WL-TR-94-8033

NEXT GENERATION CONTROLLER SPECIFICATION FOR AN OPEN SYSTEMS ARCHITECTURE STANDARD

Martin Marietta Astronautics
PO Box 179
Denver, CO 80201

28 September 1994

Final Report For the Period March 1989 - August 1994

Approved for Public Release; Distribution is Unlimited.

Manufacturing Technology Directorate
Wright Laboratory
Air Force Materiel Command
Wright-Patterson Air Force Base, Ohio 45433-7739

NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely related Government procurement operation, the United States Government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

This report has been reviewed by the Office of Public Affairs (ASC/PA) and is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication.

Daniel R. Lewallen
DANIEL R. LEWALLEN
Project Manager

Michael F. Hitchcock
MICHAEL F. HITCHCOCK
Supervisor

Bruce Rasmussen
BRUCE A. RASMUSSEN, Chief
Integration Technology Division
Manufacturing Technology Directorate

"If your address has changed, if you wish to be removed from our mailing list, or if the addressee is no longer employed by your organization please notify WL/MTIA, Bldg. 653, 2977 P St., Suite 6, W-PAFB, OH 45433-7739 to help us maintain a current mailing list."

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document.

REPORT DOCUMENTATION PAGE

FORM APPROVED
OMB NO. 0704-0188

1. AGENCY USE ONLY *(Leave Blank)*

2. REPORT DATE

28 September 1994

3. REPORT TYPE AND DATES COVERED

Final March 1989 - August 1994

4. TITLE AND SUBTITLE

Next Generation Controller
Specification for an Open Systems Architecture Standard

5. FUNDING NUMBERS

C F33615-89-C-5706
PE 78011
PR 3095
TA 06
WU 18

6. AUTHOR(S)

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Martin Marietta Astronautics
PO Box 179
Denver, CO 80201

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Manufacturing Technology Directorate
Wright Laboratory
Air Force Materiel Command
Wright-Patterson AFB, OH 45433-7739

10. SPONSORING/MONITORING AGENCY REP NUMBER

WL-TR-94-8033

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for Public Release; Distribution is Unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT

This report describes an open architecture framework for advanced manufacturing controllers including machine tools, robotics, process control, and coordinate measurement. The methodology described is applicable to both system design as well as system retrofit applications. A component-based approach is used that allows system architectures to be developed from generic building blocks that address system requirements and resulting functionality requirements. Initial architectures are used to produce an "application architecture" that addresses specific system messaging requirement. An "application system" is then synthesized using off-the-shelf, reusable "implementation components" for the application software. Hardware/software specific platform issues are addressed through the use of system "profiles," a convention adopted form the IEEE Portable Operating system Interface (POSIX) standard that is now gaining wide acceptance. A number of examples are provided demonstrating use of all aspects of the methodology on a typical machine tool controller application. Appendices provide details with respect to specific components, requirements, and domain models.

14. SUBJECT TERMS

Open System Architecture, Computer Numerical Control (CNC), Machine Tool Control, Reusable Software, Hardware/Software Platform

15. NUMBER OF PAGES

315

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASS OF THIS PAGE.**	**19. SECURITY CLASS OF ABSTRACT**	**20. LIMITATION ABSTRACT**
Unclassified	Unclassified	Unclassified	UL

FOREWORD

The purpose of this document is to provide a description of the Next Generation Workstation/Machine Controller (NGC) program philosophy and objectives in sufficient detail so to be able to quickly and efficiently assess the impact on emerging system designs and concepts. The focus for this document is the Specification for an Open System Architecture Standard (SOSAS) that is the primary deliverable of the NGC program. The SOSAS will capture definitions, conventions, and standards as they relate to the final NGC family of controllers. The document presented here is not meant to be construed as the final SOSAS, but rather it represents the key elements of the SOSAS necessary to achieve the desired objectives relating to open systems, interchangeability, interoperability, portability, etc. It is fully anticipated that subsequent discussion of this document will provide some of the most important feedback with respect to the production of the final SOSAS document. In this respect, comments relating to this document are strongly encouraged.

