) (print val)
```

Text Box 10: CBR LISP Code for Implementing Pecking Cycle

Example: Adjusting Cut Amounts Based on Material Hardness and Fixture Rigidity

The amount of cut taken on each pass of a milling tool path is dependent on the working conditions for a particular job. Fixtures, tools, machines, and material type have a major influence on the selection of appropriate cutting parameters. Cut depth and width amounts are influenced most by the diameter of the cutting tool being used for the feature. For this reason, side and base cut increments are most often calculated as a function of the tool diameter. An extreme case occurs when the material of the workpiece is hard (ie BH > 200) and the fixturing orientation is weak. In this case, a very light cut would be taken to reduce the forces acting on the tooling and setup. For this reason, a set of rules were devised to address this situation.

This type of upgrade is best demonstrated by processing a feature using the current knowledge, changing the rules used to calculate the cut values, and finally, by re-processing the same feature to see the effects of rule changes. Once values are verified, the new rules can be made part of normal operations. A part boundary and a notch are processed here to show the effects of enhancements described above. Dotted lines represent the center line of the tool as it completes it's motions. The after pictures show more passes (less material removed per pass) due to the knowledge change for the harder material and lighter fixturing.

Figure 41: Boundary Tool Path Before Change.

Figure 42: Boundary Tool Path After Change

Figure 43: Notch Tool Path Before Change

Figure 44: Notch Tool Path After Change

Figure 45 is a postscript file of the MA-Learn window for the part boundary case. Feature data passed from ANVIL-5000 is specified with an asterisk while the remaining parameters in the upper left slots are determined by the CBR. One slot specifies the amount of side cut taken per roughing pass (max-rough-side-cut). This value would commonly be varied by machinists based on the material hardness and the fixturing rigidity. The conditions and actions specified in the MA-Learn window reflect such a situation. When the accept button is pressed, a new actions file is written for the current feature and a map is placed in a file for inclusion into the core knowledge base for similar scenarios in the future.

Feature: I-M-PART-BOUNDARY			
SIZE	0.0	CLEARANCE-PLANE*	2.0
PROFILE-APPROACH-X	-0.38992602	SURFACE-FINISH-SIDE*	125
PROFILE-APPROACH-Y	3.8209674	SURFACE-FINISH-BOTTOM*	125
MAX-ROUGH-BASE-CUT	0.75	SIDE-TOLERANCE*	0.005
BASE-STOCK	0.5	BOTTOM-TOLERANCE*	0.005
PLAN	I-M-PLAN.4	BOTTOM-FILLET*	0.0
EXTEND-DISTANCE	0	CLEAN-UP-STOCK*	0.01
ROUGH-SIDE-STOCK	0.48	MAX-TOOL-DIA*	10.0
SIDE-THICK	0.01	FEATURE-WIDTH*	0.0
MAX-ROUGH-SIDE-CUT	0.3	START-X*	0.25
FINISH-SIDE-STOCK	0.01	START-Y*	3.6495
FINISH-SIDE-THICK	0.0	VEC-I*	-0.965926
MAX-FINISH-BASE-CUT	2.25	VEC-J*	0.258819
MAX-FINISH-SIDE-CUT	0.075	PERIMETER-LENGTH*	26.82796
BOUNDARY*	I-M-CLOSED-BOUNDARY	ROUGH-TOOL-NO*	5
SIDE-STOCK*	I-M-SIDE-STOCK.10	ROUGHING-TOOL-DIA*	0.75
MATERIAL*	I-M-MATERIAL.99	FINISH-TOOL-NO*	5
MILL-SPINDLE*	I-M-HORIZ-MILL-SPINDLE	FINISHING-TOOL-DIA*	0.75
MATERIAL-NAME*	STAINLESS STEEL	DX*	9.25
MACHINE-TOOL*	C/M T-10	DY*	5.75
FIXTURE-RIGIDITY*	5	SMALLEST-FILLET*	0.375
PART-QUANTITY*	1	XMIN*	-0.125
TOP-OF-STOCK*	0.0	YMIN*	0.125
FINAL-CUT-DEPTH*	-0.5		

CONDITION_1:	(material = m-hard-material)	<input type="button" value="Accept"/> <input type="button" value="Quit"/> <input type="button" value="Help"/> <input type="button" value="PostScript"/>
CONDITION_2:	((fixture-rigidity < 7))	
CONDITION_3:		
CONDITION_4:		
CONDITION_5:		
ACTION:	(max-rough-side-cut = roughing-tool-dia * 0.1)	

Figure 45: MA-Learn Window for Part Boundary Case
(User Specified Cut Width for Light Fixturing and Hard Material)

Interface to Case-Based Reasoning Knowledge

Milling Assistant's User Interface Utilities

The Milling Assistant employs two utilities in its interactions with users. Both of these utilities are designed to teach; one of them teaches the user, and the other teaches the Milling Assistant itself. The purpose of this document is to provide a brief description of these utilities, and then to discuss their implementation in some detail. For practical information about the use of each of these utilities, see the Milling Assistant Operating Guide.

