# Analysis of Dimensional Metrology Standards

by the

**NIST Manufacturing Engineering Laboratory**

Contributors:

John Evans
Simon Frechette
John Horst
Hui Huang
Thomas Kramer
Elena Messina
Fred Proctor
Bill Rippey
Harry Scott
Ted Vorburger
Al Wavering

## Disclaimer

Commercial equipment and materials are identified in order to specify certain procedures adequately. In no case does such identification imply recommendation or endorsement by the National Institute of Standards and Technology, not does it imply that the materials or equipment identified are necessarily the best available for the purpose.

## Acknowledgements

Partial funding for the writing of this paper was provided to Catholic University by the National Institute of Standards and Technology under cooperative agreement Number 70NANB7H0016.

## Abstract

This is an analysis of standards related to dimensional metrology.

## Keywords

dimensional, DMIS, inspection, metrology, NIST, numerical control, process planning, standard

# CONTENTS

# 1 Executive Summary

This paper is an analysis of standards related to dimensional metrology information, with recommendations regarding standards development. The analysis focuses on the degree to which existing and developing standards provide a complete set of non-overlapping specifications for information needed to perform automated dimensional metrology. The analysis was prepared in the Manufacturing Engineering Laboratory at the National Institute of Standards and Technology. The intent of the analysis is to assist users, vendors, and standards developers in the area of dimensional metrology.

The most pressing current needs are:

1. Continue work on DMIS. DMIS is at the key interface between programming and execution for users of dimensional measuring equipment. DMIS particularly needs fixed conformance classes, conformance testing, and conformance certification. Current work on DMIS Part 2 and Version 4.0 should be pursued to completion. See, Section 4.2.2, Section 6.4 and Section 6.5.

2. Continue work on the CMM-driver specification and harmonize it with the DmeEquip portion of DMIS Part 2 and European I++ program efforts. This API will be very useful for builders of controls for dimensional measuring equipment (particularly CMMs), which receive and execute API commands. The API should be equally useful to builders of software for executing high-level inspection programs, since their software makes calls to commands in the API. In both cases a standard specification will simplify builders' work by replacing a number of differing APIs with a single API. Unless a user of dimensional measuring equipment is also building controls or writing drivers, this API will not be used directly by that user. See Section 4.2.4 and Section 6.9.

3. Work towards a standard 3D shape representation that includes tolerances. STEP AP 203, which uses boundary representation geometry, is adequate for nominal shape but needs to have tolerances added. A new version of STEP AP 203 should be created that includes tolerances. STEP AP 224, which is feature-based, does include tolerances and can be used for many parts, but is not able to represent a full range of shapes. See Section 4.2.1 and Section 6.2.

4. Work towards a standard representation for measurement data which is input to the reporting and analysis activity. The bulk of the data is point data, but other data, such as feature data, is also required. See Section 4.2.3.

5. Complete the Application Reference Model for STEP AP 219 in the form required for a committee draft as a Technical Specification under STEP. STEP AP 219 will provide a data model and format for the transfer of inspection planning data. Develop an XML-based interchange format based on this model for transfers of dimensional inspection information. See Section 4.2.5 and Section 6.7.

6. To the extent possible, the metrology community should work to avoid creating or endorsing multiple standards for the same purpose. Where standards overlap, they should be harmonized to the extent that, in the area of overlap, there is a one-to-one mapping from one standard to the other. Current work to harmonize DMIS and STEP AP 219 should continue to the point where a

one-to-one mapping is defined for the area of overlap. STEP AP 219, in turn, should be harmonized with STEP AP 224 and STEP AP 214.

Some major observations:

1. When defining standards, the dimensional metrology community should move towards methods in which the information content is separate from the file format or other exchange specification. A single information model can be used with many different file formats or database access and communication methods.

2. TCP/IP should suffice for communications in the foreseeable future. This will work for both Internet and intranet systems.

3. The metrology community should anticipate that computer software modules may be separated in three degrees: (1) both modules in the same process, (2) modules in different processes on a single computer (or different computers sharing physical memory), (3) modules in processes on different computers. Basic standards for one module using another should assume both modules are in the same process, and, hence, be written as APIs. Any API may then be wrapped in a messaging system for processes on the same computer, then wrapped again in a communications system for processes on different computers.

# 2 Introduction

## 2.1 Focus

This is an analysis of standards related to dimensional metrology information. The analysis focuses on the degree to which existing and developing standards provide a complete set of non-overlapping specifications for information needed to perform automated dimensional metrology. Information, here, means anything that can be transmitted as electronic bits, including both commands and data. We are using the term "standards" loosely to mean national and international standard formats, open proprietary standard formats, de facto non-proprietary standards, and non-standard formats in common use. This analysis covers all of those.

Section 3 describes the entire dimensional metrology function in terms of a set of activities and the software modules that perform them. Section 4 discusses the software systems into which the modules are typically divided and makes recommendations regarding the interfaces that are most important now. Section 5 discusses languages that might be used for defining standards or for standardized exchange of data. Section 6 looks at APIs and data formats for dimensional metrology, including an assessment of the situation regarding formats for each type of data or commands identified. Appendix A discusses in detail the modules of Section 3, their interfaces, and the types of data and commands that pass over the interfaces.

The dimensional metrology function of a company is tied tightly to the broader function of assessing product quality and production system performance. Thus, data for dimensional metrology must often be compatible with data used in or produced by other assessment activities. The dimensional metrology function must also work with other company functions, particularly design, resource management, and scheduling, so data compatibility with those functions is also needed. In this analysis we identify data shared with these other functions.

## 2.2 Purpose

The purpose of the overall activity of which this analysis is a part is to make it easy to assemble and operate a dimensional metrology system using components from various vendors.

Another purpose of the overall activity is to allow a manufacturer to acquire and store inspection data using a variety of types of dimensional inspection equipment and have each type of data be in the same format regardless of the equipment used to acquire the data.

The above implies that equipment of a given type from different suppliers should produce data in the same format.

## 2.3 Scope

### 2.3.1 In Scope

The activities and hardware described below are in scope. We are interested in the data used in or produced by the activities, and the commands and responses needed to conduct the activities.

We assume that almost all activities are conducted with the help of PC type (or slightly more powerful engineering workstation) computers. Any number of computers may be involved.

*2.3.1.1 Activities*

Dimensional metrology for rigid discrete objects (e.g. engine blocks and turbine blades) and semi-rigid objects (e.g. sheet metal panels for automobiles and aircraft) used in the automotive and aerospace industries are in scope.

This scope encompasses the dimensional metrology activities of many other types of companies besides those strictly in the automotive and aerospace sectors. For example, the dimensional metrology activities of machine shops of all sizes fit in this scope.

Simulation of dimensional metrology (executing a DMIS program in simulation, for example) is in scope.

Methodologies that make it feasible to handle diverse types of metrology data in a database system are in scope.

*2.3.1.2 Hardware*

The following types of devices for taking dimensional measurements are in scope:

1. hand-held electronic dimensional measurement devices (calipers, micrometers),
2. CMM (see below),
3. CNC machining center with touch probe,
4. photogrammetry equipment,
5. sensor (touch probe, non-contact probe, surface roughness sensor),
6. theodolite.

Devices which produce 3D point data in real time are considered to be CMMs in this analysis. This includes devices with mutually orthogonal physical axes (traditional CMMs) or with articulated arms. It includes laser trackers. A CMM may be fixed or portable and its axes may be positioned automatically or manually.

**2.3.2 Out of Scope**

Analysis and simulation activities not directly tied to dimensional metrology are out of scope. Finite element analysis, kinematic analysis, and throughput analysis by simulation, for example, are all out of scope.

Data used by systems performing activities that are not dimensional metrology are out of scope if such data is not shared with systems performing dimensional metrology. Such out-of-scope data includes, for example, finite element mesh data, kinematics data, and throughput data.

Activities and data for systems performing other types of metrology are out of scope. Out-of-scope data includes, for example: hardness, strength, color, density, grain structure, chemical

composition, electrical properties, and electronic properties. The activities that gather and analyze such data are out of scope.

Dimensional metrology of non-rigid bodies (ropes and clothing, for example) is out of scope.

## 2.4 Overall System Considerations

Automated metrology components and systems should be intranet and internet compatible. To do this, expect a mix of net languages (XML, Java, VRML), non-net format files transmitted via the net, database transactions via the net, and non-net format commands transmitted via the net.

Communications and database systems are the infrastructure on top of which an integrated metrology system runs.

We assume that all communications between computers are performed using ethernet.

Compatibility with services and utilities that make it easy to move communicating processes from running on the same computer to running on different computers needs to be considered. The NIST Intelligent Systems Division's NML communications system [Shackleford] is a utility of this sort. The availability of tools to make communicating processes run over internet or intranet is increasing.

We assume that all computers performing dimensional metrology are connected to a database via ethernet. The database may use files, or it may use other techniques. The database may reside on a single central server, or it may be a system of networked servers. To serve a dimensional metrology system, the database should be able to store or retrieve a set of related information (such as a file) by name. Names may, of course, be pathnames including directory names and possibly machine names.

# 3 Activities, Modules, and Interfaces

## 3.1 Activities, Software Modules, and Software Systems

In this analysis, an activity is a process performed to accomplish a goal. Making an inspection plan, for example, is an activity. The overall activity of performing dimensional metrology may be broken down into subactivities, some subset of which occur whenever any dimensional metrology activity is performed. Some subactivities involve only information, and some involve both information and dimensional measuring equipment hardware. We will not use the term subactivity much in the remainder of this analysis, but keep in mind that any activity may be a subactivity of some larger activity and may itself consist of smaller subactivities.

Each activity is performed either entirely by a computer or by a person with the help of a computer. For hand-held dimensional measuring equipment, the computer may be a chip embedded in the equipment. Where person and computer are both involved, any division of labor is possible; the computer may just act as a typewriter with the person doing the thinking, or the computer may do all the work with the person watching.

A software module includes at least a library of compiled (or compilable) computer code with an application programming interface (API). The activities identified in Section 3.2 correspond one-to-one with software modules. When a module executes, it performs the corresponding activity.

One or more software modules may be combined with supporting code (for graphics, user interface, and possibly other functions) in a software system. Almost all software systems used for dimensional metrology run on a single computer. Some software systems are contained in a single computer process. Other software systems execute in multiple processes connected by interprocess communication. A software system is what a software vendor usually sells. Software systems may be combined to form larger systems. Section 4 deals with software systems more fully.

Although it is anticipated that communications among modules will usually be done electronically, communications may be performed by moving a physical medium (such as a tape, a CD, a floppy disk, or even paper) from one place to another.

The activity breakdown described here is intended partly to match the module breakdown found in commercial software systems for dimensional metrology and partly to be a recommended module breakdown for software systems built in the future.

The activities identified here are those used for dimensional metrology for discrete parts manufacturing. Some of these activities differ between manufacturing using fixed-program transfer lines (where each part follows the same route and each machine does the same operation over and over on the same kind of part) and manufacturing more flexibly (where each machine is used for different types of parts at different times). Where there is a difference, this is noted in the detailed activity description given in Appendix A.

### 3.1.1 Connectability

Whenever software module M1 uses the services of module M2, M1 and M2 may be separated from one another in three degrees, as follows.

1. Both modules run in the same computer process. In this case, M1 makes calls to functions in M2's API, and M2 returns status and/or data to M1 as specified by the API. If M2 controls hardware, some of the API function calls make the hardware execute.
2. M1 runs on the same computer as M2, but in a different computer process. In this case, the API function calls made by M1 and executed by M2 are wrapped in an interprocess messaging scheme. The messaging scheme provides for any returned value or status from M2 to M1 (in response to a message from M1) to be sent back in a message.
3. M1 runs on a different computer from M2. In this case, each call to a function in M2's API is wrapped in a message and the message is sent between computers by a communications system. With some messaging schemes (NIST's NML messaging scheme, for example), this case (separate computers) and the preceding case (same computer, separate processes) are handled almost the same by the system builder. As in the previous case, the messaging scheme provides for any returned value or status from M2 to M1to be sent back in a message.

We assume there is a file system and/or database system to which every process with a need has access. Sets of related data (files, for example) identified by name may be put into the system and obtained from it.

## 3.2 Dimensional Metrology Activities Identified

The activities of performing dimensional metrology (listed alphabetically) are[1]:

1. Activity Coordination (deciding what other activities happen, and when)
2. CAD (creating designs with the help of a computer)
3. Hand-held Device Measuring (e.g., using calipers with electronic readout)
4. High-level Inspection Instruction Execution (e.g., executing a DMIS program)
5. Inspection Planning (deciding what to inspect and how to inspect it)
6. Inspection Programming (e.g., writing DMIS programs)
7. Low-level Inspection Instruction Execution (e.g., executing CMM-driver commands)
8. Machining Planning (deciding what to inspect when on-machine inspection is used)
9. Machining Programming (writing NC code for on-machine inspection)
10. Math Computing (feature fitting, statistical calculating, etc., excluding solid modeling)
11. Other Inspection Device Control (controlling theodolite or photogrammetric equipment)
12. Probe Instruction Execution (e.g. executing NGIS standard sensor interface commands)
13. Reporting and Analysis (reporting and analyzing data resulting from inspection)
14. Routing Planning (deciding which workstations[2] work-in-process goes to)
15. Solid Modeling (maintaining and querying a 3D model of workpieces and equipment)

1. These activity names are spelled with initial caps throughout this analysis.

Although displaying graphics is expected to be performed in many parts of the system, it is not included in the list above. This is because an interface to a graphics display system is not specific to dimensional metrology (or any other domain). Thus, defining a standard interface to graphics requires that a far broader community be involved. Graphics capabilities are generally built into commercial software systems, anyway, so the lack of a standard should not cause any difficulties for dimensional metrology.

Interacting with a user is another activity performed throughout dimensional metrology systems that is not included in the list. Standards might be developed or adopted for the look and feel of user interfaces, but that issue is not tackled in this analysis. It is too big an issue to be decided within the dimensional metrology community.

A brief description of each activity follows. The interfaces of each activity are discussed in Appendix A.

### 3.2.1 Activity Coordination

Activity Coordination is the process of (1) planning what other activities will take place and when they will take place and (2) assigning resources (fixtures, tools, people, etc.) to planned activities, and (3) giving the orders necessary to carry out the plans. Scheduling is included in Activity Coordination.

### 3.2.2 Computer-Aided Design (CAD)

CAD is the process of producing a design using a computer. There are many commercial CAD systems. Dimensions and tolerances are specified during the CAD process that drive downstream dimensional metrology activities.

The term “solid modeler” is often used to apply to CAD systems. As commonly used to describe a CAD system, it means one that makes a solid model (as opposed to a wire frame, or a facetted model, etc.). This analysis does *not* use that meaning. Our meaning for “solid modeler” is given in Section 3.2.15 and is a synonym for the term “solid modeling kernel”, often used elsewhere, but not used in this analysis.

### 3.2.3 Hand-held Device Measuring

Hand-held Device Measuring is the process of measuring things with a hand-held device. Hand-held devices are things such as electronic calipers. Typically, such a device may be carried around and is controlled by the person who carries it. These devices are included in this analysis because they produce data which is returned electronically to the database. If a hand-held dimensional measuring device does not produce electronic data directly (a manual micrometer, for example), but a human reads the device and enters data on an electronic device, the hand-held measuring device is intended to be covered by this analysis.

---

2. As used in this analysis, a “workstation” is a working area including a principal machine (such as a CMM, lathe, or machining center) for processing work-in-process, and possibly including associated equipment (such as a storage unit or powered vise).

### 3.2.4 High-level Inspection Instruction Execution

High-level Inspection Instruction Execution is the process of executing high-level inspection instructions, such as statements from a DMIS program. High-level inspection instructions may define things such as features, tolerances, and datums or may give motion control directives. The motion control directives may be high-level (such as measure a feature) or low-level (such as set a feed rate or measure a point). The high-level instructions may come from a file or from a user.

### 3.2.5 Inspection Planning

Inspection Planning is the process of deciding what to inspect and how to inspect it (which equipment, how many points, etc.), given the design for a part. Inspection Planning may also involve computing estimated uncertainties for making specific measurements using different equipment and techniques. In existing software systems, Inspection Planning may overlap with Routing Planning and/or Inspection Programming.

### 3.2.6 Inspection Programming

Inspection Programming is the process of producing a high-level inspection program, a DMIS program, for example. In existing software systems, Inspection Programming may overlap with Inspection Planning.

### 3.2.7 Low-level Inspection Instruction Execution

Low-level Inspection Instruction Execution is the process of executing low-level inspection instructions. This activity is usually carried out in the same box that sends drive signals to axis actuators. Low-level inspection instructions deal with low-level data (such as set or get search distance) or with low-level motion (such as probe a point). Low-level inspection instructions do not include defining things and do not include high-level motion control directives (such as measure a feature).

### 3.2.8 Machining Planning

Machining planning is the process of deciding what operations should be performed on a machine tool to produce a part. Many machining centers can handle touch probes, so inspection can be performed. This analysis deals with inspection operations that may be performed on a machine tool, but not with other machine tool operations.

### 3.2.9 Machining Programming

Machining Programming is the process of generating a program that may be run on a machine tool controller to cut a part. This analysis deals with inspection functions that may be programmed on a machine tool, but not with cutting and other machine functions.

### 3.2.10 Math Computing

Math Computing is the process of performing mathematical calculations. It is expected that only relatively sophisticated, difficult, or time-consuming calculations will be performed in Math

Computing, since simple math can be done in most any computer program. Such calculations would include fitting features to sets of points, for example. It is also expected that Math Computing will not perform solid modeling, since that activity is sufficiently difficult to require a separate Solid Modeling module.

### 3.2.11 Other Inspection Device Control

Other Inspection Device Control is the process of controlling other dimensional metrology equipment, such as theodolites and photogrammetry equipment.

### 3.2.12 Probe Instruction Execution

Probe Instruction Execution is the process of executing instructions sent to a probe sensor carried by a CMM.

### 3.2.13 Reporting and Analysis

Reporting and Analysis is the process of collecting inspection reports, analyzing data returned from inspection activities, and generating files and graphical representations of analyzed or unanalyzed data. It is expected that the results of Reporting and Analysis will either be presented directly to humans or be passed to activities outside the scope of this standards analysis. Reporting and Analysis may include statistical process control activities.

### 3.2.14 Routing Planning

Routing Planning is the process of deciding which fabrication and inspection activities will take place at which workstations. The output of Routing Planning is a plan for the routing of a workpiece among workstations specifying what work should be done at each workstation. Routing Planning may also generate specifications of how the workpiece is to be set up in each workstation and CAD designs for intermediate workpiece shapes.

There are at least two other common meanings of routing planning which this is not. This is not deciding on paths for pipes or cables to take. This is not planning for cutting with a router.

### 3.2.15 Solid Modeling

Solid Modeling is the process of building a representation of solid objects (usually via a boundary representation) and performing calculations done on solid objects, such as determining the mass of an object, determining if a given point lies on the surface of an object, or boolean subtracting one object from another. The Solid Modeling activity is usually performed by a software system called a solid modeler.

