3

Feature-based Process Planning Based on STEP

Thomas Kramer[1] and Frederick Proctor[1]

[1]Intelligent Systems Division, MS8230, National Institute of Standards and Technology
100 Bureau Drive, Gaithersburg, MD 20899, USA
Email: thomas.kramer@nist.gov, frederick.proctor@nist.gov

Abstract
This chapter begins by describing characteristics a process planning language should have. Then it discusses the extent to which four process planning languages have these characteristics. The first two of these are STEP standard process planning languages (all of AP 240 and part of AP 238). The third is part of ISO 14649. The fourth, called FBICS-ALPS, is a version of the ALPS language adapted for FBICS, a system that does automatic feature-based process planning. Next the chapter summarizes the machining features available in AP 238, ISO 14649, and AP 224. The way in which FBICS does feature-based process planning is presented and a system is described that translates FBICS plans into ISO 14649 plans (which are conceptually identical to AP 238 plans). Finally, the chapter is summarized, and improvements in FBICS needed to make it industrially useful are presented.

3.1 Introduction

ISO 10303 and ISO 14649 include several parts that deal with process plans and features and, hence, are used or could be used in STEP-NC. In addition, a Feature-Based Inspection and Control System (FBICS) has been built at the U.S. National Institute of Standards and Technology that uses STEP methods and standards. This chapter discusses process plans, features, and process planning (the act of making a process plan that uses features). For each of these three topics, issues are discussed, the relevant sections of ISO 10303, ISO 14649, and FBICS are presented, and these sections are compared.

This chapter uses many STEP concepts introduced in Chapter 1. Readers who are not familiar with STEP should read Chapter 1 before tackling this chapter. Throughout this chapter, it should be borne in mind that (with a few exceptions) AP 238 and ISO 14649 have identical semantics, since ISO 14649 provides the ARM for AP 238.

3.2 Process Plans

The first subsection of this section defines process plan and discusses the desirable characteristics of a process planning language. The other four subsections are devoted to presenting three STEP-based process planning languages and discussing the extent to which each of them has the desirable characteristics.

3.2.1 Definition and Desiderata

A process plan, in general, is a recipe for transforming some input materials or partially finished products into finished or (more) partially finished products. Process plans may be for continuous processes (as in oil refining) or discrete processes. Discrete process plans consist of individual operations. Since we are interested in manufacturing using NC machines, which is done with individual operations, we deal here only with discrete process plans, primarily plans for machining by milling or turning. FBICS can also write and execute plans for inspection using a coordinate measuring machine (CMM) or a machining center equipped with a touch trigger probe, but few details of that aspect of FBICS are given here.

Since process plans must be stored, a file format for representing them is required. The STEP way to get a file format, as explained in Chapter 1, is to write an EXPRESS schema giving the semantic content of an information model and then to use either the Part 21 rules or the Part 28 rules to obtain a file format from the schema. We will assume in the rest of this chapter that EXPRESS is used to define information models for process plans, and Part 21 is used for writing process plan files. The combination of EXPRESS and Part 21 constitutes a ***process planning language***. If Part 28 or some other format were used for writing process plans according to the same EXPRESS model, that would constitute a different language. It would have the same semantics but different grammar and syntax.

Several characteristics of a process planning language for feature-based manufacturing are desirable. Namely, the language should:

- be machine-readable.
- be machine processable into a serviceable application programming interface (API).
- have a generic core suitable for all types of machining (and other discrete processes).
- be vertically extensible to be suitable for several levels of a hierarchical control system.
- be horizontally extensible so that it is suitable for various types of machines.
- be capable of refinement in stages.

Machine-readable

Since we have assumed the language is modelled using EXPRESS and Part 21, the language is sure to be machine-readable. Several of the STEP systems described in Chapter 1 are able to read EXPRESS and Part 21 files. Many of those that read Part

21 files are able to save a usable representation of what they read and to provide access to the data.

Machine processable into a Serviceable API

By ***serviceable*** we mean that the API produced from a model of the language by an automatic EXPRESS processing system should be readily usable by a programmer who is also a domain expert.

Several of the STEP systems listed in Chapter 1 can read an EXPRESS file and produce an API from it in a language such as C++ or Java. Although these tools can produce an API from any EXPRESS schema, the API may fail to be serviceable for either of two reasons.

First, the data in which an applications programmer is interested may have been atomized so that it is spread across several instances of entities from the STEP integrated resources, and, individually, the instances have very little meaning. This happens when the EXPRESS model being processed is an AIM for which the ARM entities and types do not correspond closely to any entities and types found in the STEP integrated resources. The domains for which this is a problem include almost everything except CAD, for which topology and geometry in AIMs (or AMs) correspond very closely to topology and geometry in Part 42. When an API fails to be serviceable for this reason, it is necessary to buy or write a set of software that implements a serviceable API on top of the correct but unserviceable API produced by an automatic EXPRESS processor.

Second, the ARM model itself may be generic by the virtue of requiring text strings in many places. When this is done, an API produced from the EXPRESS model will often return a string to the application programmer who will then have to produce a routine manually to deal with the string. Such an API is not serviceable.

Generic Core

Many functionalities of a discrete process planning language are essential or desirable regardless of the domain for which the language is designed. These include functions that:

- identify what the plan acts on,
- identify resources used in the plan,
- represent individual plan steps,
- specify the order in which steps may be executed, possibly including loops,
- allow alternative sets of steps to be executed,
- refer to external documents (most especially process plans for the next hierarchical level down),
- provide methods of making decisions in order to choose between alternatives (usually this means having Boolean and numerical expressions),
- provide variables for use in expressions or steps.

For feature-based planning, of course, it is essential that a method of describing features be added to the core.

The generic core should be extended (using EXPRESS!) as needed for defining process planning languages needed for specific purposes. Using the same core for all

types of plans needed in a system will speed system development, minimize the amount of relearning system programmers must do, enable reuse of basic execution routines, and make the overall system more maintainable.

Vertically Extensible

Hierarchical control is a proven method of operating a shop that produces discrete parts. While control hierarchies may be seven or more levels deep, here we consider only three: cell, workstation, and task, as shown in Figure 3.1. The figure shows superiors above their subordinates. Arrows indicate commands. The figure has two subordinates for each superior, but each superior may have any number of subordinates.

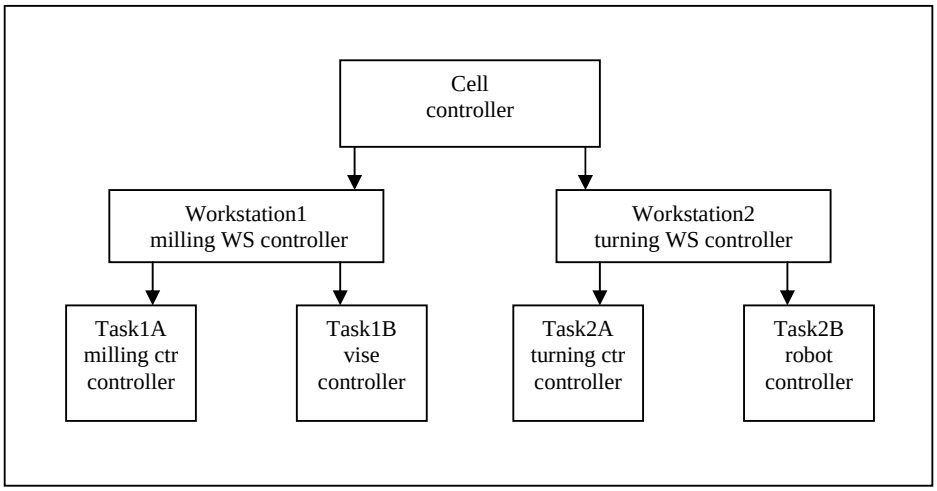

Figure 3.1 Prototypical Controller Hierarchy

The cell level controls a whole shop or a large portion of a shop. A cell has a collection of machines that, among them, can produce a finished part. At the cell level, a process plan is made to produce a finished part from stock or a near net shape such as a casting. A cell process plan says which features to make or inspect on which machines in which fixturings, and in what order the workpiece in process should be sent to the workstations of the cell.

