Computer-Aided Design 33 (2001) 653–669

COMPUTER-AIDED DESIGN

www.elsevier.com/locate/cad

A feature-based inspection and machining system

T.R. Kramer*, H. Huang, E. Messina, F.M. Proctor, H. Scott

Intelligent Systems Division, MS8230, National Institute of Standards and Technology, Gaithersburg, MD 20899, USA

Received 1 November 1999; revised 16 September 2000; accepted 15 October 2000

Abstract

This paper[1] describes an architecture for a system for machining and inspecting mechanical piece parts and an implementation of it called the Feature-Based Inspection and Control System (FBICS). In FBICS, the controller of a machining center or coordinate measuring machine uses a standard feature-based description of the shape of the object to be made as a principal input for machining and/or inspection. FBICS is a hierarchical control system and performs automated hierarchical process planning. FBICS serves: (1) to demonstrate feature-based inspection and control in an open-architecture control system; (2) as a testbed for solving problems in feature-based manufacturing; and (3) to test the usability of STEP methods and models. © 2001 Published by Elsevier Science Ltd.

Keywords: Inspection; NC milling; Process planning; Form features

1. Introduction

This paper describes an architecture for a system for machining and inspecting mechanical piece parts, and an implementation of it developed at the National Institute of Standards and Technology (NIST). The system is called the Feature-Based Inspection and Control System (FBICS). This paper covers all aspects of FBICS. Other papers and reports on FBICS include: a much deeper overall description of FBICS [1]; a brief overview of FBICS [2]; a view of FBICS as used in an inspection workstation [3]; and a discussion of process planning in FBICS [4].

1.1. FBICS summarized

While the architecture is quite general, FBICS as implemented controls a machining center or coordinate measuring machine (CMM). FBICS uses a feature-based description of the shape of the object to be made or measured as a principal input for machining and/or inspection. As used in FBICS, a feature is a volume whose shape is an instance of some member of a predefined set of shape types—a hole or a pocket, for example. The predefined set is an international standard which is part of STEP, the Standard for the Exchange of Product Model Data.

The primary purposes of FBICS are:

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

FBICS exists both as (1) a stand-alone system using minimally functional controllers but fully functional planners, with simulated inspection or machining, and (2) as part of three loosely integrated systems using the same planners but more fully functional controllers with graphically simulated inspection, actual inspection, and actual machining. Software for all versions of FBICS is in the C++ language. Machines run using FBICS include a 3-axis 'mini-mill' machining center, a Bridgeport 3-axis machining center, a hexapod machining center, and a Cordax coordinate measuring machine.

The two principal capabilities of FBICS are: to generate process plans automatically at each level of a control hierarchy; and to execute the plans in order to make and/or inspect piece parts.

* Corresponding author. Tel.: +1-301-975-3518; fax: +1-301-990-9688. *E-mail address:* kramer@cme.nist.gov (T.R. Kramer).

[1] Certain commercial products are identified in this paper to describe our study adequately. Such identification is not intended to imply recommendation or endorsement by NIST, neither is it intended to imply that the products identified are necessarily the best available for the purpose.

1.2. Approach to machining and inspecting mechanical piece parts

We focus here on piece part production in small batches using commercial off-the-shelf machines, as in a typical job shop (but we would put FBICS controllers on the machines). Work activity falls in two large sections: design and manufacturing. Manufacturing activity may be divided into manufacturing planning and manufacturing execution. Planning includes task planning, resources planning, and scheduling. Execution includes resource allocation and task execution. FBICS deals with (1) manufacturing task planning, and (2) manufacturing task execution where fabrication is done by machining and inspection is done either on a machining center or on a coordinate measuring machine.

In many job shops, task planning is divided between two groups. The first group (called process planning) usually does high-level planning—which portions of the shop should do which portions of the fabrication. The second group (called programming) usually does generation of machine control programs for cutting and inspection (a form of planning). FBICS also divides task planning, but into three levels rather than two. FBICS aims for more automated operation, improved use of feedback, and tighter integration, as compared with typical job shops.

High-level process plans used in industry are generally sequential process plans. Each plan is a simple list of steps, all steps must be performed, and they must be done in the order they appear on the list. Such plans are not usable by computers and do not allow for alternative, parameters, or using current data during plan execution. High-level plans in FBICS are tree-like plans using alternatives, parameters, and current data. They are two-stage plans that may be executed by computers.

Integration of subsystems is a major problem in manufacturing, generally, and in piece-part manufacturing, particularly. It would be desirable to be able to compose manufacturing systems using as subsystems entire commercial systems or modules of commercial systems. Composing systems this way using subsystems from different companies is often not possible because (1) commercial systems are usually not built to be compatible with systems from other sources upstream, downstream, or laterally, (2) when commercial systems are modular, the interfaces between their modules are usually not open, so that modules must be bought from a single vendor or cooperating vendors, (3) even when commercial systems are modular and open, the data formats used at the interfaces are not standard, so only modules that support several formats are likely to be widely usable, and (4) the behavior of each module is often not fully specified. In addition, the algorithms underlying the behavior of each module are usually not disclosed. This does not prevent the use of modules, but knowledge of the underlying algorithms would facilitate their use by allowing the user to predict behavior and performance.

Most users would prefer to be able to mix and match subsystems, as one does with audio systems. This is possible only if there are open, standardized interfaces allowing the subsystems to work together.

1.3. Needs FBICS seeks to meet

FBICS seeks to provide the following desirable characteristics for piece-part manufacturing systems in an open-architecture environment.

1. Tight integration of separate modules.
2. Agreed-on module definition, at the right granularity and with open interfaces.
3. Easily constructed feedback loops.
4. Full utilization of the abilities of both computers and humans.
5. Full use of available data.
6. Use of standard data representations and modeling languages.
7. Effective use of off-line and on-line planning.

2. FBICS background

2.1. Real-time control system (RCS) architecture

For many years, the Intelligent Systems Division (ISD) at NIST has been developing a control architecture known as the Real-time Control System (RCS) architecture. Under the tenets of RCS [5–7], 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 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 FBICS conforms to the RCS architecture. In RCS, each controller includes components that perform sensory processing, world modeling, planning, job assignment, and execution. The emphasis in FBICS development has been on planning and world modeling.

2.2. Machine control languages

Standard machine control languages exist: RS274 for machining [8,9]; and Dimensional Measuring Interface Standard (DMIS) for inspection [10]. Interpreters for these languages were developed at NIST independently of the FBICS project [11,12]. Since the languages are standard and interpreters were available, these two languages have been used in FBICS.

2.3. Architecture and knowledge engineering projects

The Intelligent Systems Division has a continuing project in RCS architecture development. This project has provided much of the support for the development of FBICS. The architecture project has provided support, as well, for the development of RCS controller templates. Controllers built using the templates have been used both in a version of FBICS in which planning functions are integrated with simulated inspection and actual machining and in a version of FBICS used to control a CMM, part of an integrated inspection workstation.

The Knowledge Engineering Program is studying world modeling and knowledge representation issues in intelligent control. Jointly with the Architecture Project, it developed an integrated inspection workstation [3] which incorporated FBICS.

2.4. Enhanced machine controller project

The objective of the NIST Enhanced Machine Controller (EMC) project, conducted in ISD, is to build a testbed for evaluating application programming interfaces (APIs) for open-architecture machine controllers. FBICS has been supported by the EMC project and is being used as a component 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 [13]. That controller and variants of it are called 'the EMC controller.' EMC controllers have been installed on several machining centers in commercial machine shops. The EMC controller incorporates an NC-program interpreter [12] for programs written in the RS274 language.

2.5. Earlier work at NIST leading to FBICS

The Vertical Workstation System (VWS) of the NIST Automated Manufacturing Research Facility, developed between 1986 and 1989, was a feature-based system for piece-part machining [14]. It included a design-by-feature subsystem. Software for the system was written in Lisp.

The Off-Line Programming System (OLPS) was an NC-code generation system developed between 1988 and 1990 intended to be used in a larger system in which machining features are defined separately from the design [15]. A library of parametric machining features was defined for use with OLPS [16]. OLPS code was written in Lisp.

In 1995 and 1996 a prototype Feature-Based Control System was developed which included many of the elements of FBICS [17]. It served to show the feasibility of feature-based control, but did not include many key FBICS elements.

In the late 1980s and early 1990s, the Department of Defense supported a 'Next Generation Controller' (NGC) project. ISD prepared a report 'NIST Support to the Next Generation Controller Program: 1991 Final Technical Report' [18], containing a variety of suggestions. Appendix C proposed three sets of commands for 3-axis machining, one set for each of three proposed hierarchical control levels. The suite proposed for the middle level of control evolved into the one used in FBICS for Workstation-level tasks, described in Fig. 2. The suite proposed for the lowest control level evolved into a suite, known in the EMC project as the 'canonical machining functions,' for 3-axis to 6-axis machining [19]. This suite is used in FBICS in the NC-code interpreter.

The ALPS language (A Language for Process Specification) was developed at NIST [20]. ALPS is a domain-independent language for writing process plans for discrete operations. Additions to ALPS, and EXPRESS models containing them, were made by the NIST Manufacturing Systems Integration (MSI) project [21]. The MSI project also developed resource concepts which have been used in FBICS.