## 3.3 Simulation

Simulation of dimensional metrology activities has not been listed among dimensional metrology activities. This is not because it is not important or useful; it certainly is. Simulation is more a mode of conducting an activity than it is an activity and therefore does not affect interfaces, since it must use the same interfaces that the real activity uses. Simulation occurs when any executing

activity (or collection of activities) is playing make-believe rather than dealing with the real world. To conduct a simulation, all that is necessary is for a faker to be on one end of an interface. A simulation is usually conducted by reconfiguring the system before the simulation starts. One or more faking modules are swapped in and the real ones swapped out. Modules downstream of a fake module may be removed entirely.

Some or all modules participating in a simulation may be the same when real work is being done as when a simulation is being done, because they need to behave exactly the same in either case.

A simulation of running a program to inspect part X might be conducted in the absence of the part by having a human stand by the CMM and trip the probe with a finger. In a simulation of this sort, the entire system is the real system. Only the object being inspected is faked.

A simulation of the same program for probing part X might be conducted at a higher level by swapping the real Low-level Inspection Instruction Execution module for a fake one that feeds fake data back to High-level Instruction Execution. In this case, no Probe Instruction Execution module executes at all.

Almost all commercial software systems for dimensional metrology include some type of simulation. For Inspection Programming and Machining Programming, the simulation is usually of high-level instruction execution and shows on a computer monitor a picture of the process with the workpiece and tool.

## 3.4 Interfaces and Architecture

The focus of this analysis is on interfaces between modules. For efficient automatic operation, each interface must be formally defined. To help identify the interfaces, we present an architecture in this section. By the architecture of a system, we mean a description of which software modules are connected to which other software modules by an interface. Figure 1 shows a dimensional metrology system architecture that might be used. The figure includes the 15 software modules for the activities listed in Section 3.2. The software modules shown on the figures may be separated in any of the three ways described in Section 3.1.1. User interfaces are not included in the figure, but are expected to be present.

The presented architecture is not intended as a recommended architecture, but only as a representative architecture showing all the interfaces.

Appendix A discusses each of the fifteen software modules individually, including a diagram of its interfaces to other modules (as shown in Figure 1) and a discussion of each interface.

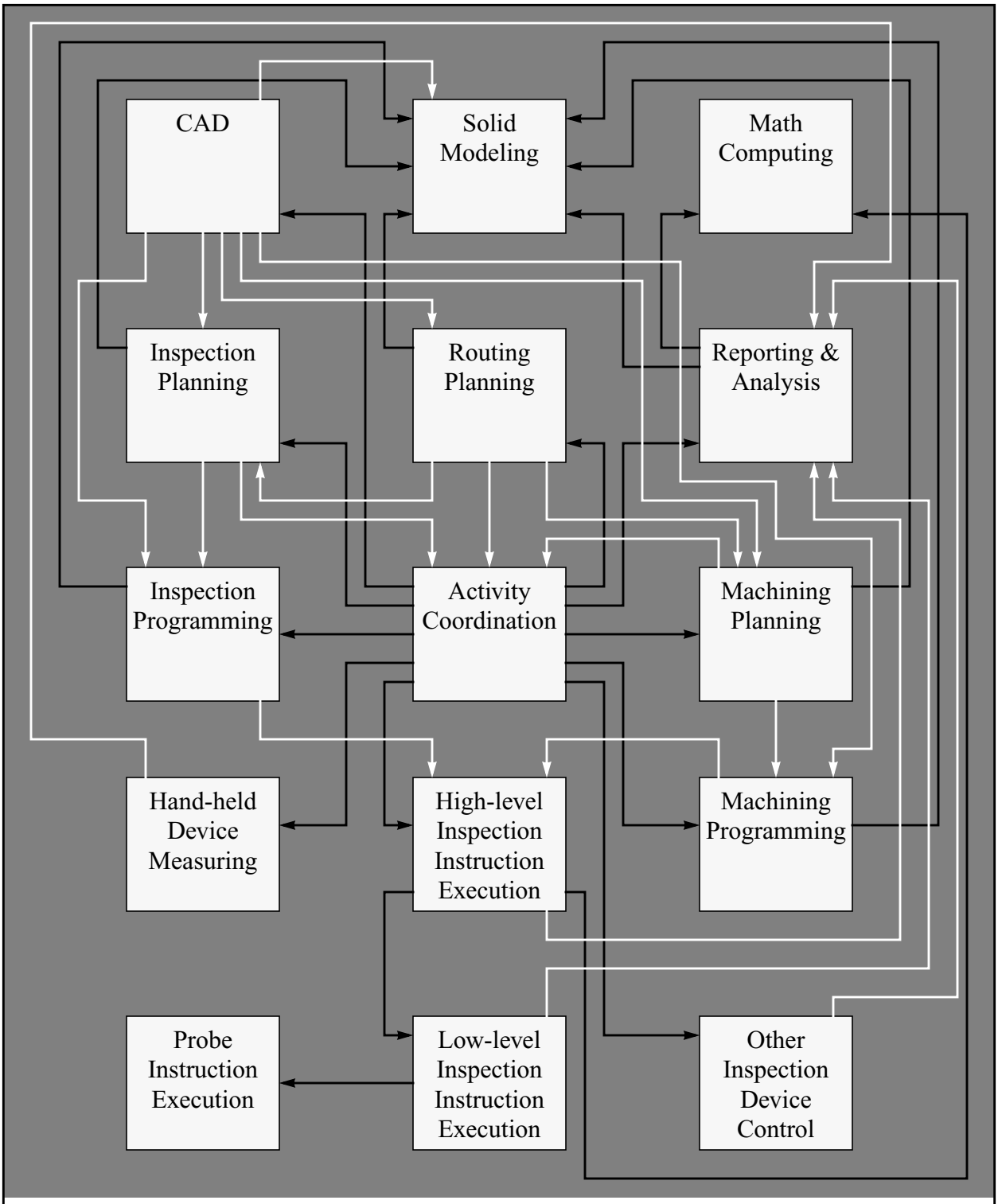


**Figure 1 Interfaces in a Dimensional Metrology System Architecture**

White arrows show producer → consumer for data interfaces.
Black arrows show user→used for active interfaces.

### 3.4.1 Active Interface

An active interface is one in which a command is given. The relationship between the two parties attached by an active interface may be either supervisor-subordinate or client-server. In both cases there is a user module and a used module. Active interfaces are shown on Figure 1 with black arrows leading from the user module to the used module. Any of the three degrees of separation discussed in Section 3.1.1 may occur between the user and used modules. The black arrows are shown going in the direction from user to used. As discussed in Section 3.1.1, status or data is expected to be sent back in the reverse direction to the user module from the used module.

In systems that are not fully automated (all existing systems) some of the active interfaces are between two humans. These interfaces will be called person-to-person interfaces. In this document, person-to-person interfaces are assumed to be implemented using natural language, spoken or written; the boss tells the worker what to do. Person-to-person interfaces could be implemented using a formally defined language, but this is not done.

Some other active interfaces are between two software modules. A formally specified and automated interface is required.

The remaining active interfaces are between a human user and a software module. In this case, the user interface with which the human interacts may be assumed to send commands to the used module in the same format a software user module would have employed.

### 3.4.2 Data Interface

A data interface is one in which a producer puts data into a file system or database system, and a consumer retrieves data from the file system or database system. Although there is no direct contact between producer and consumer, both must have the same understanding of the format and meaning of the data. Data interfaces are shown on Figure 1 with white arrows leading from the producer software module to the consumer module.

Unlike the black active interface arrows, no status or data travels back along the white data interface arrows.

### 3.4.3 Distinction May be Blurred

A command passing over an active interface may include a pointer (such as a file name) to data. The used module must then retrieve the data from a file system or database. In these cases, the data is not considered to pass over the active interface. Rather, there is considered to be a data interface between the source of the data (usually different from the user module that gave the command) and the used module.

If a command includes data (such as the “1, 2, 3” in “GOTO 1, 2, 3”), the data is considered to be part of the command and no data interface is identified.

The distinction between the two types of interface is easily blurred. A simple form of this is to send commands via a file in the database that is re-written each time there is a new command. The commanding software module writes the file whenever it wants to send a command, and the

commanded module reads the file each time the file is updated. Conversely, it is also possible to send large amounts of data in a command-and-status API. This analysis identifies the types of interface commonly used now or most likely to be used in the future.

### 3.4.4 Independence of Architecture and Interfaces

Interfaces can be made independent of architecture.

A module writing data to a file system or database does not know which modules might use the data. A module reading data from a file system or database does not know what module put it there. Thus, data interfaces are independent of architecture. Of course, the content of the data must be such that it serves the purposes of the modules that use it, so consideration must be given to possible uses when the information model for the data is defined, but that is only a loose connection to architecture.

Each module has certain capabilities and an API is written that allows those capabilities to be used. Any other module might make calls to that API. Thus, active interfaces are independent of architecture. As with data, consideration must be given to what capabilities other modules might need when the API for a module is built, but this is also only a loose connection to architecture.

### 3.4.5 NIST ISD Implementation

Thirteen of the fifteen activities shown on Figure 1 have been implemented within two workstations of the NIST Intelligent Systems Division (ISD): the Inspection Workstation [Messina] and the EMC Bridgeport Machining Workstation. Both workstations have a multi-level RCS hierarchical control architecture and use the Feature-Based Inspection and Control System (FBICS) for the upper two or three levels of the hierarchy [Kramer]. The two activities from Figure 1 not implemented in either ISD system are Hand-held Device Control and Other Inspection Device Control.

Almost all the interfaces shown on Figure 1 (apart from those to the two unimplemented activities) are implemented in these ISD systems and are defined formally. The Activity Coordination activity is distributed in these ISD systems, not centralized. Thus, while Figure 1 includes the interfaces used in the NIST systems, it does not show their architecture. In the two ISD systems, instances of all three degrees of separation described in Section 3.1.1 may be found.

# 4 Software Systems

Understanding the software modules and interfaces for dimensional metrology presented in Section 3 and Appendix A is essential for systems builders and for formulating standards, but may not be directly useful to users. This section focuses on software systems a user might buy today and the interfaces between them.

## 4.1 Systems

Where several modules are combined in a commercial software system, the interfaces between those modules become invisible to the user. The interfaces between systems, however, are exposed. From a user perspective, it is important that there be effective standards for these exposed interfaces. To the extent that different commercial system encompass different sets of modules, however, different sets of interfaces become exposed. The primary groupings of modules into systems and the most important associated interfaces are shown on Figure 2. The figure lists the names of several major commercial systems (or systems makers) of each type[1]. The figure also shows the hot interfaces — those for which having a standard is now of high priority.

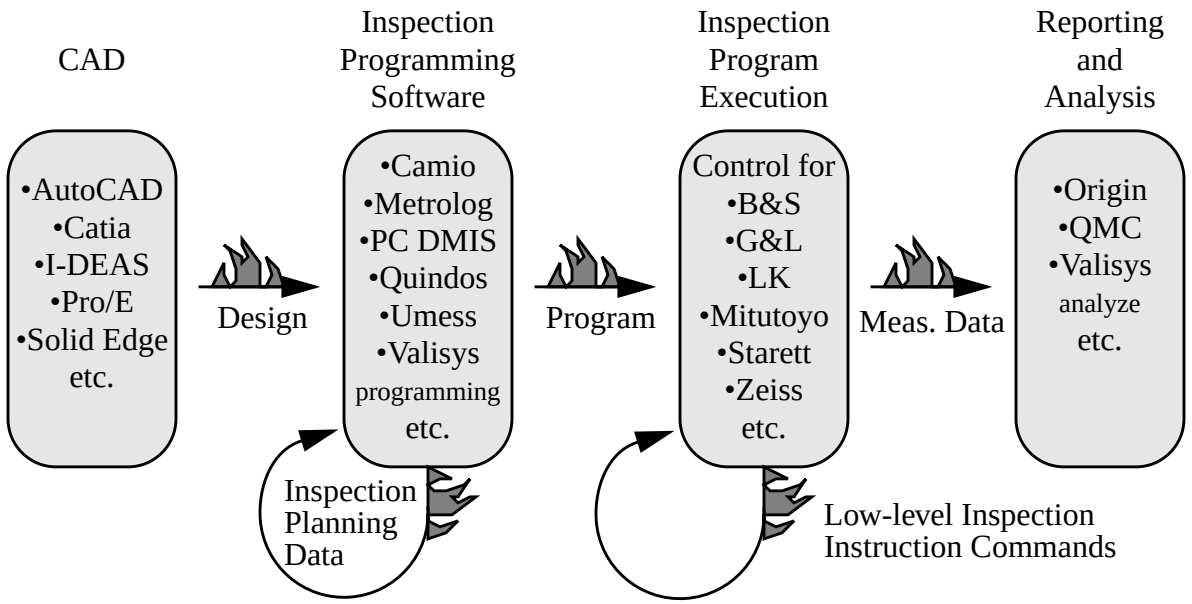


**Figure 2 Metrology Automation Major Systems and Hot Interfaces**

The CAD system of Figure 2 has the same function as the CAD module of Figure 1. A solid modeler is assumed to be part of the CAD system.

1. The Valisys systems listed on Figure 2 are now Tecnomatix systems with different names.

The CMM Programming Software system of Figure 2 includes the functions of the Inspection Planning and Inspection Programming modules of Figure 1. Commercial versions of this system always include the capability to simulate the execution of a CMM program graphically (as a picture on a computer monitor) and may include collision checking and simulated data output and analysis. Thus, many other modules from Figure 1 may be involved, but the involvement of other modules does not matter to the user, since their interfaces are not exposed and the end output of the system is still a program for a CMM. The interface of Inspection Planning Data between the Inspection Planning and Inspection Programming modules, which is actually internal to the Inspection Programming Software system, is shown on Figure 2 because of its current importance. STEP AP 219 is expected to provide this interface.

The CMM Program Execution system of Figure 2 includes the functions of the High-level Inspection Instruction Execution, Low-level Inspection Instruction Execution, and Probe Instruction Execution modules of Figure 1. Some commercial versions of this system include the Reporting and Analysis module (and the modules it uses) as well. Most commercial versions enable the export of point data to a separate Analysis and Reporting system, even if they encompass one themselves. The interface of CMM-driver commands between the High- and Low-level Inspection Instruction Execution modules, which is actually internal to the CMM Program Execution system, is shown on Figure 2 because of its current importance.

The Reporting and Analysis System of Figure 2 includes the functions of the Reporting and Analysis, Math Computing, and Solid Modeling modules of Figure 1.

Although the systems shown on Figure 2 have the functionality described above in terms of the functions of modules from Figure 1, actual commercial systems are not necessarily divided into those modules. They may be modularized some other way. They may not be modularized at all, though that is very unlikely.

## 4.2 Hot Interfaces

Five currently hot interfaces are shown on Figure 2. **These are high-priority interfaces** which should be the focus of attention now and in the near future.

### 4.2.1 Design

Designs pass from CAD to the Inspection Programming system. This interface is a tower of Babel. Details are given in Section 6.2. The situation is very bad but has persisted for so long and has received so much attention that people are used to it. STEP is making some headway with this interface but needs improvement. The most pressing need here is to get tolerances into STEP AP 203.

### 4.2.2 Program

Inspection programs pass from the Inspection Programming system to the Inspection Program Execution system. Here again there are many commercial proprietary languages and a standard, DMIS. The course of action here is to strengthen DMIS. Details are given in Section 6.4.

### 4.2.3 Measurement Data

The bulk of measurement data is point data, but Reporting and Analysis may also need other data (nominal point and feature data, for example) it cannot get from the CAD model but can get from CMMs and other measurement devices. Work on standardizing measurement data is just beginning. Details are given in Section 6.17.

### 4.2.4 API for Low-level Inspection Instruction Execution

The active interface inside the Inspection Program Execution system between the High-level Inspection Instruction Execution module and the Low-level Inspection Instruction Execution module is the focus of three standardization efforts currently in progress (the CMM-driver Specification, the DmeEquip module of DMIS Part 2, and some part of the European I++ effort). **Harmonizing these efforts is of high priority**. It would be very unfortunate to have multiple standards for low-level inspection instructions. This work should continue until a single formal standard is established. Further details are given in Section 6.9.

### 4.2.5 Inspection Planning Data

STEP AP 219, Dimensional Inspection Information Exchange, is being developed now. AP 219 is expected to provide data that may be used for inspection planning. The scope of AP 219 includes planning, execution, and analysis of dimensional inspection activities, so AP 219 may have an impact on other interfaces identified in this analysis. Details are given in Section 6.7.1.

# 5 Languages

This section describes formal languages relevant to dimensional metrology. It is necessary to understand the choices to be made among languages before tackling specific interfaces. The choice of language for a data standard or API makes a great difference in how clearly the standard or API can be stated, how easily it can be understood, and how easily it can be implemented. The choice of language, therefore, is a large factor in determining how widely accepted an API or data standard will be.

## 5.1 Separate vs. Combined Information Model and Representation

A language may provide for modeling both (a) the information content of statements and (b) representations of statements, or it may specify only the information content. If only information content is specified, then an additional specification is needed for how to make representations. An example is given in Appendix C

Languages which separate the two enjoy a great advantage over languages which do both. Software tools for working with languages which separate information content from representation are widely available. Such tools are enabled by the fact that, working from information models, it is possible, given a specific information model, to produce source code automatically for dealing with data corresponding to the model. Moreover, tools may be built for any computer programming language for which a binding from the information modeling language has been defined.

## 5.2 General-Purpose Programming Languages

General-purpose programming languages have not been used much for defining information models, although some could be. General-purpose programming languages usually do not have accompanying representation languages, which is a disadvantage. Two such languages that might be suitable for defining information models are C++ and Java.

C++ is currently the language preferred by most developers for engineering applications. A C++ class hierarchy would be a usable information model.

Java is the language of choice for net applications. Java runs uncompiled, and has been implemented on almost all computer architectures for networked computers. Java is similar to a simplified form of C++.

## 5.3 Syntax Specification Languages

Syntax specification languages are not used directly for expressing programs or data, but are used for defining languages in which programs and data may be written. In deciding how to represent the definition of a language, it is important to understand how syntax specification languages may be useful.

The definition of a language (C++ or DMIS, for example) may be made unambiguous for parsing by defining the language in terms of a syntax specification language. The best known of these is

Extended Backus Naur Form (EBNF), ISO/IEC14977:1996 (E). A well-written definition of a language given in a syntax specification language makes it possible to determine if any sequence of characters (from one line to an entire file) is an allowed statement of the language. This is tremendously useful for determining if implementations of a language purported to conform to the specification of the language actually conform. Any other type of specification is apt to leave ambiguities that will be resolved differently by different implementors, leading to incompatible implementations.

An EBNF specification of a language is often split into two parts. The first part defines words[1] in terms of characters. The second part (1) defines non-terminal symbols which stand for various combinations of words and (2) specifies the allowed sequences of words and non-terminal symbols.

The two parts of an EBNF specification have a parallel in the lex and YACC input specification languages. Lex is a language for defining a lexical scanner (a program that recognizes character strings that form words). YACC (Yet Another Compiler Compiler) is a language for defining a parser of sequences of words. Both lex and YACC have compilers that automatically build C language source code for a parser; this code may also be compiled in C++. Given an EBNF specification, it is relatively straightforward (as compared to starting with some other type of language specification) to write an equivalent lex and YACC specification. Thus, building a standard parser for a language is greatly facilitated by having an EBNF specification of the language. Having a standard parser available to developers and users of a language makes it feasible to test the conformance of an implementation relatively easily.