The workstation level is the next level down from the cell level. A workstation is part of a cell and contains a single principal machine, possibly attended by ancillary machines such as powered fixtures, roller tables, or material handling robots. A workstation process plan says which features of the part design or the workpiece are

to be made in a single fixturing. A single step in a feature-based workstation plan for a milling center or turning center identifies the feature to process and the operation to perform on the feature. The step usually indicates a specific tool or type of tool to use and may include feed and speed rates and strategies for using the tool.

The task level is the next level down from the workstation level. A step in a task level process plan for machining typically says what tool path, feed and speed to use. In current practice outside of STEP, an NC programming language such as RS274 [3.4] is used at the task level.

In the hierarchy, each superior controller can give either planning commands or execution commands to its subordinates.

The core process planning language should be capable of being extended vertically to multiple levels of the control hierarchy.

Horizontally Extensible

At the workstation control level, where a single principal machine is used, different languages are needed for different types of workstations (those using a milling center, a turning center, or a CMM, for example). The core process planning language should be capable of being extended by defining specific types of plan steps for each different kind of machine. Since the workstations are all at the same hierarchical level, this is horizontal extension.

Refinable in Stages

Process plans are usually made so that some or most resources that can enter and leave a workstation (such as cutting tools) are specified by type, without distinguishing between instances of things of the same type. When it is time to execute a process plan, specific instances of these resources must be selected and linked to the plan. It is also possible that two workstations are almost the same but have slight differences in their fixed resources or that resources in a single workstation become unavailable (from breakage or maintenance, for example). It is useful, therefore, to make a process plan that includes alternate methods of doing the same thing, where the way the alternatives are done depends on the resources available.

In the operation of a shop, jobs are usually scheduled. In order to control the shop, timing information (such as earliest start and latest completion) may be inserted in the process plan, and actual times may be reported while the plan is being executed.

A process planning language should be such that process plans may be readily refined from (1) the stage at which generic resources and resource-dependent alternatives are specified to (2) the stage at which specific resources are identified but jobs are not scheduled to (3) the stage at which specific resources are identified and a schedule has been made.

Process plans may also be generated containing alternate operations for which the resources are all expected to be available. In order to optimize the plan, it may then be useful to simulate a number of different ways of executing the plan, select

the best alternatives, and discard the other alternatives. Doing this is another way in which a plan may be refined, and process planning languages should support it.

The shelf life of a process plan gets shorter and shorter as the plan becomes more and more refined. Where a plan has been optimized based on conditions that may change, some record of those conditions should be saved with the plan so that at execution time, it can be checked that the conditions still hold.

3.2.2 ISO 14649 and AP 238 Process Planning Languages

ISO 14649 and AP 238 (of ISO 10303) use process planning languages that have the same semantics, but AP 238 adds a level of encoding in order to use the STEP integrated resources. ISO 14649 models the core language in Part 10 [3.9], a language for milling machine process plans in Part 11 [3.10], and a language for turning center process plans in Part 12 [3.11]. Language models for other types of machines are being developed in ISO 14649. AP 238 [3.7] puts everything in one giant model.

Machine processable into a Serviceable API

The EXPRESS schemas for Parts 10 and 11 of ISO 14649 are readily processable into a serviceable API. This was done by the authors for building an ISO 14649 interpreter [3.18] with functionality sufficient to interpret Example 1 given in Annex E of Part 11 of ISO 14649. The API was generated using commercially available STEP tools. Although Part 12 of ISO 14649 was not processed, we believe it could be processed readily and would give a serviceable API.

The EXPRESS schema for AP 238 is automatically processable (because all EXPRESS schemas are automatically processable), but the API that results is not serviceable. In order to get a serviceable API from the automatically generated code resulting from processing AP 238, it is necessary to add a large, complex layer of manually written code on top of the automatically generated code. This is because the EXPRESS schema for AP 238 is an AIM type schema in which (as in almost all AIMs) data from several AIM entities must be recombined in order to obtain data that would be provided directly by an API built from the ARM schema.

The authors built an AP 238 interpreter with the same functionality as the ISO 14649 interpreter. The output of both interpreters was calls to functions implementing canonical machining commands [3.19]. An implementation of the canonical machining commands in which each command just prints itself was compiled into both interpreters. An AP 238 Part 21 file representing the same information as the ISO 14649 Part 21 file for the example was written by hand. When the ISO 14649 file was processed by the ISO 14649 interpreter and the AP 238 file was processed by the AP 238 interpreter, the files of canonical commands produced by the two interpreters were identical. Identical output files were also produced when other pairs of equivalent test files were used.

Generic Core

The core process planning languages defined in Part 10 of ISO 14649 and in AP 238 are almost adequate. They do everything a core language should do except the following.

- The plan structure is innately hierarchical in that an executable may be a workplan, and a workplan includes a list of executables, but no provision was made for the system executing a plan to be part of a hierarchy of controllers. In particular, there is no way to reference a plan for another controller.
- The only kind of expression provided is Boolean expression. Arithmetic expressions are not defined. This omits some important functionalities. For example, although numeric variables are provided, and a "while" loop is defined with a Boolean test for getting out of the loop, there is no way to increase or decrease the value of a variable, so that the results of a test can change. The most common way of running a loop is to increment a counter variable each time around the loop and stop when the counter reaches a particular value. This cannot be done in ISO 14649.

Vertically Extensible

The process planning languages defined in Part 10 of ISO 14649 and in AP 238 are not currently vertically extensible, but they would be if the problems noted immediately above were fixed.

Horizontally Extensible

The process planning languages defined in Part 10 of ISO 14649 and in AP 238 are horizontally extensible and have been extended for milling and turning. The languages were constructed so as to be horizontally extensible.

Refinable in Stages

There is nothing in ISO 14649 that discusses the notion of refinement in stages, but ISO 14649 will support refinement. Chung and Suh [3.2] reported implementing refinement by first generating ISO 14649 process plans containing alternatives whose feasibility depends on machine capabilities and then refining the plan by selecting from among the alternatives. The unrefined plan is suitable for a variety of turning machines. Its structure is called a neutral process sequence graph. The refined plan is suitable for a specific turning machine and is called a hardware-dependent process sequence graph.

The authors of Part 111 of ISO 14649, "Tools for milling machines" [3.12], clearly intend that process plans be refinable by describing tools in a plan only in terms of nominal tool parameters and then selecting specific tools at execution time. This might be implemented either by refining the plan immediately before it is executed or by selecting specific tools as the plan is executed. Part 10 gives each tool a tool_id that is a label and says the tool_id must be unique, but it does not identify the collection among which uniqueness should apply. If the tool_id is

intended to identify a specific tool in a tool inventory, then specifying a tool_id can be the method of going between planning stages. If the tool_id is intended to identify a type of tool in a tool catalog, then it is not useful for going between planning stages.