As part of a Rapid Response Manufacturing program at NIST, a model for cutting tools and tooling components was developed. The requirements specification document [22] was translated into an EXPRESS model. This model, with modifications, is being used in FBICS. Efforts for standardization of cutting tool data, based on the model, are in progress in Technical Committee 29 Working Group 34 of the International Organization for Standardization (ISO TC29/WG34). The developing standard is ISO 13399.

2.6. STEP

STEP (Standard for the Exchange of Product Model Data) 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 [23]. 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 [24]. 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).

2.6.1. AP 224

STEP AP 224, the ISO standard for 'Mechanical Product Definition for Processing Planning Using Machining Features' [25] is being used in FBICS. AP 224 is largely a specification of a library of 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. FBICS uses the library but does not use the boundary representation method. The library defined in AP 224 is quite similar to that defined in [16].

The AP 224 feature library 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, pocket, rectangular_pattern, rectangular_pocket, round_hole, slot, spherical cap, thread. Each manufacturing_feature has a stereotypical shape governed by several parameters. To specify a round_hole, for example, the parameters include diameter, hole_depth, and bottom condition (among others). A particular instance of a round_hole is defined by specifying a value for each parameter. The AP 224 feature types implemented in FBICS are round_hole, counterbore_hole, and rectangular_pocket. Implementation of many other feature types from AP 224 are expected to be feasible.

The EXPRESS schema used in FBICS for STEP AP 224 is a schema provided by the person who was the ISO 'owner' of AP 224 while it was being developed. The schema has been modified slightly. In STEP terms, it is an Application Reference Model (ARM) type of model.

AP 224 provides two general types of tolerance: (1) tolerances on numeric parameters; and (2) geometric and dimensional tolerances. In the first case, the tolerance is an attribute of a parameter which is an attribute of a feature. The hole_depth of a round_hole, for example, may be a numeric_parameter_with_tolerance. In the second case, each tolerance is a self-standing entity which may be applied to more than one feature, and several tolerances may apply to the same feature. The geometric and dimensional tolerances are a full suite of tolerances from ASME Y14.5.

AP 224 STEP Part 21 files are used in FBICS for designs of piece parts (starting, finished, or partially finished) and for feature sets, as shown in Table 2. Each file is expected to define a single (AP 224) Part. The shape of a piece part is determined by its base shape as modified by a list of features. When an AP 224 file is used to describe a design in FBICS, the features are assumed to be closed solids which are Boolean subtracted from the base shape to determine the Part shape. When an AP 224 file is used to contain a feature set, the base shape is simply ignored.

2.6.2. AP 203 and AP 214

AP 203 covers configuration controlled design. A large portion of it provides a method of describing the design of a part in a boundary representation. AP 214 is a huge data model that includes all of the content of STEP APs 203 and 224 and much more. AP 203 is used for some purposes in FBICS and is used in many commercial systems. AP 214 is not used in FBICS, but is used in some similar systems, as described below.

2.7. Other integrated systems

A comparison of FBICS with other integrated systems is shown in Table 1. The table lists all the major capabilities of FBICS, plus a few capabilities that it would be desirable to add to FBICS, and shows which of these capabilities are possessed by various other systems. Some of these systems have capabilities not listed in the table that FBICS does not have. Table 1 includes systems from universities and commercial firms in addition to FBICS. Understand that in cases where commercial systems and non-commercial systems perform similar functions, the commercial systems almost always have much broader and deeper functionality.

As may be seen in Table1, two FBICS traits shared by no other system are its embedding in a hierarchical control system and its use of multistage process planning. Few of the other systems are aimed at both inspection and machining, and only one other uses feedback from inspection to machining. Those that use STEP data use AP 203 and/or AP214. None of the others uses STEP AP 224 features.

2.7.1. Commercial integrated systems

Commercial systems exist that perform many of the functions of FBICS. Only a few of the larger and more similar (to FBICS) of them are covered here. In addition to these, there are several dozen systems that generate NC code for machining, and a handful that generate code for inspection. The commercial integrated systems included in Table 1 are modular. The various functional modules share a common database. In most cases, users may purchase different combinations of modules. To avoid the appearance that NIST is evaluating commercial systems, the commercial systems included in Table 1 are not identified by name, and references to them are not included in this paper. The information in Table 1 regarding commercial systems was extracted from archival papers, web sites, demonstrations, and a little direct system use.

2.7.2. Academic integrated systems

The information in Table 1 on university systems was extracted from the sources cited in the text of this section. It is possible that the systems do less or more than indicated in the table.

The Purdue University Quick Turnaround Cell (QTC) [26–28] has many capabilities in common with FBICS. With it, a knowledgeable user can design and machine a simple part in an hour or two. Early versions of the QTC were described as including vision inspection.

Cybercut at U. C. Berkeley is similar in function to the Purdue QTC but includes touch probe inspection and may be used over the internet [29–31].

An Intelligent Machining Workstation (IMW) has been built at Ohio University [32]. This differs from the other integrated systems described here in that it uses several large subsystems from different vendors: (1) for design, any CAD system that will output a STEP file; (2) the ICEM Part system for process planning and NC-code generation; (3) Deneb's Virtual NC for NC-code verification; (4) Cognition's Cost Advantage cost estimator; and (5) An Oracle database. The transition from a STEP design output to suitable input for the Part system requires manual

Table 1
FBICS compared to other systems

Trait[a]	System							
	FBICS	Commercial A	Commercial B	Commercial C	Cybercut	IMW	Purdue QTC	SEW
Control system included	X	0	0	0	X	0	X	0
Coord. Measuring machine targeted/used	X	0	0	X	0	0	0	0
Design system included	0	X	0	X	X	X	X	X
Machining feature recognition auto.	0	0	X	0	0	X	0	0
Feedback from inspection used	X	0	0	0	X	0	0	0
Fixturing selected automatically	L	0	X	0	X	X	X	0
Hierarchical control used	X	0	0	0	0	0	0	0
Inspection control code generated auto.	X	0	0	X	L	0	0	0
Inspection performed	X	0	0	X	L	0	L	0
Machining control code generated auto.	X	X	X	X	X	X	X	X
Machining parameters selected auto.	X	X	X	X	X	X	X	X
Machining center targeted/used	X	X	X	X	X	X	X	X
Multiple setups handled automatically	X	0	X	0	L	X	L	0
Processing planning multistage	X	0	0	0	0	0	0	0
Process plans generated automatically	X	L	X	0	X	X	X	X
Rule-based methods used	L	X	X	0	X	X	0	X
Setup files generated automatically	X	0	0	0	X	X	X	0
Simulation graphics shown	X	X	X	X	0	X	X	0
Solid modeler used	X	X	X	X	X	X	X	X
STEP data used	X	X	X	X	0	X	0	0
Tolerances on features used	X	X	X	X	X	X	X	X
Tools selected automatically	X	X	X	0	X	X	X	X
User-friendly interfaces to data included	0	X	X	X	X	X	X	X

[a] X = has trait; L = has limited trait; 0 = does not have trait (as documented).

assistance. Other details about the interfaces between subsystems are not provided.

The Simultaneous Engineering Workstation (SEW), built at the University of Abertay Dundee, integrates design-by-feature with automated feature-based process planning and NC-code generation [33].

3. FBICS architecture

The FBICS controller hierarchy, shown in Fig. 1, is generally as described in RCS literature. In stand-alone FBICS, levels of controller are included at the RCS Cell, Workstation, and Task levels. In integrated FBICS, additional levels (Elementary move, Primitive, and Servo) are also present. Each controller runs in a separate process or processes (in the operating system sense). Processes may be on the same computer or different computers. Controllers communicate via a messaging system.

This hierarchy corresponds to the way many large machine shops actually run. The shop is divided into groups of machines, each called a cell. A cell is expected to be able to make and inspect a finished part from a starting workpiece. A cell includes several workstations. Each workstation has a principal machine and, possibly, auxiliary machines. Each machine can perform several types of task, each task is composed of several elementary moves, each elementary move is decomposed into primitive motions, and each primitive motion is controlled by a servo system.

In addition to the controllers, a Solid Modeling Server (Modeler, for short), a Data Repository, and a Graphic Display are included. The Modeler works for the planners, in the Cell, Workstation, and Task controllers. The Graphic Display is connected directly only to the Modeler. Commands for displaying solid objects are available to the three planners, but they are addressed to the Modeler.