The Milling Assistant user interface is comprised of two utilities: MA-Garnet, and MA-Learn. The first of these, MA-Garnet, displays an interactive graph of the Milling Assistant's hierarchy of abstractions. Mathematically, the graph is really a tree, with a single node serving as the root at the highest level of the hierarchy. Progressively lower levels of abstraction are represented by nodes further away from the root. The lowest levels of abstraction are represented by the leaf nodes of the tree. Each node bears the name of the abstraction it represents.

If an element at some arbitrary position in the abstraction hierarchy has a node below it, then that node is called the parent of the lower node, which is in turn called the child of that parent. When MA-Garnet is first started, the tree is drawn with only the root and the first level of the abstraction hierarchy (the root's immediate children" visible. To view greater portions of the hierarchy, the user may use the mouse button to click on any node whose label appears in bold-face type. Bold-face labels indicate that the node has children, and clicking on such a node "expands" the node, adding nodes representing each of the node's children to the graph and re-drawing it. If any of the new children have children of their own, their labels will indicate that fact and they can be expanded themselves in the same manner. Clearly, MA-Garnet allows simple, graphical exploration of the Milling Assistant's abstraction hierarchy. Such exploration can greatly enhance the user's understanding of the Milling Assistant's decision-making process. Figure 46 diagrams a representative section of the abstraction hierarchy related to milled features.

The second of the Milling Assistant's user interface utilities, called MA-Learn, was created to facilitate the addition of new knowledge to the CBR knowledge base. In order to describe the use of MA-Learn, we first need to delve into the manner in which the Milling Assistant works.

The Milling Assistant considers each feature to be milled, say a notch, an individual case. Each case has associated with it a number of slots, each of which represents a particular property of the case. For example, a notch will have slots for depth of cut and material hardness among others. Given a particular case to mill, the Milling Assistant scans its knowledge base to determine whether it's milled such a feature before. This process is essentially a series of comparisons between the slots of the case to be milled and the slots of all the previous cases processed by the system. If an exact match is discovered, then the Milling Assistant simply produces the same output for the case that it did when it milled the case before. If no exact match is found, the Milling Assistant builds a new case, filling its various slots with information

Figure 46: Milling Assistant's Abstraction Hierarchy

from the most closely matching cases and returning appropriate output.

Occasionally, based on the user's expertise or personal preferences, the user may deem the Milling Assistant's output inappropriate. MA-Learn provides a simple, graphical means of modifying the Milling Assistant's knowledge base to reflect the wishes of the user. As an example, suppose the user would like to mill a part boundary. To remove a certain amount of side stock from the blank, the machine needs to make some number of passes around it. This is determined by the diameter and type of tool being used along with the part material and

machine tool to be utilized for the operation. If the material is very hard, and not rigidly held down by the fixture, it probably would be prudent to remove a smaller percentage of the tool's diameter on each pass. So instead of removing an amount of side stock equal to, say, 30% of the tool diameter, the user may want to remove only 10% of the diameter on each pass. Doing so would naturally increase the number of passes required to remove all the side stock necessary.

At this point, the user can invoke MA-Learn to make the required changes. When MA-Learn starts, it presents the user with a window containing the name of the feature being milled, followed by a list of all of the feature's slots together with their current values. At the bottom of the window is a rectangle containing six editable text boxes. Five of these are reserved for the input of conditions, and one is used for the action the Milling Assistant is to take when presented with the specified set of conditions. Under "Condition One," the user may type an expression which tells the Milling Assistant "the material is hard." Similarly, under "Condition Two" the user may type an expression which means "the part is not rigidly held by the fixture." Finally, the user may specify the action to take by filling the "Action" box with an expression which means "remove only an amount of side stock equal to 10% of the current tool diameter on each pass." Once the conditions and action have been specified to the user's satisfaction, all it takes is a click on a button to incorporate the new knowledge into the Milling Assistant's knowledge base. Thus, MA-Learn provides a powerful means of bringing the user's expertise to bear in the generation of appropriate output. Figure 47 shows the MA-Learn window with slots for a part boundary, conditions and action filled in for the above example.

The Garnet User Interface Development Environment

Central to the creation of the Milling Assistant's user interface utilities was a development environment called the Garnet User Interface Development Environment [10] (abbreviated to "Garnet," an acronym for "Generating an Amalgam of Real-time, Novel Editors and Toolkits"). Developed by the Garnet research project in the School of Computer Science at Carnegie Mellon University, Garnet provides a comprehensive set of tools that aid in the design and implementation of highly interactive, graphical, direct-manipulation user interfaces.