Workstation Process Plan Contents

The main content of an ISO 14649 or AP 238 workstation level process plan, excluding administrative and flow of control items, consists of instances of the following types of entity:

- workingstep: Each workingstep points to a feature and an operation; the operation makes the feature or partially makes the feature (in the case of roughing operations).
- feature: Each feature should be referenced by at least one workingstep.
- operation: Each operation should be used in at least one workingstep and may be used in several workingsteps. An operation usually has four major components: (1) machine_functions, (2) a tool, (3) a technology, and (4) either a list of toolpaths or one to three strategies (toolpath and strategies are both optional). Most operations also have one or more parameters specific to the type of operation (such as overcut_length for milling).
- machine_functions: coolant, chip removal, axis clamps, etc. – things the machine can do other than turning the spindle and moving the tool relative to the part.
- tool: Each tool should be used in at least one operation.
- technology: Each technology should be used by at least one operation. A technology specifies feed rate, spindle speed, and other motion control items.
- strategy: Each strategy should be used by at least one operation. A strategy specifies the nature of a portion of a toolpath. For example, to start cutting a pocket, the strategy might be (1) pass through air (if a starter hole in the pocket has been made), or (2) plunge straight down into the material, or (3) spiral down into the material.
- toolpath: Each toolpath should be used by at least one operation. A toolpath gives the path the tool should follow. It may also give (1) the feed rate of the tool at each point on the path, (2) machine functions on the path, and (3) technology on the path. The path of the tool may be specified by pointing at explicit geometry or by giving a rule for generating explicit geometry.

The general idea behind having a list of toolpaths for a single operation is that it gives full control over everything the machine can do at each point on the path of the tool while performing the operation. If a toolpath is not specified, the technology and the machine_functions are constant throughout the operation. With a toolpath, they can be varied. Whenever a list of toolpaths is given for an operation and it is given high precedence, there is no use for a strategy, and the operation's machine_functions and technology may be overridden.

Conceptually, toolpaths are at the task hierarchical level and do not belong in a workstation level plan. However, if toolpaths were not provided at the workstation level, in order to allow the user complete control over machining operations (which is definitely needed now and is likely to be needed for the foreseeable future), it

would be necessary to have task level process plans. Such plans would be on the same level as M and G codes traditionally used for machine control programs. It is not clear whether having plans at both the workstation level and the task level would be worth the trouble.

3.2.3 AP 240 process planning language

Machine processable into a Serviceable API

The EXPRESS schema for AP 240 is automatically processable (because all EXPRESS schemas are automatically processable), but the API that results is not serviceable. As with AP 238, this is because AP 240 uses an AIM schema. Also as with AP 238, it is possible to build an API that provides access to ARM type data by writing a large, complex library of functions that extract ARM-like data from AIM data. Unlike AP 238, however, even doing that is not enough to produce an API usable in a specific field. This is because many of the attributes of AP 240 entities are strings, and it is intended that the strings have meaning in a specific field. To use AP 240 in a specific field, an additional document must be written that explains which strings may be used and what they mean. Then, to make a usable API, it is necessary for a programmer to read and understand the document and add a third layer of code that knows what to do when faced with particular values for the strings.

Generic Core

The core process planning language defined in AP 240 has some of the things a core language should have. It can:

- identify what the plan acts on,
- identify resources used in the plan,
- represent individual plan steps,
- specify the order in which steps may be executed. Only sequential execution is provided,
- allow alternative sets of steps to be executed. This is supported in only the following two ways: (1) a process plan may identify a list of alternate process plans, and (2) alternate_activity is defined (though it is not clear how to use it),
- refer to external documents.

The core language defined in AP 240 does not provide:

- expressions of any sort. Thus, no execution time decisions can be made automatically by executing steps specified in an AP 240 plan,
- variables of any type.

Vertically Extensible

AP 240 explicitly identifies the work_cell, workstation, and machine hierarchical levels. The introduction says a process plan is to be “used by programmers to

generate machine tool control programs", which implies that only the workstation level is supported. AP 240 defines enough types of data, however, to support both cell level and workstation level process plans. In most current discussions of AP 240, it is said to be intended to be useful for "macro" process plans, and macro appears to mean the cell level.

Horizontally Extensible

AP 240 is horizontally extensible by using agreed-upon sets of strings in the right places. Extending AP 240 horizontally by EXPRESS subtyping could be done only by ignoring the values of many existing attributes.

Refinable in Stages

It is possible to refine an AP 240 process plan in stages by eliminating alternatives. It may also be possible to refine an AP240 plan by specifying id numbers for resources such as cutting tools. Id numbers, however, are not optional, so to refine by going from specifying a type of tool to specifying a particular instance of a tool, a bogus id number would need to be used in the stage one plan indicating that any tool of the type described will do. Then a real tool id would be used in the refined plan.

AP 240 does not appear to have been constructed with refinement in mind. There is no data element for indicating the stage of a plan.

Workstation Process Plan Contents

It is difficult to determine by studying AP 240 what the contents of a workstation level process plan are intended to be. The standard does not provide any examples of plans at any level, and the authors of this chapter have not been able to find any examples. It appears that a workstation process plan could be built by having data of the sort shown in Figure 3.2, which is a pseudo-Part 21 file snippet. Each entity instance in the figure contains zero, one, or two of the values that would be required in a real Part 21 file, and the names of attributes (which would not appear in a real Part 21 file) are shown in *italics*. The value of each named attribute is separated from the name by an equals sign (which also would not appear in a real Part 21 file). Moreover, the entities in the figure are ARM entities, not AIM entities.

Note that the names of the processes and their parameters are simply strings. No specific processes or process parameters are defined in AP 240. Thus, to make use of an AP 240 workstation level process plan of the sort shown in Figure 3.2, it would be necessary to write a document for the particular type of workstation similar to Part 11 or Part 12 of ISO 14649 explaining what the various terms and values mean. This same problem would arise with an AP 240 process plan for any other hierarchical level.

```
#1 = PROCESS_PLAN_VERSION
        (activities_to_produce_part = (#2, #3));
#2 = MANUFACTURING_PROCESS
        (assigned_feature = #4, assigned_operation = #6);
#3 = MANUFACTURING_PROCESS
        (assigned_feature = #5, assigned_operation = #7);
#4 = MANUFACTURING_PROCESS_FEATURE ( );
#5 = MANUFACTURING_PROCESS_FEATURE ( );
#6 = PROCESS_ACTIVITY
        (process_parameters = (#8), uses_to_perform = #10);
#7 = PROCESS_ACTIVITY
        (process_parameters = (#9), uses_to_perform = #11);
#8 = PROCESS_PROPERTY
        (process_name = 'drill', process_characteristics = (#12, #13));
#9 = PROCESS_PROPERTY
        (process_name = 'countersink', process_characteristics = (#14, #15));
#10 = TOOL_ASSEMBLY ( );
#11 = TOOL_ASSEMBLY ( );
#12 = NUMERIC_PARAMETER
         (parameter_name = 'speed', parameter_value = 2000);
#13 = NUMERIC_PARAMETER
         (parameter_name = 'feed', parameter_value = 507);
#14 = NUMERIC_PARAMETER
         (parameter_name = 'speed', parameter_value = 1254);
#15 = NUMERIC_PARAMETER
         (parameter_name = 'feed', parameter_value = 333);
```

Figure 3.2 AP 240 Workstation Level Process Plan File Snippet

3.2.4 FBICS-ALPS process planning languages

The process planning languages used in FBICS have a modified version of ALPS (A Language for Process Specification [3.1], [3.20]) as their core. The language is called FBICS-ALPS and is modelled in the FBICS_ALPS schema. ALPS itself does not define expressions, but FBICS_ALPS imports a separate EXPRESS schema for expressions written for FBICS. The FBICS_ALPS core has nothing specific to a particular hierarchical level (cell, workstation, or task).