The FBICS architecture is designed to allow for human participation at every significant step of the process. Elements of this include using appropriate data types at each interface between control levels, having an open file format for all data types passing across interfaces, having

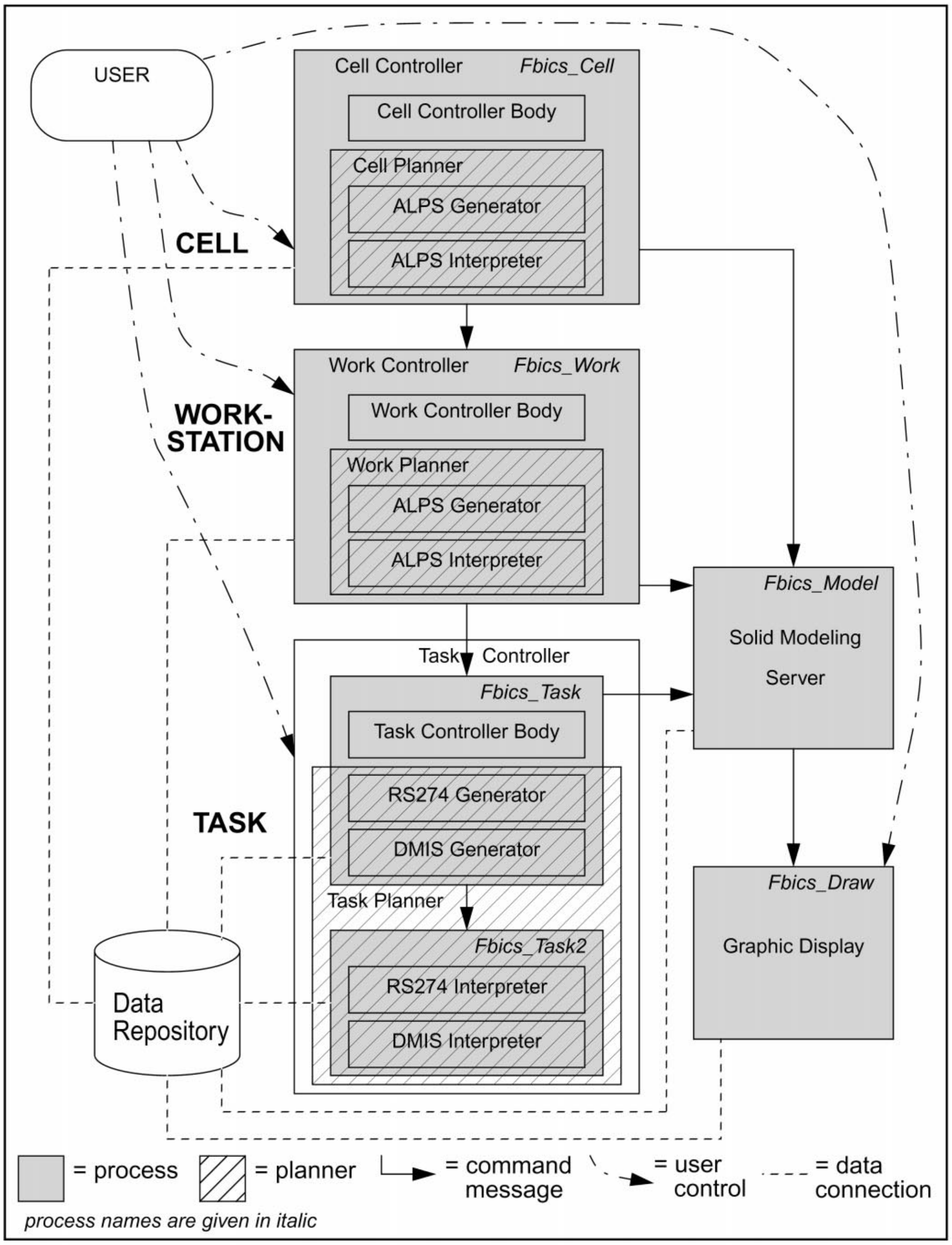


Fig. 1. FBICS stand-alone architecture.

variable planning depth, and allowing either off-line or on-line planning. The user-friendly editors and interfaces required for effective human participation in FBICS activities, however, have not been built.

The notion of a ‘set up’ is central to FBICS. A setup is a fixturing of a workpiece on the table of a machining center or coordinate measuring machining. The workpiece does not move with respect to its fixturing during a single setup. It does move from one setup to the next. Some parts can be handled in a single setup, but more commonly, two or more setups are required.

The ALPS process plan language is used for plans at the Cell and Workstation levels. The fundamental object of ALPS is the plan. A plan is a recipe for performing a specific task. A plan in ALPS is a one-level breakdown of a task into subtasks, expressing the requirements of each subtask and its interrelation with other subtasks in the plan. A plan contains a set of nodes (or steps) which provide sequencing information and detail how to perform the task in terms of individual subtasks. Every plan has an associated target system which may execute the plan. ALPS provides for parallel operations, alternatives, synchronization of events, parameters, and resource allocation. To make ALPS usable in FBICS, two suites of subtypes of the ALPS primitive_-task_node have been added to ALPS. One suite, for the Workstation level, is discussed in Section 5. The other suite, for the Cell level, includes only one subtype: run_setup.

The stand-alone FBICS architecture is shown in Fig. 1. In stand-alone FBICS, emphasis has been placed on the planning component of each controller. A clean, small, function call interface has been defined for each planner, to be used by the rest of the controller. In the stand-alone version, the rest of each controller includes only enough functionality to make the system run and to help with debugging during development.

Fig. 1 shows only one subordinate for each superior controller, since that is how stand-alone FBICS is built. In general, in RCS, each superior controller may have several subordinates. The FBICS Task-level controller might be readily split into two task-level controllers, one for a machining center and one for a coordinate measuring machine, for example.

The ‘command message’ arrows on Fig. 1 show the direction of flow of commands. In every case (although no return arrows are drawn), a status message flows back to the issuer of the command. Almost all the status messages are simple, meaning ‘I did it’, or ‘I could not do it.’ Status messages from the Solid Modeling Server may be more meaty. Many of them include a Boolean yes/no answer to a question, some include a single numerical value, and others include several related values.

3.1. Cell Controller

The (top-level) Cell Controller is focused on making a part from a part blank or on inspecting an entire previously made part. The Cell Controller can (1) make a Cell-level process plan for the part, (2) execute a Cell-level process plan, (3) give planning commands to the Workstation Controller, and (4) give execution commands to the Workstation Controller.

3.2. Workstation Controller

The (second-level) Workstation Controller handles a single setup of the part (i.e. processing the part without moving it). The Workstation Controller focuses on transforming a workpiece from an incoming shape to an outgoing shape, possibly with intermittent inspection, or on inspecting those features that may be inspected in a single setup. The Workstation Controller can (1) make a Workstation-level process plan for one setup, (2) execute a Workstation-level process plan, (3) give planning commands to the Task Controller, and (4) give execution commands to the Task Controller.

3.3. Task Controller

The (third-level) Task Controller focuses on making or inspecting single features. The Task Controller can make a Task-level process plan for machining one feature or inspecting one feature. At the Task level, a machining process plan is an NC-code file (in the RS274 language), and an inspection plan is an inspection code file (in the DMIS language). The Task Controller can execute Task-level plans.

In stand-alone FBICS, the Task Planner is split between two processes. The Fbics_Task2 process contains the parts of the planner that run during plan execution. During execution, the Fbics_Task process sends messages to the Fbics_Task2 process, telling it to run a file of RS274 code or DMIS code. Splitting the planner between two processes was originally done for convenience. The split has been kept in stand-alone FBICS so that loose integrations are easy to build. In the loose integration that has been used most, messages that would be sent to the Fbics_Task2 process in the stand-alone are redirected to another Task-level controller.

The FBICS_Task2 process includes version three of the NIST DMIS interpreter and version three of the 3-axis NIST RS274/NGC interpreter.

The two messages Fbics_Task2 understands are ‘run a DMIS file’ and ‘run an NC file’. In response to these messages, the appropriate interpreter reads the file and executes the instructions in the file. In the stand-alone version, this results in some degree of emulation of machining or inspection.

3.4. Solid modeling server

The Solid Modeling Server (often shortened to ‘Modeler’ in this document) provides solid modeling services to its clients, which are the controllers. The Modeler maintains a separate view of the shape of objects for each client. The types of service the Modeler provides include, for example, maintaining a model of the current shape of the part, faceting a model and telling the Graphic Display to show it, and determining if a candidate touch point for probing is present on the current part. The Parasolid solid modeler serves as the underlying modeling engine.

3.5. Graphic Display

The Graphic Display shows 2D views of 3D objects on a color monitor, using the ‘movie camera’ paradigm [34]. Display of objects is handled by the Modeler and Graphic Display jointly. For each type of object to display, the Modeler facets the object, writes a modeling file and sends a message to the Graphic Display, telling the Graphic Display to read the file and display it as directed by the user. The display serves to let the user see how planning or execution is progressing. The display is manipulable by the user in the usual ways: move around the object; zoom in or out, etc.

The types of object which can be shown are: intended outgoing part shape; incoming part shape; current part shape; feature shape; feature access volume; and fixture shape. Each object type may be displayed as a solid object, as a wire frame, or not at all—under immediate control of the user. The HOOPS graphics system is the underlying graphics engine.

3.6. Data Repository

FBICS uses a file system as the Data Repository. Data in the repository includes, for example, part designs, process plans, and user option files. During FBICS operation, data files are written by one process and read by others. Data files are used as a supplement to messaging for sending commands from one controller to another. Using a file system as the Data Repository has been fully adequate. As FBICS matures it may become desirable to use a database system for the Data Repository, but the need has not yet arisen.

3.7. Interfaces between FBICS modules

An FBICS module is a set of related computer code for performing functions of the system. FBICS uses three types of interfaces between modules: application programming interfaces (APIs); messaging interfaces; and file interfaces. An API is a set of functions calls available for use between modules. A messaging interface is a set of messages that may be sent between processes by interprocess communication methods. A file interface is a set of file types which may be written by one process and read by another. For interprocess communication, FBICS uses a messaging interface in every case [35], with most of them being supplemented by a file interface.