Garnet is written in Common Lisp for the X window manager, and is portable to virtually any Common Lisp environment (for technical reasons, it may be slightly easier to run Garnet under CMU Common Lisp as opposed to some other Common Lisp environment). Currently, Garnet supports X/11 R4 or R5 using the standard CLX interface. Garnet does not use CLOS (Common Lisp Object System) or any X toolkit (such as Xtk or CLIM). A project is currently underway in the Computing Research Laboratory at New Mexico State University to port Garnet to the Apple Macintosh using Macintosh Common Lisp.

Garnet is comprised of two layers. The "toolkit" layer provides programs with an object-oriented, constraint-based system which allows the properties of various graphical objects, such as windows, buttons, etc., to be specified in a simple, declarative manner. Once these objects have been specified, they are maintained automatically by the Garnet system.

Feature: I-M-PART-BOUNDARY			
SIZE	0.0	CLEARANCE-PLANE*	2.0
PROFILE-APPROACH-X	-0.38992602	SURFACE-FINISH-SIDE*	125
PROFILE-APPROACH-Y	3.8209674	SURFACE-FINISH-BOTTOM*	125
MAX-ROUGH-BASE-CUT	0.75	SIDE-TOLERANCE*	0.005
BASE-STOCK	0.5	BOTTOM-TOLERANCE*	0.005
PLAN	I-M-PLAN.4	BOTTOM-FILLET*	0.0
EXTEND-DISTANCE	0	CLEAN-UP-STOCK*	0.01
ROUGH-SIDE-STOCK	0.48	MAX-TOOL-DIA*	10.0
SIDE-THICK	0.01	FEATURE-WIDTH*	0.0
MAX-ROUGH-SIDE-CUT	0.3	START-X*	0.25
FINISH-SIDE-STOCK	0.01	START-Y*	3.6495
FINISH-SIDE-THICK	0.0	VEC-I*	-0.965926
MAX-FINISH-BASE-CUT	2.25	VEC-J*	0.258819
MAX-FINISH-SIDE-CUT	0.075	PERIMETER-LENGTH*	26.82796
BOUNDARY*	I-M-CLOSED-BOUNDARY	ROUGH-TOOL-NO*	5
SIDE-STOCK*	I-M-SIDE-STOCK.10	ROUGHING-TOOL-DIA*	0.75
MATERIAL*	I-M-MATERIAL.99	FINISH-TOOL-NO*	5
MILL-SPINDLE*	I-M-HORIZ-MILL-SPINDLE	FINISHING-TOOL-DIA*	0.75
MATERIAL-NAME*	STAINLESS STEEL	DX*	9.25
MACHINE-TOOL*	C/M T-10	DY*	5.75
FIXTURE-RIGIDITY*	5	SMALLEST-FILLET*	0.375
PART-QUANTITY*	1	XMIN*	-0.125
TOP-OF-STOCK*	0.0	YMIN*	0.125
FINAL-CUT-DEPTH*	-0.5		

CONDITION_1:	(material = m-hard-material)	<input type="button" value="Accept"/> <input type="button" value="Exit"/> <input type="button" value="Help"/> <input type="button" value="PostScript"/>
CONDITION_2:	(fixture-rigidity < 7)	
CONDITION_3:		
CONDITION_4:		
CONDITION_5:		
ACTION:	(max-rough-side-cut = roughing-tool-dia * 0.1)	

Figure 47: MA-Learn Window with Slot Values and User Interface Area.

The "interactor" layer allows the programmer to separately attach dynamic interactive behaviors to the objects specified in the toolkit layer. Additionally, this layer provides tools which allow programmers and non-programmers alike to create user interfaces by simply drawing pictures of what the interface should look like.

Since Garnet is being developed as a research project, it has several nice features which made it a particularly attractive development environment for the Milling Assistant project. One of these features is that, in Garnet, constraints are integrated with the object system, so any slot (instance variable) of any object can contain either a value or a formula. Whenever a value that is referenced in the formula changes, as might happen when a window is moved, for example, the formula is automatically re-evaluated.

In Garnet, the programmer specifies the handling of input from the user at a high level using

abstract "interactor" objects. Typical user actions, such as clicking a mouse button, are encapsulated into a few different types of interactors. A programmer need only provide a small number of parameters to get objects to respond to user behavior in sophisticated ways.

Two complete sets of "gadgets" are provided by the Garnet system. Among these gadgets are buttons, menus, and scrollable windows. Further, Garnet provides built-in support for laying out objects in rows and columns or in graphs or trees, making it very easy to create objects such as panels of buttons or display directory structures. Finally, Garnet will automatically produce Postscript code for any picture on the screen.

Use of Garnet in the Implementation of the Milling Assistant's User Interface Utilities

The Garnet User Interface Development Environment was used in the implementation of both MA-Garnet and MA-Learn. Each of these utilities is displayed in a window created using Garnet, and all of the graphical objects present in the windows were produced using Garnet.

