Feature-Based Control of a Machining Center

Thomas R. Kramer
Frederick M. Proctor

Intelligent Systems Division
National Institute of Standards and Technology
Technology Administration
U.S. Department of Commerce
Gaithersburg, Maryland 20899

NISTIR 5926
December 2, 1996

Disclaimer

Acknowledgements

Partial funding for the work described in this paper was provided to Catholic University by the National Institute of Standards and Technology under cooperative agreement Number 70NANB2H1213.

Copyright

CONTENTS

1 Introduction

This report describes an architecture for a Feature-Based Control System (FBCS) for machining mechanical piece parts, and an early prototype implementation of it. By “feature-based control”, we mean that the controller of a machining center uses, as a principal input for machining, a feature-based description of the shape of the object to be made. The report focuses on the feature-based aspects of the system and deals only briefly with control.

2 Background

2.1 Real-time Control System (RCS) Architecture

James Albus, Chief of the Intelligent Systems Division at the National Institute of Standards and Technology (NIST) has for many years spearheaded development of a control architecture known as the Real-time Control System (RCS) and is currently Chief Architect in a continuing controller development project which has fostered FBCS development. We deal only briefly with RCS here. For those interested in more detail, dozens of papers on RCS are available, including [Albus1, Albus2, Albus3].

Under the tenets of RCS, controllers are arranged in a hierarchy. Each controller, save one at the top, is subordinate to a single superior controller, and each controller may have several subordinates. Those at the bottom of the hierarchy have no subordinate controllers but control actuators. Controllers interact by superior controllers sending commands to subordinates, in response to which the subordinates perform actuation or send commands to their subordinates and then send status back. Each controller performs some type of real-time planning, which may range from selecting a pre-made plan without change to totally generative planning.

The field of discrete parts manufacture is amenable to RCS control, and the FBCS conforms to the RCS architecture.

2.2 Enhanced Machine Controller Project

The Enhanced Machine Controller (EMC) project is conducted in the Intelligent Systems Division. The EMC project is headed by one of the authors (Proctor). The primary objective of the EMC project is to build a testbed for evaluating application programming interfaces (APIs) for open-architecture machine controllers. The FBCS is being built in the EMC project and comprises part of the testbed.

In prior work, the EMC project built a machine tool controller and retrofitted a 4-axis machining center at General Motors with it [Proctor]. That controller and variants of it are called “the EMC controller.” The EMC controller incorporated an NC-program interpreter for programs written in the RS274 language [Kramer1].

2.3 Machining Features

2.3.1 General

In this report, a machining feature is a closed volume in space which is related to a machining operation as follows: when the operation is finished, there must be no material remaining inside the feature, and the operation may remove no material outside the feature. In use, the machining features are defined so that a boolean subtraction of the features from the original workpiece (stock or a partially finished part) will result in the desired final shape for the workpiece (finished or partially finished part).

The machining features used to make a specific part may be instances of a fixed library of parametrically defined features, or they may be defined without a library by making boundary or constructive solid geometry representations.

There are, of course, many alternative definitions of machining features; we do not deal with them here. Issues regarding machining features are discussed in [Kramer2].

To machine a part, both the machining features and the machining operations must be defined, and the operations must be sequenced. We have used operations which are instances of parametrically defined operations from a fixed library of operations.

2.3.2 Earlier Work

In the Vertical Workstation of the NIST Automated Manufacturing Research Facility [Kramer3], a primitive CAD system was used with a design protocol [Kramer4] which constrained the user to design in terms of machining features (although the features did not conform fully to the definition just given, because several operations might be required to make one feature). The operations for cutting the features were automatically defined and partially sequenced by a generative process planner [Kramer5]. The individual components of this system were unsophisticated but they were very well integrated. Within a limited range of design, a part could be designed and cut within an hour using this system. An NC-code generator, controllers, and a feature recognition module which could extract features of the required sort from a boundary representation were included [Kramer6]. The Vertical Workstation System stopped just short of feature-based control, using short turnaround off-line NC code generation instead.

From experience with the Vertical Workstation, it became apparent that using machining features for design, while a good technique for a few special situations, is not a general solution for piece part manufacture. It is common to be able to machine a part more effectively (faster, cheaper, tighter tolerances, etc.) if machining features may be used which are not explicit in the design. Many other researchers have come to the same conclusion.

We then started work on the Off-Line Programming System (OLPS) an NC-code generation system which is intended to be used in a larger system in which machining features are defined separately from the design [Kramer7]. A library of parametric machining features was defined for use with OLPS [Kramer8].

2.3.3 STEP

STEP is the common name for standard 10303 of the International Organization for Standardization (ISO). This standard is composed of individual documents known as STEP “Parts”. STEP Part 11 defines the EXPRESS data modeling language. An EXPRESS model definition is contained in one or more constructs called EXPRESS “schemas”. STEP Part 21 defines an exchange file format for transmitting instances of data which has been modeled in EXPRESS schemas. STEP also provides data models for various domains. The models fall in several classes. The class of model intended to be used is called an “Application Protocol” (AP).

A proposed standard set of machining features is defined STEP AP 224 [Slovensky]. This is largely a parametric library of machining features; more detail is given in Section 5. The library defined in AP 224 is quite similar to that defined in [Kramer8] and is used in the FBCS. The specific feature types implemented in the FBCS prototype are round hole, counterbore hole, and rectangular pocket. Implementations of many other feature types from AP 224 are expected to be feasible.

3 Overview of FBCS

The primary purposes of the FBCS are:

1. to demonstrate feature-based control in an open-architecture control system.
2. to serve as a testbed for solving problems in feature-based manufacturing, particularly the partitioning of manufacturing activities into separate activities and the definition of interfaces between activities.
3. to test the usability of STEP methods and models.