This is the complete SOSAS document. An overview is also available entitled "Next Generation Controller Specification for an Open Systems Architecture Standard - Overview".

TABLE OF CONTENTS

1.0 INTRODUCTION 1

Technology Infusion 1
Maintenance 2
Operator Training 2
Document Summary 3

2.0 NGC SYSTEM PHILOSOPHY AND STRUCTURE 6

2.1 Introduction 6
2.2 Industrial Control Domains 8
2.3 NGC Planning and Execution in the Manufacturing Context 10
2.4 NGC Development Process 13
2.5 Structure of a Working NGC 15
2.6 Timing Considerations 17
2.7 Open System Architecture Specification 18
2.8 Components 19
2.9 NGC Development and Validation Tools 19
2.10 Open System Environment Overview and Profiles 20
2.11 Conformance Overview 21
2.12 Growth and Evolution 21

3.0 DETAILED ARCHITECTURE 23

3.1 Reference Architecture 24
3.1.1 Levels of Abstraction 24
3.1.2 Components and Realization 27
3.2 Integration Architecture 29
3.2.1 Service Groups 31
3.2.2 Domain-Independent Components 32
3.3 NGC OSE Framework 32
3.3.1 NGC OSE Reference Model 33
3.3.2 NGC OSE Taxonomy 34
3.3.2.1 Application Programming Interface (API) 36
3.3.2.1.1 POSIX and ANSI C
An Introduction 39
3.3.2.1.2 Platform Services API 42
3.3.2.1.3 Presentation Management Services (PM API) 44
3.3.2.1.4 Data Management Services (DM API) 48
3.3.2.1.5 Geometric Modeling Services (GM API) 51
3.3.2.1.6 Communications Services (Comm API) 52
3.3.2.1.7 Device Input/Output Services (Device I/O API) 59

3.3.2.2 External Environment Interface (EEI) 61
3.3.3 Profiles 70
3.4 Development Process and Implementation 80
3.4.1 Integration Process 81
3.4.2 Application Framework, Architecture, and Implementation 82
3.4.3 Standardization and Document Vision 85
3.4.4 Implementation Component Library & Tools 88
3.4.5 Implementation 89
3.5 NGC Conformance 92

4.0 NGC Development Example 96

4.1 Application Architecture for Machining 98
4.2 Machining Scenario 101
4.2.1 Factory Interaction 103
4.2.2 Resource Verification 105
4.2.3 Job Release 106
4.2.4 Part Program Setup 106
4.2.5 Code Augmentation 108
4.2.7 Tool Motion 109
4.2.8 IEC 1131 Environment 112
4.2.9 Implementation Component Messages 113
4.3 Implementation 114
4.3.1 Integration Process Review 114
4.3.2 Implementation Example - Machining Center 117
4.3.3 Implementation Example—Low End Controller 124

5.0 SUMMARY AND RECOMMENDATIONS 132

5.1 Summary 132
5.2 Synergistic Activities 134
5.3 Recommendation 135

6.0 GLOSSARY OF TERMS 136

APPENDIX A — REFERENCE REQUIREMENTS A-1
A.1 General Requirements A-1
A.2 Axis Motion Control Requirements A-4
A.3 Auxiliary Machine Control Functions A-5
A.4 Programming Requirements A-7

APPENDIX B — PRIMITIVE COMPONENTS B-1

APPENDIX C — ARCHITECTURE DESCRIPTION RULES C-1

C.1 ADL Rules Accessing Multiple Primitive Components C-1
C.2 ADL Rules Linking Primitive Components C-2
C.3 ADL Rules Accessing Single Primitive Components C-2

APPENDIX D — DOMAIN MODELS D-1

APPENDIX E — DOMAIN DICTIONARY E-1
E.1 Bibliography E-2
E.2 Glossary of Terms E-3
E.3 Notation E-58
E.4 Category Index E-61

APPENDIX F — ARCHITECTURE DESCRIPTION LANGUAGE (ADL) F-1
The Adaptive Semantic Language F-1