MA-Garnet makes extensive use of Garnet's "aggregraph" aggregadget. An aggregadget is an object which provides the designer with a simple method for the definition and use of sets of Garnet objects and interactors. When an aggregraph or other type of aggregadget is supplied with a list of object definitions, the Garnet system internally creates instances of those objects and adds them to the aggregadget as components, actually grouping the objects and interactors into a single group.

Creation of the abstraction hierarchy graph described in Section 1 of this document minimally requires filling three aggregraph slots: children-function, which is a function that generates a list of child nodes from a parent node; info-function, a function which associates a string with a source node so that graph nodes can have labels; and source-roots, a list of the graph's root node(s). For MA-Garnet, the children-function is simply a function which provided information about what entities were present in the hierarchy below the entity given as a parameter. The info-function is a function that returns the name of the object in the hierarchy as a string. The source-roots slot is filled with the name of a root node, which can be either the root of the entire abstraction hierarchy (m-root) or any other node in the hierarchy. If m-root is given as the root node, the graph will represent the entire abstraction hierarchy; if any other node is given as the root node, the graph will represent that portion of the hierarchy beginning at that node and extending downward through the tree.

With the addition of interactors to the aggregraph, MA-Garnet's graph becomes a powerful exploratory tool. The particular interactor used by the graph handles mouse clicks on the nodes of the graph. The single interactor can distinguish between clicks with the left, middle, or right mouse buttons, with or without modifier keys like shift and control. Moreover, different functions can be assigned to each different type of mouse click. For example, clicking the left mouse button on a node that has children causes that node to expand, displaying its children in the graph. At the same time, clicking the right mouse button on the same node will cause the

node to contract again, and clicking the left mouse button on any node while holding the shift and control modifier keys down will cause a completely new graph to be drawn using the target node as the root. these disparate actions are implemented using the same interactor by merely specifying a few parameters.

MA-Learn uses several of Garnet's gadgets to perform its tasks. Most predominant among these are gadgets of the type scrolling-labeled-box. Scrolling-labeled-boxes are really a combination of labeled-box gadgets and scrolling-input-string gadgets. Labeled-boxes are comprised of a permanent (unchangeable by the user) label string which usually indicates something about the contents of an attached, dynamically-dimensioned editable text box. The size of the box changes with the size of the input text string.

Scrolling-input-strings allow the user to enter and edit a single line of text of arbitrary length. The string is automatically scrolled to the left or right as necessary, and so the gadget requires only some fixed amount of screen "real estate". The presence of an ellipsis at either end of the text box indicates the presence of text scrolled off that end.

The difference between scrolling-input-strings and scrolling-labeled-box is simply that the latter have permanent, descriptive labels attached to them, while the former do not. MA-Learn uses its scrolling-labeled-boxes to display a feature's slots together with their values. The values are displayed whether they are numerical values or formulas. While these values are fully editable, changes made to the slot values are not saved and have no effect on the actual values.

MA-Learn also uses scrolling-labeled-boxes to display and edit the conditions and action used to modify the Milling Assistant's knowledge base as described in Section 1 above. These scrolling-labeled-boxes serve as receptacles for the user's input text which describes the action to take based on the presence of certain conditions. Once the user has entered the appropriate text, another gadget, called a text-button is used to submit the changes to the Milling Assistant.

Garnet's buttons are simply graphical objects that respond to mouse clicks in user-defined ways. MA-Learn contains four buttons: "Accept", "Quit", "Help", and "Postscript" (there are actually five: the fifth button displays the name of the current feature at the top of the window and does not respond to mouse clicks). These buttons are grouped together into a button panel, which centralizes control of the button actions and relieves the user of having to align the buttons together on the screen. Clicking the left mouse button on the "Accept" text-button causes the Milling Assistant to respond by incorporating the user's new "rule" into its knowledge base. Internally, when the user clicks on the "Accept" button, the button panel associates the button with a function which is to be called whenever the button is clicked.

The "Postscript" button in MA-Garnet and MA-Learn calls a function when clicked which creates a file containing Postscript description of the current window's contents. The Postscript file can be downloaded to a Postscript-compatible printer to generate a hard copy. This is very useful for purposes of documentation and tutoring.

Merit of the Use of Garnet in the Development of Milling Assistant's User Interface

The use of the Garnet User Interface Development Environment in the implementation of the Milling Assistant's user interface utilities greatly simplified and expedited the development process. It is a comprehensive package, containing a great many pre-defined gadgets that may be incorporated into a user interface with very little effort.

The Garnet User Interface Development Environment works very well under Common Lisp, the Lisp environment used to develop the case-based reasoning portion of the Milling Assistant. Support for Garnet is excellent, and takes advantage of an Internet mailing list called `garne-users@cs.cmu.edu` created especially for that purpose. Brad A. Myers, principle investigator on the Garnet project at Carnegie Mellon University, mediates the list along with other Garnet researchers. To subscribe to the Garnet mailing list, send mail to `Brad_Myers@BAM.GARNET.CS.CMU.EDU` or call (412) 268-5158.