We have built an integrated prototype FBCS in the C++ language using RCS controllers. The prototype system controls a 3-axis “mini-mill” machining center.

3.1 FBCS Architecture

The FBCS controller hierarchy is as has been described often in RCS literature - Figure 3 of [Albus3] under “Machine Part”, for example. System controller architecture is shown in Figure 1. Levels of controller are included at the RCS “Workstation”, “Task”, and “Emove” levels. The first two of these were built for the FBCS system. Each controller runs in a separate process (process in the UNIX sense). Processes may be on the same computer or different computers. Controllers communicate via a messaging system [Shackleford].

3.1.1 Part Controller

The top level “Part Controller” is focused on making a part (actually on transforming a workpiece from an initial shape to a final shape in a single setup)[1]. This controller has a “Part Planner” embedded in it. The planner is part of the same process as the rest of the controller and interacts with the rest of the controller via an API for part planners.

3.1.2 Feature Controller

The mid-level “Feature Controller”[2] focuses on making single features. This controller has a “Feature Planner” embedded in it. The planner is part of the same process as the rest of the controller and interacts with the rest of the controller via an API for feature planners.

3.1.3 Fixture Controller

A second mid-level controller, the “Fixture Controller” controls the fixturing of the workpiece. In the prototype we have implemented this by having the Fixture Controller write instructions to a user interface, where a human is expected to read what to do to fixture the part, to fixture the part, and then to tell the system that the fixturing is completed. Unfixturing is provided for similarly.

1. Because we are using Next Generation Controller (NGC) project terminology, we actually call the Part Controller the “Workstation Planning Standard Application” (WPSA) controller. We are using clearer terms in this report.
2. “Workstation Management Standard Application” (WMSA) controller in NGC terminology.

3.1.4 Other Controllers

The third-level “Host Machine Executive” (HME) controller is the top-level controller of the previously existing EMC controller. The HME runs NC programs. The HME has two subordinates: Input/Output (I/O) and Trajectory (Traj). All three of these controllers are conceptually at the RCS EMOVE level. I/O and Traj have subordinates not shown in the figure.