## 5.4 Interface Definition Languages and Information Modeling Languages

In theory, interface definition languages (IDLs) and information modeling languages (IMLs) differ in that IDLs specify how you can get and set information, while IMLs specify what information is needed to define things. In practice, as long as a language is talking about the properties of things (which is how such languages are being used for metrology), the two types of language are equivalent. Both types have inheritance hierarchies. Both types can be processed by utilities that will automatically generate C++ code, and the C++ code sets generated from the two language types are very similar. For example, the C++ code for a circle produced by a generator for either type of language will almost certainly have methods for getting and setting the radius or diameter of the circle.

So, we will treat IDLs and IMLs together. Well-written descriptions of these languages may be found at http://www.objs.com/x3h7/fmindex.htm, which is a comparison of object-oriented languages.

IDLs and IMLs are well-suited to defining object-oriented versions of standards. This is convenient for building implementations in object-oriented computer languages. IDLs and IMLs are workable, but slightly un-natural (to many users) for defining declarative languages (ones that

1. As intended here, a “word” is any sequence of characters (such as “inspect” or “3.1415”) having a meaning taken as a whole. “Terminal symbol” is a synonym for “word” used in this sense.

say: do A, do B, etc.). IDLs have the disadvantage, compared to IMLs, of not having a standard mapping for data from a model to a file format.

### 5.4.1 CORBA IDL and Microsoft IDL

The central thing IDLs provide is the “interface” definition. The heart of an interface is a list of function signatures (function name and number and types of arguments) for the functions in the interface. An indication of which arguments are set and sent out with a call to the function and which are returned and set as a result of the function executing is included with each signature.

IDLs do not provide data “attributes” in the sense that IMLs do. CORBA IDL defines “attributes”, but they are described as simply being shorthand for declaring interfaces.

The CORBA IDL [CORBA] is the official object modeling language of the Object Management Group. It is well-documented. It has been used for building DMIS Part 2 and was adequate for that task. Commercial software tools exist that will take a CORBA IDL information model and automatically generate C++ code for manipulating data conforming to the model.

Microsoft IDL (MIDL) also exists. Documentation is harder to find than for CORBA IDL and tends to be oriented to explaining how to use MIDL to build Windows-based software.

CORBA IDL is somewhat more expressive than MIDL in that it provides for multiple inheritance and exceptions, which MIDL does not. MIDL provides multiple interfaces rather than multiple inheritance.

### 5.4.2 EXPRESS

The central thing EXPRESS provides is the “entity” definition. The heart of an entity is a list of attributes with their data types. EXPRESS also provides powerful methods for putting constraints on the attributes of entities and on data sets (sets of entity instances). EXPRESS does not provide for the declaring interfaces composed of functions.

EXPRESS (Part 11 of international standard ISO 10303) [ISO1] is the official STEP information modeling language. EXPRESS is used for representing all the STEP application protocols (the things that are to be used as data standards). Many commercial software tools exist that will take an EXPRESS information model and automatically generate C++ code for manipulating data conforming to the model. The generated C++ invariably contains functions for setting and getting the values of attributes. So (presto!) attributes have turned into functions, and entities resemble IDL interfaces. The commercial packages usually also include a library of functions for manipulating data in many ways, such as reading and writing data files or selecting all objects of a given type from a data set.

EXPRESS works together with a generic exchange file format specified in Part 21 of STEP. Tools for manipulating STEP data are generally able to read and write Part 21 files corresponding to an EXPRESS information model. Each data file gives the name of the EXPRESS information model(s) to which it corresponds. IDLs and C++ lack this key feature of EXPRESS (because there is no file equivalent of a function).

### 5.4.3 C++ Header Files

The central thing C++ language header files provide is the "class" definition. A class definition is generally a combined declaration of interface functions and attributes. C++ header files do not provide for constraints as EXPRESS does, and they do not provide for identifying outgoing and incoming function arguments the way IDLs do. C++ could be used for writing standards but has not been used much. Standards writers prefer not to have standards tied to specific computer languages.

C++ header files, however, are the principal target for automatic code generators that take either IDLs or IMLs as input. In some cases, there is a standard for the conversion from an IDL or IML to C++, so that various implementation can expect the C++ code to be interoperable.

### 5.4.4 XML

XML (eXtensible Markup Language) [XML] provides a generic file format for data and two methods for describing formats. XML is a particularly convenient language because many existing internet-based communications systems (including web browsers) use it. Thus, the infrastructure for using XML in communications relating to dimensional metrology may already be in place in many enterprises.

XML can communicate both data and data formats. To deal with XML data formats, senders and receivers must agree on what formats may be used and have software in place to put data into the correct format on the sending end and get it back out on the receiving end.

One method of specifying a data format in XML is to use an XML Document Type Definition (DTD). DTDs are suitable for relatively simple data formats. DTDs are likely to be suitable for active interfaces but are not likely to be useful for data interfaces.

A second method of specifying a data format in XML is to use an XML schema. An XML schema is similar to an EXPRESS schema. An information model can be represented well in an XML schema. Schemas are relatively new in XML. The language for XML schemas is large and complex. XML schemas are likely to be useful for data interfaces.

## 5.5 STEP Part 21 Generic File Format

STEP Part 21, "Clear Text Encoding of the Exchange Structure" [ISO2], provides a method, given an EXPRESS information model, of automatically and unambiguously defining the syntax of data files corresponding to that model. This is extremely convenient, since there are many commercial systems that implement the method, allowing them to write and read STEP exchange files. This frees the standards writer from having to invent syntax (How do I write that down?) and allows the standards writer to concentrate on semantics (What do I mean to say?), which is generally the important part of a standard.

STEP Part 21 files are not easily human-readable.

## 5.6 Process Planning Languages

Two domain-independent, general-purpose process planning languages have been developed at NIST. To use either of these languages in a specific domain, additional domain-specific extensions must be created.

We are not aware of any domain-independent proprietary process planning languages.

### 5.6.1 A Language for Process Specification (ALPS)

The fundamental object of ALPS [Catron] 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 use ALPS for FBICS, two suites of subtypes of the ALPS primitive_task_node were created. The first suite is for plans coming from routing planning and consists of a single task subtype, run_setup. The second suite is for plans coming from inspection planning or machining planning and consists of many subtypes. Section 6.7.2 and Section 6.11.2 provide more details.

Software for traversing ALPS plans was built for use in FBICS. With such software available, ALPS becomes executable, and may be regarded as a programming language.

ALPS appears to be usable in a commercial setting for expressing process plans, but no commercial use has been made of ALPS. It has been used only in research-grade systems.

### 5.6.2 Process Specification Language (PSL)

PSL, another domain-independent, general-purpose process planning language, has been under development at NIST for several years. The initial testing of PSL is underway, but PSL is not yet available for general use.

## 5.7 Configuration and User Option Files

Every CAX system has at least one, and usually several, configuration file or user option file. On PCs, they often have the suffix ".ini". A PC convenient to one of the authors listed 66 files when asked to find all ".ini" files. These files generally consist of attribute value pairs, such as "error_signal=bell", or "error_signal:bell", typically with one pair per line. These files are so structurally simple and so pervasive that it would be easy to define a standard format for them (not including standard contents). The interested community, however, is all computer users and builders. If a standard were to be made, the dimensional metrology community would be only one among many interested parties.

Developing a standard in this area, while intriguing, is of low priority. In the absence of a standard, it is still easy to exchange simple configuration and options files, since it is easy to write parsers for them.

If user options are more complex than a list of attribute-value pairs, a simple file format is no longer feasible. FBICS, for example, has a file of user options which consists of rules for deciding how to use cutting tools (set speeds, feeds, etc.). These rules may include complex expressions. If options are complex, it would be better to model the options in an information modeling language (such as EXPRESS) and use the default file format that works with the modeling language (such as STEP Part 21), if there is one.

# 6 APIs and Data Formats for Dimensional Metrology

This section discusses the APIs and data formats needed for dimensional metrology and assesses the status of each. The principal subsections are arranged alphabetically.

The term API is used broadly in this section to mean either an API in the narrow sense or an API-like messaging protocol. In the narrow sense of API, a command in an API is a function in some computer language which tells the executor of the call to manipulate data or perform a physical activity and (usually) returns a value or values to the caller before the caller is able to do anything else. In an API-like messaging protocol, the caller sends a message which tells the executor of the call to manipulate data or perform a physical activity, but the caller might not wait for a return message before continuing, and return values may be returned in messages received at various times in the future.

The term "data format" is used in this section to mean either an explicit data format or a data model which may be combined with an exchange format. The difference between the two is explained at the beginning of Section 5, and an example is given in Appendix C. Where a data format is for files of executable instructions (a DMIS file, for example), "language" would be a synonym for "data format".

For each type of API or data format, we look at the usage of existing specifications, the adequacy of existing specifications, and the needs of existing specifications. Where the current situation needs improvement, we recommend a situation that would be better and suggest how improvements might be made.

Standardizing the various interfaces is ranked simply as being of **high**, medium, or low priority. In a fully automated system, every interface, even those ranked here as of low priority, must be formally defined.

## 6.1 CAD APIs

The active interface from Activity Coordination to CAD (sections A.1.1 and A.2.1) could use an API to tell CAD what to design and when to design it, but no commonly used API for that purpose exists. Standardizing this API is of low priority.

All CAD systems have APIs for their design functions. When a user is designing on a CAD system, calls to functions in the CAD API are made by the user interface module in response to the user's actions. Some CAD systems hide their API. Others open their APIs to developers by providing a description of the API functions and a linkable library. CAD APIs of this sort are not further discussed in this analysis.

## 6.2 CAD Data Formats

The data interfaces from CAD to Inspection Planning (sections A.2.2 and A.5.4), Inspection Programming (sections A.2.3 and A.6.3), Machining Planning (sections A.2.6 and A.8.3), Machining Programming (sections A.2.7 and A.9.3), Routing Planning (sections A.2.5 and A.14.3), and Solid Modeling (sections A.2.4 and A.15.7) all require a CAD data format. In

addition, CAD data representing the intermediate shapes of workpieces and shapes of volumes to be inspected or removed may be: generated by Routing Planning and used by Inspection Planning (sections A.5.5 and A.14.5) generated by Inspection Planning and used by Inspection Programming (sections A.5.6 and A.6.4), generated by Routing Planning and used by Machining Planning (sections A.8.4 and A.14.6), and generated by Machining Planning and used by Machining Programming (sections A.8.5 and A.9.4).

Many widely-used CAD data formats exist. Some of these are drawing formats (i.e., their principal function is to describe a drawing) while others are model formats (i.e., their principal function is to describe all or part of the shape of a solid object). In practice, the distinction is blurred, since drawing formats may be able to represent some model data and model formats may be able to represent views (including dimensions, tolerances, and other annotations) of the object modeled. Model formats are more useful and are displacing drawing formats.

There are many varieties of model format. Simple models may be built as wire frames, in which only the part edges are represented. All CAD systems can draw wire frame representations for the user, but no major CAD systems use wire frames as the underlying representation. For some purposes, facetted models, in which all surfaces are represented as planes, may suffice. Again, all CAD systems can draw facetted models, but no major CAD system has a facetted underlying representation. Some CAD systems require modeled objects to be physically reasonable, so that, for example, the entire surface must be defined. Other CAD systems specialize in surface representation and may model only a portion of the surface, leaving the rest undefined. For some CAD systems, the external file representation is a list of modeling commands which must be rerun by the system in order to create a working representation of the object modeled. Representations of the last type are difficult or impossible to translate from one system to another. The STEP Parametrics Committee is working on a standard for representation of modeling history.

An additional difficulty is that the notion of a CAD file has no well-defined boundary, overlapping with representations of graphics, printed circuit layout, animations, etc. An example of an overlapping language is VRML (Virtual Reality Modeling Language). VRML files typically contain a great deal of geometry, but the geometry is not required to make sense (objects can share space, for example) and complex geometry is facetted. VRML is usable as CAD output, therefore, only for simple objects or for applications not requiring that the model be physically possible, such as fly-throughs.

The boundary between CAD file formats and solid modeler file formats (which are discussed in Section 6.21) is very faint.

Because so many widely-used formats exist, many commercial software systems that deal with CAD data have been equipped with translators that convert from one format to another. Product literature for such software systems usually includes a section listing the various CAD formats the system can handle. Software system vendors enter into agreements with CAD vendors to get their cooperation in dealing with CAD formats. Huge numbers of programmer-years are devoted to building and maintaining translators.

**Standardizing CAD data is of high priority**.

### 6.2.1 Non-Proprietary CAD Data Formats

*6.2.1.1 STEP AP 203*

STEP AP 203, "Configuration Controlled Design", provides a boundary representation (a type of solid model) for the design of solid objects. AP 203 is an ISO international standard. STEP AP 203 was devised specifically to help solve the problem of multiple, non-standard CAD data formats. AP 203 input and output is supported by all major CAD vendors.

AP 203 does not include tolerances, so it is not usable for transmitting design information from CAD to most inspection activities and is weak for transmitting design information to manufacturing activities.

Systems exist that will allow a user to start with an AP 203 design and add tolerances to it. The FBTol System from Allied Signal in Kansas City is an example of such a system. The output file from such a system is not an AP 203 file, however.

The most straightforward way to get a standard for CAD data that could be used for dimensional metrology activities would be to revise STEP AP 203 to include tolerances.

*6.2.1.2 STEP AP 224*

STEP AP 224, "Mechanical Product Definition for Process Planning Using Machining Features" defines a library of machining features. The 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.

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 AP 224 geometric and dimensional tolerances are a full suite of tolerances from ASME Y14.5.

There are a few commercial software systems that generate AP 224 files, but the use of AP 224 is not yet widespread.

FBICS uses AP 224 to describe parts and sets of removal volumes. AP 224 features generally consist of collections of bounded DMIS features. An AP 224 hole with a flat bottom, for example includes a DMIS cylinder for its sides and a DMIS plane for its bottom. The FBICS DMIS generator decomposes AP 224 features into their constituent DMIS features.

Compared to AP 203, AP 224 has a large advantage for dimensional metrology uses in that it provides tolerances. AP 224 has the disadvantage that not as large a range of shapes can be described with its manufacturing feature library as can be described by a boundary representation. Also, many fewer commercial CAD systems support AP224.

*6.2.1.3 STEP AP 214*

STEP AP 214, "Core Data for Automotive Design Process" is a large (several thousand pages) application protocol for automobile design and manufacturing. An outline of the scope of AP 214, alone, is a page long, including for example, eight different methods of shape representation.

In the area of machining features and tolerances, AP 214 duplicates AP 224 rather than simply referencing it. This is unfortunate because it makes translation necessary for data exchange. Because the feature suites are largely identical, translation is not difficult.

*6.2.1.4 IGES*

IGES (Initial Graphics Exchange Specification), is an open drawing format standard of the National Computer Graphics Association that is widely supported by CAD vendors. It is generally regarded as obsolescent, though still widely used.

### 6.2.2 Proprietary CAD Data Formats

CAD systems from the major CAD vendors (including, for example, PTC's Pro-Engineer, SDRC's I-DEAS, Dassault/IBM's CATIA, and UGS's Solid Edge), each have their own proprietary data format for representing designs. Designs from these systems are generally design history files and cannot be usefully manipulated without using the system that generated them.

*6.2.2.1 DXF*

The Drawing eXchange Format (DXF) is a proprietary format for AutoDesk's AutoCAD. As the name implies, it is a drawing format, not a solid model format, though it does have some 3D modeling capabilities. Many commercial software systems use DXF input. AutoDesk revises DXF from time to time. The current version is release 15, for which the specification is open and available on the internet at http://www.autodesk.com/techpubs/autocad/acad2000/dxf/index.htm.

## 6.3 Hand-Held Device Measuring APIs

An API is used by Activity Coordination to tell Hand-Held Device Measuring what to measure and when to measure it (sections A.1.4 and A.3.1). There are no existing widely-used APIs for this interface. Standardizing this API is of low priority.

## 6.4 High-Level Inspection Instruction Execution APIs

An API is used by Activity Coordination to tell High-level Inspection Instruction Execution what to do (sections A.1.6 and A.4.1). It is convenient if there are two APIs. The first API provides for

execution of any single high-level inspection instruction. The second API provides for execution of a program or file containing many such instructions.

### 6.4.1 DMIS Part 2

DMIS Part 2 does not fit neatly into any one section of this analysis because it provides not one, but three interfaces. Since the largest of the three may be used to send high-level inspection instructions to High-level Instruction Execution, it is discussed here.

DMIS Part 2 is an interface specification written in CORBA IDL, an object-oriented language (see Section 5.4.1). Since it is written in CORBA IDL, it may easily and automatically be mapped into a set of C++ class definitions. Since it is an interface specification, not a software system, DMIS Part 2 defines interfaces but does not implement them. Commercial implementations of DMIS Part 2 are expected to be built.

The Dme portion of DMIS Part 2 provides an API for giving individual high-level commands to High-level Instruction Execution. We will call it the Dme API. Two other portions[1] of DMIS Part 2, the DmeEquip portion and the DmeMath portion, are discussed in Section 6.9.2 and Section 6.13, respectively.

A call to an interface function in the Dme API might have the same meaning as commanding that a single line of text from a high-level program be executed. The Dme API might be used, for example, (1) by a user interface to DME equipment to give individual commands to the equipment or (2) internally in Inspection Programming to build a representation of a DMIS program being written by a programmer, or (3) internally in High-level Instruction Execution to build a representation of a DMIS program being read from a file.

The Dme API corresponds closely to the commands specified by DMIS Part 1. The DMIS Part 2 specification provides that the meaning of each interface in the Dme API is given by the description of the corresponding command from DMIS Part 1. This subordination of meaning does not work for the other two portions of DMIS Part 2 because there is little or nothing in DMIS Part 1 corresponding to those portions.

**Completing standardization of DMIS Part 2 is of high priority**.

### 6.4.2 Another Type of API to High-level Instruction Execution

Another type of API is used by Activity Coordination to tell High-level Instruction Execution what inspection programs to run and when to run them. In FBICS, this API includes commands such as “prepare to run program files”, “run high-level program file XYZ”, and “stop running program files”. There are no known open commercial implementations of such an API. Presumably, hidden APIs of this sort exist between user interfaces to CMM controllers and the controllers’ innards. Standardizing this API is of low priority.

---

1. DMIS Part 2 actually calls these “modules” rather than “portions”, but this analysis is using “module” to mean something else.

## 6.5 High-level Inspection Instruction Data

The data interfaces to High-level Inspection Instruction Execution from Inspection Programming (sections A.4.4 and A.6.5) and from Machining Programming (sections A.4.5 and A.9.5) require a format for inspection instructions and programs. In addition, the user interface to High-level Inspection Instruction Execution (what a CMM operator deals with) might generate command strings in this format. The only standard for inspection programs is DMIS Part 1. There are several proprietary data formats for inspection programs produced as output of commercial inspection software systems such as Camio, Metrolog, Quindos, Umess, and Valisys Programming.

The use of RS274 at the interface from Machining Programming to High-level Inspection Instruction Execution is discussed in Section A.9.