The Cell, Workstation, and Task planners in FBICS each have an API for calls to the planner. The DMIS and RS274/NGC interpreters included in the Fbics_Task2 process have APIs. The Modeler has an API interface, but, since the Modeler is in a separate process, its API is wrapped in messaging interface.

All interfaces between modules are documented in detail in Ref. [1], except the interpreter interfaces, which are explained in Refs. [11,12].

3.8. User interfaces

As shown in Fig. 1, stand-alone FBICS has user interfaces to the three controllers and to the Graphic Display. The user interface to the Graphic Display has the capabilities mentioned above and is totally mouse-driven. The stand-alone version's user interfaces to the controllers are all simple text-based interfaces that allow the user to exercise the two principal functions of the controller: plan or execute.

3.9. Integrated FBICS

Loosely-integrated versions of FBICS have been built with all upper-level components as just described, except for Fbics_Task2. For a loosely-integrated version, the Fbics_Task2 process is replaced by a fully functional RCS controller that includes an RS274 interpreter and/or a DMIS interpreter. For an integration with inspection, the message interface between Fbics_Task and Fbics_Task2 is used unaltered. For an integration with machining, the interface is modified. The fully functional RCS controllers control actual hardware for cutting or inspection [36].

A proposed architecture for tightly-integrated versions of FBICS is described in Ref. [1].

4. Two-stage planning

FBICS employs two-stage planning at the Cell, Workstation and Task levels. The use of two-stage planning is driven by the desirability of using flexible plans, combined with the requirement of doing hierarchical planning.

A flexible plan is a plan with variables, alternatives, and (possibly) omissions. Omissions might include omitting tool-changing and coolant commands from stage-one plans; FBICS makes these omissions. Using flexible plans solves the common problem that in the discrete parts and other domains where the formulation of plans is costly in time or money, it is desirable that process plans be reusable. If conditions under which plans are executed may change in a way that affects the utility of plans, however, completely fixed process plans may not be reusable. At least four types of data may change: (1) sensory data—e.g. temperature; (2) resource data—e.g. the tools in a machine carousel; (3) job data—e.g. the number of widgets to make now; and (4) planning system behavior data—e.g. which rule to use to set feed rates. If any of these types of data affects planning, a fixed plan will become obsolete when the data changes.

In FBICS, the stage-one plans prepared in the Cell and Workstation planners are flexible plans written in ALPS. A flexible plan is expected to be usable for an extended time under a variety of conditions whose variability is embodied in the variables and omissions of the plan. In addition to providing for changeable conditions, a stage-one plan may leave alternatives open because the planning system recognized alternatives but elected not to choose among them. When all alternatives seem equally desirable to a planner, rather than making an arbitrary decision, it seems a better strategy to defer making a decision. Then an executor with better information or a stronger opinion will have the opportunity to exercise its judgement.

A stage-one plan may be executed directly. When the plan is executed, it is traversed one step at a time. For each step of the plan, one or more executable operations are executed (by physical action or by sending commands to a subordinate). By waiting for each command to be executed before selecting the next step to run, the executor allows for using up-to-date values of plan variables. In FBICS, each executable operation is executed by first telling the subordinate to make a plan to carry out the operation and then telling the subordinate to execute the plan it just made. If either the subordinate's planning or its execution is unsuccessful at any point, execution of the superior's plan fails and is stopped.

A disadvantage of executing a flexible plan directly is that, for each command sent to a subordinate, it may be necessary for the subordinate to make a plan from scratch for carrying out a command received from its superior. If the subordinate planner cannot make such a plan, execution of the superior plan fails. A second disadvantage of executing a flexible plan directly is that, if it is executed more than once under the same conditions, after the first execution, the subordinate will be needlessly making the same plans over again, entailing needless cost.

If conditions will not change for a period of time during which the superior's plan is to be executed one or more times, it is, therefore, useful to prepare a stage-two plan for the superior and a set of stage-one plans in the subordinate. To do this, the superior's stage-one plan is traversed, producing a set of executable operations by selecting a specific sequence of tasks from among alternatives in the stage-one plan, selecting specific resources where the stage-one plan has generic resources, and filling in blanks where the stage-one plan has omitted items. The subordinate makes a plan for carrying out each of the superior's executable operations. The superior makes an ordered list of pairs, one for each of the executable operations. Each pair consists of the subordinate plan name and the name of the type of executable operation. This list is the stage-two plan for the superior. FBICS Cell and Workstation-level stage-two plans are modeled very simply and do not use ALPS. Modeling stage-two plans in ALPS would be easily done but cumbersome.

Another reason for having stage-two plans is to support optimizing plans across a hierarchy. To perform optimization over a hierarchy via standard search methods, a stage-one plan allowing for all cases to be tested is prepared for the top controller. Partial search plans (plan traversals) are constructed through this top-level plan. For each node (plan step) on a partial search path at the top level, the next level down is told to find an optimum execution. The total cost of any one search path at the top level is the direct cost of the path to the top level plus the cost of each node on the path to the subordinate(s) that handles it. The subordinate controllers follow exactly the same procedure in determining their own optimums. To support this type of search, it is necessary to have a format for recording the optimum path. The stage-two plan provides that format.

5. FBICS Workstation tasks

A suite of tasks has been defined to be used in ALPS stage-one process plans at the FBICS Workstation level. These tasks are the leaf nodes of the supertype-subtype tree shown in Fig. 2. The tasks are divided into two main subtypes: two for inspection and 16 for machining. In the figure, the root of the tree is the ALPS primitive_task_node. This is actually a leaf node in the generic ALPS supertype-subtype tree, which is not shown.

In Fig. 2, tasks are shown in **boldface** type. Supertypes are connected to subtypes by lines, with the supertype higher on the page. Attribute names are shown in *italic* type. Data types of attributes are not shown. Only leaf nodes may be instantiated. To find all attributes of a task, trace down the tree from 'primitive_task_node' to the task, and include the attributes of every node along the path, in order.

Each task includes a pointer to an AP 224 feature on which the task is to be performed. The form of the pointer is the index number of the feature from the list of features included in the features file used in the same setup as the process plan. The attribute containing the pointer is named removal_volume_index.

The two subtypes of inspection_task are inspect_feature_surface and inspect_feature_geometry. Only inspect_-feature_geometry has been implemented.

Of the machining tasks, FBICS currently handles counterboring, finish_mill (for rectangular pockets), finish_mil-l_adaptive, and twist_drill.

For stage-two Workstation plans, 24 types of executable operation are defined. Seventeen of these correspond one-to-one with 17 stage-one tasks—all but finish_mill_adaptive, whose execution requires three executable operations (finish_mill, inspect, finish_mill). The other seven executable operations include coolant control for machining, and three matching pairs (one for inspection and one for machining) for starting a program, changing a tool, and stopping a program.

6. FBICS data files

All the data languages employed by FBICS are text-based, using ASCII characters; no binary formats are used. Two types of languages are used: those described using STEP with semantics in EXPRESS; and grammar and syntax from STEP Part 21, and those (such as DMIS and RS274) described in a single specification covering semantics, syntax and grammar.

Table 2 shows all the file types used in FBICS. All files except the four marked *No EXPRESS Schema* are STEP Part 21 files corresponding to the schemas listed in Table 2. STEP AP 203 and AP 224 are ISO standards. The tool catalog is being standardized in ISO. The four types not modeled in EXPRESS are DMIS input format,

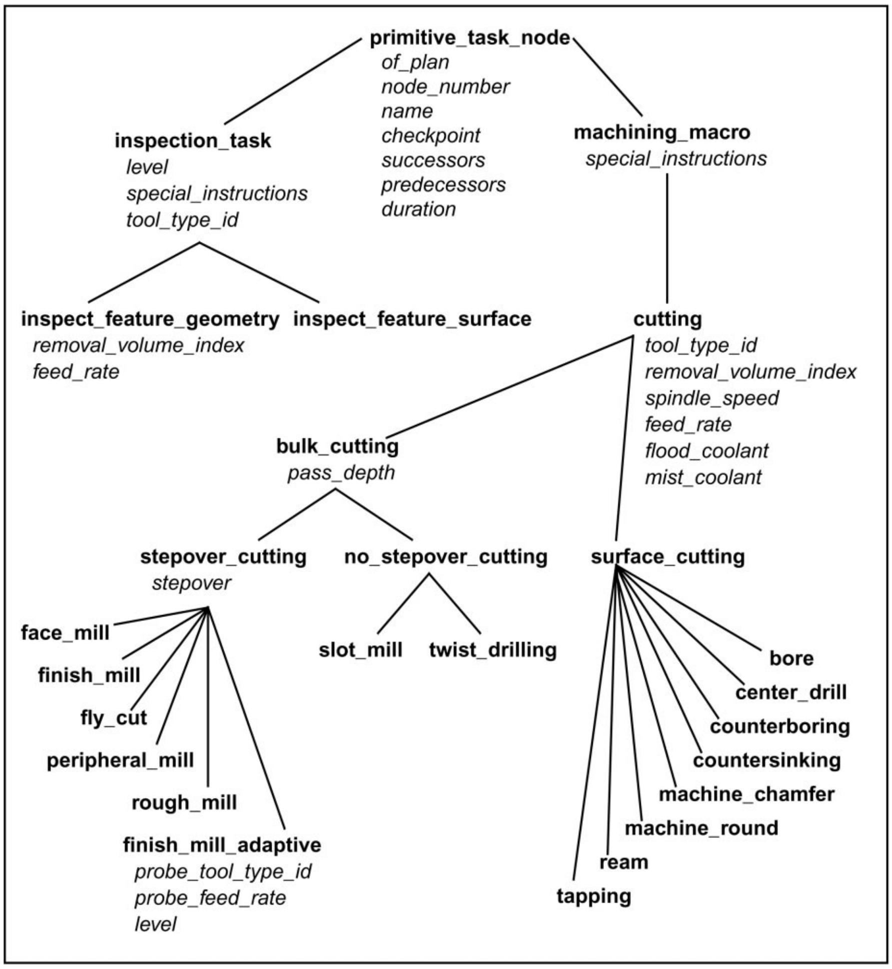


Fig. 2. Workstation-level tasks.