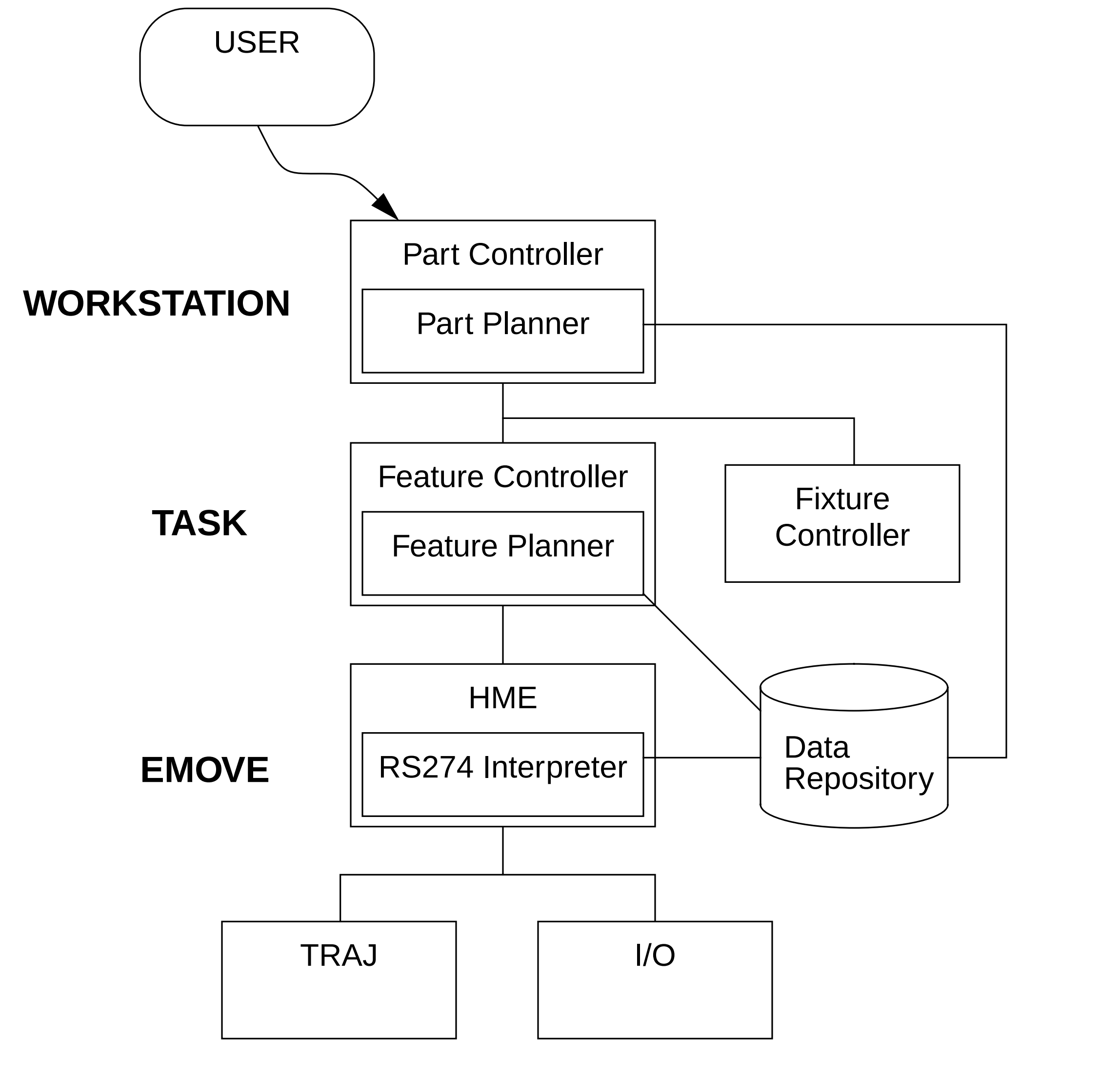

Figure 1. FBCS Architecture

3.1.5 User Interfaces

A user tells the Part Controller what to do via the user interface. Figure 1 shows only this one user interface. In actual practice, during development, we have user interfaces to every controller, and there is a user interface to a stand-alone executable for testing the two planners. We have not put much thought, yet, into interfaces for FBCS users other than developers. As the FBCS moves out of the prototype stage, it will be necessary to focus on this.

3.1.6 Data Repository

In the prototype we are using a file system as the data repository. A file system is fully adequate for the prototype and may be fully adequate for future versions of the system. At some point as the FBCS matures we will look into the desirability of using a database system for the Data Repository.

3.2 FBCS Planning

3.2.1 Architecture of Planning

In the RCS architecture, every controller does planning. In the FBCS architecture (which conforms to RCS), planning for machining a part may occur in one or two stages.

In two-stage planning, a plan with variables, alternatives and possibly omissions is prepared first. Such a plan is usually prepared off-line. A plan of this sort is intended to be usable in an environment where tooling may change somewhat, the initial workpiece may vary, and different batch sizes may be made. During system operation this plan is retrieved and simplified to make a second plan (which has no alternatives or omissions) by selecting a specific sequence of operations from among alternatives in the stage 1 plan, selecting specific resources where the stage 1 plan has generic resources, filling in blanks where the stage 1 plan has omitted items, etc.

In one-stage planning, a fixed plan of the second sort described above is prepared directly.

3.2.2 Planning Languages

The ALPS process planning language [Catron1] supports both types of planning, and we are using it in the FBCS system with the Part Planner.

At the Feature Controller level we are using the NGC dialect of the RS274 NC-programming language [NCMS] to record the results of planning. This was selected because the existing EMC controller already included an interpreter for this language, not because we believe it is the best choice. Planning at this level is primarily path planning but also includes auxiliary machine operations such as changing tools and turning coolant on or off.

3.2.3 Planning in the FBCS Prototype

Part Planner

In the FBCS prototype, the Part Planner starts with stage 1 plan in ALPS which has been prepared off-line (by a human using a text editor, currently). A plan of this sort contains descriptions of machining operations and pointers to machining features. During initialization of a setup, the Part Planner produces a stage 2 plan. The Part Planner thereafter (when asked to do so) tells the Part Controller which machining feature to make next by referring to the stage 2 plan.

The hard part of part planning is generating the stage 1 plan, i.e. deciding what the machining features should be, what operations should make them, and how to fixture the workpiece for each setup. The Part Planner in the prototype does not tackle this problem at all. We believe this problem can be approached in several ways, as described in Section 12.3.

Feature Planner

In the FBCS prototype, the Feature Planner generates NC-programs to cut features. It generates stage 2 plans (NC-programs, each of which makes one feature) from higher level information, not by refining a stage 1 plan.

The path planning capabilities of the Feature Planner are fair for the few feature types and operations it handles. The only operation of even moderate complexity currently implemented is finish milling rectangular pockets.

The Feature Planner has access to the same information about the operations to be carried out and features to be made as does the Part Planner. This is accomplished in the prototype by having both planners run identical functions during initialization of a setup, which is more convenient than providing for sharing a single copy of data (since the two planners are running in different processes).

3.3 FBCS Operation

3.3.1 System Initialization

The system is brought to a warm state by starting the various independent processes and establishing communications among them so that commands and status messages can be sent from controller to controller. Then explicit initialization commands are sent down hierarchically, with each controller in the hierarchy sending an initialize command to its subordinate, and each subordinate sending an OK message to its superior when it has completed its initialization. The Part Controller also makes a Part Planner Init function call and the Feature Controller calls Feature Planner Init. These two functions are identical in content; the planner reads three files which are independent of any particular part to be made:

1. tool catalog
2. tool inventory
3. machining options

Whenever a file is read, a working form is built to represent the information in the file. This is an internal data structure which can be used during subsequent operation.

3.3.2 Setup Initialization

After system initialization, it is expected that the FBCS will be cutting a variety of parts. For each instance of cutting a part, setup information is required. Each setup is initialized as follows.

First, the Part Controller receives a Part Remove Volumes message from the user interface. The part design, process plan, machining features and associated information for this setup have previously been stored in the Data Repository, but tool paths and NC code have not been generated.

Second, the Part Controller calls the Part Planner Open Setup function, in response to which the Part Planner reads in and processes data from the Data Repository. Specifically, it:

1. reads the setup file
2. reads the design file
3. reads the fixture file
4. reads the workpiece file
5. reads the machining feature file
6. reads the process plan file
7. builds a stage 2 process plan

8. preprocesses the machining volumes

Third, the Part Controller sends a Feature Open Setup message to the Feature Controller.

Fourth, upon receiving the Feature Open Setup message, the Feature Controller calls the Feature Planner Open Setup function, in response to which the Feature Planner does the same things the Part Planner did.

Fifth, the Part Controller sends a fixturing message to the Fixturing Controller, where a human fixtures the part and sends an OK status message back when the fixturing is completed.

3.3.3 Cutting Features

Once a setup is initialized, features may be cut. To cut a feature:

First, the Part Controller calls the Part Planner First Feature function (for the first feature) or the Part Planner Next Feature function (for all other features) to decide what feature to make next. The return value of the function gives the answer.