The Garnet system is fairly large, requiring upwards of nine megabytes of disk space. It requires a substantial amount of memory to use, and its performance can be rather slow because of that. However, not all of the Garnet code needs to be loaded; for example, the user will probably only use a few of Garnet's gadgets at a time in any one program, and so only the code for those gadgets needs to be loaded. Careful loading of Garnet code can save significant amounts of memory and time.

Some portions of Garnet don't work as well as they should. For example, we found Garnet's scrolling-windows to be unsatisfactory in a variety of ways (mostly having to do with inconsistent implementation of scrolling-window aggregates). Such instances are rare, however. Overall, we found Garnet to be a very good choice for development of our user interface utilities.

Garnet is readily available via anonymous ftp. It is free of charge, unrestricted, and requires no license. To retrieve Garnet via ftp, ftp to `a.gp.cs.cmu.edu` (128.2.242.7), log in as "anonymous" and use your name as the password. Change to the 'garnet' directory, and 'get' the files `src.tar.z`, `lib.tar.z`, and `doc.tar.z`. After "uncompressing" and "untarring" the files, further documentation will be available.

Operating Guide

This section provides a basic operating guide to running the Milling Assistant NC programming system. Instructions are given for executing ANVIL-5K (CADD/CAM/CAE) and for processing features by running the LISP programs which embody the Case Based Reasoning decision processor. The procedures described are to be used exclusively on the Sun SPARC station 2 computers in Bill Burd's office (ford: Jett Rm. 515) and the IMSL (madison: Jett Rm. 20) as well as the Tektronix X-terminals in the IMSL and adjacent Cad/Cam laboratory (Rm. 21). This guide can be used to document and improve MA operations in addition to serving as an introduction to future users.

Start-Up and Initialization of MA, ANVIL-5000, and CBR

Logging on to the UNIX environment can be accomplished with the first letter of the first name and the first seven characters of the last name along with a password if you are a member of the MA group. Otherwise the shared account can be accessed with the following login name and password.

IMSL (Sun and X-terminals)

Login: ma
Password: obtain from IMSL staff

Jett 515

Login: ma
Password: obtain from IMSL staff

Typing <anvil5kx> at the UNIX prompt (in either the session control or x-term windows) executes the ANVIL-5000 software as well as the initialization programs for the CBR. The window opened by this process to run CBR should be monitored at start-up to ensure that no errors in loading occur.

The CAD model of the part and preparation information must be inserted into the ANVIL databases before features can be processed. The first step is to recreate the part geometry using CAD options in ANVIL or convert it over from the original CAD system using XIP (XIP builds an ANVIL file from IGES or DXF data files). Milling Assistant uses a 2-D, to-scale drawing of a part as the basis for tool path decisions. Milling Assistant is built on top of CAM capabilities existing in ANVIL. It is accessed by executing a GRAPL interface program.

Tools available for feature milling must be pre-defined under CAM menu options or by attaching a tool library. Tools are picked based on the size of the particular feature and depths and input by the user. Milling Assistant chooses only tools that have already been defined.

Selecting the MA Icon from the set on the right hand side of the ANVIL screen runs the GRAPL interface program which initializes the MA. Default parameters needed by MA to determine effective cutting strategies are queried from the user at this point.

- * Stock dimensions
- * Default dimensional tolerance and surface finish characteristics
- * Fixture Rigidity Factor (Scale 1 - 10)
- * Part Quantity

A part feature can be selected from the new set of feature icons. The selected feature has associated parameters that must be input by the user. MA may also be put into a mode where the user makes manual tool selections rather than letting the system make the choice.

The CBR portion of MA can be operated manually or automatically. The first method processes each feature selected only while the constant polling mode allows features to be programmed without user intervention. A brief description of each is given below.

Case Based Reasoning: Two modes of operation for processing features

a) Feature by feature (rmdf): rmdf = read MA data file

- identify material, machine, rigidity factor, setup defaults
- select feature desired from icon set
- input values asked for by grapl menus
- verify that call is made by window where ANVIL was run from
- at * prompt type (rmdf)
- feature information is processed, call will be answered and ANVIL will echo parameters such as tool selected, RPM, and feedrate to the user
- change parameters if desired and O/C until "creating NC tool path" appears
- tool path can be changed using ANVIL modify utilities and output for verification or NC post processing

b) Constant polling mode (wfr): wfr = wait for request

In this mode of operation, the lisp window looks for a calling statement from the window where ANVIL was run. When it appears, lisp automatically responds by processing the feature and returns control to ANVIL. This mode is operated the same as part a) above except that steps 4 & 5 are automatic.