DMIS output format, RS274, and the graphics file format.

7. FBICS operation

This section describes (1) what happens during FBICS initialization, (2) how planning three levels deep for machining with inspection is done, (3) what the special problems of inspection planning are, (4) what happens during execution of the Cell-level stage-one plan made in item 2, and (5) how adaptive machining has been implemented.

7.1. FBICS initialization

FBICS is brought to a warm state when the user gives the shell command 'fbics' in a terminal window. This starts the various independent processes and establishes communications among them.

FBICS is initialized when the user gives an initialization command to the Cell Controller. This causes initialization command messages to be 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. During initialization, each planner cleans up its world model, sets default values for various world model parameters, and reads various files. The Cell Planner reads the shop_options file and resets any inspection and machining parameters in its world model that have values in that file. The Workstation Planner does the same with the shop_options file, but in addition, reads the tool catalog, tool inventory, and tool_usage_rules files named in the shop_options file. The Task Planner reads the shop_options file, the tool inventory and tool catalog files, and a task_options file.

7.2. Planning three levels deep for machining with inspection

The most complex form of FBICS planning occurs when the Cell Controller is told to plan three levels deep including both machining and inspection, so that type of planning is described here. It is assumed that this type of processing will

Table 2
FBICS data in files

Model name	Item modeled	File writers	File readers	EXPRESS schemas
AP203	Fixture	User, CAx (i.e. CAD/CAM/CAE) system	Workstation & Task planners	ap203
AP 224	Initial workpiece	User, Cell Planner	All 3 planners	arm224
	In-process workpiece	Cell Planner	Workstation & Task planners	
	Final part	User, CAx system	All 3 planners	
	Features	Cell Planner	Workstation Planner	
Cell Stage-1 Plan	Cell stage-1 plan	Cell Planner	Cell Planner	fbics_combo, fbics_alps
Cell Stage-2 Plan	Cell stage-2 plan	Cell Planner	Cell Planner	fbics_combo
DMIS input format	Control code for inspection	Task Planner (Fbics_Task)	Task Planner (Fbics_Task2)	*No EXPRESS schema*
DMIS output format	Results of inspection	Task Planner (Fbics_Task2)	Task Planner (Fbics_Task2)	*No EXPRESS schema*
Graphics file	Access volumes, features, fixture, final part, workpieces	Modeler	Graphic Display	*No EXPRESS schema*
RS274/NGC	Control code for machining	Task Planner (Fbics_Task)	Task Planner (Fbics_Task2)	*No EXPRESS schema*
Setup	Setup	Cell Planner	All 3 planners	setup
Shop options	Shop options	User	All 3 planners	shop_options
Task options	Task options	User	Task Planner	task_options
Tool catalog	Tool catalog	User	Workstation & Task Planners	tool_catalog
Tool inventory	Tool inventory	User	Workstation & Task Planners	fbics_combo, expressions
Workstation stage-one plan	Workstation stage-one plan	Workstation Planner	Workstation Planner	fbics_combo, fbics_alps, arm224
Workstation stage-two plan	Workstation stage-two plan	Workstation Planner	Workstation Planner	fbics_combo
Workstation executable operations	Workstation executable operation	Workstation Planner	Task Planner	fbics_combo

be performed by a machining center equipped with cutting tools and a touch-trigger probe. Other types of planning for a machining center are similar but simpler, usually including a subset of the three-level planning activities.

7.2.1. Cell controller planning

When the user gives a plan_part command to the Cell Controller, the user provides the file name of the design of the part to be made, the file name of the design of the incoming workpiece, a flag indicating whether the incoming workpiece design file already exists or should be written, the base name for setup files, the base name for plan files, the name of the fixture file to use, and the number of levels to plan. We are assuming here that the number of levels is three. The general plan of attack is to make a stage-one Cell-level plan first, then make a stage-two plan from the stage-one plan, giving planning commands to the Workstation Controller as the stage-two Cell-level plan is made.

To make a stage-one plan, the Cell Controller reads the files describing the design of the part to be made and the incoming workpiece. The Cell Controller makes a few initial checks, such as requiring that the shape to be made is contained in the incoming workpiece and winnowing down the features to be made by throwing out those whose intersection with the initial workpiece is empty.

The set of features to be made is divided into subsets called direction-sets, all of whose native z-axes point in the same direction (since these are all machinable from the same direction). Some of the features to be made will block access to making other features in the same direction-set or other direction-sets. Blocking features must be made before blocked features. With the help of the Modeler, an analysis of the blocking situation is made. Using the results of this analysis, the features to be made are assigned to setup-sets (each of which is either an entire direction-set or a subset of a single direction-set) and the setup-sets are partially ordered. The partial ordering is captured in a Cell-level stage-one ALPS process plan. The plan nodes that are partially ordered are all run_setup nodes, one for each setup-set.

For each run_setup plan node, a metafile called a setup file is made and its name is recorded in the run_setup node. Several data files referenced in the setup file are made, as well, including (1) a file describing the features to be made in the setup, (2) a file describing the shape of the workpiece coming into the setup (if there is only one possible such shape), and (3) a file describing the shape of the workpiece leaving the setup (if there is only one possible such shape). The names of these additional files are recorded in the setup file along with the base name of a Workstation plan for making the features, the name of a fixture file, and the direction of the setup-set.

To make a stage-two plan, the Cell traverses the stage-one plan. Each plan step the Cell decides to do next is a run_setup step, so one of two types of run_setup operation is put into the stage-two plan. Since three levels are being planned, a run_setup2 operation (indicating a stage-two Workstation plan should be run) is used and a stage-two Workstation plan is named in the operation. Also during the traversal, the Cell determines the shape of the incoming and outgoing workpiece for each setup. If either shape was unknown when the setup file for the setup was written, the setup file is rewritten and a file (or two) describing the previously unknown shape(s) is written. The Cell then commands the Workstation to make a plan two hierarchical levels deep for the (now fully specified) setup.

7.2.2. Workstation controller planning

Each time the Cell Controller commands the Workstation Controller to plan a setup two levels deep, the Workstation Controller first makes a stage-one plan for the setup, then makes a stage-two plan from the stage-one plan (generating command files to be used later by the Task Controller as it builds its stage-two plan). Finally, the Workstation Controller traverses its stage-two plan.

To make a stage-one plan, the Workstation Controller first reads the setup file. This tells it what the name of the plan it writes should be, as well as the names of files to read. The Workstation Controller reads the files and calls on the Modeler to set up solid models of all parts and features in the correct orientation. Then the Workstation Controller analyzes the feature blocking situation and constructs a partial ordering for making the features. It also decides which features are to be inspected. For each feature to be machined, the Workstation Controller selects a machining operation appropriate to make the feature and a type of cutting tool with which to perform the operation. Values are selected for tool use parameters (feed, speed, etc.). For each feature to be inspected, the Workstation Controller selects an inspection operation and a touch-trigger probe with which to perform the operation.

To make a stage-two plan, the Workstation Controller traverses the stage-one plan. For each step of the stage-one plan the Workstation Controller decides to execute, the Workstation Controller defines one or more Task-level operations required to accomplish that step. For each operation, the Workstation Controller writes a file describing the operation.

Finally, the Workstation Controller traverses the stage-two plan and, for each operation, sends a planning command to the Task Controller.

7.2.3. Task controller planning

When the Task Controller receives an open_setup command, it reads the setup file whose name is received with the command, reads the files describing the incoming part and fixture, and calls the Modeler to model the incoming part and fixture.

A generate_nc command includes (1) the name of a file describing the machining operation and feature to make and (2) the name of an NC-code file to write. When the Task Controller receives a generate_nc command, it reads the input file, calls the Modeler to model the feature and the workpiece as it will be when that feature is subtracted from it, and calls an NC-code generator to write the output file.

A generate_dmis command includes (1) the name of a file describing the inspection operation and feature to inspect and (2) the name of a DMIS code file to write. When the Task Controller receives a generate_dmis command, it calls a DMIS code generator to write the output file. DMIS generation details are given in Section 7.3.4.

The Task Controller does not make stage-two plans. A Task-level stage-two plan would be a list of canonical machining and/or inspection commands as generated by the DMIS and RS274 interpreters. Since sensory data from the next lower control level often affect the generation of canonical commands, pre-generated lists of canonical commands are usually not valid.