Second, the Part Controller sends a Feature Make Feature message to the Feature Controller. The message includes the number of the feature to make.

Third, the Feature Controller calls the Feature Planner Generate function. In response, the Feature Planner generates and stores an NC program which will cut the feature. Then the Feature Controller sends the HME an HME Run Program message. In response, the HME runs the program, making the feature. Several operations may be necessary for cutting a single feature; all necessary operations are included in the program which is generated.

4 Data

All data files are kept in the Data Repository.

4.1 Types of Data

Data sets of each of the nine following types are kept in separate files.

The first three types of data are independent of which part is to be machined. One of these is for user preferences regarding machining options, while the other two are for tooling (cutters, probes, holders, etc.). The other six types of data depend upon the work being done in a specific setup of a part.

1. Machining Options - the user’s preferences regarding such things as retract height, tool change position, deep hole drilling, etc.
2. Tool Catalog - a catalog of types of tools that are usually available.
3. Tool Inventory - an inventory of specific tools that are currently actually available.

For the machining done in a single setup of a part, one of each of the following types of files is required.

1. Machining Features - a description of the machining features to be cut away.
2. Process Plan - a plan which includes at least one operation to make each machining feature. The plan may contain alternatives and describe resources generically. Each step of the plan which calls for material removal must reference a machining feature.

3. Fixture - a description of the fixture which holds the workpiece being machined.
4. Workpiece - a description of the shape of the workpiece as it is before machining is begun in the current setup. This may be raw stock or a partially machined part.
5. Design - a description of the intended shape of the workpiece after machining in the current setup. This may be a finished part or a partially machined part.
6. Setup - the setup file specifies:
   a. a set of data about data, primarily the names of the files associated with the machining done in one setup for a specific part.
   b. a description of the location of the fixture, workpiece, design, and machining features with respect to the coordinate system of the machining center.
   c. a description of a box-shaped volume containing the workpiece.

Output NC programs are prepared as pseudocode in working form and printed to files as RS274/NGC programs.

The process plan and machining feature files may include constructs (such as macros and variables) which need to be concretized for generating NC code. Concrete versions of process plans and machining features (with macros and variables removed and setup specifications considered) are built during the initialization of a setup (see Section 10 for details). These data sets are kept in active memory. In future versions of the FBCS it will be possible to write concretized data out to files and use the FBCS in a mode where the concretized files are taken as input.

4.2 Data Formats

Files in the Data Repository are STEP Part 21 exchange files. For each file type, there is an EXPRESS schema providing a model of data of that type.

Four types of data are prepared using STEP standard EXPRESS schemas as follows:

1. machining features - STEP AP 224
2. fixture - STEP AP 203
3. workpiece - STEP AP 203
4. design - STEP AP 203

The EXPRESS schema used in the FBCS for STEP AP 203 is the AP 203 schema provided by STEPTools Inc. with functions and “where” clauses removed. STEPTools, Inc. is a commercial venture which provides tools for dealing with STEP methods, models, and data.

The EXPRESS schema used in the FBCS for STEP AP 224 is a schema provided by Len Slovensky, owner of AP 224. In STEP terms it is an Application Resource Model (ARM) type of model.

The following EXPRESS schemas being used are not STEP standards:

1. machining options - a special-purpose schema written for the FBCS system.
2. process plans - ALPS plus 3-axis machining operations.
3. setup - a special-purpose schema written for the FBCS system.
4. tool catalog and tool inventory - a subset of the model built in EXPRESS by the NIST Manufacturing Systems Integration Division and contractors of that division [Jurrens], with a tool_instance entity added.

4.3 Using Data During System Operation

Files in the Data Repository are converted into working forms in computer memory during initialization of a setup.

Working forms for all nine types of data listed above are built by using the STEPTools "useDesign" function, which automatically builds a working form, given a file name. To enable this, C++ class definitions and access functions were created automatically as part of the system software by using the STEPTools express2c++ utility (which uses the EXPRESS schema for that type of data as input). STEPTools library functions are called by the automatically generated code for accessing the working forms; the source code we wrote does not call the library functions.

5 STEP AP 224 Machining Features

STEP AP 224, the proposed ISO standard for "Mechanical Product Definition for Process Planning Using Machining Features" [Slovensky] is being used in the FBCS. AP 224 is largely a specification of a library of parametric machining features, but also provides for defining machining features in terms of a boundary representation and provides for related data, such as design_exceptions and requisitions.

AP 224 provides 51 parametric "manufacturing_features" including, for example: boss, chamfer, circular_pattern, compound_feature, counterbore_hole, countersunk_hole, edge_round, fillet, general_pattern, general_pocket, groove, hole, pocket, rectangular_pattern, rectangular_pocket, slot, spherical cap, thread. These feature types are useful in the FBCS because machining operations which make them can be defined and tool path generators can be devised for them.

Having features defined in terms of a boundary representation is less useful because it is very difficult, if not completely infeasible, to identify machining operations for making entities found in boundary representations without grouping the entities (i.e., recognizing features). It is correspondingly difficult to generate tool paths.

The AP 224 feature library provides rich methods of describing details of features. Thirteen types of (planar) feature_profile are provided, for example. Feature_profiles may be swept in various ways to produce manufacturing_features. Twenty-nine types of feature_definition_item are provided. This includes feature details such as hole_bottom_condition, path, and taper.

6 APIs and Messaging Interfaces

APIs are used where two functional modules are in the same process (planner and rest of controller in the FBCS), while messaging interfaces are used where functional modules are in different processes. It is feasible to change an API into a messaging interface, in the event functional modules are put into separate processes — and vice versa if processes are merged.