**Solidifying the position of DMIS as the one and only standard for inspection programs is of high priority**.

### 6.5.1 DMIS

DMIS (Dimensional Measuring Interface Standard) is a high-level language for control of dimensional measuring equipment. Executing a DMIS program in High-level Inspection Instruction Execution will result in low-level inspection instructions being given to Low-Level Inspection Instruction Execution.

DMIS includes both an input language and an output language. Part of the DMIS input language is for defining objects of interest (features, tolerances, sensors, etc.), and the rest is a programming language with action commands, flow of control commands, variable settings, etc. The output language serves both as a partial log of commands given and as a report of inspection results, with actual and nominal point data, features, and tolerances.

DMIS is a U. S. national standard and may soon be an ISO international standard. DMIS 4.0 [CAM-I2] was approved in April 2001 by ANSI as a U.S. national standard. Most existing implementations are of DMIS 3.0 [CAM-I1]. DMIS is maintained by the DMIS National Standards Committee (DNSC), a committee of CAM-I. The DNSC has put excellent procedures in place for collecting and acting on suggestions for improvements in DMIS.

DMIS, however, needs additional infrastructural support to be a strong and useful standard. Specifically, it needs (1) fixed conformance classes (not just implementor-defined conformance), (2) a set of conformance tests for each conformance class, providing complete coverage of the functionality of the class (3) a conformance testing service (not NIST) that will test a DMIS implementation in a conformance class and certify (if it passes) that the implementation conforms to the DMIS standard for that class.

Fixed conformance classes are needed because it is too time-consuming, expensive, and unreliable for a DMIS user to try to analyze implementor-defined conformance statements in order to determine which implementations fit the user's needs (even assuming the implementations conform to the implementors' claims). Few users are sophisticated enough to

determine reliably from a conformance statement whether a system will do the job the user needs done. If there are fixed conformance classes, a body of experience can be built up so that it will be common knowledge among frequent DMIS users which conformance class is required to do which type of job.

A set of conformance tests is needed because you cannot test conformance without them. Conformance test suites should be freely available to everyone, especially implementors, so that implementors can assure themselves their implementations are correct before releasing them.

A conformance testing service is needed because without it, poor implementations which claim to conform but do not may drive out truly conforming implementations. Implementors can save money by cutting corners. Users lose money as a result. A standard without a conformance testing service is like a speed limit sign with no cops; it will soon be outrageously violated.

To make DMIS easily usable, standard alternatives to the textual form of DMIS should be devised. DMIS Part 2 is one alternative and is being standardized. A mapping of DMIS to EXPRESS, for example, would allow automatic conversion of DMIS files to and from STEP Part 21 files. That would be a useful alternative because it would allow DMIS to be included seamlessly in the STEP world.

NIST can help provide the needed infrastructure by devising conformance tests, participating in standardizing DMIS Part 2, and generating a mapping of DMIS to EXPRESS.

Portions of developing standard AP 219 (primarily feature, tolerance, and datum definitions) overlap with DMIS. Several meetings have been held to harmonize these aspects of DMIS and AP 219. **Completing the harmonization is of high priority**.

## 6.6 Inspection Planning APIs

An API is used by Activity Coordination to tell Inspection Planning what to plan for and when to plan (sections A.1.2 and A.5.1). This API is defined in FBICS, but there are no existing widely-used APIs for this interface. Standardizing this API is of low priority.

## 6.7 Inspection Plan Data

The data interface from Inspection Planning to Activity Coordination (sections A.1.12 and A.5.3) requires a data format for inspection plans. An inspection plan at this level says what to inspect in one fixturing of a part being manufactured. There is no existing widely-used data format for inspection planning. This interface is person-to-person in many industrial systems.

Commercial systems dealing with inspection usually combine inspection planning with inspection programming, so that the interface is invisible or non-existent. FBICS uses the ALPS high-level process planning language augmented with inspection tasks, as described in Section 6.7.2. STEP AP 219, currently being developed, is expected to provide data that may be used on this interface. **Standardizing this data is of high priority**.

### 6.7.1 STEP AP 219

Application Protocol 219, Dimensional Inspection Information Exchange, is planned as the information model for dimensional inspection operating under STEP. It aims to be the principal link between dimensional inspection implementation software and the system of STEP standards that deal with inspection information such as AP 224, "Mechanical Product Definition for Process Plans Using Machining Features" and AP 214, "Core Data for Automotive Mechanical Design". The developing AP 219 information model includes an activity model, a data dictionary of terms in the activity model, a draft EXPRESS-G and EXPRESS reference model, and a draft XML Document Type Definition (DTD) based on the EXPRESS model. The next component to be added will be the set of object definitions of elements in the EXPRESS reference model. The scope of AP 219 includes planning, execution, and analysis of dimensional inspection activities. The AP 219 data model will be implemented in data formats of clear text encoding (STEP Part 21) and XML DTDs (STEP Part 28).

From the scope of AP 219, it is apparent that AP 219 may provide other types of data in addition to inspection plan data.

### 6.7.2 ALPS with Inspection Operations

The ALPS language is described in Section 5.6.1. For FBICS inspection, the "inspection_task" was defined as a subtype of the ALPS primitive_task_node. The inspection_task has two subtypes itself: inspect_feature_geometry, and inspect_feature_surface. The inspect_feature_geometry subtype is intended to work with AP 224 features. For FBICS, this data is modeled in an EXPRESS information model. Specific plans are generated automatically and written in STEP Part 21 files.

## 6.8 Inspection Programming APIs

An API is used by Activity Coordination to tell Inspection Programming what to write programs for and when to write them (sections A.1.3 and A.6.1). This API is defined in FBICS, but there are no existing widely-used APIs for this interface. Standardizing this API is of low priority.

## 6.9 Low-Level Inspection Instruction APIs

An API is used by High-level Inspection Instruction Execution to tell Low-level Inspection Instruction Execution what to do (sections A.4.3 and A.7.1). **Standardizing this API is of high priority**.

Low-level instructions are generated by High-level Instruction Execution and executed in real time, not read from a file or sent by a human user. Thus, for low-level inspection instructions, it is not expected that there will be files of instructions (except for testing), so no standard data format is required.

A low-level inspection instruction API was defined for the NIST DMIS Interpreter and implemented in FBICS. Two, possibly three, standardization efforts are currently in progress (CMM-driver Specification, the DmeEquip portion of DMIS Part 2, and possibly the European

I++ effort), each of which defines a low-level inspection instruction API. **Harmonizing these efforts is of high priority**. It would be very unfortunate to have different standards for low-level inspection instructions.

### 6.9.1 CMM-driver Specification

This is primarily an API for low-level commands for control of a CMM. It is actually an API-like messaging protocol plus a communications method. The initial statement and testing of this specification was done by a group of three CMM vendors (Zeiss, LK, and B&S) working together. NIST has been working to assist developing the specification and test methods for it.

NIST has developed a CMM-driver command file format for testing purposes, not further discussed here.

### 6.9.2 DmeEquip Portion of DMIS Part 2

The DmeEquip portion of DMIS Part 2 includes almost all of the functionality of the current version of the CMM-driver specification, plus additional functionality. Other discussion of DMIS Part 2 is given in Section 6.4.1 and Section 6.13.

### 6.9.3 I++

A European effort named I++ is said to be working on a low-level inspection instruction API and possibly related data. Further information about what I++ is doing is needed.

## 6.10 Machining Planning APIs

Machining planning is relevant to the extent that it includes planning for in-process inspection.

An API is used by Activity Coordination to tell Machining Planning what to plan for and when to plan (sections A.1.8 and A.8.1). This API is defined in FBICS, but there are no existing widely-used APIs for this interface. Standardizing this API is of low priority.

## 6.11 Machining Plan Data

The data interface from Machining Planning to Activity Coordination (sections A.1.13 and A.8.6) requires a data format for machining plans (which may contain in-process inspection steps). There are no widely-used data formats for machining plans, but three existing formats are described here. In addition to the three plan formats described below, it is expected that PSL (see Section 5.6.2) will be used in the future to express machining process plans. Standardizing this data is of medium priority.

### 6.11.1 ISO 14649

ISO 14649, “Industrial Automation Systems and Integration — Physical Device Control — Data Model for Computerized Numerical Controllers” is a committee draft international standard intended to replace M and G codes for controlling NC machines. ISO 14649 provides higher-level data that can be used directly for machine control. A closely related, but not quite synonymous,

effort is called STEP NC, which implements ISO 14649 using STEP techniques. ISO 14649 includes only two basic touch probing capabilities: (1) probe a point, and (2) probe an entire part to establish its orientation.

### 6.11.2 ALPS with Machining Operations

For machining process plans, FBICS uses ALPS (see Section 5.6.1) with a suite of machining tasks added. One of these tasks is finish_mill_adaptive, which may be used for machining pockets. This is performed by milling a smaller pocket than specified, measuring the results with a touch probe, calculating a size error, and then milling the pocket to the specified size by adjusting the final pass to offset the expected size error. FBICS machining plans may also include any of the inspection tasks described in Section 6.7.2.

### 6.11.3 STEP AP 213

STEP AP 213 is a draft international standard for process plans for machined parts. AP 213 is unlikely to become a full international standard in its present form. Its progress has been stopped for several years. While it has in-process inspection in its scope, there is little support for the expression of tolerances or for inspection activities. AP 213 is thus not very relevant to inspection and is not further discussed in this analysis.

## 6.12 Machining Programming APIs

Machining programming is relevant to the extent that machining programs may include in-process inspection.

An API is used by Activity Coordination to tell Machining Programming what to write programs for and when to write them (sections A.1.9 and A.9.1). This API is defined in FBICS, but there are no existing widely-used APIs for this interface. Standardizing this API is of low priority.

## 6.13 Math Computing APIs

An API is used by High-level Inspection Instruction Execution (sections A.4.2 and A.10.1) and Reporting and Analysis (sections A.10.2 and A.13.3) to tell Math Computing what to compute and to get answers back. There is no widely used API for this interface. Each commercial Math Computing package has its own API. Standardizing this API is of medium priority.

FBICS defines this API with only a set of feature-fitting functions. FBICS passes a set of touch points and a nominal feature description to Math Computing and gets back an actual feature description. Math Computing is implemented using feature fitters provided by the NIST Algorithm Testing System.

The DmeMath portion of DMIS Part 2 defines this interface with a generic feature-fitting function, feature constructions, tolerance evaluation, installation of feature-fitting functions (to be used by the generic feature-fitting function), and transformation matrix transformations.

There are many other types of mathematical calculations beyond those of FBICS and DMIS Part 2 that might be done by Math Computing, particularly statistical analyses on behalf of Reporting and Analysis.

Commercially available mathematics packages include Mathematica and Matlab. These do not include feature-fitting functions of the sort required by High-level Inspection Instruction Execution, but they do have statistical computing capabilities that might be used by Reporting and Analysis.

## 6.14 Other Inspection Device APIs

An API is used by Activity Coordination to tell Other Inspection Device Control what other device control activities to perform and when to perform them (sections A.1.10 and A.11.1). There are no existing widely-used APIs for this interface. Standardizing this API is of low priority.

## 6.15 Probe Instruction APIs

An API is used by Low-level Instruction Execution to tell Probe Instruction Execution what to do and to get data back (sections A.7.2 and A.12.1).

It would be very useful to CMM vendors, integrators of dimensional metrology equipment, and (probably) probe vendors if there were a standard API (and standard hardware connections) for probes. There is not much activity in standardizing this API at this time, but it appears to be a worthwhile thing to do and is of medium priority. From a technical viewpoint, standardizing this interface seems nearly as important as standardizing the interface to Low-level Inspection Instruction Execution, for which there is currently a substantial effort, as described in Section 6.9.

Commercially available probes come equipped with APIs and hardware interfaces. These are all proprietary and none is used by a vendor other than the owning vendor.

There are two existing possibilities for standardizing this interface: either finish development of the NGIS SIM specification or apply the IEEE 1451 Intelligent Sensor Standard.

### 6.15.1 NGIS SIM Specification

A “Next Generation Inspection System (NGIS) Sensor Interface Module (SIM) Specification” was drafted in 1997 and 1998 by a “SIM API Working Group” comprised of probe vendors, users, and researchers under the auspices of the National Center for Manufacturing Sciences. NIST staff served on the working group and served as editors for the draft “NGIS SIM Specification”, which was published in 1998 as NISTIR 6116. This specification addresses the integration of commercial probe hardware and software with commercial controllers. The application is control of coordinate measurement machines (CMMs) and numerically controlled (NC) machine tools to perform part inspection.

The document defines a standard hardware component, the SIM, and an API to the SIM. The focus of this specification is on touch-trigger probes and scanning probes, but the specification may be applicable to other types of probes.

The intent is that the SIM will allow part inspection probes to be integrated easily into control systems for CMMs and NC machine tools. This specification provides for the capability to integrate a new probe at controller run time, without the need to recompile or reconfigure the controller software. The goal is to provide sufficient guidance on technical properties of the SIM that independently developed probe and controller products will be compatible right out of their "shrink-wrapped" packages.

The SIM API was implemented by one vendor, who continues to use it in commercial products. The draft was never entered into any standardization process, and was not refined after 1998. It might be worthwhile to revive and standardize this NGIS SIM Specification.

### 6.15.2 IEEE 1451 Intelligent Sensor Standard

The intent of IEEE 1451 Intelligent Sensor Standard is to permit integration of multi-vendor intelligent sensors and actuators on a network. The standard is not specific to probes. The standard was developed for applications such as process control and building energy control, but it may be suitable for probing. Researchers at NIST took a preliminary look at using IEEE 1451 for probing, and constructed a mapping from the NGIS SIM Specification to the IEEE 1451 standard. Further work would be needed to determine if IEEE 1451 is suitable in principle. If it is suitable in principle, a prototype probing implementation would be needed to determine if it is practical.

## 6.16 Reporting and Analysis APIs

An API is used by Activity Coordination to tell Reporting and Analysis what data to analyze and when to do the analysis (sections A.1.7 and A.13.1).

A set of XML DTDs (see Section 5.4.4) in a file named dml_analysis_request.dtd has recently been drafted by DaimlerChrysler for this interface.

Standardizing this API is of medium priority.

## 6.17 Reporting and Analysis Data

The data interface from High-level Inspection Instruction Execution to Reporting and Analysis (sections A.4.6 and A.13.7) requires a data format for high-level inspection data. Data for this interface may be provided in DMIS output file format. DMIS output files might be analyzed, for example, to determine if the same feature is often out of tolerance on parts of the same design, or if there is a trend over time in making good parts of a specific design. Even if no analysis is done on DMIS output files in Reporting and Analysis, it would be natural and desirable that Reporting and Analysis keep records of what DMIS output files were generated.

The data interfaces from Other Inspection Device Control (sections A.11.2 and A.13.4), Hand-held Device Measuring (sections A.3.2 and A.13.5), and Low-level Inspection Instruction

Execution (sections A.7.3 and A.13.6) require a data format for low-level inspection data. There are many proprietary formats for this data but no widely used format. DMIS programs may be written so that low-level data is saved in DMIS output files. Thus, the DMIS output file format may be used for low-level inspection data. If data from several sources (a hand-held electronic micrometer and a CMM, for example) are to be used together, it is necessary to have data in the same (or equivalent) formats. It would be very useful to have a standard for this low-level data. **Standardizing this data is of high priority**.

It is not clear to the authors whether DMIS output files meet the needs of users of either high-level or low-level inspection data. If DMIS output files produced by existing DMIS program execution systems are not meeting users' needs, any of four causes are possible: (1) the DMIS input format does not provide a sufficient range for specifying types of output, (2) the programs being executed are not written so as to provided the desired output, (3) the analysis capabilities of the execution systems (fitting features to sets of points, primarily) are not adequate, (4) the DMIS output file format does not provide a sufficient range of types of output.

## 6.18 Routing Planning APIs

An API is used by Activity Coordination to tell Routing Planning what to plan for and when to plan (sections A.1.5 and A.14.1). There is no commonly used API for this purpose. This is a person-to-person interface in all existing industrial systems but has been defined as a command for a human user to give from the user interface in FBICS. Standardizing this API is of low priority.

## 6.19 Routing Plan Data

The data interface from Routing Planning to Activity Coordination (sections A.1.11 and A.14.4) requires a format for a process plan for directing the activity of performing all machining and inspection required on a part being manufactured. This is a high-level plan saying which workstations a workpiece should go to and what should be done at each workstation. In current systems, this is often a person-to-person interface. FBICS uses the ALPS high-level process planning language augmented with the run_setup task, defined in EXPRESS information models. Specific plans are written in STEP Part 21 files. Standardizing this data is of low priority.

## 6.20 Setup Data

The data interfaces from Routing Planning to Inspection Planning (sections A.5.5 and A.14.5) and to Machining Planning (sections A.8.4 and A.14.6), from Inspection Planning to Inspection Programming (sections A.5.6 and A.6.4), and from Machining Planning to Machining Programming (sections A.8.5 and A.9.4) require a format for setups. A setup includes a description of where various things are in the coordinate system of the machine (CMM or machine tool) on which the processing is being done, plus pointers to the files needed to do the processing. There is no widely used format for setup data. FBICS models setups in EXPRESS and uses STEP Part 21 files to represent specific setups. Standardizing this data is of low priority.

## 6.21 Solid Modeling APIs

An API to Solid Modeling is used by Inspection Planning (sections A.5.2 and A.15.2), Inspection Programming (sections A.6.2 and A.15.3), Machining Planning (sections A.8.2 and A.15.5), Machining Programming (sections A.9.2 and A.15.6), Reporting and Analysis (sections A.13.2 and A.15.4), and Routing Planning (sections A.14.2 and A.15.1) to tell Solid Modeling what to model, to make inquiries about solid models, and to get answers back. Standardizing this API is of low priority.

### 6.21.1 Proprietary Solid Modeling APIs

As noted in Section 3.2, this analysis distinguishes between CAD systems and solid modelers. Users interact directly with CAD systems to design things. Users do not usually interact directly with solid modelers. Solid modelers are used by CAD and other software systems via an API. Software developers include calls to solid modeler API functions in the programs they write.

The two predominant solid modelers are ACIS (from Spatial Technology) and Parasolid (from Unigraphics Solutions, Inc.).

ACIS and Parasolid each have their own API for telling the modeler what to do and getting answers back.

### 6.21.2 Non-Proprietary Solid Modeling APIs

The Application Interface Specification, developed in the 1980s under the auspices of Consortium for Advanced Manufacturing - International, is a generic solid modeler API but has not been standardized. The most recent version of the specification is version 2.1 from 1994. Early versions of the Parasolid API were similar to the specification.

A generic solid modeler API has been developed at Allied Signal, along with a binding of the generic API to the ACIS API. The generic API is being used in Allied Signal's FBMach and FBTol systems. A binding to Parasolid is contemplated.

## 6.22 Solid Modeling Data

As shown on Figure 1, this analysis does not anticipate that solid model data files will be used outside of Solid Modeling. There are formats for solid modeling data, similar to but distinct from CAD data.

Both ACIS and Parasolid make their file format publicly available on the internet. The ACIS format is called the "SAT" file format and is currently at version 6.0. The Parasolid file format is called the "XT" format. These file formats provide structures for the data required by a solid modeler but do not provide structures for tolerances, datums, etc.