7.3. Inspection planning

The behavior of the system when planning for pure inspection is much the same as in machining planning. One factor makes inspection planning easier: the workpiece does not continually change shape during processing. A second factor makes inspection planning harder: until it is decided which features are to be inspected, it is not known whether any proposed setup will be required. Thus, the Cell Controller must use the same decision rules as the Workstation Controller while the Cell Controller plans inspection setups. Using the same decision rules is implemented by putting them in the shop_options file, which is used by both controllers.

In machining planning, deciding what to do is simple—every bit of material in all the features not already on the workpiece has to be removed. In inspection planning, deciding what to do is not so simple. Which AP 224 features should be inspected? How thoroughly should they be inspected? When should they be inspected? The approach to these questions taken in FBICS is that there is no 'right' answer, so let the user decide. This is implemented by user preference files and (if user preferences specify it) by asking the user on a case-by-case basis.

7.3.1. Which features to inspect

Options regarding which features to inspect are used in the Workstation Planner. Five choices are provided in the shop_options file: (1) inspect all features; (2) inspect no features; (3) inspect any feature with any parameter having any tolerance; (4) inspect any feature which has any parameter with a tight tolerance; and (5) let the user decide for each feature. The definition of what is a 'tight' tolerance for milling is another user option.

7.3.2. How thoroughly to inspect a feature

It is impossible to inspect every point on a surface, so a decision must be made on which points to inspect. FBICS uses the qualitative idea of high, medium, and low inspection intensity. The user may specify in the task_options file how many points to inspect at each of these three levels for each DMIS feature type.

7.3.3. When to inspect

FBICS currently inspects each feature immediately after cutting. This wastes time, since doing it requires that the tool be changed and the coolant turned on or off before and after every cutting operation. For complex parts, waiting to inspect until cutting is complete might also be wasteful, since if any cutting operation spoils the part, all the following cutting operations are wasted effort. An option for how many features to cut before inspecting any features to be inspected has been defined and may be included in shop_options files, but the information is not used in the current implementation.

7.3.4. DMIS code generation from AP 224 features

The DMIS language defines a number of surface features. These include plane, cylinder, cone, and sphere, among others. These are simpler than features defined in AP 224, which has, for example, pocket and hole. To generate DMIS code from AP 224 features, it is necessary to decompose the surface of each AP 224 feature into a set of DMIS features. The surface of the prototypical AP 224 pocket, for example, may be decomposed into nine DMIS features: five planes (four sides and a bottom) and four cylinders (the corners of the pocket—each is a quarter cylinder). For each DMIS feature present in principle, DMIS code is written (1) to define the feature, (2) to measure the feature, and (3) to report the results of the measurement. If the feature is a cylinder with a diameter tolerance (inherited from a tolerance on the corner radius of an AP 224 pocket), DMIS code is also written to define the tolerance, calculate the error in the diameter, and report whether the cylinder diameter is in-tolerance or not.

7.4. Executing a stage-one cell-level plan

This section describes what happens when the user tells FBICS to execute the stage-one Cell-level plan for machining described in Section 7.2. Stage-two plan execution, which is much simpler (essentially: do the operations on the list in order) is not described in this paper, but is in Refs. [4,11].

During execution of a stage-one Cell-level plan, the Cell Controller traverses the plan one step at a time. For each plan step (they are all run_setup steps), the Cell Controller first sends a plan_setup command to the Workstation Controller, and then a run_setup command (indicating a stage-one plan should be used) immediately after. The way in which the Cell Controller traverses its stage-one plan and disambiguates data is identical to the way it is done in preparing a stage-two plan from a stage-one plan (described in Section 7.2.1). A slightly different naming convention is used for the disambiguated data, which is intended to be short-lived in the case of executing a stage-one plan.

For each plan_setup command it receives, the Workstation Controller makes a stage-one plan exactly as described in Section 7.2.2. For each run_setup (using a stage-one plan) command it receives, the Workstation Controller reads the stage-one plan and traverses the stage-one plan (also exactly as described in Section 7.2.2), writing operations files as described in that section. During execution, however, as soon as each operation is selected, the Workstation Controller first sends a generate_nc or generate_dmis command to the Task Controller then sends an execute_nc or execute_dmis command immediately afterwards.

Generating NC code and DMIS code in the Task Controller is the same during executing as during planning. Executing the code is done only during execution. As shown in Fig. 1, the Task Controller includes two separate processes. Code generation is performed entirely by the Fbics_Task process. Code execution is performed entirely by the Fbics_Task2 process. For each generate_nc command it receives specifying that an AP 224 feature should be cut, the Task Controller reads the file describing the operation and generates the tool path to use to cut the feature. Tool use parameters are given in operation files for cutting. Thus, the code that is generated may include commands to reset the values of spindle speed or feed rate. For each generate_dmis command the Task Controller receives specifying that an AP 224 feature should be inspected, the Task Controller generates code as described in Section 7.3.4. For each execute_dmis or execute_nc command the Task Controller receives, the Task Controller (by call to either the DMIS interpreter or the RS274 interpreter) interprets the file named in the command, adjusts its internal model appropriately, and makes calls to canonical machining or inspection functions. In stand-alone FBICS, these canonical calls simply print themselves and update a dummy external world model required by the interpreter. In integrated FBICS, the canonical calls are used for machine control and actual cutting or inspection.

7.5. Feedback from inspection to planning

FBICS includes using feedback from inspection to planning, as follows. If there is a tight tolerance on any of the length, width, or corner radius of a pocket to be finish-milled, the Workstation Planner puts a finish_mill_adaptive step for milling the pocket in the stage-one plan. Both a cutting tool and a probe tool are used in the step. Tool-use parameters for both tools are also included. When operations are created for carrying out that step during execution of the plan, three primary operations are defined (in addition to tool changing and coolant control, if needed). The first primary operation is to finish mill a test pocket in the same place as, but slightly smaller than the specified pocket. The second is to inspect the test pocket, and the third is to finish mill the final pocket. When the Task Controller executes the DMIS file for inspecting the test pocket, the DMIS interpreter (because the inspection program says to do so) writes a file containing the results of the inspection. Before generating code for the final cut on the pocket, the Workstation reads this file. A correction factor for the final cut is calculated as the average error in the radii of the four corners of the test pocket. If the test pocket corner radii were too big on average, NC code is written so the final cut is made inside the nominal tool path for the final pocket by the correction factor. If the test pocket radii were too small, NC code is written so that the final cut is outside the nominal path by the correction factor.

8. Results and a simple example

FBICS has been tested successfully with simulated machining and inspection using moderately complex parts, including the much-used ANC101 and TEAM test parts. Integrated tests with machining or inspection have not been as complex. Planning to depth two for machining the 51-feature the TEAM part with in-process inspection of six holes (starting with a featureless block) and executing the plans in simulation results in one Cell plan, Workstation plans for eight setups, 157 NC-code files, nine DMIS files, and associated data. A total of 373 files is written. On a Sun SPARCstation 20, planning the TEAM part took 16 min and execution took 12 min. Over 90% of the time is used in the Modeler and Graphics Display. Results for the ANC101 part are similar. Describing the processing of the TEAM or ANC101 part would take far too much space.

This section, therefore, provides a much simpler example of FBICS in action. The example part, 'simple_part', is a block with a pocket on top and a hole on one side. A picture of simple_part is shown in Fig. 3. This example involves stage-one plans only. Planning one level deep for simple_part (Cell only) took 15 s on a Sun SPARCstation 20. Executing the Cell plan (which triggers Workstation level planning for two setups, as well as NC-code generation and simulated execution) took 50 s. For simple_part, 41 files are written, as summarized in Fig. 3B. An abbreviated description of what happens during planning and execution for simple_part follows. All mention of what the Modeler and Graphics Display do is omitted.

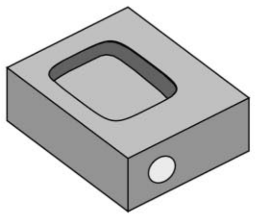
Planning for the simple part is started by typing the following command at the user interface for the Cell controller: **plan_part(simple_part.stp, OFF, block.stp, plans, feats, fi1, setups, 1)**. The **1** at the end of this command means that FBICS should plan one level deep (i.e. at the Cell level only). The name of the AP 224 STEP Part 21 file giving the design of this part to make is **simple_part.stp**. The data section of this file is shown in Fig. 3A. **OFF** means that the file giving the design of the shape to start with (**block.stp**) does not yet exist and should be created. **plans** is the root name for process plan files to write (including ALPS, DMIS and RS724 files). **feats** is the root name for feature files to write. **fi1** is the name of the fixture to use, which must already exist. **setups** is the root name for setups to write.