User Interface Utilities

Two utilities are available for working with CBR. The first utility is a lisp tool kit application which displays a diagram showing the structure of the Case-Based Reasoning (CBR) decision processor used by the Milling Assistant. The second utility is a user interface to the knowledge base that allows access to feature information as well as captures desired modifications to the core knowledge.

A) Display Abstraction Hierarchy

Description

This utility displays an abstraction hierarchy tree showing the structure of knowledge used in building tool paths for 2 1/2 axis milled and point-to point features. [See Figure 46] The graphical representation is made up of inter-linked nodes which form a tree. The root node is the "parent" of all others which may or may not have "children" of their own. A node in bold face type has children which can be viewed by opening up the branches of the hierarchy tree. A scrolling vertical and horizontal window, the ability to output encapsulated postscript files, and user control over visible nodes/branches at any one time contribute to making ma-garnet an effective presentation and learning tool.

Details

ma-garnet .lisp contains the code which customizes the display and user-interaction capabilities provided with the abstraction hierarchy. It can be found in the following directory:

/home/ford2/ma/cases/ma-extensions

Operation

- 1) Type "(do-go)" at the lisp "*" prompt. After a brief 20-25 second delay, the hierarchy is displayed on the screen.
- 2) The scrolling window is operated by clicking the arrows or dragging the view box (both with left mouse button).
- 3) Any node in bold-face type has "children" which can be displayed by clicking the left mouse button once on that node. A middle mouse button clicked on a node opens a gemacs file which displays the related mops.
- 4) Any expanded node can be closed by clicking that node with the right mouse button.
- 5) A <shift> <ctrl> left mouse button on any node makes it replace the root node in the display. It can be expanded if it is in bold-face type.
- 6) The help box (clicked with the left mouse button) opens a gemacs editor file which contains a limited amount of user information.

- 7) Clicking on the postscript box (left mouse button) outputs a screenG##.EPS file to the /home/ford2/ma/screens directory. The file can be printed directly to a postscript printer using the "lpr filename.EPS" command or imported to word-perfect for inclusion in a document. The word-perfect document must subsequently be sent to a postscript printer to obtain satisfactory results.
- 8) The quit box (left mouse button) ends the session and puts the ma-garnet window away. A "(stop-it)" in the lisp window accomplishes the same task.

Alternative Method

The abstraction hierarchy can be displayed in text form by typing the following at the "*" prompt in the lisp window:

"(dah 'm-root)"

The same can be done for any other node by replacing "m-root" above with the abstraction name.

Usage Notes

If, at any time during operations, a problem is encountered, it will most likely show up in the window where lisp is running. The debug prompt is a left bracket ([). The specific sequence of events leading up to the error and the text that precedes the debug prompt should be documented and communicated to someone in the "ma" group. A "q" should be typed at the debug prompt to allow a "(quit)" to end the lisp session.

B) Feature Information Display and Knowledge Modification Area

Description

The second utility provides a screen with both a table of feature attributes and a conditions/actions area where knowledge can be added or modified. The user can view the parameters input to CBR and the variables determined by CBR. The feature currently under consideration is named in the top middle box on the screen. The quit, postscript, and help boxes clicked-on with the left mouse button respond the same as described in part one.

[See Figure 47]

A) Current Feature Attributes:

- 1- Attributes passed into CBR by GRAPL-IV (marked with an *)
- 2- Attributes determined by strategies embedded in CBR

B) User Interface to Text Based Knowledge Repair Utility

Both types of data mentioned in A above can be used to set-up conditions under which some specified action will take place. The knowledge changes can later be used to modify the tool path currently being generated and/or to change the core knowledge that will be used for similar features in the future.

Details

The file `ma-learn.lisp` contains the code for displaying the slots of information related to the **new-instance** (ie. the feature last processed). The `ma-learn` file resides in the `ma-extensions` directory as did the `ma-garnet.lisp` file. The `ma-learn` utility both echoes information to the user for inspection and interfaces to the text based knowledge repair utility (`repair-ma`). The accept box to the right of the conditions will be discussed below.

Operation

- 1) Process a feature using either the constant polling mode or feature at a time mode available with CBR. The first mode is activated using a `"(wfr)"` command in the lisp window and is exited using two `"<ctrl>c"` keystrokes. The second is activated for each feature processed using the `"(rmdf)"` command in the lisp window. The return call to ANVIL5K and subsequent tool path generation happens automatically. If using constant polling mode, you must exit it before using `ma-learn`.
- 2) The `ma-learn` window is accessed by typing `"(do-it)"` at the `"*"` prompt in the lisp mag window.
- 3) This window takes about 35-40 seconds to initially paint.
- 4) The top half includes detailed parameter settings input to and output from CBR. Variables passed into CBR from GRAPL-IV are identified by an `"*"` and the others are generated within CBR by using the input data and machining strategies that are expressed as mops (memory organization packages).