Separate sections of another EXPRESS schema written for FBICS, the FBICS_COMBO schema, define additions to the ALPS core for three specific types of stage 1 plans: for a cell, for a machining workstation, and for an inspection

workstation. Since machining and inspection are both in the same schema, hybrid plans containing both machining and inspection tasks may be written. Since interpreters for the RS274 and DMIS languages (for machining and inspection, respectively) were available to the FBICS project, no attempt was made to replace those languages with ALPS-based EXPRESS schemas. Those languages are used at the FBICS task level.

Machine processable into a Serviceable API

The FBICS process planning languages have all been processed into usable APIs using commercially available STEP tools. The APIs have all been extensively used and tested.

Generic Core

FBICS-ALPS is the generic core of FBICS stage 1 plans for the cell and for inspection and machining (or both) workstations. FBICS-ALPS is entirely generic.

FBICS-ALPS has all of the desirable characteristics listed in Section 3.2.1 plus other functionalities that may be useful in some situations. These include allocating and deallocating resources during execution, and synchronizing the execution of plan steps. These other functionalities are not used in FBICS, however.

The resources used in FBICS are limited to cutting tools and touch trigger probes. Cutters and probes available for planning are given in a Part 21 file tool catalog while those available for execution are given in a Part 21 file tool inventory. An EXPRESS cutting tool catalog model developed at NIST was enhanced for use in FBICS by adding entities for probes and tool instances. The enhanced model is used for the FBICS tool inventory and tool catalog Part 21 files.

Vertically Extensible

FBICS-ALPS was constructed so as to be vertically extensible and is fully vertically extensible. Vertical extensions are made in other schemas by defining sets of subtypes of primitive_task_node.

The version of ALPS with which the FBICS project started was a souped-up version designed for production management. It had one impediment to being extended using EXPRESS. That is, although primitive_task_node had only one attribute (named subtask), the attribute got in the way of gracefully building subtypes of primitive_task_node. The value of subtask was a work_element, an entity that had only a name. If primitive_task_node had been used as a supertype, the subtask attribute would have become useless.

In the FBICS_ALPS schema, the subtask attribute of primitive_task_node has been removed. It is intended and implemented that sets of subtypes of primitive_task_node should be defined in EXPRESS schemas that extend the core language. Each subtype of primitive_task_node (defined in other schemas) adds whatever attributes it needs.

Horizontally Extensible

FBICS-ALPS was constructed so as to be horizontally extensible and is fully horizontally extensible. Horizontal extensions, like vertical extensions, are made in other schemas by defining sets of subtypes of primitive_task_node. Indeed, FBICS grew horizontally from its original implementation, FBCS, which covered only machining, to include inspection.

Refinable in Stages

The production management version of ALPS with which the FBICS project started was carefully designed to support three stages, which it called process plan, production-managed plan, and production plan. This was simplified in FBICS-ALPS to only two stages. A stage one plan is called a process plan, has alternative execution paths, and has various different types of steps. A stage two plan is called an operation plan, has only one linear execution path, and has only one kind of step, which is called one_operation. A one_operation has two attributes: an operation type identifier and the name of a file to be used at next hierarchical level down that will carry out the operation. The simplicity of stage two plans made it very easy to implement building and executing them, but it introduced yet another information model that an application programmer would need to understand. It would probably have been a better idea to use the same format for stage two plans as for stage one plans, even though that would have required more code for generating and traversing stage two plans.

Workstation Process Plan Contents

The main content of a FBICS stage one workstation level process plan, excluding administrative and flow of control items, consists of instances of operations. Each operation specifies the index in the associated features file of the feature to operate on, the tool catalog name of a tool to use, the feed rate, the spindle speed, the coolant use, and values of parameters associated with the specific kind of operation (such as pass depth for a peck drilling operation).

A FBICS stage one workstation level process plan points to a number of other files whose data is implicitly included. These other files include:

- the feature-based design of the workpiece before the plan is executed.
- the feature-based design of the workpiece as it should be after the plan is executed.
- the feature-based design of the fixture used to hold the workpiece while the plan is executed.
- the features referenced in the plan.
- a setup file giving the locations in machine coordinates of the four previous items.

Each FBICS workstation level process plan is generated using a tool catalog that is selected when FBICS is initialized. Since a stage one plan may be executed or refined long after it is generated, the name of the tool catalog used by a plan really should be added to the plan.

As mentioned in the previous subsection, stage two FBICS plans consist of ordered lists of one_operations, each of which identifies a type of operation and names the file a subordinate should use to carry out the operation. In the case of the workstation level, the subordinate's file is either a DMIS file for inspection [3.3] or an RS274 file for machining [3.15].

3.2.5 Summary Table

Table 3.1 gives a summary of the extent to which each of the four languages just described has the desirable characteristics of a process planning language.

Language → ***Characteristics*** ↓	ISO 14649	AP 238	AP 240	FBICS-ALPS
machine-readable	yes	yes	yes	yes
machine-processable into serviceable API	yes	no	doubly no	yes
generic core	almost	almost	almost	yes
vertically extensible	almost	almost	yes	yes
horizontally extensible	yes	yes	somewhat	yes
refinable in stages	somewhat	somewhat	somewhat	yes

Table 3.1. Characteristics of Four Process Planning Languages Modeled in EXPRESS

3.3 Features

Across all domains of discourse, the word ***feature*** has many different meanings. In this chapter a feature is a type of shape (such as a pocket or hole) associated with a process plan or with the design of a piece part. In FBICS, the definition of a feature is narrowed so that a feature must be a closed volume in space.

A discussion of the issues to be considered in building a suite of features for material removal may be found in [3.13]. One of the issues is how a feature relates to a machining operation acting on the feature. In many systems, the relation is not stated explicitly. The relation used in FBICS is this: when the operation is finished, all of the material in the feature must be removed, and the operation may not remove any material outside of the feature.

Many feature models intended to be useful for machining have been proposed. In this section we deal with only those features defined in AP 224 of STEP and Parts 10 and 12 of ISO 14649. FBICS uses AP 224. Part 11 of ISO 14649 (for milling) uses only the features defined in Part 10 of ISO 14649. Part 10 defines general features plus specialized feature types for milling but not for turning. Part 12 of ISO 14649 (for turning) defines and uses turning features. These are subtypes of general features defined in Part 10. AP 238 uses all the features defined in Parts 10 and 12 of ISO 14649, but some of them are modified so as to be more like AP 224 features.

The reasons that stereotyped features are useful are:

- Humans think in terms of stereotyped features.
- Automatic recognition of stereotyped features from a boundary representation may be feasible.
- Sterotyped operations may usually be defined for cutting stereotyped features.
- Automatic process planning for parts with stereotyped features may be feasible.
- Automatic toolpath generation may be feasible for stereotyped features.

Part 224 features and ISO 14649 features are similar in shape but not identical. The EXPRESS supertype-subtype hierarchies are also similar but not identical. Both define features in a native coordinate system and locate each feature by locating the native coordinate system of the feature in a setup or part coordinate system. Both have all the features listed in Figure 3.3, but not by the names shown below or in the hierarchical arrangement shown below. In particular, AP 224 does not separate features into milling features and turning features.

Other similarities between AP 224 features and ISO 14649 features include:

- AP 224 and ISO 14649 both allow tolerances to be applied to parameters representing distances and angles.
- AP 224 features and ISO 14649 features both have a lot of ambiguities that should be fixed. For example, in both standards, a description of the shape of a chamfer is given only for the case where the chamfer is made between two planes, but a chamfer may be applied between any two faces.