LIST OF FIGURES

Figure 2.1a. Manufacturing Enterprise Hierarchy 7
Figure 2.1b Integration Architecture 8
Figure 2.2. Domains of Practice for Industrial Control 9
Figure 2.3-1. Traditional Flow of NC Machine Tool Programming 10
Figure 2.3-2. NGC Concept of Product Design and Production Planning 12
Figure 2.3-3. The Concept of NGC Planning and Execution 12
Figure 2.4. NGC Development Process 14
Figure 2.5. Components and the Common Execution Environment (CEE) 15
Figure 2.6. NGC Timing Domains 17
Figure 2.7 NGC Document Vision 22
Figure 3.1.1 NGC Abstraction Levels 25
Figure 3.1.2 NGC Realization Process 29
Figure 3.2-1 NGC Integration Architecture 30
Figure 3.2-2 NGC Integration Architecture & Related Standards 30
Figure 3.2.2 Domain-Independent Implementation Components 32
Figure 3.3.1 NGC OSE Reference Model 34
Figure 3.3.2 NGC OSE Taxonomy 35
Figure 3.3.2.1 NGC Architecture Perspective 38
Figure 3.3.2.1.3 Presentation Management Services—API Levels and Implementation 47
Figure 3.3.2.1.6-1 Communications Services—Trollius Example APIs Network Dependent 54
Figure 3.3.2.1.6-2 Communications Standards Architectural Hierarchy 56
Figure 3.3.2.1.6-3 CORBA-DCE IDL & RPCs 58
Figure 3.3.2.2-1. NGC EEI System Context—Physical View 62
Figure 3.3.2.2-2. NGC EEI Interoperability Standards—Physical Mapping 63
Figure 3.3.3-1 NGC Profiles—Template Structure 72
Figure 3.3.3-2 API Profile Suite 75
Figure 3.3.3-3 POSIX Profile Level 1 76
Figure 3.3.3-4 POSIX.1 Profile Level 2 77
Figure 3.3.3-5 POSIX.4 Profile Level 2 77
Figure 3.3.3-6 POSIX.4a Profile Level 2 78
Figure 3.3.3-7 POSIX.4b Profile Level 2 78
Figure 3.3.3-8 C Language Profile 79
Figure 3.3.3-9 EEI Profile Suite—Platform/Application 79
Figure 3.4.1 NGC Integration Process 81
Figure 3.4.2 From Reference Requirements to Implementation 84
Figure 3.4.3-1 NGC Document Vision 85
Figure 3.4.3-2 ANSI Standardization Process Model—ACS Method 87
Figure 3.4.4 Implementation Component Library & Tools—Structure & Contents 88
Figure 3.4.5-1 From Application Architecture to Implementation 90
Figure 3.4.5-2 Application Architecture 91

Figure 3.4.5-3 Generic Implementation 92
Figure 4.0. Horizontal Machining Center 96
Figure 4.1. Example of an Application Architecture for a Machining Center 99
Figure 4.2a Application Architecture Components and Messages—Part 1 102
Figure 4.2c Application Architecture Components and Messages—Part 3 103
Figure 4.3.1. NGC Integration Process 116
Figure 4.3.2-1 Platform Implementation Examples 118
Figure 4.3.2-2 Profile 119
Figure 4.3.2-3 Profile 119
Figure 4.3.2-4 Profile 120
Figure 4.3.2-5 Profile 121
Figure 4.3.2-6 Profiles 121
Figure 4.3.2-7 Platform Partitioning, Example 1 123
Figure 4.3.2-8 Platform Partitioning, Example 2 123
Figure 4.3.3-1a From Application Architecture to Implementation (Simplified) 126
Figure 4.3.3-1b From Application Architecture to Implementation (Detailed) 127
Figure 4.3.3-2 Application Architecture 127
Figure 4.3.3-3 Generic Implementation 128
Figure F-1. A Tower of the n-Towers Model F-4
Figure F-2. A Factory Application of the n-Towers Model F-5
Figure F-3. Situational Dependency Nodes with Theta Relations F-14
Figure F-4. Epsilon Relations of Situational Dependency Nodes F-15
Figure F-5. Other Relations of Situational Dependency Nodes F-16