- 5) The bottom section provides an interface to the MA knowledge base. Existing knowledge can be modified and extended by communicating up to five conditions and the resulting desired action. This knowledge can be changed for the current tool path or for all similar features processed later.
- 6) The user can enter into one of the table slots at the top or bottom of the screen by depressing the left mouse button in that area. The cursor will appear and the user can use the left and right arrow keys on the keyboard to move around within the cell.
- 7) The enter key must be depressed (on the numeric keypad) in order to go on to some other operation once the cursor is in one of the cell positions.
- 8) The bottom area (where knowledge is modified) appears empty when the `ma-learn` window is first activated using `"(do-it)"`. It can be filled in with information by the user.
- 9) Conditions based on either parameters passed into CBR by GRAPL-IV or those calculated by CBR can be typed in the condition boxes (must be enclosed by parenthesis).

- 10) A single action based on up to 5 conditions is specified in the bottom cell on the screen. Both must be a function of parameters appearing in the top half of the ma-learn window. The words used must correspond to syntax used by the CBR or they will not be accepted. An error will appear in the window where the CBR is running if this occurs.
- 11) When the conditions and action are entered, the accept box is clicked (left mouse button). This writes out a file "repair.mops" to the root directory of the current user. This file contains the changes made in mop format and allows multiple modifications to be stored for the current feature.
- 12) A "(write-mops)" command in the lisp window finishes the knowledge updating process by linking the new strategies with the core knowledge base (ma.mops) again found in the root directory of the user. Subsequent features processed will utilize the added knowledge for similar features from now on. To use this knowledge the command "(use-mine)" must be typed in the lisp window before processing.
- 13) The help, postscript and quit buttons work the same as with the previously discussed utility. The user can quit, get a postscript dump of the learn window, or view a gemacs help file.
- 14) Quit can also be executed in the lisp window with a "(stop-it)" command at the "*" prompt.

Alternative Method

A "(show)" command in the lisp window at the "*" prompt allows the user to see the slots of information in the *new-instance* (ie the feature just processed). The display is in text format rather than the tabular form found in the ma-learn window.

Usage Notes

If, at any time during operations, a problem is encountered, it will most likely show up in the window where lisp is running. The debug prompt is a left bracket ([). The specific sequence of events leading up to the error and the text that precedes the debug prompt should be documented and communicated to someone in the "ma" group. A "q" should be typed at the debug prompt to allow a "(quit)" to end the lisp session.

Conclusions

Several Milling Assistant Project accomplishments were listed in the introduction to this report and are not repeated here. This section draws conclusions from the project which the authors feel should influence the direction of future work.

The MA project combined Artificial Intelligence and CAD/CAM systems to demonstrate an automated technique for programming CNC milling machines from feature based CAD models. The benefits realized by the MA work include the reduction of time associated with NC programming tasks and improved product quality due to the use of "best practices" in the automated decision processor. The research associated with the MA project extended the MA capabilities and defined areas for future MA activities and extensions into other manufacturing processes.

A major conclusion demonstrated by the MA project is that the decision processor must be knowledge-rich and must contain an integrated knowledge acquisition technique (such as MA-Learn) to collect and extend machining knowledge as the system is used. The actual AI implementation methods are of less importance than the amount of machining knowledge captured by the system. Furthermore, any reliance on skilled programmers (LISP, C++, or other) to modify or add new knowledge to the decision system seriously limits the implementation to the laboratory environment.

The knowledge acquisition technique must directly interact with the domain expert to capture machining knowledge. The domain expert is the skilled NC programmer operating the MA system. Future systems must build on the demonstrated concept of a "learn cycle", where the MA proposes a solution, the user defines logic for variations to the proposed solution, and then the knowledge structures are updated. Furthermore, it is desirable to represent the machining knowledge and strategy in an abstract form (such as IDEF), independent of the decision processor implementation. The formalization of machining knowledge using IDEF3 has demonstrated a valuable step in converting practical expertise to specific AI rules for the development of automated systems. A technique to generate decision processor code from the abstract representation is desirable.

Several future directions can be taken from the MA project. The MA techniques can be applied to Coordinate Measuring Machines and Turning Centers. The system can be extended to process multiple features, do more process planning tasks, use information from feature based solid models, and use PDES/STEP definitions directly. Alternate methods of decision processing and knowledge acquisition need to be integrated into the MA framework to reach the goal of a "knowledge-rich" system.

References

- [1] ANVIL-5000 CAD/CAM System, Manufacturing and Consulting Services, Inc., 7560 Redfield Road, Scottsdale, AZ 85260 602-991-8700.
- [2] Brooks, S. L., Hummel, K. E., Wolf, M. L., *XCUT: A Rule-Based Expert System for the Automated Process Planning of Machined Parts*, Allied-Signal Aerospace, Kansas City Division, Kansas City, Missouri, 64141.