AP 224 features and ISO 14649 features have some fundamental differences.

- AP224 allows protrusions and bosses that add material to a base shape. ISO 14649 has no protrusions and allows bosses only as material inside a step, pocket, or planarFace that should not be milled away when the rest of the feature is milled away.
- AP224 allows fillet as a transition feature. Fillets are not a type of transition feature in ISO 14649 (probably because fillets, like protrusions, add material to the part). ISO 14649 allows fillet shapes on the interior of features by providing for bottom corner and side corner radii. Curiously, neither provides for a flat fillet (which is easily machined at the bottom of a feature with the right type of endmill).
- ISO 14649 has toolpath feature, which is a dummy feature with no geometry that may be associated with an operation for which the toolpath is specified.

.

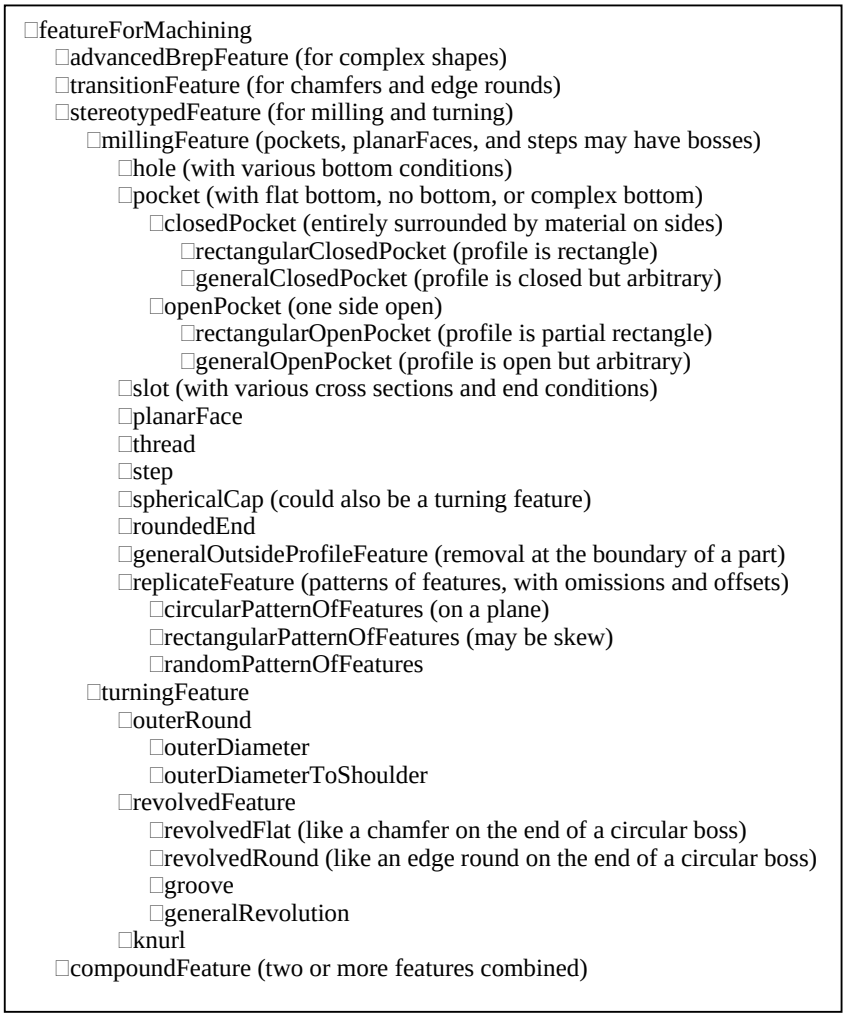

Figure 3.3. Features in AP 224 and ISO 14649

Figure 3.4 shows a part with a variety of stereotyped features.

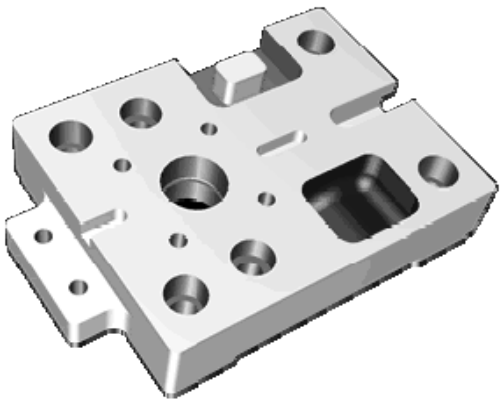

Figure 3.4. Some Stereotypical Features: holes, pockets, slots, planar faces, steps, bosses and freeform surfaces.

3.4 Feature-Based Process Planning

3.4.1 Overview of Feature-Based Process Planning

Process planning is the activity of creating or refining a process plan (or a set of process plans). Here we focus on process planning for machining a part using machining features. Process planning may be done automatically, manually, or partially automatically and partially manually. In current industrial practice, most feature-based process planning is primarily manual at the cell and workstation levels but primarily automated using a computer-aided machining (CAM) system at the task level (where toolpaths are created).

3.4.2 Features and Process Planning

The sequence of activities for making a part of a given design starts with recognizing machining features from the design. Part designs are usually given as boundary representations, but may be given as design features (which may not be usable as machining features), or via constructive solid geometry. Feature recognition may be entirely automatic, done manually with a CAM system, or partly automatic and partly manual. Describing a part entirely in terms of features is most feasible when the features are either rotational (i.e. of the sort that can be made on a turning center) or have one or a few characteristic directions (i.e. of the sort that can be made on a 3-axis machining center in one or a few fixturings).

In some descriptions of feature-based process planning, it is assumed that once features have been identified, they cannot be changed during process planning. This is not a practical approach. It is common to start process planning and realize that a

better plan can be built by changing the features. In an industrially usable system, it must be possible to change features during the construction of a process plan.

- If one of the features is the outside boundary of the part, it may be obstructed by clamps. In this case, the closed profile of the outside boundary may be divided into two or more open profiles, each of which is part of the boundary. The clamps can be moved in between cutting the two open profiles.
- If a large pocket with a complex profile is to be machined, and there is a narrow gap in the pocket through which only a small diameter cutter can pass, then it will be best to separate the pocket into three parts: two large pockets (one on either side of the gap) and one small pocket that passes through the gap. The large pockets can be cut efficiently with a large diameter tool while the gap pocket can be cut with a small diameter tool.
- If the profile of a pocket has concave corners with small radii, it may be useful to define a similar pocket with larger radii in the concave corners and add small pockets that fit into the corners with small radii. Then the large pocket may be machined with a large tool, and the small pockets may be machined with small tools.
- If two pockets intersect and their bottoms are on the same plane, it may be desirable to combine them into a single pocket with a more complex profile.

3.4.3 Feature-Based Process Planning in FBICS

FBICS implements automatic two-stage hierarchical feature-based process planning starting from a design described in a STEP Part 21 file based on the AP 224 ARM EXPRESS model. FBICS plans for 3-axis machining. Only the round_hole, counterbore_hole, and rectangular_pocket AP 224 features are implemented in FBICS. The equivalent of an AP 224 step, however, may be made in FBICS by using a rectangular_pocket to remove the same volume as the step. FBICS makes plans for the cell level, the workstation level, and the task level, with only one controller at each level. Each of the three controllers is in a separate computer process. An interprocess communications system is used to connect the controllers with each other and with other FBICS processes. At the cell and workstation levels, stage one plans are built in FBICS-ALPS with operation types appropriate to the level and the capabilities of the equipment.

The architecture of FBICS as implemented is shown in Figure 3.5.