# Appendix A Modules and Interfaces in Detail

This appendix describes interfaces on a module-by-module basis for each of the 15 modules previously identified (as shown on Figure 1). A separate figure is provided for each module, showing the same interfaces as shown on Figure 1. A brief description of the content of each interface is provided. Cross-references from arrows to text are included on the figures.

Bear in mind, as stated earlier, that the intent of showing connections is only to make it clear what the content of the interfaces must be, not to espouse a particular architecture.

For each interface, a reference is provided to the part of Section 6 describing the APIs or data formats for the interface. Priorities for standardizing interfaces are given in this section and in Section 6. Interfaces for which developing a standard is of high priority are also discussed in Section 4.

For active interfaces, it is expected that the used module will always return at least an “I did it” or error response to the user. These status responses are not further described here. Data may also be returned from the used to the user via an active interface. If data is returned, that is described here.

## A.1 Activity Coordination

For the purposes of this analysis, Activity Coordination is treated as a central function. In actual systems, Activity Coordination is almost always distributed. If Activity Coordination is distributed, the same interfaces to other modules will be needed as if Activity Coordination were centralized, but additional active interfaces will be required among the various Activity Coordination modules.

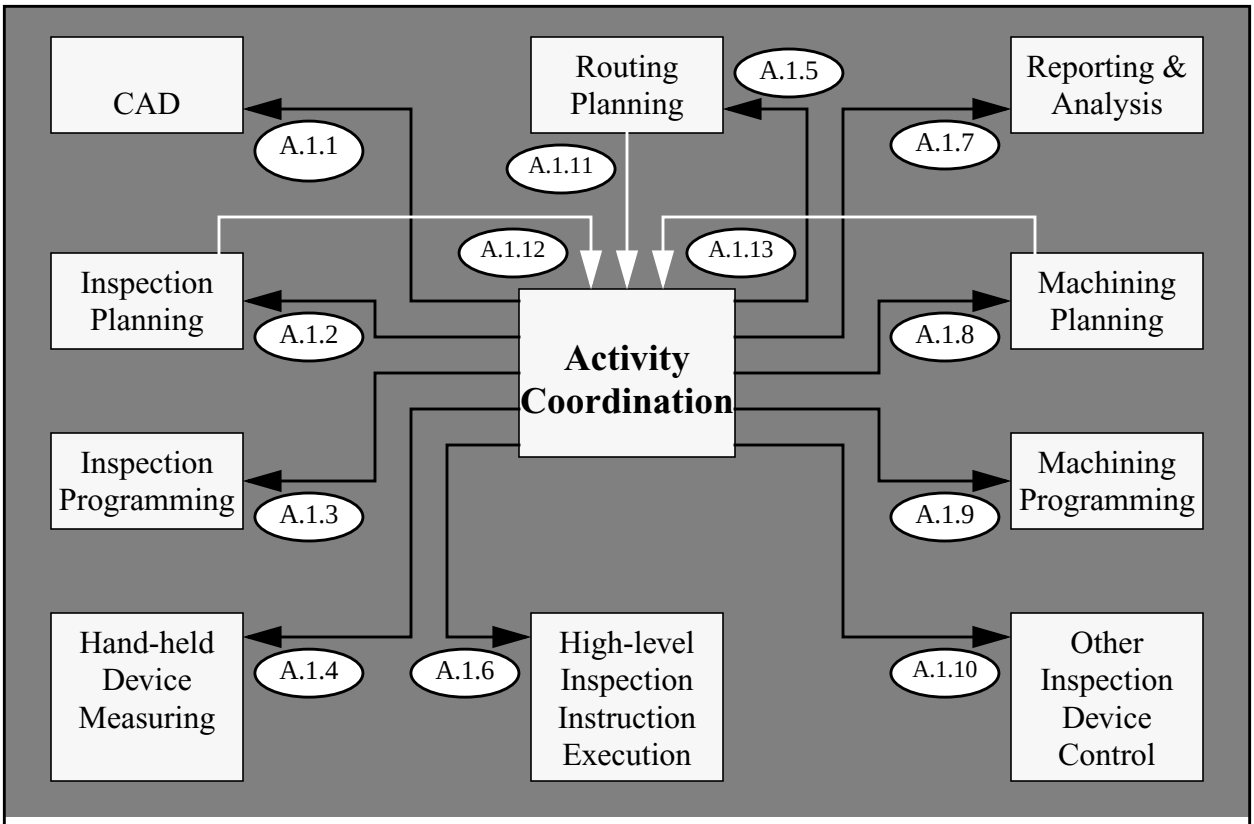


**Figure 3 Activity Coordination Interfaces**

White arrows show producer → consumer for data interfaces.
Black arrows show user→used for active interfaces.

A.1.1 Active to CAD

Activity Coordination tells CAD what to design and when to design it. There is no commonly used API for this purpose. This is a person-to-person interface in all existing industrial systems and is not likely to be automated in the near future. Standardizing this interface is of low priority (see Section 6.1).

A.1.2 Active to Inspection Planning

Activity Coordination tells Inspection Planning what to plan for and when to plan. There is no commonly used API for this purpose. This is a person-to-person interface in most existing industrial systems but has been defined as a command-and-status interface and automated in FBICS. Standardizing this interface is of low priority (see Section 6.6).

### A.1.3 Active to Inspection Programming

Activity Coordination tells Inspection Programming what to write programs for and when to write them. There is no commonly used API for this purpose. This is a person-to-person interface in most existing industrial systems but has been defined as a command-and-status interface and automated in FBICS. Standardizing this interface is of low priority (see Section 6.8).

### A.1.4 Active to Hand-held Device Measuring

Activity Coordination tells Hand-held Device Measuring what to measure and when to measure. There is no commonly used API for this purpose. This is a person-to-person interface in all existing industrial systems and is not likely to be automated. Standardizing this interface is of low priority (see Section 6.3).

### A.1.5 Active to Routing Planning

Activity Coordination tells Routing Planning what to plan for and when to plan. There is no commonly used API for this purpose. This is a person-to-person interface in all existing industrial systems but has been defined as a command for a human user to give from the user interface in FBICS. Standardizing this interface is of low priority (see Section 6.18).

### A.1.6 Active to High-level Inspection Instruction Execution

Activity Coordination tells High-level Inspection Instruction Execution what inspection programs to run and when to run them. There is no commonly used API for this purpose. This is usually a person-to-person or person-to-machine interface in existing industrial systems but has been defined as a command-and-status interface and automated in FBICS. Standardizing this interface is of low priority (see Section 6.4).

### A.1.7 Active to Reporting and Analysis

Activity Coordination tells Reporting and Analysis what data to analyze and when to do the analysis. There is no commonly used API for this purpose. An XML DTD has been drafted by DaimlerChrysler. This is a person-to-person interface in most existing industrial systems. Standardizing this interface is of medium priority (see Section 6.16).

### A.1.8 Active to Machining Planning

Activity Coordination tells Machining Planning what to plan for and when to plan. There is no commonly used API for this purpose. This is a person-to-person interface in most existing industrial systems but has been defined as a command-and-status interface and automated in FBICS. Standardizing this interface is of low priority (see Section 6.10).

### A.1.9 Active to Machining Programming

Activity Coordination tells Machining Programming what to write programs for and when to write them. There is no commonly used API for this purpose. This is a person-to-person interface

in most existing industrial systems but has been defined as a command-and-status interface and automated in FBICS. Standardizing this interface is of low priority (see Section 6.12).

A.1.10 Active to Other Inspection Device Control

Activity Coordination tells Other Inspection Device Control what other device control activities to perform and when to perform them. There is no commonly used API for this purpose. This is a person-to-person interface in all existing industrial systems. Standardizing this interface is of low priority (see Section 6.14).

A.1.11 Data from Routing Planning

Activity Coordination uses a process plan generated by Routing Planning for directing the activity of performing all machining and inspection required on a part being manufactured. This is a high-level plan saying which workstations a workpiece should go to and what should be done at each workstation. In current systems, this is often a person-to-person interface. Standardizing this interface is of low priority (see Section 6.19).

A.1.12 Data from Inspection Planning

Activity Coordination uses a process plan generated by Inspection Planning for directing the activity of performing the inspection required in one fixturing of a part being manufactured. **Standardizing this interface is of high priority** (see Section 6.6).

A.1.13 Data from Machining Planning

Activity Coordination uses a process plan generated by Machining Planning for directing the activity of performing all machining (including in-process inspection) required in one fixturing of a part being manufactured. Standardizing this interface is of medium priority (see Section 6.11).

**A.2 Computer-Aided Design**

CAD generates designs used by other modules. Different modules have differing needs for design data.

CAD is clearly not confined to the dimensional metrology domain. CAD is important as a self-contained activity and is also linked closely to other activities, particularly part forming (machining, turning, etc.). Thus, it does not make sense to develop CAD standards within the dimensional metrology community. That community should, however, participate in setting CAD standards.

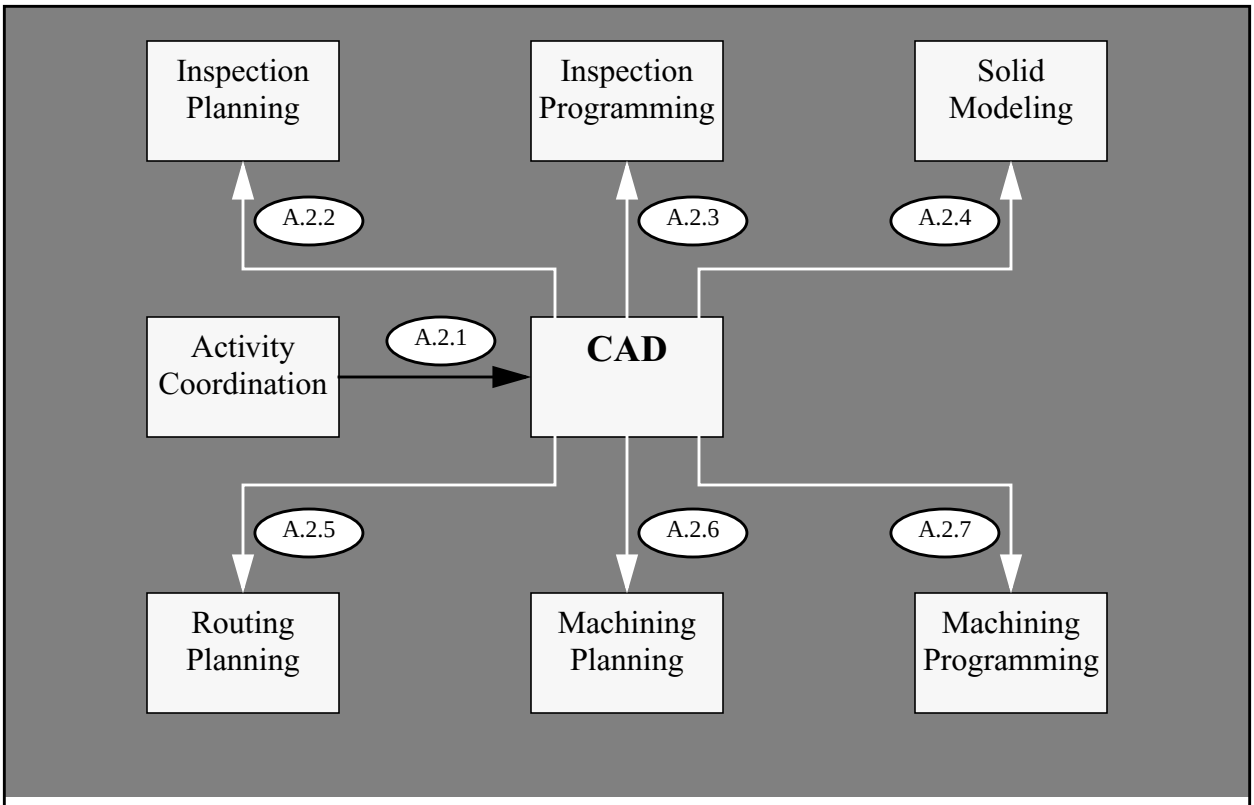


**Figure 4 CAD Interfaces**

White arrows show producer → consumer for data interfaces.
Black arrows show user→used for active interfaces.

A.2.1 Active from Activity Coordination

Activity Coordination tells CAD what to design and when to design it. There is no commonly used API for this purpose. This is a person-to-person interface in all existing industrial systems and is not likely to be automated in the near future. Standardizing this interface is of low priority (see Section 6.1).

A.2.2 Data to Inspection Planning

Inspection Planning uses designs generated by CAD. Inspection Planning requires tolerance data (including datums) in the design and may be facilitated by a feature-based design. **Standardizing this interface is of high priority** (see Section 6.2).

A.2.3 Data to Inspection Programming

Inspection Programming uses designs generated by CAD. Inspection Programming requires tolerance data (including datums) in the design and may be facilitated by a feature-based design. **Standardizing this interface is of high priority** (see Section 6.2).

A.2.4 Data to Solid Modeling

Solid Modeling uses designs generated by CAD. Solid modelers usually use boundary representations, so Solid Modeling is facilitated if CAD provides a boundary representation. **Standardizing this interface is of high priority** (see Section 6.2).

A.2.5 Data to Routing Planning

Routing Planning uses designs generated by CAD. **Standardizing this interface is of high priority** (see Section 6.2).

A.2.6 Data to Machining Planning

Machining Planning uses designs generated by CAD. For prismatic parts (those having standard features like pockets and holes), Machining Planning is helped by having a feature-based design. If Machining Planning does not receive a feature-based design from CAD, it may create one. Machining Planning requires tolerance information in the design, both for determining what machining operations to use and for determining what in-process inspection to perform. **Standardizing this interface is of high priority** (see Section 6.2).

A.2.7 Data to Machining Programming

Machining Programming uses designs generated by CAD. For prismatic parts (those having standard features like pockets and holes), Machining Programming is helped by having a feature-based design. **Standardizing this interface is of high priority** (see Section 6.2).

**A.3 Hand-held Device Measuring**

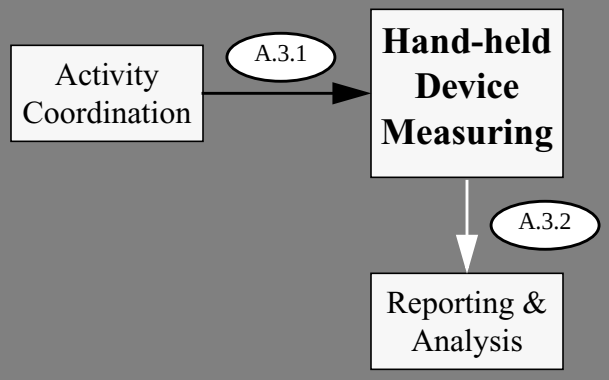


**Figure 5 Hand-held Device Measuring Interfaces**

White arrows show producer → consumer for data interfaces.
Black arrows show user → used for active interfaces.

A.3.1 Active from Activity Coordination

Activity Coordination tells Hand-held Device Measuring what to measure and when to measure. There is no commonly used API for this purpose. This is a person-to-person interface in all existing industrial systems and is not likely to be automated. Standardizing this interface is of low priority (see Section 6.3).

A.3.2 Data to Reporting and Analysis

Reporting and Analysis uses data generated by Hand-held Device Measuring. Some existing systems automate the recording of data from hand-held devices. **Standardizing this interface is of high priority** (see Section 6.17).

**A.4 High-level Inspection Instruction Execution**

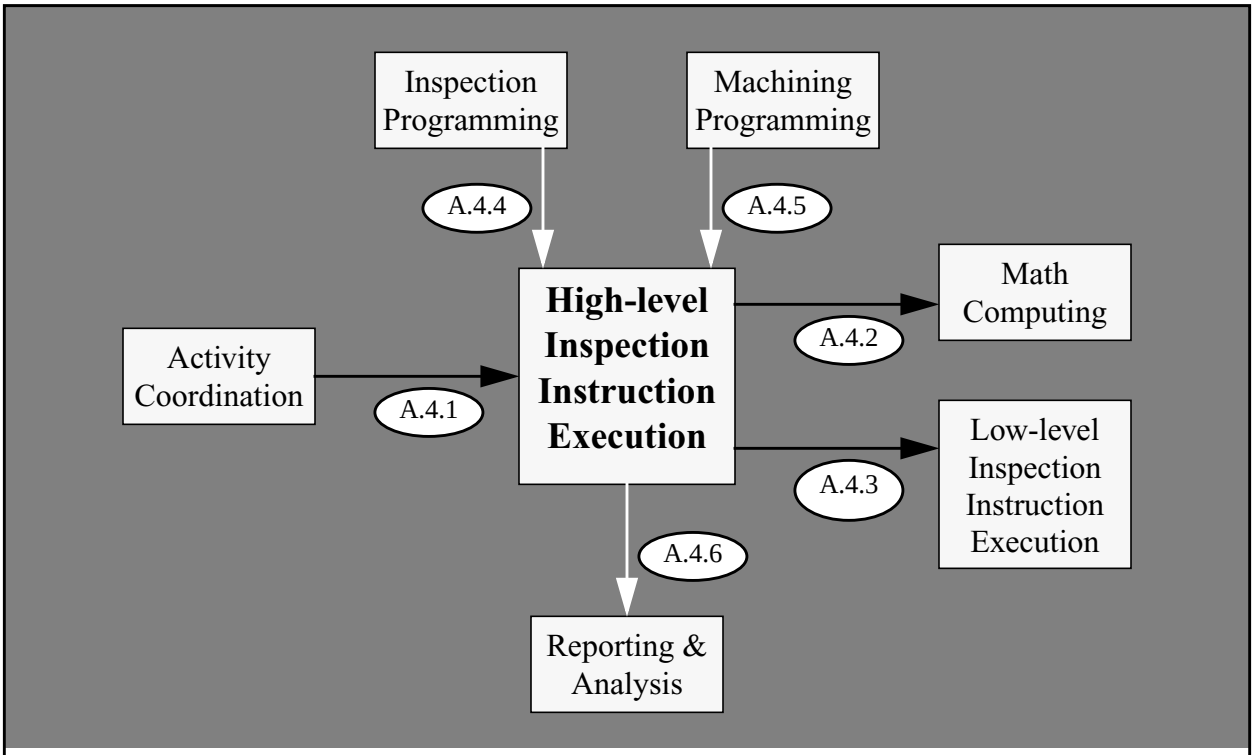


**Figure 6 High-level Inspection Instruction Execution Interfaces**

White arrows show producer → consumer for data interfaces.
Black arrows show user → used for active interfaces.

A.4.1 Active from Activity Coordination

Activity Coordination tells High-level Inspection Instruction Execution what inspection programs to run and when to run them. There is no commonly used API for this purpose. This is usually a person-to-person or person-to-machine interface in existing industrial systems but has been

defined as a command-and-status interface and automated in FBICS. Standardizing this interface is of low priority (see Section 6.4).

A.4.2 Active to Math Computing

High-level Inspection Instruction Execution requires specialized math computations to be performed if (as in a DMIS interpreter) it does data analysis while it runs. Fitting a cylinder to a set of data points is an example of such a computation. Commercial solid modelers do not do these computations. High-level Inspection Instruction Execution has an active interface to Math Computing for this purpose. Math Computing may also be given less demanding tasks, such as handling coordinate transformations. Math computing returns numerical data to High-level Inspection Instruction Execution on this interface. This interface is automated in all systems that have it; this includes FBICS. Standardizing this interface is of medium priority (see Section 6.13).