To carry out this command, the Cell planner:

1. Reads simple_part.stp and builds a 'feature_plus'—a structure with additional data about the feature—for each feature of simple_part.
2. Creates the starting shape for machining by making a copy of the base shape of simple_part and writes the file block.stp.
3. Divides the features into sets which may be made in one setup and puts the sets into batches which may be in any order. In the case of simple_part there are two sets (each containing one feature) and one batch (containing both sets).
4. Makes a Cell-level stage-one plan and prints it to the file plans_1cell.stp. The data section of this file and the graph of the plan are shown in Fig. 3C and D. The plan indicates that the two setups may be made in any order.
5. Makes a setup file for each of the two setup, setups_1.stp and setups_2.stp. The structure (not the text) of setups_2.stp is shown in Fig. 3E. The setups_1.stp file is similar. Because the two setups may be made in any order, it is not known what the incoming or outgoing part will be for either setup (hence the 'not_set' entries in Fig. 3E), although it is known what features will be made during each setup. The setup file positions the part_in, part_out, and features with respect to the working coordinate system of the machine so that the axis of the hole to be drilled is vertical and the features (i.e. the hole), part_in and part_out are in the correct relative location. The fixture is in the same location for both setups. The base name, 'plans_2.stp', for the Workstation-level and Task-level plans is known, even though they have not yet been written.

To execute the plan plans_1cell.stp, the user types at the user interface for the Cell Controller: **run_part1(plans)**. In response:

1. The Cell reads the file plans_1cell.stp and starts traversing the plan graph (shown in Fig. 3D). When it comes to the split node (item 2 in Fig. 3D), it decides to process node 3 (run the first setup) first.
2. The Cell reads the file setups_1.stp. Since it knows from its plan that the part_in for the entire process is 'block.stp' and it knows this is the first setup, it replaces the 'not_set' part_in for the first setup with 'block.stp'. It also changes the part_out in the first setup from 'not_set' to 'simple_part_temp_1.stp'. It then writes setups_1_temp.stp, a revised version of setups_1.stp.
3. The Cell sends a command message to the Workstation telling it to make a plan for executing the first setup as specified by setups_1_temp.stp.
4. The Workstation, on receiving this message, reads setups_1_temp.stp and all the files it names. Then it makes a stage-one plan for doing the work specified by the setup and sends a done status message to the Cell. The plan it makes is very simple, since there is only one feature to make—just start, mill a pocket, and end. It writes this plan in the file plans_1_1work.stp.



```
#1  = direction_element((0.0, 0.0, 1.0));
#2  = direction_element((1.0, 0.0, 0.0));
#3  = location_element((50.0, 37.5, 0.0));
#4  = orientation(#1, #2, #3);
#5  = numeric_parameter('block Y', 75.0, 'mm');
#6  = numeric_parameter('block X', 100.0, 'mm');
#7  = numeric_parameter('block Z', 35.0, 'mm');
#8  = block_base_shape(#7, #4, #5, #6);
#9  = numeric_parameter('tip angle', 118.0, 'degrees');
#10 = numeric_parameter('floor radius', 0.0, 'mm');
#11 = conical_hole_bottom(.T., #9, $);
#21 = numeric_parameter('diameter', 12.7, 'mm');
#22 = circular_closed_profile(#21);
#23 = numeric_parameter('depth', 25.0, 'mm');
#24 = location_element((100.0, 20.0, 15.0));
#25 = orientation(#2, #1, #24);
#26 = round_hole(#25, 'hole', #11, $, #22, #23);
#27 = shape_aspect((), (), #26);
#31 = numeric_parameter('corner_radius', 10.0, 'mm');
#32 = numeric_parameter('width', 50.0, 'mm');
#33 = numeric_parameter('length', 70.0, 'mm');
#34 = location_element((50.0, 30.0, 35.0));
#35 = orientation(#1, #2, #34);
#36 = rectangular_closed_profile (#31, #32, #33);
#37 = location_element((0.0, 0.0, -10.0));
#38 = planar_pocket_bottom_condition(.T., #37, #1, #10);
#39 = rectangular_closed_pocket(#35, 'pocket', #38, $, #36);
#40 = shape_aspect((), (), #39);
#50 = shape((#27, #40), #8, ());
#60 = material('aluminum', 'soft aluminum', $, (), ());
#70 = part('block 100 x 75 x 35', 'rev1', 'block100_75_35',
      'simple part', '', (), #50, (), (), (), $, (), (#60), (), ());
```

A. STEP part 21 AP224 data model of simple part

During Planning

starting shape (1)	cell plan (1)
features to be made in setup (2)	setup (2)
graphics (4)	modeler journal (1)

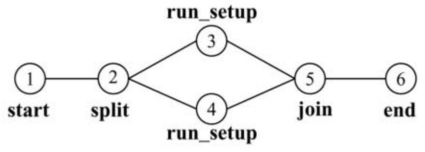
During Execution

workstation plan (2)	revised setup (2)
commands to Task (12)	NC code (12)
graphics (5)	modeler journal (1)
part after first setup (1)	

B. Summary of files written

```
#1 = plan('plans_1cell', 'version1', (), (#2, #3, #10), 'cell',
     'a part', 'generated by FBICS', (#4, #5, #6, #7, #8, #9);
#2 = internal_string(#1, 'part_in_file_name', .T.,'block.stp');
#3 = internal_string(#1, 'part_out_file_name',. T.,
     'simple_part.stp');
#4 = start_plan_node(#1, 1, $, $, (#5), ());
#5 = parameterized_split_node(#1, 2, $, $, (#6, #7), (), 0,
     .SPLIT_TIMING_SERIAL.);
#6 = run_setup(#1, 3, $, $, (#8), (), 1, 'setups_1.stp', ());
#7 = run_setup(#1, 4, $, $, (#8), (), 1, 'setups_2.stp', ());
#8 = path_join_node(#1, 5, $, $, (#9), ());
#9 = end_plan_node(#1, 6, $, $, (), ());
#10 = internal_real(#1, 'setup_total', .T., $, 2.0);
```

C. STEP part 21 ALPS cell plan



D. Graph structure of ALPS cell plan

```
file_names
    setup (this file): 'setups_2.stp'
    part_out: 'not_set'
    fixture: 'fi1.stp'
    workstation process_plan: 'plans_2.stp'
    removal volumes: 'feats_2.stp'
    part_in: 'not_set'
part_out_location:
removal volume location:  → See text
part_in location:
fixture location:
machining_box: (has corners at origin and (35, 75, 100))
inspect only: FALSE
```

E. Structure of second setup file

```
#1 = twist_drilling_ex(#2, #3, 'drill-0.5-2-3', 1);
#2 = twist_drilling($, 2, '12.7 hole in right side',$,($),($),1,(),
     'drill-0.5-2', 0, 2100, 133.4, .T., .F., 12.7);
#3 = round_hole(#4, '12.7 hole in right side', #8,$, #10, #12);
#4 = orientation(#5, #6, #7);
#5 = direction_element((0.0, 0.0, 1.0));
#6 = direction_element((-1.0, 0.0, 0.-));
#7 = location_element((20.0, 20.0, 100.0));
#8 = conical_hole_bottom(.T., #9, $);
#9 = numeric_parameter('drill tip angle', 118.0, 'degrees');
#10 = circular_closed_profile(#11);
#11 = numeric_parameter('drill hole diameter', 12.7, 'mm');
#12 = numeric_parameter('drill hole depth', 25.0, 'mm');
```

F. STEP part 21 drilling command

Fig. 3. Simple example.

The node to mill a pocket includes the name of the type of tool to use, the feed rate, the spindle speed, the horizontal stepover to use, and the vertical pass depth. The plan includes the name of the setup file.

5. The Cell, on learning that the Workstation has completed planning for the first setup, sends a command message to the Workstation telling it to execute that plan.
6. The Workstation, on receiving this message, reads the file plans_1_1work.stp, the file setups_1_temp.stp, and the files named in that setup file. The Workstation sends an open setup command message to the Task, which reads the setup file and gets ready to run. This implicitly includes fixturing the workpiece as specified by the setup file.
7. The Workstation starts traversing the (start-mill-end) plan graph. Traversing the start node results only in checking. When the mill node is reached, the Workstation plans four executable operations to carry out that node: start_nc; change tool; adjust coolant (on); and finish_mill. It then executes these operations in order. Executing the last of the operations results in the pocket being milled. For any operation requiring a tool, a specific tool is selected from the tool inventory. The method of executing an operation is described immediately following.

8. To execute an operation, the Workstation first writes a command file, and then sends a command message to the Task, telling it to generate NC code and giving it the name of the command file. The data section of a sample command file (for the hole drilling operation that occurs later) is shown in Fig. 3F. Each command file describes the operation and the plan step that gave rise to the operation. If the operation is performed on a feature (as the finish milling operation is performed on a pocket, for example) the command file also describes the feature. The Task, on receiving the message, generates an NC-code file and returns a done status message. The Workstation then sends an execute_nc message to the Task, telling it to execute the NC code, which the Task does.
9. The Workstation next handles the end node of its plan, for which it generates two operations, adjust coolant (off) and end_nc. These are executed by Task as described immediately above. That completes execution of the Workstation plan for the first setup. The Workstation then sends a done status message to the Cell.
10. The Cell continues to traverse its plan, first going back to the split node (to see what might be done next) and then deciding to do the second run_setup (item 4 in Fig. 3D). The process outlined in paragraphs 2–9 above is repeated for the second setup, resulting in the hole being drilled. Then the Cell finishes traversing its plan, cleans up, and is ready to do something else.