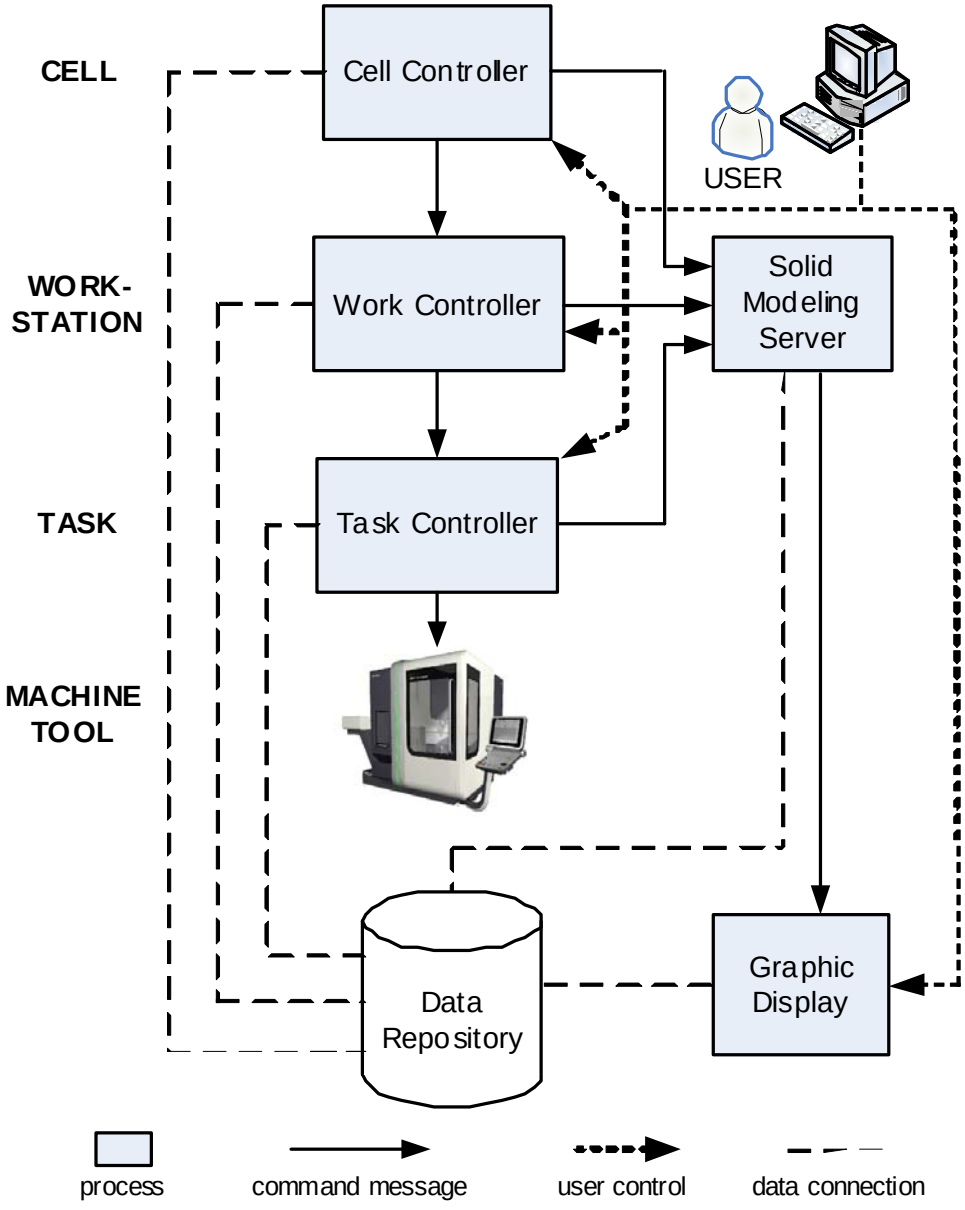

Figure 3.5 FBICS Architecture

FBICS can plan for machining alone, machining with intermittent on-machine inspection, or inspection alone. How it does each of these is described in detail in [3.17]. In this chapter, only FBICS planning for machining is described.

FBICS uses a solid modeller in its own process. Each of the three controllers can use the modeller. A set of FBICS interface functions for the modeller was written to make using it independent of the underlying commercial solid modeller.

FBICS has a user interface to each of the three controllers. Each user interface can transmit a command from the user to make a plan or to execute a plan. The most complex activities occur when the user commands the cell to make a plan and then commands the cell to execute the plan, so that is described here. FBICS can plan automatically off-line for 1, 2, or all 3 hierarchical levels. In the example that follows, we will assume the cell has been commanded to plan 2 levels deep.

User Gives Planning Command

When the user gives a planning command to the cell, the user specifies:

- the name of a Part 21 file containing the AP 224 design of the part. This file describes a block and the features that must be removed from the block to make the final intended shape.
- whether a file giving the shape of the incoming workpiece exists. If the file exists, FBICS will read it. If the file does not exist, FBICS will write it; in this case the workpiece design will be the block shape given in the AP 224 design of the part.
- the name the workpiece file to read or write (according to whether it exists or not). This is also an AP 224 Part 21 file.
- the base name to use for the Part 21 plan files FBICS will write.
- the base name to use for the Part 21 AP 224 feature files FBICS will write.
- the base name to use for the Part 21 setup files FBICS will write.
- the number of hierarchical levels for which FBICS should plan (2 in this example; 1 and 3 are the alternatives).

When FBICS executes, it generates a large number of files (often in the hundreds). Many of these files refer to other files, so it is necessary for FBICS to have an elaborate system of generating file names. To simplify this discussion of how FBICS works, in the remainder of this section, file names will be ignored.

Cell Makes Cell Level Stage One Plan

When the cell gets the user command, the cell reads in the design of the part and calls on the modeller to make a solid model of the part. The solid model is made by creating a solid from each of the features in the design, creating a solid from the block in the design, and boolean subtracting all the features from the block.

The cell tells the modeller to make a solid model of the workpiece, which is done the same way as making a model of the design. In order to reason about how to make the part and to produce a graphical display of what it is thinking about, the cell

tells the modeller to make a copy of the workpiece to use for the part-in-progress. The part-in-progress model changes as planning proceeds.

With the help of the modeller, the cell generates a lot of information about each feature in the design.

The cell checks whether each of the design features intersects the workpiece. Those design features that do not intersect the workpiece do not need to be machined. The other design features are put on a list of features to make.

The cell checks that the designed part is contained within the workpiece. If not, that means material needs to be added to the workpiece in order to make the part. Adding material is not possible by machining, so planning stops if the part is not contained in the workpiece.

The cell then determines what setups of the part-in-progress are needed and which features should be machined in each setup. To do this, the set of features to be made is divided into subsets called direction_sets. The features in a direction_set all have their Z axes pointing in the same direction (so that they are all machinable from the same direction). The features in a direction_set cannot necessarily all be made in the same fixturing, since other features may be blocking access to them. Hence, each direction_set is separated into two subsets, makeables and unmakeables. The makeables are initially those features that are not blocked in the Z direction by any other features (i.e., the makeables are those features in the direction_set that are immediately accessible for machining on the workpiece). The unmakeables are the rest of the features in the direction_set. After the initial assignment of makeables and unmakeables, any feature in the unmakeables that is blocked only by features in the makeables is transferred to the makeables (since it will be makeable in the same fixturing after the features that block access to it are made). The features in the unmakeables are all repeatedly checked until no more features can be transferred.

Setup_sets (sets of features that can be made in a single setup) and a partial ordering for making the setup_sets are then constructed concurrently by the following iterative procedure.