Function and message names used in this section are intended to be descriptive. The actual names in the prototype software are different.

6.1 Part Planner API

The Part Planner API includes six interface functions. They are intended to be called by the Part Controller. Every function has return values which indicate an error has occurred. Functions which do not return feature numbers return a value indicating OK, as long as no error has occurred.

When a more capable Part Planner is built (see Section 12.3) additional interface functions will be required.

6.1.1 Part Planner Init

Part Planner Init takes no arguments. The general meaning is: get ready to run for making one or more parts. In the prototype, when Part Planner Init is called, the Part Planner reads three data files and makes internal representations of data.

6.1.2 Part Planner Open Setup

Part Planner Open Setup takes one argument: the name of a setup file. The general meaning is: get ready to make a number of features on a workpiece using the data included or referenced in this setup file. In the prototype, when Part Planner Open Setup is called, the Part Planner reads six data files, makes internal representations of data, checks the data, and runs preprocesses.

6.1.3 Part Planner First Feature

Part Planner First Feature takes no arguments. The meaning is: return the number of the first feature to make.

6.1.4 Part Planner Next Feature

Part Planner Next Feature takes one argument: the number of the last feature made. The meaning is: return the number of the next feature to make.

6.1.5 Part Planner Close Setup

Part Planner Close Setup takes no arguments. The general meaning is: get ready to exit or to run another part; finish any and all work associated with the current setup. The working forms for the current setup are deleted from memory and references to them are removed from the Part Planner world model.

6.1.6 Part Planner Exit

Part Planner Exit takes no arguments. The general meaning is: stop running.

6.2 Feature Planner API

The Feature Planner API includes five interface functions. They are intended to be called by the Feature Controller.

6.2.1 Feature Planner Init

Feature Planner Init takes no arguments. The general meaning is: get ready to run for making one or more parts. In the prototype, when Feature Planner Init is called, the Feature Planner reads three data files and makes internal representations of data.

6.2.2 Feature Planner Open Setup

Feature Planner Open Setup takes one argument: the name of a setup file. The general meaning is: get ready to make a number of features on a workpiece using the data included or referenced in this setup file. In the prototype, when Feature Planner Open Setup is called, the Feature Planner reads six data files, makes internal representations of data, checks the data, and runs preprocesses.

6.2.3 Feature Planner Generate

Feature Planner Generate takes two arguments: a feature number and an NC file name (the name of the file to write). The general meaning is: write an NC-program to cut a feature. In the prototype, when Feature Planner Generate is called, the Feature Planner generates and stores an NC program of the given name which will cut the given machining feature (from the original machining features). This may require only one operation, or it may require many (if the original machining feature was an array of holes, for example).

6.2.4 Feature Planner Close Setup

Feature Planner Close Setup takes no arguments. The general meaning is: get ready to exit or to run another part; finish any and all work associated with the current setup. The working forms for the current setup are deleted from memory and references to them are removed from the Feature Planner world model.

6.2.5 Feature Planner Exit

Feature Planner Exit takes no arguments. The general meaning is: stop running.

6.3 Message Interface to the Part Controller

The messages which the Part Controller may be sent (aside from initialize and exit) are as follows. In normal usage these are sent from the user interface, but the FBCS might be embedded in a larger system, in which case the messages would come from a superior controller.

6.3.1 Part Remove Volumes

Part Remove Volumes takes one argument: the name of a setup file. This does the machining of one setup. All the data files named in the setup file must exist already. This is implemented in the prototype.

6.3.2 Part Machine Part

Part Machine Part takes three arguments: a setup file name, a design file name, and a workpiece file name. This does the machining of one setup. If the setup file exists, it is retrieved and used as in Part Remove Volumes. If not, the Feature Planner is called to generate the setup file and all the files it references which do not yet exist. The data is then used as in Part Remove Volumes. This message is not implemented in the prototype.

6.4 Message Interface to the Feature Controller

The messages which the Feature Controller may be sent (aside from initialize and exit) are as follows. In normal operation, these messages are sent by the Part Controller.

6.4.1 Feature Open Setup

Feature Open Setup takes one argument: a setup file name. This message causes the Feature Controller to:

1. check that it is going to be possible to make the part using the data which been provided and the current tooling. In the long run, this step might include changing the tools in the carousel, if needed.
2. make a Feature Planner Open Setup function call.
3. generate and run a small NC program to get things started. This might not be needed

in all cases and is not included in the prototype.

6.4.2 Feature Make Feature

Feature Make Feature takes one argument: a feature number. It is understood that the feature is to be found in the currently open set of machining features. This message causes the Feature Controller to:

1. make a Feature Planner Generate function call, in response to which the Feature Planner generates and stores an NC program to make the feature.
2. send an HME Run Program message to the HME, which runs the program and makes the feature.

6.4.3 Feature Close Setup

Feature Close Setup takes no arguments. This message causes the Feature Controller to:

1. generate and run an NC program to shut things down, including:
   a. retract the tool from the vicinity of the workpiece.
   b. turn coolant off if on.
   c. move the machine to park position.
2. send a Feature Planner Close Setup message to the Feature Planner.

7 Verification

For the FBCS to be effective, it must use every opportunity to avoid errors while it runs, or stop before it makes one. A single crash can put a machining center out of action for weeks. The approach to error prevention should be paranoid (Murphy was right).

A full discussion of error prevention is not given here. An extensive discussion is given in [Kramer9]. Some of the important topics are given below. Machining feature checking (Section 7.2) and machining operation verification (Section 7.3) have been partially implemented in the stand-alone prototype.