- [3] Burd, W.C., *The Milling Assistant*, SAND89-1519, Computer Aided Manufacturing Division, 7483, Sandia National Laboratories, Albuquerque, New Mexico, 1989.
- [4] Burd, W.C., *The NC Assistant -- Interfacing Knowledge Based Manufacturing Tools to CAD/CAM Systems*, SAND88-2088C, Sandia National Laboratories, Albuquerque, NM. 87185, Oct. 1988.
- [5] DeGarmo, Black, and Kosher, Materials and Processes in Manufacturing, 6th Ed., Macmillan, 1984.
- [6] DeVries, Warren R., Analysis of Material Removal Processes, 3rd Ed., New York Pub., 1992, p. 3.
- [7] DeWitte, Paula S., Mayer, Richard, Painter, Michael K., *IDEF Family of Methods for Concurrent Engineering and Business Re-engineering Applications*, Knowledge Based Systems, Inc., 1992, pp. 18-23.
- [8] Hoffman, Edward G., Student's Shop Reference Handbook, Industrial Press Inc., 1986.
- [9] Lascoe, Nelson, and Porter, Machine Shop Operations and Setups, American Tech. Publishers, Inc., 4th Ed., 1973.
- [10] LeClair, Steven R., IDEF the Method, Architecturing the Means to Improved Manufacturing Productivity, SME Tech Paper #MS82-902, 1982, SME.
- [11] Mitchell, F. H. Jr., CIM Systems: An introduction to computer-integrated manufacturing., pp. 18-21.
- [12] Myers, Brad A., Dario, A. Giuse, Roger B. Dannenberg, Brad Vander Zanden, Davis S. Kosbie, Edward Pervin, Andrew Mickish and Philippe Marchal. *Garnet: Comprehensive support for graphical, highly-interactive user interfaces*. IEEE Computer 23(11):71-85, November 1990.

- [13] Myers, Brad A., *An object-oriented, constraint-based, user interface development environment for X in Common Lisp*. In 4th Annual X Technical Conference, Boston, MA. January 1990.
- [14] Riesbeck, C.R. and Schank, R.C., Inside Case-Based Reasoning, Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1989.
- [15] Shah, Jami J., *Current Status of Features Technology*, R-88-GM-04.1, et. Computer Aided Manufacturing-Intl., Inc., 1250 E. Copeland Rd., Suite 500, Arlington, TX, 76001, Nov. 1988.
- [16] VERICUT, CGTech, 22706 Aspan St. Suite 310, El Toro, CA. 92630. (714)-380-8022.

Distribution

Internal Sandia Distribution

0035	J. Lovato
0323	R. G. Easterling
1404	J. A. Ang
1561	H. S. Morgan
1611	C. S. Loucks
1621	D. R. Strip
2100	D. W. Doak
2400	J. Q. Searcy
2400	D. L. Plymale
2481	T. M. Simpson
2482	T. L. Cleveland
2483	P. W. Plomp
2483	W. C. Burd (20)
2483	J. Dunton
2483	R. Helmick
2483	J. Pankey
2483	T. Smith
2483	F. Whiston
2486	P. C. McKey
2606	J. H. Barnette
2801	J. R. Yoder
2814	M. Olson
2882	D. Robbins
2900	W. E. Alzheimer
2901	G. L. Laughlin
2901	T. J. Young
4112	T. M. Bomber
4112	J. Rogers
4200	J. Otts
5408	G. M. Ferguson
7141	Technical Library (5)
7151	Technical Publications
7613-2	Document Processing for DOE/OSTI (10)
8284	L. N. Tallerico
8284	A. J. Hazelton
8284-2	B. E. Affeldt
8523-2	CTF
9911	J. Polito

Distribution (continued)

New Mexico State University Distribution

3449	D. J. Morgan
3449	L. K. Matthews
3449	C. Woods
3450	G. Mulholland (5)
3450	R. Colbaugh
3450	R. J. Pederson
3450	H. Chappelle
3450	F. Burris
3566	G. D. Alexander
4230	S. Kamat
4230	L. Cox (10)
4230	P. R. Finch
4230	J. Mullen
4230	B. Lambert
4230	D. E. Culler (30)
3ARP	(IRSI) Tom Eskridge (5)
3ARP	(IRSI) Ted Slater

Distribution (Continued)

External Distribution

CGTech (1)
15375 Barranca Parkway, Bldg L
Irvine, CA 92603
Attention:
Jon Prun

Manufacturing and Consulting Services. Inc. (5)
7560 E. Redfield Road
Scottsdale, AZ 85206
Attention:
P. J. Hanratty, Ph.D.
B. J. Hanratty
S. Hanratty
W. E. Hoverter, Jr.
E. L. Rose

END

**DATE
FILMED**

11 / 12 / 93