1. If one or more direction_sets has no unmakeables, those direction_sets are used as a group of setup_sets representing setups that can be made in any order. Otherwise, if one or more direction_sets has non-empty makeables, those makeables are used as a group of setup_sets representing setups that can be made in any order. The features in the setup_sets that have been identified are considered to have been made, and the part-in-progress is updated by boolean subtraction of those features.
2. The detailed information about all the features that have not yet been made is recomputed. Any direction_set with no remaining features is deleted from the direction_sets. The makeables and unmakeables of each direction_set are recomputed.

The two steps above are repeated until all features have been considered to have been made. This produces a total ordering of groups of setup_sets in which the setup_sets of each group may be made in any order. This ordering is captured in a cell-level stage one FBICS-ALPS process plan. The steps in the plan are all run_setup steps, one for each setup_set. Each step gives the name of a setup file.

For each setup, a features file is generated and saved, a fixture is selected to hold the part during the setup, and the names of the files describing the features and the fixture are inserted in the setup file. In FBICS, the selection of fixtures is naive – really just a placeholder for more intelligent fixture selection routines yet to be developed.

If it is possible to execute setups in more than one order, for some of the setups, it will not be known what the shapes of the incoming workpiece and outgoing workpiece are because it is not known what machining will have been done previously. Thus, the setup files in stage one plans refer to incoming and outgoing workpiece designs that do not exist. When a stage one plan is executed or refined, however, when any setup is about to be run, it will be known what the shapes are, and files describing the shapes will be available.

It is possible that the design has features that are not accessible. In this case planning fails. If planning seems to have succeeded, the modeller is used to check that the part-in-progress is now the same shape as the design.

Cell Makes Stage Two Plan

The cell has now made a stage one plan. It could execute this plan directly, but it has been commanded to plan two levels deep. The cell could traverse its stage one plan in every legal order and tell the workstation to make a plan for each possible traversal of the cell's stage one plan, but that may cause a combinatorial explosion of plans. One of the test parts used in FBICS required seven setups, and they could be done in any order. There would be 7! (= 5040) different ways to do that. To optimize the set of plans being generated, it would probably be useful to explore some of the different ways in which the stage one plan could be executed, but that has not yet been implemented. What has been implemented is to pick one legal execution order arbitrarily and use it.

The stage two cell plan will be linearly ordered and will consist of run_setup operations, each of which names a workstation level process plan that the workstation should execute. Since the cell has been commanded to plan two levels deep, the workstation level plans named in the cell stage two plan will be stage one plans. If the cell had been commanded to plan three levels deep, the workstation plans would be stage two plans.

Now that the cell has fully determined the order in which setups will be run, the shape of the workpiece as it leaves each setup and enters the next is found, and Part 21 AP 224 files describing these shape are saved. The setup files are modified by inserting the names of these intermediate shape files.

Cell Tells Workstation to Plan

Now the cell traverses the stage two plan it just made. For each of the setups it has decided to use, the cell gives a command to the workstation to plan for that setup. Each command contains only the name of the setup file to read, the name of the plan to write, and the number of levels deep to plan. In the scenario we are describing, each command tells the workstation to plan one level deep since the cell was commanded to plan two levels deep.

Workstation Makes Stage One Plans

Upon receiving a planning command from the cell, the workstation reads files for the setup, the features to make, the fixture, the incoming workpiece, and the outgoing workpiece. It then calls on the modeller to make solid models of the features, fixture, and workpieces in the positions given in the setup file. A part-in-progress is built by copying the incoming workpiece. A check is made that the outgoing workpiece is contained in the incoming workpiece.

The workstation knows that all the features in the setup can be made, but it does not know in what order they can be made. Therefore, the workstation goes through the same kind of accessibility analysis the cell performed in dealing with the makeables and unmakeables of a setup-set, but the workstation forms a partial ordering of the features as it does so.

For each feature to be machined, the workstation selects a cutting operation to make the feature and a type of tool to cut with. Values are selected for tool use parameters (feed, speed, etc.) by using tool usage rules. These rules are loaded when FBICS is initialised from a file prepared by the user. Thus, the user has local control over how tool use parameters are found. The tool usage rules use variables and may contain logical and arithmetic expressions. The variables may be any of ten variables whose names are hard-coded in FBICS. They include, for example: “material” (meaning the material the workpiece is made from) and “diameter” (meaning the diameter of the tool). The values of the hard-coded variables are set automatically in various ways. For example, the value of material is set when the workpiece file is read. Allowed values of non-numerical variables such as material must be given in the tool usage rules file.

The workstation writes a stage one process plan containing the operations that have been planned with the partial ordering of operations that has been determined.

That completes the planning commanded by the user.

Cell Executes Stage Two Plan

Now that planning is finished, the user gives the cell a command to execute a stage two plan. The command from the user gives only the name of the cell stage two plan to execute. To execute the cell plan (the gist of which is to run each of a list of workstation stage one plans), for each workstation plan, the cell sends the workstation a command to run the named plan. After sending each command, the cell waits until the workstation says it is done before sending the next command.

Workstation Executes Stage One Plans

Each time the workstation gets a command from the cell to execute a stage one plan, the workstation reads the plan and the files named in the plan (setup, features, etc.) and commands the task controller to open the setup also. Then the workstation repeatedly decides what plan step to execute next, decides what operations are necessary to carry out the plan step, and, for each operation, writes a file describing the operation in detail, commands the task controller to write a segment of NC code to carry out the operation described in the file, and (after the task controller reports

that it has written the code) commands the task controller to execute the segment of code it has just written.

Some of the workstation stage one plan steps may require more than one operation. For example, steps for changing the tool and turning coolant on and off are not included in a workstation stage one plan because they may or may not be needed, according to the order in which the cutting steps are executed. But changing the tool and turning coolant on and off must be done from time to time, so for each plan step it executes, the workstation inserts these operations if they are needed.

When traversal of the workstation stage one plan is finished and all required operations have been carried out, the workstation closes its view of the setup and commands the task controller to close its view of the setup.

Task Controller Writes Code Segment

Each time the workstation commands the task controller to write a segment of NC code to carry out an operation, the task controller reads the file that the workstation wrote describing the operation and calls a code generating function that knows how to generate code for the particular kind of operation. The code is placed in a file.

Task Controller Executes Code Segment

Interpreters for both the RS274NGC language [3.15] and the DMIS language [3.14] are built into the FBICS task controller. Each time the workstation commands the task controller to execute a segment of NC code (contained in a named file), the task controller calls on the appropriate interpreter to interpret the code. Interpreting the code produces calls to canonical machining functions or canonical inspection functions, which are then executed to do the required cutting or inspection.

3.5 FBICS to ISO 14649

FBICS workstation level stage one machining plans are conceptually very similar to ISO 14649 plans for milling. They are similar enough that it has been possible to build a translator for Part 21 files. The translator reads a Part 21 file based on the FBICS_ALPS and FBICS_COMBO schemas and writes a Part 21 file based on the MILLING_SCHEMA and MACHINING_SCHEMA of ISO 14649. The input files may be generated automatically by FBICS or they may be generated any other way, as long as only those entities and types are used that could be in a schema generated by FBICS.

FBICS stage one plan files are shorter than ISO 14649 plan files because FBICS plan files:

- refer to external setup files,
- include the data from an ISO 14649 operation and technology in a FBICS operation, and
- refer to tools by tool type id.