A solid modeler is necessary for several types of verification. The Parasolid solid modeler from Electronic Data Systems Corporation has been obtained, and it comes with a suitable API, but it has not been used in the prototype. It will be the underlying solid modeling engine in future versions of the FBCS.

7.1 Workpiece Shape Verification

Two level shape verification seems best.

1. Do booleans with machining features to check that the final workpiece will, in principle, be the shape it is supposed to be if the process plan is carried out and to check that fixtures are not cut.

2. Do booleans with individual tool sweep volumes to check no material is left in a machining feature, that no material outside a machining feature is removed, and that fixtures are not machined. Do booleans with shank and tool-holder with individual moves to check that no crashes should occur.

7.2 Machining Feature Checking

Each machining feature should be checked to make sure it is a realizable machining feature. This includes checking that each parameter is within a range appropriate for it, and that rules regarding the relations between parameters are not violated.

7.3 Machining Operation Verification

1. The tool and the operation should be compatible. For example the diameter of a drill making a hole should be the same as the diameter of the hole.

2. Feeds and speeds should be in a reasonable range.

3. Pass depths and stepovers should be in a reasonable range.

7.4 Other Types of Verification.

1. Entry methods should be reasonable.

2. Fixturing should be sufficient. For example, machining forces should not move the workpiece.

3. Check that the machining operations specified should be able to achieve required tolerances and surface finishes.

8 Tool Data

The FBCS requires both tool catalog data and tool inventory data.

8.1 Tool Catalog

The tool catalog lists all types of tools available in principle for use. Tool catalog data includes, for example:

1. tool type id
2. nominal dimensions (such as length and diameter)
3. material from which the tool is made
4. materials the tool can cut
5. number of flutes
6. maximum RPM of use (not currently)
7. maximum number of reworks

8.2 Tool Inventory

The tool inventory lists the tools available. “Available” might mean “in the carousel” or “easily obtainable for putting in the carousel.” In the prototype, each tool_instance has a slot number in the carousel.

Each tool in the inventory is an instance of some tool type described in the catalog and inherits all the information about that type of tool contained in the catalog. In addition, each tool in the inventory has some information associated with it which does not exist in the catalog, including, for example:

1. tool_instance_id
2. location
3. actual dimensions
4. time in service
5. number of reworks

9 Machining Operations

The EXPRESS schema for FBCS process plans defines the following operations. Of these, the NC code generators in the prototype handle: change_tool, counterbore, finish_mill (for rectangular pockets), flood_off, flood_on, and twist_drill.

1. bore
2. center_drill
3. chamfer
4. change_tool
5. counterbore
6. countersink
7. face_mill
8. finish_mill
9. flood_off
10. flood_on
11. fly_cut
12. peripheral_mill
13. plunge
14. ramp
15. ream
16. rough_mill
17. round
18. slot
19. spiral
20. tap
21. twist_drill

10 Preprocessing

During preprocessing, the process plan and machining features are modified to make them specified in detail. We call the modification operation concretizing. Currently the Part Planner and the Feature Planner use identical software for concretizing. This is simple and effective in insuring both planners use the same data when necessary, but it is somewhat inefficient because much of the data is needed by only one of the two planners. In future versions of the FBCS, it may be desirable either to eliminate that part of the redundancy that is useless, or to provide for data sharing between planners, so that the concretizing need only be done once.

10.1 Concretize Plan

The process plan concretizer takes in an ALPS process plan and puts out an array of operations to be performed in order.

In the original process plan, the feature referenced by a single plan step may be an AP224 macro feature, such as a circular pattern of holes. In the case of such a step, the array of operations contains an operation for each hole in the pattern.

Steps in the original process plan may omit values for process attributes, such as spindle speed or stepover. The operations in the operation array contain values for all attributes for each step.

The input process plan may omit steps such as turning coolant on and off or changing tools. These are inserted in the array of operations.

The input process plan may have alternative steps for machining a feature. The array of operations will have no alternatives.

So that the plan will be carried out fully by performing operations which reference features, each step of the plan must reference some feature. All the steps associated with a feature are grouped together in the array of operations. So that the ordering requirements of the plan may be met under this constraint, no two steps in the plan may reference the same feature. A plan index is constructed from features to operations; for each feature, the plan index lists the array index of the first operation and the number of operations associated with the feature.

10.2 Concretize Machining Features

The machining feature concretizer takes in the working form created when the machining features file was read in and builds a new working form.

In the initial machining feature file there may be macro features, such as a circular pattern of bolt holes. These are defined by defining one of the holes and the pattern. The Feature Planner requires a specific fully defined feature for each operation. The machining feature concretizer generates these from the initial machining feature form. There may be other constructs in the initial machining feature form such as mirror images and copied groups that require similar processing.

In addition, the machining feature concretizer moves the features to the location specified by the setup.

Any feature in the input machining feature file which is not to be made will not be included in the concretized machining feature file. This will occur whenever a feature has been defined in the initial machining feature file which is referenced only by an alternative in the input plan not used when the plan is concretized.

11 Code Generator

The code generator takes in a machining operation description and generates the tool path (or other operation) in pseudocode. The pseudocode is stored briefly in active memory. An NC code printer translates the pseudocode into RS274/NGC NC code.

In the prototype, existing generators which were written in Lisp for the OLPS system have been rewritten in C++ and revised somewhat. Commercial path generators are available and might be used in the future.

Since Feature Planner Generate currently uses feature numbers which refer to the input machining features set (as opposed to features from the concretized set, or process plan steps from the initial plan, or operations from the concretized array of operations), each feature is made by some number of operations in the operations array. When a Feature Planner Generate function call is made, the Feature Planner makes:

1. a start_program function call.
2. a number of generate_one_step function calls
   (one for each operation needed to make the feature).
3. an end_program function call.