A.4.3 Active to Low-level Inspection Instruction Execution

High-level Inspection Instruction Execution gives commands to Low-level Inspection Instruction Execution in order to carry out a high-level program. Data (the location of a probe trip point, for example) is returned by Low-level Inspection Instruction Execution. There are several proprietary sets of commands used for this interface in commercial systems. A standard set is under development. This interface is automated in all systems. **Standardizing this interface is of high priority** (see Section 6.9).

A.4.4 Data from Inspection Programming

High-level Inspection Instruction Execution uses inspection programs generated by Inspection Programming. There is one standard language (DMIS) plus several proprietary languages for expressing inspection programs. **Standardizing this interface is of high priority** (see Section 6.5).

A.4.5 Data from Machining Programming

High-level Inspection Instruction Execution uses instructions generated by Machining Programming. See the beginning of Section A.9 for a discussion of this.

A.4.6 Data to Reporting and Analysis

Reporting and Analysis uses data generated by High-level Inspection Instruction Execution. The data coming out of High-level Inspection Instruction Execution may be raw data (such as probe trip positions), low-level processed data (such as points on a surface), or high-level processed data (such as the diameter of a measured cylinder). **Standardizing this interface is of high priority** (see Section 6.17).

### A.5 Inspection Planning

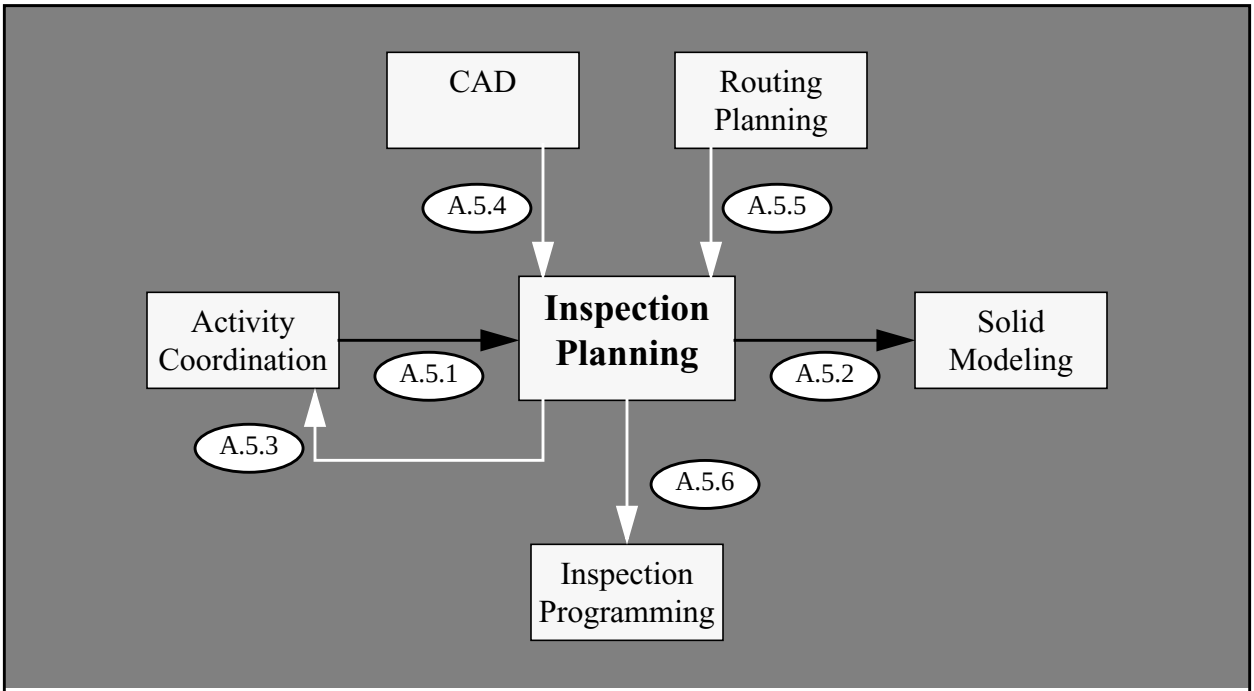


**Figure 7 Inspection Planning Interfaces**

White arrows show producer → consumer for data interfaces.
Black arrows show user → used for active interfaces.

A.5.1 Active from Activity Coordination

Activity Coordination tells Inspection Planning what to plan for and when to plan. There is no commonly used API for this purpose. This is a person-to-person interface in most existing industrial systems but has been defined as a command-and-status interface and automated in FBICS. Standardizing this interface is of low priority (see Section 6.6).

A.5.2 Active to Solid Modeling

Inspection Planning uses Solid Modeling as a source of geometry data (what is the direction of the axis of a hole, for example). This interface is automated in all systems that have the interface; this includes FBICS. There is no standard set of commands for using Solid Modeling. Existing commercial solid modelers each have their own proprietary API. Standardizing this interface is of low priority (see Section 6.21).

A.5.3 Data to Activity Coordination

Activity Coordination uses a process plan generated by Inspection Planning for directing the activity of performing the inspection required in one fixturing of a part being manufactured. **Standardizing this interface is of high priority** (see Section 6.6).

A.5.4 Data from CAD

Inspection Planning uses designs generated by CAD. Inspection Planning requires tolerance data (including datums) in the design and may be facilitated by a feature-based design. **Standardizing this interface is of high priority** (see Section 6.2).

A.5.5 Data from Routing Planning

Inspection Planning uses inspection setup data and geometric feature data from Routing Planning. This is a barely formalized interface in existing industrial systems and is likely to be tied closely to the active interface from Activity Coordination. In FBICS, this interface is fully formalized, including a setup file and a list of features to inspect generated by Routing Planning. Standardizing this interface is of low priority (see Section 6.20).

A.5.6 Data to Inspection Programming

Inspection Programming uses geometry and tolerance data from Inspection Planning. This data is referenced in commands to Inspection Programming from Activity Coordination. Existing commercial systems generally use some proprietary CAD format. FBICS uses STEP AP 224 data, defined in an EXPRESS information model. In FBICS, data regarding individual geometric features and tolerances are extracted from AP 224 CAD data and written in STEP exchange files. STEP AP 219, currently being developed, is expected to provide data that may be used on this interface. **Standardizing this interface is of high priority** (see Section 6.2).

### A.6 Inspection Programming

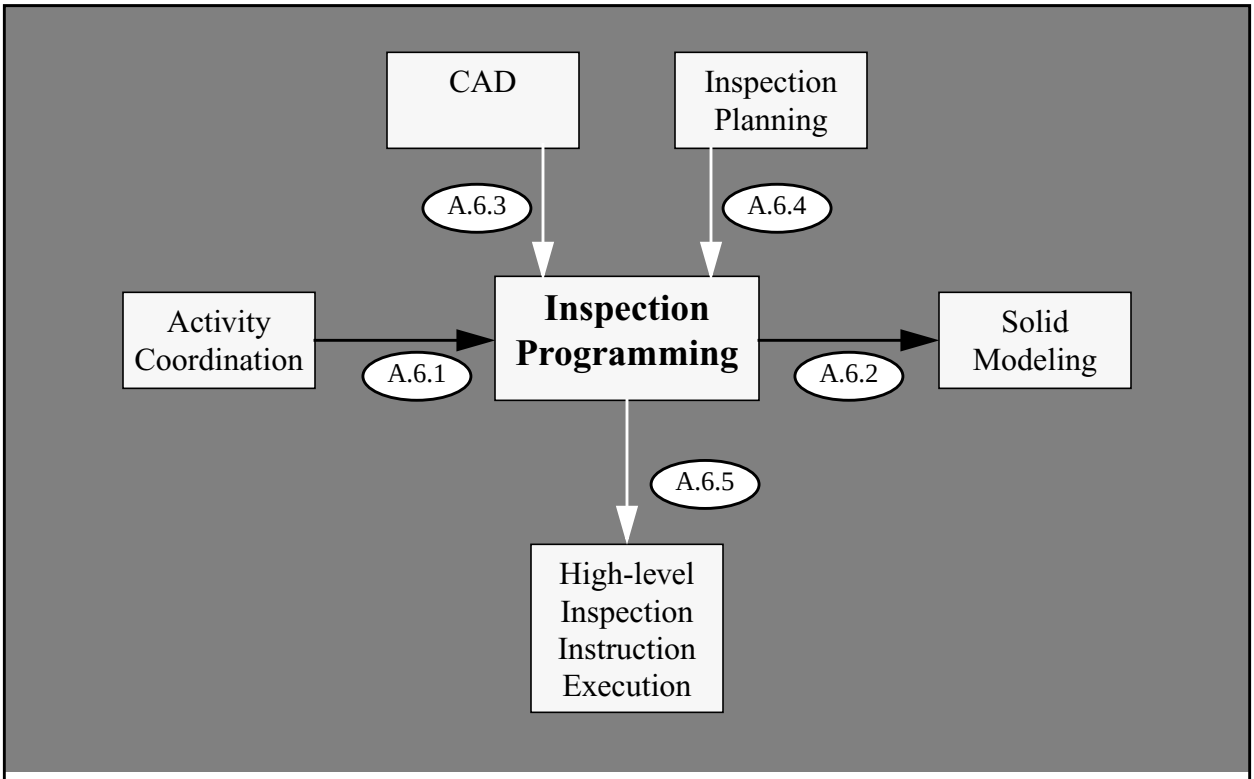


**Figure 8 Inspection Programming Interfaces**

White arrows show producer → consumer for data interfaces.
Black arrows show user → used for active interfaces.

A.6.1 Active from Activity Coordination

Activity Coordination tells Inspection Programming what to write programs for and when to write them. There is no commonly used API for this purpose. This is a person-to-person interface in most existing industrial systems but has been defined as a command-and-status interface and automated in FBICS. Standardizing this interface is of low priority (see Section 6.8).

A.6.2 Active to Solid Modeling

Inspection Programming uses Solid Modeling as a source of geometry data (is a candidate inspection point on the surface of the part, for example). This interface is automated in all systems that have the interface; this includes FBICS. There is no standard set of commands for using Solid Modeling. Existing commercial solid modelers each have their own proprietary API. Standardizing this interface is of low priority (see Section 6.21).

A.6.3 Data from CAD

Inspection Programming uses designs generated by CAD. Inspection Programming requires tolerance data (including datums) in the design and may be facilitated by a feature-based design. **Standardizing this interface is of high priority** (see Section 6.2).

A.6.4 Data from Inspection Planning

Inspection Programming uses geometry and tolerance data from Inspection Planning. This data is referenced in commands to Inspection Programming from Activity Coordination. Existing commercial systems generally use some proprietary CAD format. FBICS uses STEP AP 224 data, defined in an EXPRESS information model. In FBICS, data regarding individual geometric features and tolerances are extracted from AP 224 CAD data and written in STEP exchange files. STEP AP 219, currently being developed, is expected to provide data that may be used on this interface. **Standardizing this interface is of high priority** (see Section 6.2).

A.6.5 Data to High-level Inspection Instruction Execution

High-level Inspection Instruction Execution uses inspection programs generated by Inspection Programming. There is one standard language (DMIS) plus several proprietary languages for expressing inspection programs. **Standardizing this interface is of high priority** (see Section 6.5).

**A.7 Low-level Inspection Instruction Execution**

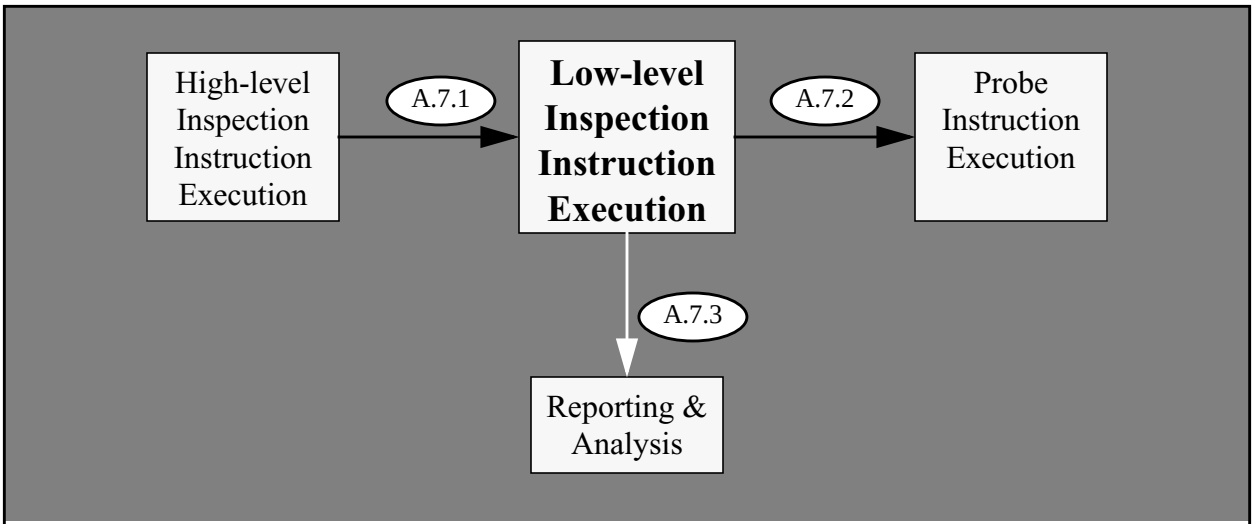


**Figure 9 Low-level Inspection Instruction Execution Interfaces**

White arrows show producer → consumer for data interfaces.
Black arrows show user → used for active interfaces.

A.7.1 Active From High-level Inspection Instruction Execution

High-level Inspection Instruction Execution gives commands to Low-level Inspection Instruction Execution in order to carry out a high-level program. Data (the location of a probe trip point, for example) is returned by Low-level Inspection Instruction Execution. There are several proprietary sets of commands used for this interface in commercial systems. A standard set is under development. This interface is automated in all systems. **Standardizing this interface is of high priority** (see Section 6.9).

A.7.2 Active To Probe Instruction Execution

Low-level Inspection Instruction Execution gives commands to Probe Instruction Execution in order to carry out low-level inspection commands. This interface is automated in all systems that have it. Data (the location of a probe trip point, for example) is returned by Probe Instruction Execution. Proprietary sets of commands are used for this interface in commercial systems. A proposed standard set has been drafted. Standardizing this interface is of medium priority (see Section 6.15).

A.7.3 Data To Reporting and Analysis

Reporting and Analysis uses data from Low-level Inspection Instruction Execution. There is no widely used format for this data. This interface is automated in all systems that have it, since the same data (probe trip points, normally) might be sent by High-level Inspection Instruction Execution. FBICS does not have this interface. **Standardizing this interface is of high priority** (see Section 6.17).

### A.8 Machining Planning

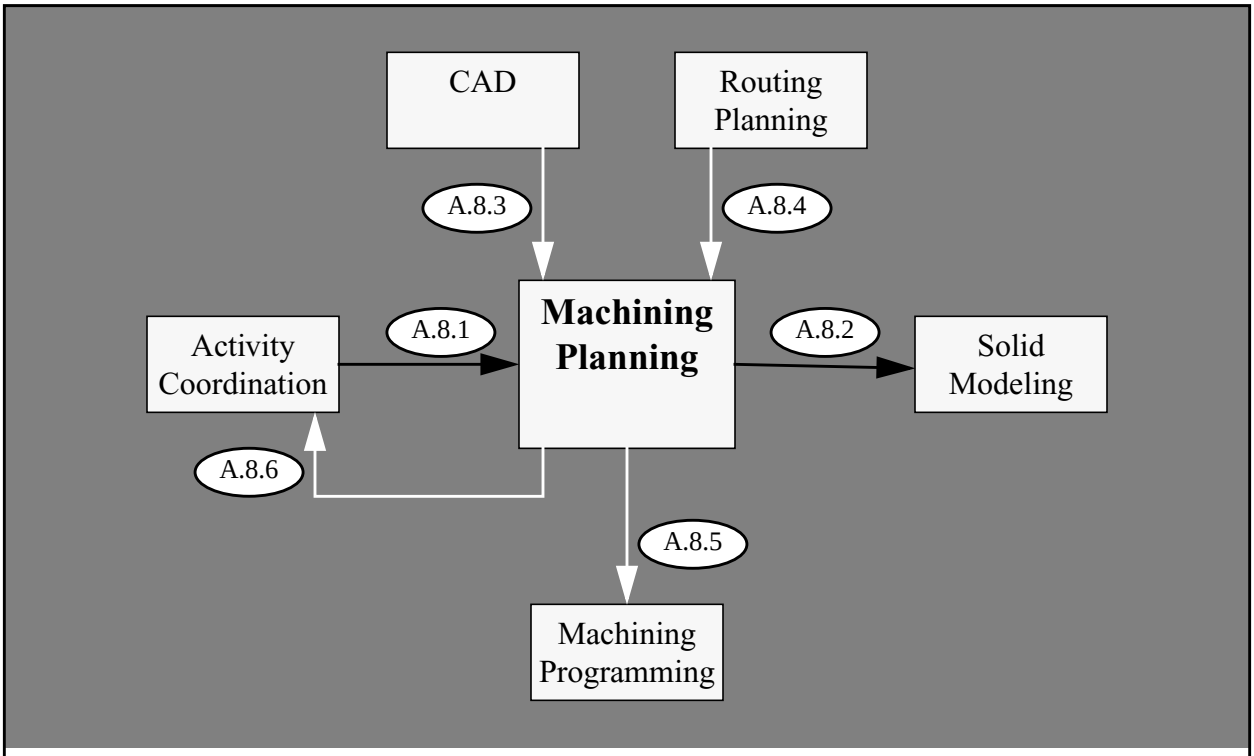


**Figure 10 Machining Planning Interfaces**

White arrows show producer → consumer for data interfaces.
Black arrows show user → used for active interfaces.

A.8.1 Active From Activity Coordination

Activity Coordination tells Machining Planning what to plan for and when to plan. There is no commonly used API for this purpose. This is a person-to-person interface in most existing industrial systems but has been defined as a command-and-status interface and automated in FBICS. Standardizing this interface is of low priority (see Section 6.10).

A.8.2 Active To Solid Modeling

Machining Planning uses Solid Modeling as a source of geometry data (what is the direction of the axis of a hole, for example). This interface is automated in all systems that have the interface; this includes FBICS. There is no standard set of commands for using Solid Modeling. Existing commercial solid modelers each have their own proprietary API. Standardizing this interface is of low priority (see Section 6.21).

A.8.3 Data from CAD

Machining Planning uses designs generated by CAD. For prismatic parts (those having standard features like pockets and holes), Machining Planning is helped by having a feature-based design.

If Machining Planning does not receive a feature-based design from CAD, it may create one. Machining Planning requires tolerance information in the design, both for determining what machining operations to use and for determining what in-process inspection to perform. **Standardizing this interface is of high priority** (see Section 6.2).

A.8.4 Data From Routing Planning

Machining Planning uses machining setup data and geometric feature data from Routing Planning. This is a barely formalized interface in existing industrial systems and is likely to be tied closely to the active interface from Activity Coordination. In FBICS, this interface is fully formalized, including a setup file and a list of features to machine (possibly with in-process inspection) generated by Routing Planning. Standardizing this interface is of low priority (see Section 6.20).