Hence, to make an ISO plan file from a FBICS plan file, in addition to handling two plan files, the translator must:

- read the FBICS setup file and use it to build a setup entity in the 14649 plan file,
- create a 14649 operation and a technology from each FBICS operation, and
- have on hand a tool catalog data base and a tool inventory data base, which when combined with the tool type id from the FBICS plan, will enable the construction of a 14649 tool from each tool type id mentioned in the FBICS plan.

Other reformatting is done, but it is more straightforward than the details just mentioned.

3.6 Conclusion

This chapter has identified the desirable features of a process planning language: be machine-readable, be machine-processable into a serviceable API, have a generic core, be vertically extensible, be horizontally extensible, and be capable of refinement in stages.

Four languages for writing feature-based process plans have been developed using the STEP methodology of an EXPRESS schema plus Part 21. Two of these (AP 238 and AP 240) are Parts of STEP (ISO 10303). The third (ISO 14649) is an ISO standard. The fourth (FBICS-ALPS) is not a standard.

The AP 238 and AP 240 process planning languages are difficult to use because their EXPRESS schemas are AIM type schemas, making it necessary to build complex software that decodes semantics. AP 240 suffers from the additional liability of making extensive use of strings, so that interoperability will be possible in a particular application domain only if documents are written specifying what strings may be used in the domain and what they mean.

The ISO 14649 process planning languages are usable now for workstation level plans for machining and turning, but ISO 14649 has not been built to support hierarchical control.

The architecture of FBICS and the approach to feature-based process planning implemented in FBICS provide the kernel of an industrial strength automated machining and inspection system, but many improvements are needed to make FBICS workable in an industrial setting, as follows.

- Currently, FBICS handles only a few feature types. The FBICS methods need to be extended to cover a fuller set of features. This should be straightforward, but a large amount of code of various types is needed for each type of feature.
- FBICS does not include making an analysis of whether machining features that have been recognized should be subdivided or combined. This kind of analysis is an essential part of cost-effective process planning and should be added.
- FBICS does not include an analysis of how a part is to be located and held while it is being machined, and it does not include process plan steps for moving clamps in a single setup. It also does not have a method of planning for clamps and locating surfaces automatically. Those items are needed.

- FBICS does not optimize the plans that it makes. Where there are not many ways in which plan steps can be executed, it would be feasible to find the cost of every possible way to execute a set of plans and pick the cheapest, but combinatorial explosion is quite likely (there are N! orders in which to drill an array of N holes, for example). To optimize effectively, the scenarios in which explosions are likely would have to be identified and heuristics developed for each scenario.
- FBICS has a fancy method (not described here) of using tolerances on parameters of pockets to determine when to do in-process inspection. FBICS uses the results of the inspection to modify the tool path used for final finish milling. FBICS does not make any use of geometric tolerances, however. The use of tolerances in planning for machining must be expanded. For example, if a hole that has been drilled has a tight location tolerance, it might be planned to center drill before drilling. Or, if the hole has a tight cylindricity tolerance, it might be planned to ream the hole after drilling.
- FBICS currently does not make particularly efficient plans. For example, FBICS does not plan to do rough milling. The planning methods should be changed to make more efficient plans.
- Currently, FBICS planning rules are hard-coded. The planning rules should be changed to be in loadable files, so that each shop could use the planning rules it prefers.
- Humans are much more flexible and innovative in problem solving than any automated process planning system is now or will be for the foreseeable future. An effective system must allow for human intervention at each major stage of process planning. To provide this opportunity, FBICS includes many interfaces where files are used for data. For each of these data interfaces, a friendly, graphical user interface is needed so that a human can intervene in the planning process easily and intuitively.

References

[3.1] Catron, B., Ray, S.R. 1991. *ALPS – A language for process specification*, International Journal of Computer-Integrated Manufacturing, Vol 4, No 2, 105-113.

[3.2] Chung, D-H., Suh, S-H. 2008. *ISO 14649-based nonlinear process planning implementation for complex machining*, Computer-Aided Design 40, 521-536.

[3.3] Dimensional Measurement Standards Consortium. 2007. *Dimensional Measurement Interface Standard 5.1*, ANSI/DMIS 105.1-2007, Part 1.

[3.4] Electronic Industries Association. 1979. *Interchangeable Variable Block Data Format for Positioning, Contouring, and Contouring/Positioning Numerically Controlled Machines*, EIA Standard EIA-274-D.

[3.5] ISO 10303-11. 2004. *Industrial automation systems and integration – Product data representation and exchange – Part 11: Description methods: The EXPRESS language reference manual*, Geneva, Switzerland: International Organisation for Standardisation (ISO).

[3.6] ISO 10303-224. 2006. *Industrial automation systems and integration – Product data representation and exchange – Part 224: Application protocol: Mechanical product definition for process planning using machining features*, Geneva, Switzerland: International Organisation for Standardisation (ISO).

[3.7] ISO 10303-238. 2007. *Industrial automation systems and integration – Product data representation and exchange – Part 238: Application protocol: Application interpreted model for computerized numerical controllers*, Geneva, Switzerland: International Organisation for Standardisation (ISO).

[3.8] ISO 10303-240. 2005. *Industrial automation systems and integration – Product data representation and exchange – Part 240: Application protocol: Process plans for machined products*, Geneva, Switzerland: International Organisation for Standardisation (ISO).

[3.9] ISO 14649-10. 2004. *Industrial automation systems and integration – Physical device control – Data model for computerized numerical controllers - Part 10: General process data*, Geneva, Switzerland: International Organisation for Standardisation (ISO).

[3.10] ISO 14649-11. 2004. *Industrial automation systems and integration – Physical device control – Data model for computerized numerical controllers - Part 11: Process data for milling*, Geneva, Switzerland: International Organisation for Standardisation (ISO).

[3.11] ISO 14649-12. 2005. *Industrial automation systems and integration – Physical device control – Data model for computerized numerical controllers - Part 12: Process data for turning*, Geneva, Switzerland: International Organisation for Standardisation (ISO).

[3.12] ISO 14649-111 (FDIS). 2004. *Industrial automation systems and integration – Physical device control – Data model for computerized numerical controllers - Part 111: Tools for milling machines*, Geneva, Switzerland: International Organisation for Standardisation (ISO).

[3.13] Kramer, T.R. 1994. *Issues Concerning Material Removal Shape Element Volumes (MRSEVs)*, International Journal of Computer Integrated Manufacturing, Vol. 7, No. 3, 139-151.

[3.14] Kramer, T.R., Proctor, F.M., Rippey, W.G., Scott, H. 1998. *The NIST DMIS Interpreter – Version 2*, NISTIR 6252, National Institute of Standards and Technology.

[3.15] Kramer, T.R., Proctor, F.M., Messina, E. 2000. *The NISTRS274NGC Interpreter – Version 3*, NISTIR 6556, National Institute of Standards and Technology.

[3.16] Kramer, T.R., et al. 2001. *A Feature-based inspection and machining system*, Computer-Aided Design 33, 653-669.

[3.17] Kramer, T.R., et al. 2004. *Feature-based Inspection and ControlSystem*, NISTIR 7098, National Institute of Standards and Technology.

[3.18] Kramer, T.R., Proctor, F., Xu, X., Michaloski, J.L. 2006. *Run-time Interpretation of STEP-NC: Implementation and Performance*, International Journal of Computer Integrated Manufacturing, Vol. 19, No. 6, 495-507.

[3.19] Proctor, F., Kramer, T.R., Michaloski, J.L. 1997. Canonical Machining Commands, NISTIR 5970, National Institute of Standards and Technology.

[3.20] Wallace, S., et al.. 1993. *Control Entity Interface Specification*, NISTIR 5272, National Institute of Standards and Technology.