For each operation, the code generator will:

1. get the tool away from where it was left at the end of the preceding operation.
2. move the tool to where it can start the current operation.
3. generate the path for the current operation.

Executing the Feature Planner Generate message results in one or more generate_one_operation function calls. Each generate_one_operation function call refers to an operation description (which will have been generated when the plan is concretized). A generating function for the given type of operation is selected and called by generate_one_operation.

To avoid confusion and make it easy to use function pointers, the argument lists for all the functions which generate code for named operations (i.e., the ones called by generate_one_operation) are the same. Where an argument is not required by a given operation, it is given a filler value in the call, and is ignored during execution.

Arguments are:

1. removal_volume - a pointer to a machining feature description.
2. tool_instance - a pointer to a specific instance of a tool.
3. stepover (radial offset specifying horizontal cut depth).
4. pass depth (upper bound of vertical depth increment).
5. spindle speed.
6. general feed rate - used if only one rate is required and whenever plunging or slotting is not being done.
7. plunge feed rate - used if the operation includes a plunge cut.
8. slotting feed rate - used if the operation includes slotting (cutting full tool width with material on both sides).
9. entry strategy - used for pockets. There are five alternate strategies.
   a. plunge
   b. spiral hole
   c. ramp
   d. open side
   e. existing opening
10. machining strategy - different strategies are embodied in different functions applicable to the same feature and operation type. For example, a pocket may be cut zig-zag or spiral out from center.

12 Issues

12.1 Features vs. Plan Steps

The FBCS prototype focuses on features. This approach was taken to keep the FBCS compliant with existing RCS literature. We believe it is better to focus on steps from the concretized process plan, since the steps which make features are usually a proper subset of the steps of the concretized plan. Some steps (changing tools, turning coolant on and off) do not make features and do not conceptually belong to a specific feature; such steps are assigned to specific features in the prototype.

In a plan-focused system, finer control is exercised because NC programs generated under a feature focus may include several operations; these programs will be split into separate NC programs under plan focus. Finer control is desirable, we believe.

Focusing on features rather than steps forces a slightly awkward approach to concretizing the process plan. As described in Section 10.1, currently an index from the original features to the concretized plan is required. If the focus is on steps from the concretized plan, no index is required.

Under a plan focus, the “Feature Controller” would become the “Operation Controller”, and the “Feature Planner” would become the “Operation Planner”. The APIs and messaging interfaces to these two entities would change accordingly. An API for an Operation Controller is described in [Michaloski].

12.2 A Principle of Modularization

Past experience with feature-based machining indicates that the range of parts which can be made when human intervention is allowed in an otherwise automated system is (loosely speaking) a factor of ten larger than the range without human intervention.

One basic principle of modularization in the FBCS system is that modules are defined so that data required to be transferred from one module to another provides a convenient, suitable focus for human input. Having a human modify data as it passes from one module to another is an effective method of harnessing human capabilities.

As an example, handling tool changes, coolant, and other minor details could have been made a function of either the process plan concretizer or the trajectory generator. In keeping with the principle of providing rich output data for humans to modify, these functions were placed in the process plan concretizer. That way, operations for changing tools and turning coolant on and off appear in the concretized process plan. Also in the concretized process plan, each operation identifies only one sub-volume of a machining feature which was originally a pattern or group of sub-volumes.

Since future versions of the system will able to take concrete plan and machining feature files as input, it will be convenient for a human to either delete or add such operations to the plan or to change the order of operations before the trajectory generator starts to work on it. If the functions had been assigned to the trajectory generator, there would not be an opportunity for human intervention until the NC code was produced.

It would be useful to add user-friendly editors to the system for each kind of data passed between modules.

12.3 Part Planning

The core of part planning is to take the design for a part, determine how many setups are required to make the part (and on what sort of machines), and then, for each setup, determine what the machining features should be. This functionality is not included at all in the prototype.

The difficulty of planning for a given design ranges from fairly easy to quite hard, depending upon the nature of the design and the degree of design for manufacturability applied in the design process. It is assumed here that the planner has access to all output design data (often not the case in practice).

If a part was designed on a CAD system which constrains the designer to design using a library of parameterized manufacturable features (features that map easily to machining features), then it is relatively straightforward to plan for manufacturing the part. The NIST Vertical Workstation system was of this type, and a system of this type was built and used internally by Sun Hydraulics. Since the library is fixed, rules for what machining operations to use to make specific feature types can be devised. A given rule may apply within specific ranges of feature parameters and specific tolerance and surface finish limits. A rule might decompose a manufacturable feature into several machining features (as defined earlier) and might use several operations to make a single manufacturable feature.

If a part was designed (on any kind of system) so that it has the same shape as a part that could have been designed using manufacturable features, then the manufacturable features can be recognized (automatically or by a human), and thereafter the planning problem is the same as if the part had been designed using manufacturable features [Pratt].

If the part was designed using parametric features from a fixed library which are not *a priori* manufacturable features, then it may be feasible to devise a mapping from the design feature library to a library of manufacturable features, reducing the problem to an apparently easy one. Since the design process will not have considered manufacturability directly, however, making good parts by this method will be more difficult than in a system where design is by manufacturable feature.

If the part was designed without using features or by using features which were not from a fixed library, the planning problem becomes yet harder.

The first version of an improved Part Planner for the FBCS can be built assuming that a design is available in terms of manufacturable features. Part designs which the system can handle may then be obtained by building a design-by-manufacturable-feature system, by building a feature mapper for an existing design-by-feature system, or by building a feature recognition system — which might use a boundary representation as input and might be automatic, manual, or a hybrid. The Navy RAMP project has used the features provided by STEP AP 224 as design features. Using designs made this way should readily implementable.