A.8.5 Data To Machining Programming

Machining Programming uses geometry data from Machining Planning. This data is referenced in commands to Machining Programming from Activity Coordination. Existing commercial systems generally use some proprietary CAD format. FBICS uses STEP AP 224 data, defined in an EXPRESS information model. In FBICS, data regarding individual geometric features are extracted from AP 224 CAD data and written in STEP exchange files. **Standardizing this interface is of high priority** (see Section 6.2).

A.8.6 Data to Activity Coordination

Activity Coordination uses a process plan generated by Machining Planning for directing the activity of performing all machining (including in-process inspection) required in one fixturing of a part being manufactured. Standardizing this interface is of medium priority (see Section 6.11).

**A.9 Machining Programming**

The usual method of dealing with a touch probe on a machining center is to put the probe in a tool holder, so that it may be put into the spindle and taken out like any other tool A dialect of RS274 is used that includes M and G codes for probing, so that the probe may be programmed like any other tool. Results of probing may be saved in the controller's table of machine parameters. The results may be used in control of subsequent machining.

Machining Programming includes an interface to High-level Inspection Instruction Execution. This means executing high-level inspection instructions on a machining center. The machining commands in a machining program would, of course, be executed by a module for high-level machining instruction execution, but machining instructions are out of scope and not shown on the diagrams in this analysis.

In existing systems, high-level inspection instructions executed on a machining center are written using non-standard M and G codes. These allow only for simple inspection routines. Thought should be given as to whether high-level inspection instructions for a machining center should be written according to the same standard (DMIS) as for a CMM. This would allow for more

advanced inspection methods but would require that the machine tool controller include a DMIS interpreter as well as an M and G code interpreter. Settling this issue is of low priority.

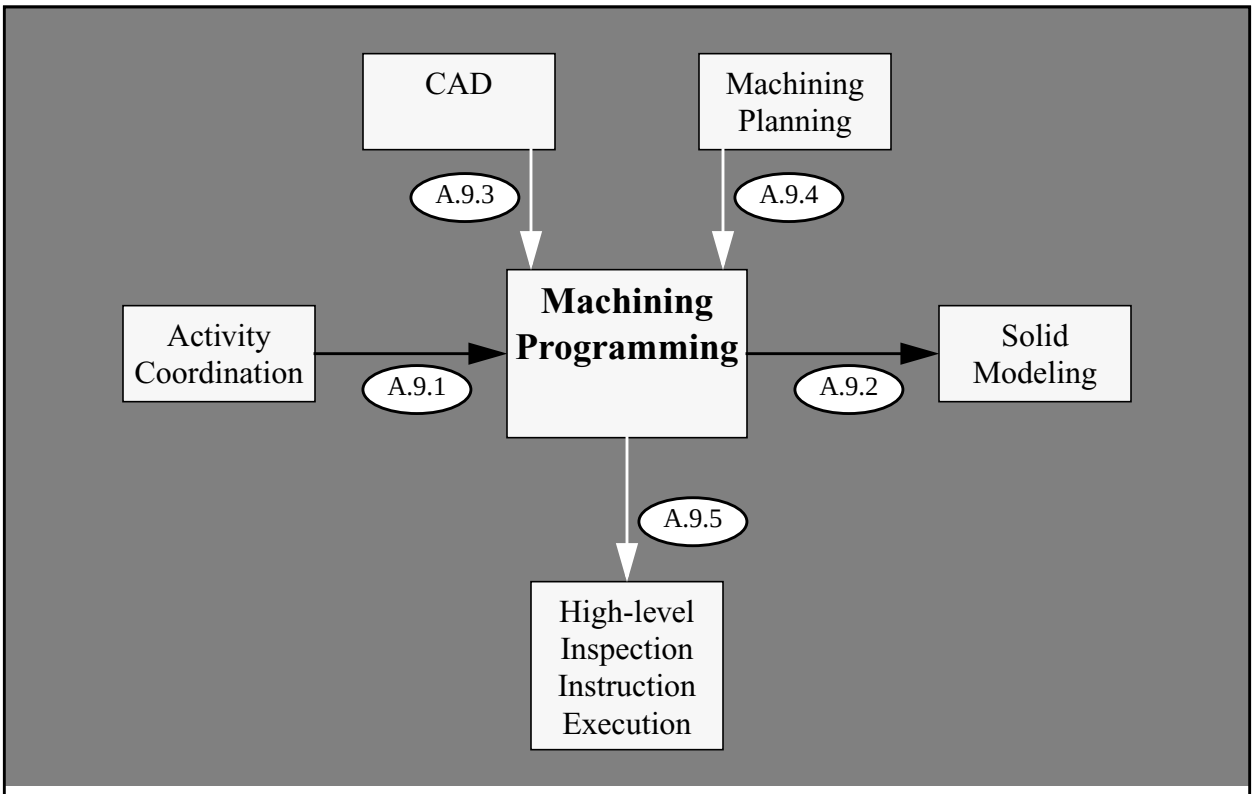


**Figure 11 Machining Programming Interfaces**

White arrows show producer → consumer for data interfaces.
Black arrows show user → used for active interfaces.

A.9.1 Active From Activity Coordination

Activity Coordination tells Machining Programming what to write programs for and when to write them. There is no commonly used API for this purpose. This is a person-to-person interface in most existing industrial systems but has been defined as a command-and-status interface and automated in FBICS. Standardizing this interface is of low priority (see Section 6.12).

A.9.2 Active To Solid Modeling

Machining Programming uses Solid Modeling as a source of geometry data (for picking nominal probe points, for example). This interface is automated in all systems that have the interface; this includes FBICS. There is no standard set of commands for using Solid Modeling. Existing commercial solid modelers each have their own proprietary API. Standardizing this interface is of low priority (see Section 6.21).

A.9.3 Data From CAD

Machining Programming uses designs generated by CAD. For prismatic parts (those having standard features like pockets and holes), Machining Programming is helped by having a feature-based design. **Standardizing this interface is of high priority** (see Section 6.2).

A.9.4 Data From Machining Planning

Machining Programming uses geometry data from Machining Planning. This data is referenced in commands to Machining Programming from Activity Coordination. Existing commercial systems generally use some proprietary CAD format. FBICS uses STEP AP 224 data, defined in an EXPRESS information model. In FBICS, data regarding individual geometric features are extracted from AP 224 CAD data and written in STEP exchange files. **Standardizing this interface is of high priority** (see Section 6.2).

A.9.5 Data to High-level Inspection Instruction Execution

High-level Inspection Instruction Execution uses instructions generated by Machining Programming. See the beginning of Section A.9 for a discussion of this.

**A.10 Math Computing**

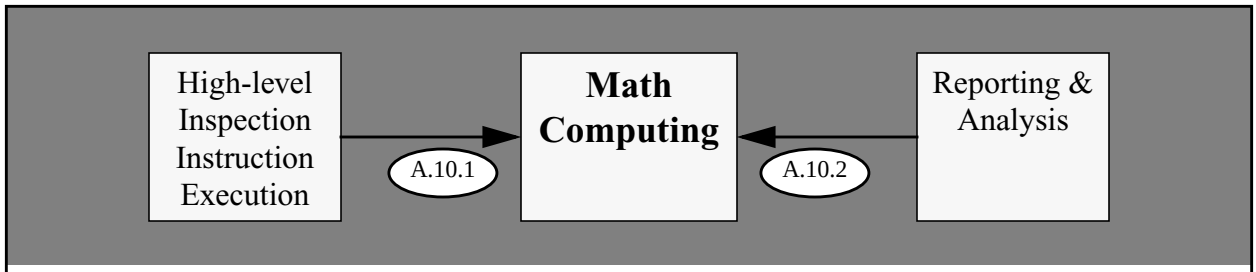


**Figure 12 Math Computing Interfaces**

White arrows show producer → consumer for data interfaces.
Black arrows show user → used for active interfaces.

A.10.1 Active from High-level Inspection Instruction Execution

High-level Inspection Instruction Execution requires specialized math computations to be performed if (as in a DMIS interpreter) it does data analysis while it runs. Fitting a cylinder to a set of data points is an example of such a computation. Commercial solid modelers do not do these computations. High-level Inspection Instruction Execution has an active interface to Math Computing for this purpose. Math Computing may also be given less demanding tasks, such as handling coordinate transformations. Math computing returns numerical data to High-level Inspection Instruction Execution on this interface. This interface is automated in all systems that have it; this includes FBICS. Standardizing this interface is of medium priority (see Section 6.13).

A.10.2 Active from Reporting and Analysis

Reporting and Analysis requires specialized math computations to be performed. Fitting a cylinder to a set of data points is an example of such a computation. Commercial solid modelers do not do these computations. Reporting and Analysis has an active interface to Math Computing for this purpose. Math Computing returns numerical data to Reporting and Analysis on this interface. This interface is automated in all systems that have it. FBICS does not have this interface. Standardizing this interface is of medium priority (see Section 6.13).

**A.11 Other Inspection Device Control**

Other Inspection Device Control requires mathematical processing, such as extracting point data from range data. Since the type of processing is specific to the type of other device (photogrammetry equipment or theodolite), we assume that this processing is done by the controller of the other device, not by Math Computing. Thus, we do not include an interface to Math Computing. It may be that this interface should be added, since Math Computing might well be used for further processing of point data.

A data interface from Inspection Programming to Other Inspection Device Control might be added. Other devices may be programmable and, if so, need to get programs from somewhere.

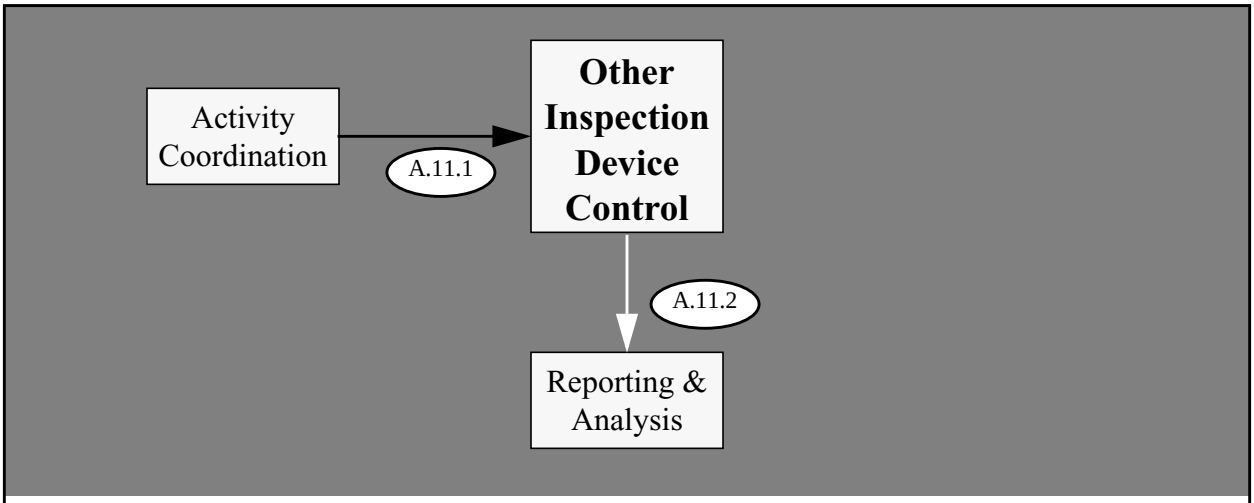


**Figure 13 Other Inspection Device Control Interfaces**

White arrows show producer → consumer for data interfaces.
Black arrows show user → used for active interfaces.

A.11.1 Active from Activity Coordination

Activity Coordination tells Other Inspection Device Control what other device control activities to perform and when to perform them. There is no commonly used API for this purpose. This is a person-to-person interface in all existing industrial systems. Standardizing this interface is of low priority (see Section 6.14).

A.11.2 Data to Reporting and Analysis

Reporting and Analysis uses data generated by Other Inspection Device Control. Some existing systems automate the recording of data from hand-held devices. **Standardizing this interface is of high priority** (see Section 6.17).

### A.12 Probe Instruction Execution

Probe Instruction Execution has no downward interfaces to other dimensional metrology activities.

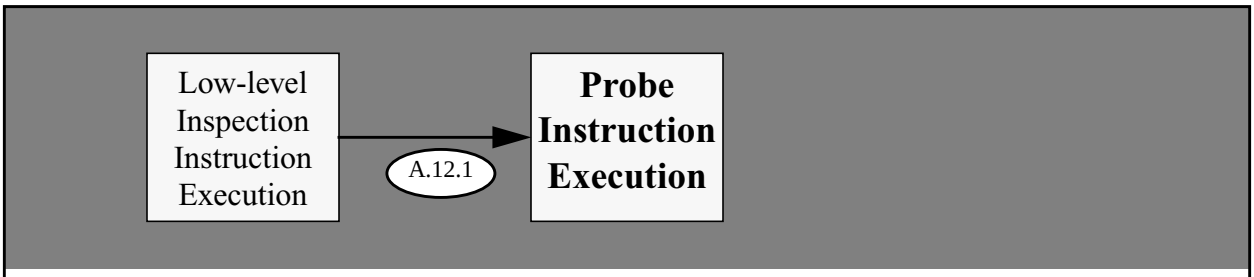


**Figure 14 Probe Instruction Execution Interfaces**

White arrows show producer → consumer for data interfaces.
Black arrows show user → used for active interfaces.

A.12.1 Active from Low-Level Inspection Instruction Execution

Low-level Inspection Instruction Execution gives commands to Probe Instruction Execution in order to carry out low-level inspection commands. This interface is automated in all systems that have it. Data (the location of a probe trip point, for example) is returned by Probe Instruction Execution. Proprietary sets of commands are used for this interface in commercial systems. A standard set has been drafted. Standardizing this interface is of medium priority (see Section 6.15).

### A.13 Reporting and Analysis

Reporting and Analysis is shown in Figure 15 with no CAD data going into it. This is not because Reporting and Analysis does not need CAD data to compare with actual data; that is needed. It is assumed, however, that Solid Modeling (which is used by Reporting and Analysis, and does have CAD data as input) will deal with the CAD data.

Reporting and Analysis is shown in Figure 15 with no data coming out of it. This is misleading, since the whole point of Reporting and Analysis is to prepare useful data. The data is currently prepared for the use of humans only and does not feed into other modules. A person reads a report or looks at data displayed pictorially, decides if the data looks OK, and, if not, what to do about it.

In the future, Reporting and Analysis should return digested data directly to other modules. For example, it might suggest to Inspection Programming that more points be used to inspect a particular feature, or that another feature does not need to be inspected on every part of a given

design because that feature has never been out of tolerance on any part of that design. Data interfaces of this sort are currently rare in industry. Standardizing such interfaces is of very low priority at this time.

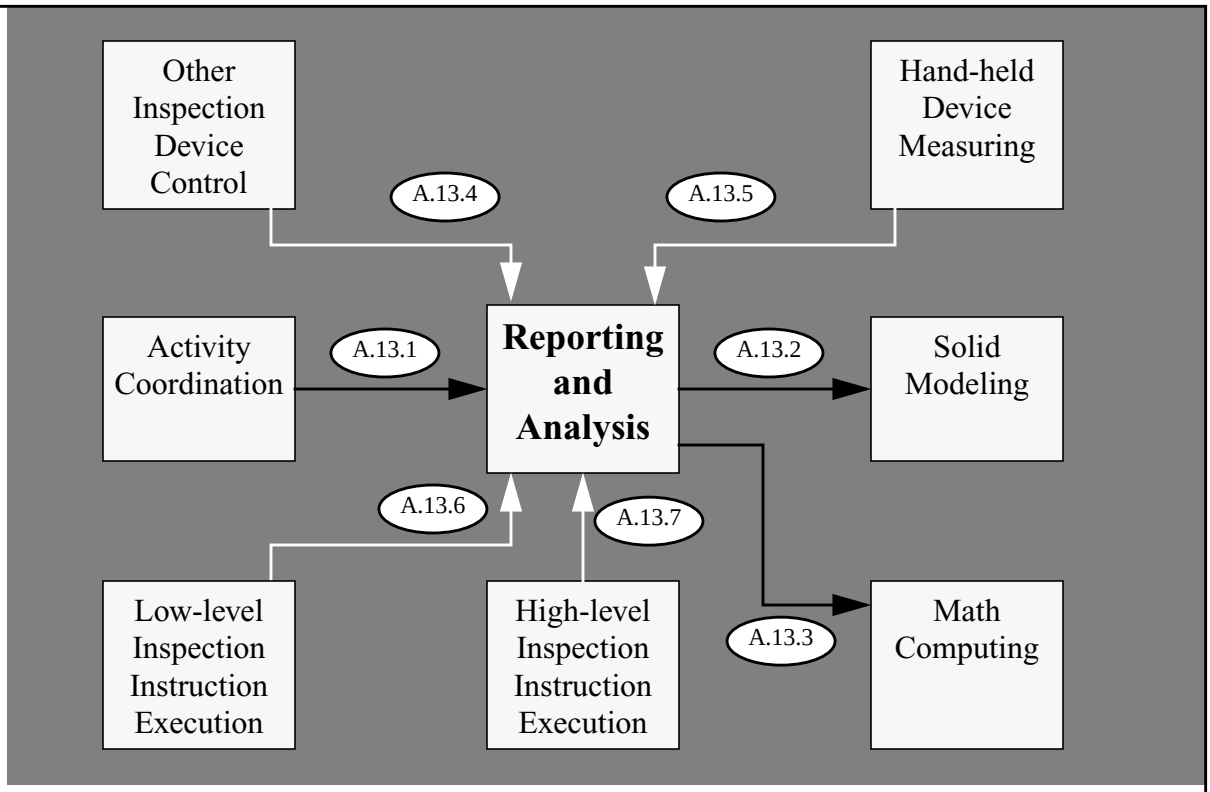


**Figure 15 Reporting and Analysis Interfaces**

White arrows show producer → consumer for data interfaces.
Black arrows show user → used for active interfaces.

A.13.1 Active from Activity Coordination

Activity Coordination tells Reporting and Analysis what data to analyze and when to do the analysis. There is no commonly used API for this purpose. An XML DTD has been drafted by DaimlerChrysler. This is a person-to-person interface in most existing industrial systems. Standardizing this interface is of medium priority (see Section 6.16).

A.13.2 Active to Solid Modeling

Reporting and Analysis may need the services of Solid Modeling to do analyses. This interface is automated in all systems that have it. There is no standard set of commands for using Solid Modeling. Existing commercial solid modelers each have their own proprietary API. Standardizing this interface is of low priority (see Section 6.21).

A.13.3 Active to Math Computing

Reporting and Analysis requires specialized math computations to be performed. Fitting a cylinder to a set of data points is an example of such a computation. Commercial solid modelers do not do these computations. Reporting and Analysis has an active interface to Math Computing for this purpose. Math Computing returns numerical data to Reporting and Analysis on this interface. This interface is automated in all systems that have it. FBICS does not have this interface. Standardizing this interface is of medium priority (see Section 6.13).

A.13.4 Data from Other Inspection Device Control

Reporting and Analysis uses data generated by Other Inspection Device Control. Some existing systems automate the recording of data from hand-held devices. **Standardizing this interface is of high priority** (see Section 6.17).