9. Conclusion

The FBICS architecture provides for planning and control of machining and inspection of discrete parts in a hierarchical control system. The FBICS implementations demonstrate that the architecture, including its specific modularization of manufacturing activities, is usable for industrial application. The implementations also demonstrate the feasibility of feature-based manufacturing and the usability of the ALPS process planning language, STEP AP 224, and STEP data handling techniques and tools, generally. FBICS demonstrates the feasibility and desirability of two-stage planning. FBICS is unusual—possibly unique—in (i) its ability to generate plans for several levels of a hierarchical control system, (ii) its implementation of two-stage planning, and (iii) its use of STEP AP 224 for part modeling.

References

[1] Kramer TR, Huang H-M, Messina E, Proctor FM, Scott H. Feature-based inspection and control system. NISTIR, 2000 (in draft).

[2] Proctor FM, Kramer TR. A feature-based machining system using STEP. Proceedings of SPIE Conference on Sensors and Controls for Intelligent Machining, Agile Manufacturing, and Mechatronics, November, Boston, MA. SPIE 1998;3518:156–63.

[3] Messina E, Horst J, Kramer T, Huang H, Tsai T, Amatucci E. A knowledge-based inspection workstation. Proceedings of the IEEE International Conference on Information, Intelligence, and Systems, Bethesda, MD, 1999.

[4] Kramer TR, Huang H-M, Messina E, Proctor FM, Scott H. An automatic hierarchical process planning system, 2000 (in draft).

[5] Albus JS. A theory of intelligent systems. Control and Dynamic Systems 1991;45:197–248.

[6] Albus JS, McCain HG, Lumia R. NASA/NBS standard reference model for telerobot control system architecture (NASREM): NIST Technical Note 1235. ed. NIST 1989:1989.

[7] Albus JS. A reference model architecture for intelligent systems design, NISTIR 5502. NIST, 1994.

[8] Electronic Industries Association. EIA Standard EIA-274-D interchangeable variable block data format for positioning, contouring, and contouring/positioning numerically controlled machines. Washington, DC: Electronic Industries Association, 1979.

[9] National Center for Manufacturing Sciences. The next generation controller part programming functional specification (RS-274/NGC), draft. NCMS, 1994.

[10] Consortium for Advanced Manufacturing—International. Dimensional measuring interface standard; revision 3.0, ANSI/CAM-I 101-1995. Arlington, TX: CAM-I, 1995.

[11] Kramer TR, Proctor FM. The NIST RS274/NGC interpreter version 2, NISTIR 5739. NIST, 1995.

[12] Kramer TR, Proctor FM, Rippey WG, Scott H. The NIST DMIS interpreter version 2, NISTIR 6252. NIST, 1998.

[13] Proctor FM 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.

[14] Kramer TR, Jun J-S. Software for an automated machining workstation. Proceedings of the International Machine Tool Technical Conference. Chicago, IL: National Machine Tool Builders Association, 1986. p. 1986;12-9:12–44.

[15] Kramer TR. The off-line programming system (OLPS): a prototype STEP-based NC-program generator. Proceedings of a Seminar (Product Data Exchange for the 1990s), vol. 2. New Orleans, LA: NCGA, 1991

[16] Kramer TR. A library of material removal shape element volumes (MRSEVs), NISTIR 4809. NIST, 1992.

[17] Kramer TR, Proctor FM. Feature-based control of a machining center, NISTIR 5926. NIST, 1996.

[18] Albus J.S., et al. NIST support to the next generation controller program. final technical report, 4888. NIST, 1992.

[19] Proctor FM, Kramer TR. Canonical machining commands, NISTIR 5970. NIST, 1997.

[20] Catron B, Ray SR. ALPS—A language for process specification. International Journal of Computer Integrated Manufacturing 1991;4(2):105–13.

[21] Wallace S, Senehi MK, Barkmeyer E, Ray S, Wallace EK. Control entity interface specification, NISTIR 5272. NIST, 1993.

[22] Jurrens KK, Fowler JE, Algeo ME. Modeling of manufacturing resource information, NISTIR 5707. NIST, 1995.

[23] ISO 10303-11: Industrial automation systems and integration—product data representation and exchange–Part 11: the EXPRESS language reference manual. Geneva, Switzerland: ISO 1994:1994.

[24] ISO 10303-21: Industrial automation systems and integration—product data representation and exchange—Part 21: clear text encoding of the exchange structure. Geneva, Switzerland: ISO 1994:1994.

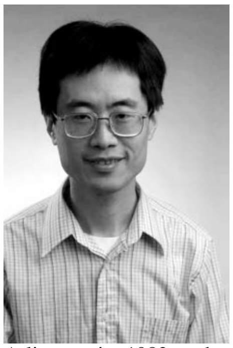
[25] ISO 10303-224: Industrial automation systems and integration—product data representation and exchange—Part 224: application protocol: mechanical product definition for process planning using machining features. Geneva, Switzerland: ISO 1998:1998.

[26] Anderson DC, Chang TC. Geometric reasoning in feature based design and process planning. Computers & Graphics an International Journal 1990;14(2):225–35.

[27] Anderson DC, Chang TC. Automated process planning using

object-oriented feature based design; advanced geometric modelling for engineering applications. North-Holland, 1990.
[28] Anderson D. Purdue quick turnaround cell. http://cadlab.www.ecn.-purdue.edu/qtcplm 1999.
[29] http://cybercut.berkeley.edu U. C. Berkeley Cybercut website.
[30] Smith CS, Wright PK. Cybercut: a World Wide Web based design-to-fabrication tool. Journal of Manufacturing Systems 1996;15(6):432–42.
[31] Wang F-CF, Wright PK, Web-based CAD. tools for a networked manufacturing service. Proc. of the ASME Design Engineering Technical Conference, Atlanta, GA 1998:1998.
[32] Judd RP et al. The intelligent machining workstation. Prototyping Technology International 3:52–5.
[33] Naish JC, Mill FG, Salmon JC. An industry-based study of cutting process capability representation requirements for an integrated engineering workstation. IIE Transactions 1997;29:573–84.
[34] Leler W, Merry J. 3D with HOOPS. Addison-Wesley, Reading, MA, 1996.
[35] Shackleford W. Real-time control systems library documents. http://isd.cme.nist.gov/proj/rcs_lib 1996.
[36] Horst JA. Real-time and object-oriented issues for an inspection workstation application. Proceedings of the Fifth International Workshop on Object-Oriented Real-Time Dependable Systems, 1999.

Thomas Kramer is a Guest Researcher in the Intelligent Systems Division at the National Institute of Standards and Technology and a Research Associate of the Catholic University of America. He has worked full-time at NIST since 1984. Mr. Kramer has been developing concepts and software for feature-based manufacturing systems since 1986. His primary research interest is automated reasoning. At NIST he has also worked on control architectures, interpreters for machine control languages, and data standards related to manufacturing. He received a BA in Physics from Swarthmore College in 1965 and an MA and PhD in Mathematics from Duke University in 1968 and 1971, respectively.



Hui-Min Huang is a Mechanical Engineer with the National Institute of Standards and Technology (NIST), US Department of Commerce. His major research areas include the architectures and software engineering tools and methodologies for real-time, intelligent control systems. His prior professional experiences include technical positions with the Advanced Technology and Research Company (Burtonsville, MD), the Singer Company Link Simulation Division (Silver Spring, MD) and the Timex Watches Company (Taiwan). Mr. Huang received an MS degree in Mechanical Engineering from the University of Texas at Arlington in 1982 and an MS degree in Computer Science from the Johns Hopkins University in 1986.

Elena Messina is the Group Leader of the Knowledge Systems Group in the Intelligent Systems Division at the National Institute of Standards and Technology. She is also Program Manager for Research and Engineering of Intelligent Systems. Her work responsibilities include knowledge representation and planning for control, establishment of performance metrics for intelligent systems, and development of software architectures, tools, and methodologies for building controllers. Current testbeds include an autonomous scout vehicle and an inspection workstation with feature-based planning, control and part localization. Prior to joining NIST, she was Manager of Geometric Modeling for the I-DEAS Master Series CAD/CAE Software at the Structural Dynamics Research Corporation. Ms. Messina also worked at Cincinnati Milacron on their industrial robots, where she received two patents for her arc welding-related enhancements. She received a BS in Engineering Science from the University of Cincinnati.

Harry Scott is the project manager for the Reference Model Architecture project within the Manufacturing Engineering Lab at NIST. This program addresses interoperability issues in intelligent control systems, in part, through the development and testing of a reference model for design and implementation of real-time, intelligent systems. Prior work has included integration of the Inspection Workstation Testbed into the National Advanced Manufacturing Testbed (NAMT) CIM Framework, wireless communication system implementations for DOD and DOT intelligent vehicle programs, a DOT investigation into vehicle-to-roadside communication technologies, and control system development and integration for the Advanced Manufacturing Research Facility at NIST in the 1980s. Mr. Scott holds a BS in Electrical Engineering from the University of Maryland.

Frederick Proctor is the Group Leader of the Control Systems Group at the National Institute of Standards and Technology. He received a BS in Electrical Engineering from the Univesity of Maryland in 1986, and an MS in Computer Science from the John Hopkins Univesity in 1993. His research interests include control of manufacturing systems, real-time software, and validation testing for open architecture control standardization. Recently, Mr. Proctor has been contributing to an open-source project on computer numerical control and programmable logic controllers, in collaboration with small manufacturers.