A truly powerful Part Planner would consider alternative sets of machining features and perform optimization [Gupta].

Most work on automatic process planning has assumed that the process planner already knows what the machining features are when the planning activity starts. This makes the problem easier, but also makes it unrealistic. To make an effective and efficient process plan, it is necessary to define the machining features and to plan how to make them concurrently. Determining how to do that is an important area for further research.

References

[Albus1] Albus, James S.; *A Theory of Intelligent Systems*; Control and Dynamic Systems; Vol. 45; 1991; pp. 197 - 248

[Albus2] Albus, James S.; McCain, Harry G.; Lumia, Ronald; *NASA/NBS Standard Reference Model for Telerobot Control System Architecture (NASREM)*; NIST Technical Note 1235, 1989 Edition; National Institute of Standards and Technology; April 1989

[Albus3] Albus, James S.; *A Reference Model Architecture for Intelligent Systems Design*; NISTIR 5502; National Institute of Standards and Technology; September 1994

[Catron1] Catron, Bryan; Ray, Steven R.; *ALPS - A Language for Process Specification*; International Journal of Computer Integrated Manufacturing; Vol. 4, No. 2; 1991; pp 105 -113

[Gupta] Gupta, Satyandra K.; et al; *Building MRSEV Models for CAM Applications*; Advances in Engineering Software; Vol. 20, No. 2/3; 1994; pp. 121-139

[Jurrens] Jurrens, Kevin K.; Fowler, James E.; Algeo, Mary E.; *Modeling of Manufacturing Resource Information*; NISTIR 5707; National Institute of Standards and Technology; July 1995

[Kramer1] Kramer, Thomas R.; Proctor, Frederick M.; *The NIST RS274KT Interpreter*; NISTIR 5738; National Institute of Standards and Technology, Gaithersburg, MD; October 1995

[Kramer2] Kramer, Thomas R.; *Issues Concerning Material Removal Shape Element Volumes (MRSEVs)*; International Journal of Computer Integrated Manufacturing; Vol. 7, No. 3; 1994; pp. 139-151; (also published as NISTIR 4804; National Institute of Standards and Technology; March 1992*)*

[Kramer3] Kramer, Thomas R.; Jun, Jau-Shi; *Software for an Automated Machining Workstation*; Proceedings of the 1986 International Machine Tool Technical Conference; September 1986; Chicago, Illinois; National Machine Tool Builders Association; 1986; pp. 12-9 through 12-44

[Kramer4] Kramer, Thomas R.; Jun, Jau-Shi; *The Design Protocol, Part Design Editor, and Geometry Library of the Vertical Workstation of the Automated Manufacturing Research Facility at the National Bureau of Standards*; NBSIR 88-3717; National Institute of Standards and Technology (formerly National Bureau of Standards), January 1988

[Kramer5] Kramer, Thomas R.; *Process Plan Expression, Generation, and Enhancement for the Vertical Workstation Milling Machine in the Automated Manufacturing Research Facility at the National Bureau of Standards*; NBSIR 87 - 3678; National Institute of Standards and Technology (formerly National Bureau of Standards); November 1987

[Kramer6]	Kramer, Thomas R.; *A Parser that Converts a Boundary Representation into a Features Representation*; International Journal of Computer Integrated Manufacturing; Vol. 2, No. 3, May-June 1989; pp. 154-163; (also published as NISTIR 88-3864; National Institute of Standards and Technology; September 1988)
[Kramer7]	Kramer, Thomas R.; *The Off-Line Programming System (OLPS): A Prototype STEP-Based NC-Program Generator*, proceedings of a seminar Product Data Exchange for the 1990's; New Orleans, Louisiana; NCGA; February 1991; Vol. 2
[Kramer8]	Kramer, Thomas R.; *A Library of Material Removal Shape Element Volumes (MRSEVs)*; NISTIR 4809; National Institute of Standards and Technology; March 1992
[Kramer9]	Kramer, Thomas R.; Strayer, W. Timothy; *Error Prevention and Detection in Data Preparation for the Vertical Workstation Milling Machine in the Automated Manufacturing Research Facility at the National Bureau of Standards*; NBSIR 87-3677; National Institute of Standards and Technology (formerly National Bureau of Standards); November 1987
[Michaloski]	Michaloski, John; et al; *Enhanced Machine Controller (EMC) Application Programming Interface (API) Design Guide*; unpublished draft; National Institute of Standards and Technology; August 1995
[NCMS]	National Center for Manufacturing Sciences; *The Next Generation Controller Part Programming Functional Specification (RS-274/NGC)*; Draft; NCMS; August 1994
[Pratt]	Pratt, Michael J.; *Automated Feature Recognition and its Role in Product Modeling*; Geometric Modelling, Computing Suppl. 8; 1993; G. Farin, H. Hagen, H. Noltemeier (eds); Springer-Verlag; pp. 241-250
[Proctor]	Proctor, Frederick M; et al; *Simulation and Implementation of an Open Architecture Controller*; Proceedings of the SPIE International Symposium on Intelligent Systems and Advanced Manufacturing; Philadelphia, PA; 1995
[Shackleford]	Shackleford, Will; *Real-Time Control Systems Library Documents*; http://isd.cme.nist.gov/proj/rcs_lib; 1996
[Slovensky]	Slovensky, Len; *Industrial Automation Systems and Integration - Product Data Representation and Exchange - Part 224: Application Protocol: Mechanical Product Definition for Process Planning Using Machining Features*; Committee Draft; International Organization for Standardization; 1995