A.13.5 Data from Hand-held Device Measuring

Reporting and Analysis uses data generated by Hand-held Device Measuring. Some existing systems automate the recording of data from hand-held devices. **Standardizing this interface is of high priority** (see Section 6.17).

A.13.6 Data from Low-level Inspection Instruction Execution

Reporting and Analysis uses data from Low-level Inspection Instruction Execution. There is no widely used format for this data. This interface is automated in all systems that have it, since the same data (probe trip points, normally) might be sent by High-level Inspection Instruction Execution. FBICS does not have this interface. **Standardizing this interface is of high priority** (see Section 6.17).

A.13.7 Data from High-level Inspection Instruction Execution

Reporting and Analysis uses data generated by High-level Inspection Instruction Execution. The data coming out of High-level Inspection Instruction Execution may be raw data (such as probe trip positions), low-level processed data (such as points on a surface), or high-level processed data (such as the diameter of a measured cylinder). **Standardizing this interface is of high priority** (see Section 6.17).

**A.14 Routing Planning**

For production line rather than discrete part manufacturing, the functions of Routing Planning may be performed when the production line is laid out.

Routing Planning will typically produce designs of in-process workpieces, and may also produce setup instructions, in addition to producing a routing plan.

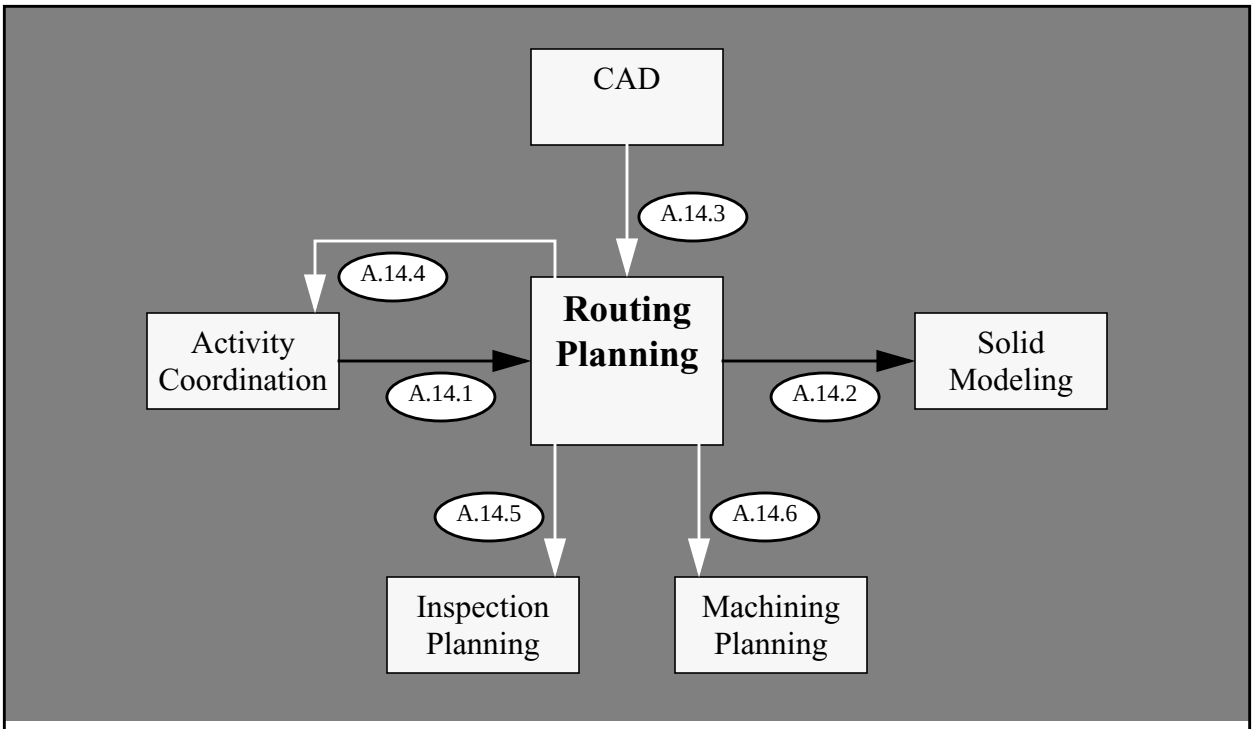


**Figure 16 Routing Planning Interfaces**

White arrows show producer → consumer for data interfaces.
Black arrows show user → used for active interfaces.

A.14.1 Active from Activity Coordination

Activity Coordination tells Routing Planning what to plan for and when to plan. There is no commonly used API for this purpose. This is a person-to-person interface in all existing industrial systems but has been defined as a command for a human user to give from the user interface in FBICS. Standardizing this interface is of low priority (see Section 6.18).

A.14.2 Active to Solid Modeling

Routing Planning uses Solid Modeling to help with geometric reasoning (what features do not need to be machined because they do not intersect the initial workpiece, for example). This interface is automated in all systems that have the interface; this includes FBICS. There is no standard set of commands for using Solid Modeling. Existing commercial solid modelers each have their own proprietary API. Standardizing this interface is of low priority (see Section 6.21).

A.14.3 Data from CAD

Routing Planning uses designs generated by CAD. **Standardizing this interface is of high priority** (see Section 6.2).

### A.14.4 Data to Activity Coordination

Activity Coordination uses a process plan generated by Routing Planning for directing the activity of performing all machining and inspection required on a part being manufactured. This is a high-level plan saying which workstations a workpiece should go to and what should be done at each workstation. In current systems, this is often a person-to-person interface. Standardizing this interface is of low priority (see Section 6.19).

### A.14.5 Data to Inspection Planning

Inspection Planning uses inspection setup data and geometric feature data from Routing Planning. This is a barely formalized interface in existing industrial systems and is likely to be tied closely to the active interface from Activity Coordination. In FBICS, this interface is fully formalized, including a setup file and a list of features to inspect generated by Routing Planning. Standardizing this interface is of low priority (see Section 6.20).

### A.14.6 Data to Machining Planning

Machining Planning uses machining setup data and geometric feature data from Routing Planning. This is a barely formalized interface in existing industrial systems and is likely to be tied closely to the active interface from Activity Coordination. In FBICS, this interface is fully formalized, including a setup file and a list of features to machine (possibly with in-process inspection) generated by Routing Planning. Standardizing this interface is of low priority (see Section 6.20).

### A.15 Solid Modeling

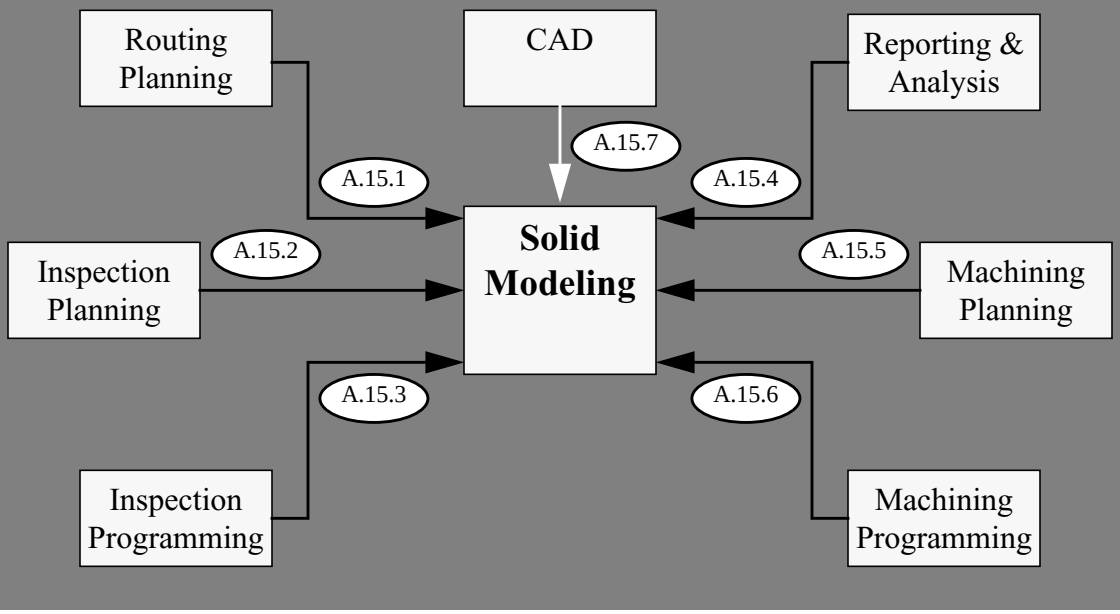


**Figure 17 Solid Modeling Interfaces**

White arrows show producer → consumer for data interfaces.
Black arrows show user → used for active interfaces.

A solid modeler may be needed by CAD, Inspection Planning, Inspection Programming, Machining Planning, Machining Programming, and Routing Planning. Solid Modeling is performed by a solid modeler. Solid modelers are large, complex programs available from only a handful of vendors. Hundreds of person-years are required to build a good one. The best-known solid modelers are ACIS and Parasolid.

CAD systems generally come with a solid modeler built in, so the independent solid modeler used by the rest of a dimensional metrology system may not have to be used by CAD. A solid modeler built into a CAD system cannot be expected to be available for use by other modules.

Solid modelers have not traditionally been connected to many other modules, but more and more modules are using them. Since many modules can potentially benefit from using them and modern PCs have the required speed and memory for fast solid modeling, it is expected that in the future many modules will use them.

It is not clear whether a single central solid modeler will be used, or several. In this analysis, we use a central solid modeler. Solid modelers require a lot of RAM memory (100 Mbytes when dealing with a large model is not unusual), so for most PCs today, running a central solid modeler is more practical than running several modelers. Also, if a central modeler is used, it is relatively straightforward to compare solid models used by different modules. This is a useful capability, since it is often desirable to check that different modules are dealing with the same shape.

A central modeler for a dimensional metrology system will require a standard API that can serve all the modules that use it. Distributed modelers can get by with a separate API for each domain.

Existing solid modelers come with very big, domain-independent APIs. A higher-level, much smaller, domain-specific API was developed and is used for FBICS. The higher-level API is implemented via calls to a commercial low-level API. An interesting research issue is whether it is feasible to define a smallish solid modeler API covering all the requirements of dimensional metrology. If so, it would be feasible and useful to have it as a standard.

### A.15.1 Active from Routing Planning

Routing Planning uses Solid Modeling to help with geometric reasoning (what features do not need to be machined because they do not intersect the initial workpiece, for example). This interface is automated in all systems that have the interface; this includes FBICS. There is no standard set of commands for using Solid Modeling. Existing commercial solid modelers each have their own proprietary API. Standardizing this interface is of low priority (see Section 6.21).

### A.15.2 Active from Inspection Planning

Inspection Planning uses Solid Modeling as a source of geometry data (what is the direction of the axis of a hole, for example). This interface is automated in all systems that have the interface; this includes FBICS. There is no standard set of commands for using Solid Modeling. Existing commercial solid modelers each have their own proprietary API. Standardizing this interface is of low priority (see Section 6.21).

### A.15.3 Active from Inspection Programming

Inspection Programming uses Solid Modeling as a source of geometry data (is a candidate inspection point on the surface of the part, for example). This interface is automated in all systems that have the interface; this includes FBICS. There is no standard set of commands for using Solid Modeling. Existing commercial solid modelers each have their own proprietary API. Standardizing this interface is of low priority (see Section 6.21).

### A.15.4 Active from Reporting and Analysis

Reporting and Analysis may need the services of Solid Modeling to do analyses. This interface is automated in all systems that have it. There is no standard set of commands for using Solid Modeling. Existing commercial solid modelers each have their own proprietary API. Standardizing this interface is of low priority (see Section 6.21).

### A.15.5 Active from Machining Planning

Machining Planning uses Solid Modeling as a source of geometry data (what is the direction of the axis of a hole, for example). This interface is automated in all systems that have the interface; this includes FBICS. There is no standard set of commands for using Solid Modeling. Existing commercial solid modelers each have their own proprietary API. Standardizing this interface is of low priority (see Section 6.21).

A.15.6 Active from Machining Programming

Machining Programming uses Solid Modeling as a source of geometry data (for picking nominal probe points, for example). This interface is automated in all systems that have the interface; this includes FBICS. There is no standard set of commands for using Solid Modeling. Existing commercial solid modelers each have their own proprietary API. Standardizing this interface is of low priority (see Section 6.21).

A.15.7 Data from CAD

Solid Modeling uses designs generated by CAD. Solid modelers usually use boundary representations, so Solid Modeling is facilitated if CAD provides a boundary representation. **Standardizing this interface is of high priority** (see Section 6.2).

## Appendix B Glossary

ACIS — a commercial solid modeler
ALPS — “A Language for Process Specification” (a process planning language)
ANSI — “American National Standards Institute” (a U.S. standards-approving organization)
AP — “application protocol” (a generic STEP term for a standard to be applied)
AP 203 — an ISO STEP international standard for configuration controlled design
AP 214 — an ISO STEP international standard for automotive data
AP 213 — a draft ISO STEP international standard for machining process plans
AP 219 — a developing ISO STEP international standard for dimensional metrology
AP 224 — an ISO STEP international standard for machining features
API — “application programming interface” (a generic term)
B&S — “Brown and Sharpe” (a company)
CAM-I — “Consortium for Advanced Manufacturing - International”
C++ — a widely used programming language
CMM — “coordinate measuring machine”
CMM-driver — a language under development for low-level CMM control
CORBA — “Common Object Request Broker Architecture”
DMIS — “Dimensional Measuring Interface Standard” (an ANSI/CAM-I standard)
DMIS Part 1 — a combined file format and semantic specification of DMIS
DMIS Part 2 — an object version of DMIS with semantics subordinate to DMIS Part 1
DNSC — “DMIS National Standards Committee” (a CAM-I committee)
DOT — “DMIS Object Technology” (an implementation of DMIS Part 2)
DTD — “Document Type Definition” (in XML)
DXF — “Drawing eXchange Format” (a CAD file data format from AutoDesk’s AutoCAD)
EBNF — “Extended Backus Naur Form” (a language for defining languages)
EXPRESS — an ISO STEP international standard information modeling language
FBICS — “Feature-Based Inspection and Control System” (a NIST ISD automated system)
G&L — “Giddings and Lewis” (a company)
IDL — “interface definition language” (a generic term)
IGES — “Initial Graphics Exchange Specification” (a drawing-oriented CAD file format)
IML — “Information Modeling Language”
IP — “internet protocol”, usually seen as part of TCP/IP
ISD — “Intelligent Systems Division” (part of NIST’s Manufacturing Engineering Laboratory)
ISO — “International Organization for Standardization”
ISO 10303 — the ISO number for STEP

I++ — a European dimensional metrology standards effort (maybe)
Java — a widely used net programming language
lex — a language and processor for building lexical scanners
MAA — “Metrology Automation Association”
NGIS — “Next Generation Inspection System”
NIST — “National Institute of Standards and Technology” (a federal agency)
NML — Neutral Message Language (a NIST ISD interprocess communications utility)
Parasolid — a commercial solid modeler
PC — “Personal Computer”
PSL — “Process Specification Language” (a process planning language)
RCS — “Real-time Control System” (a control system architecture developed in ISD)
SIM — “Sensor Interface Module”
Socket — a widely used net communications software item with an API
STEP — “STandard for the Exchange of Product model data” (a set of standards, ISO 10303)
TCP — “Transmission Control Protocol”, usually seen as part of TCP/IP
XML — “eXtensible Markup Language” (an international standard language for net use)
VRML — “Virtual Reality Modeling Language”
YACC — “Yet Another Compiler Compiler” (a language and processor for building compilers)

## Appendix C Combined vs. Separate Model and Representation

This appendix uses the example of a circle to show how the handling of modeling and representation differs between a language that does both, and a language that does only modeling and, hence, requires separate specification of a file format.

A language that does both information content and representation (DMIS, for example) might describe how to represent a circle in a DMIS program file by the following (simplified a little from real DMIS):

```
F(label) = FEAT/CIRCLE, var_1, x, y, z, i, j, k, diam
```

Where the label is an identifier, var_1 could be either INNER or OUTER and x, y, z, i, j, k, diam are all real numbers. Then an instance of a circle in a DMIS program might be:

```
F(circle1) = FEAT/CIRCLE, INNER, 5.0, 5.0, 0.0, 0.0, 0.0, 1.0, 4.0.
```

A language that does information content but not representation (EXPRESS, for example) might describe exactly the same information needed to define a circle by the following. Note that the label is omitted:

```
TYPE inner_outer_type = ENUMERATION OF (inner, outer);
END_TYPE;

ENTITY circle;
in_or_out : inner_outer type;
x : REAL;
y : REAL;
z : REAL;
i : REAL;
j : REAL;
k : REAL;
diam : REAL;
END_ENTITY;
```

That says what information is needed to define a circle, but does not specify how to write the definition of a given circle in a file. There is a separate specification for a file format. Alternative file formats might even be provided. Under the usual file format used with EXPRESS models, the DMIS circle instance above would be represented as:

```
#43 = CIRCLE(.INNER., 5.0, 5.0, 0.0, 0.0, 0.0, 1.0, 4.0);
```

The #43 (which might be #2 or #901 or anything else in that format as long as it is unique in the file) serves as the label for this instance of a circle.

Under a different file format specification using the same EXPRESS definition, the same circle might be represented as:

```
circle1 = CIRCLE(in_or_out .INNER., diam 4.0, i 0.0, j 0.0, k 1.0, x 5.0, y 5.0, z 0.0);
```

In this file format, the identifier (circle1) is once again a name, and, since the attribute names are included, the order in which the values of attributes are specified is arbitrary.

## References


[CAM-I1] Consortium for Advanced Manufacturing - International; *Dimensional Measuring Interface Standard*; Revision 3.0, ANSI/CAM-I 101-1995; CAM-I, Arlington, Texas; 1995

[CAM-I2] Consortium for Advanced Manufacturing - International; *Dimensional Measuring Interface Standard*; Revision 4.0, ANSI/CAM-I 104.0-2001; CAM-I, Arlington, Texas; 2001

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

[CORBA] http://www.omg.org/technology/documents

[ISO1] ISO 10303-11:1994; *Industrial automation systems and integration - Product data representation and exchange - Part 11: The EXPRESS Language Reference Manual*; ISO; Geneva, Switzerland; 1994

[ISO2] ISO 10303-21:1994; *Industrial automation systems and integration - Product data representation and exchange - Part 21: Clear Text Encoding of the Exchange Structure*; ISO; Geneva, Switzerland; 1994

[ISO3] ISO 10303-224:1998; *Industrial automation systems and integration - Product data representation and exchange - Part 224: Application Protocol: Mechanical Product Definition for Process Planning Using Machining Features*; ISO; Geneva, Switzerland; 1998

[Kramer] Kramer, Thomas R.; Huang, Hui-Minh; Messina, Elena; Proctor, Frederick M.; Scott, Harry; *A Feature-Based Inspection and Machining System*; Computer-Aided Design, Vol. 33, Issue 9; August 2001

[Messina] 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; November 1999

[Shackleford] Shackleford, Will; *Real-Time Control Systems Library Documents*; http://www.isd.mel.nist.gov/projects/rcslib/index.html; 2001

[XML] http://www.xml.com