ACRONYMS

ACL	ASCII Cutter Location
A/D	Analog-to-Digital
ADA	Ada programming language, in honor of Augusta Ada Byron, Countess of Lovelace (1815-1852), daughter of Lord Byron, was assistant and patron of Charles Babbage.
ADL	Architecture Definition Language
AIS	Application Interface Specification
ANSI	American National Standards Institute
API	Application Programming Interface
APT	Automatically Programmed Tools
ASCII	American National Standard Code for Information Interchange
AT	IBM's 80286-based PC bus archictecture
BCL	Binary Cutter Location
BSD	Berkeley Software Distributions
CAD	Computer-Aided Design
CAM	Computer-Aided Manufacturing
CAM-I	Consortium for Advanced Manufacturing International
CAN	Controller Area Network
CCITT	International Consultative Committee on Telegraphy and Telephony
CEE	Common Execution Environment
CLI	Command Line Interpreter
CORBA	Common Object Request Broker Architecture
COSE	Common Open Systems Environment
COTS	Commercial Off-the-Shelf
CNC	Computer Numerical(ly) Controlled
CPU	Central Processing Unit
CSMA/CD	Carrier Sense Multiple Access/ Collision Detection
D/A	Digital-to-Analog
DCE	Distributed Computing Environment
DEC	Digital Equipment Corporation

DFS	Distributed File Service
DM	Data Management
DNI	Detailed Network Interface
DOS	Disk Operating System
EDI	Electronic Data Interchange
EDIF	Electronic Design Interchange Format
EEI	External Environment Interface
EISA	Extended Industry Standard Architecture
FIP	Federal Information Processing
FIPSPUB	Federal Information Processing Standards Publication
FORTRAN	FORmula TRANslation (programming language)
FTAM	File Transfer, Access, and Manipulation
FTP	File Transfer Protocol
GDI	Graphical Device Interface
GKS	Graphical Kernel System
GM	Geometric Modeling
GPIB	General-Purpose Interface Board (IEEE 488)
GUI	Graphical User Interface
HUI	Human-User Interface
H/W	Hardware
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers, Inc.
I/F	Interface
IGES	Initial Graphics Exchange Specification
I/O	Input/Output
ISA	Industry Standard Architecture OR Instrument Society of America
ISO	International Organization for Standardization

ISR	Interrupt Service Routine
ITB	Inside-the-box
LAN	Local Area Network
LIS	Language-Independent Specification
MCU	Machine Control Unit
MIMD	Multiple Instruction, Multiple Data
MIT	Massachusetts Institute of Technology
MS	Microsoft
NASREM	NASA/NIST Standard Reference Model for Telerobot Control System Architecture
NC	Numerical Control, Numerically Controlled
NDL	Neutral Database Language
NFS	Network File System
NGC	Next Generation Workstation/ Machine Controller
OBIOS	Open Basic Input/Output System
OMG	Object Management Group
OS	Operating System
OSE	Open Systems Environment
OSF	Open Software Foundation
OSI	International Organization for Standardization (ISO) Open Systems Interconnection
PC	Personal Computer
PDES	Product Data Exchange using STEP
PEX	PHIGS Extension (to MIT's X Windows
PHIGS	Programmer's Hierarchical Interactive Graphics System
PII	Protocol Independent Interface
PLC	Programmable Logic Controller
POSIX	Portable Operating System Interface
PWI	Public Windows Interface
RDA	Remote Data Access
RISC	Reduced Instruction Set Computer

RPC	Remote Procedure Call
RT	Real-Time
RTOS	Real-Time Operating System
SBX	Specification for an I/O Expansion Bus (ANSI/IEEE 959)
SCO	Santa Cruz Operation, Inc.
SCSI	Small Computer Systems Interface (ISO 9316)
SERCOS	Serial Data Link for Real-time Communication between Contols and Drives
SNI	Simple Network Interface
SOSAS	Specification for an Open System Architecture Standard
SQL	Structured Query Language
STD	8-bit Microcomputer Bus System (ANSI/IEEE 961)
STE	Specification for the Exchange of Product Model Data
STEP	Standard for the Exchange of Product Model Data
S/W	Software
TCOS	Technical Committee on Operating Systems (and Application Environments)
TCP/IP	Transmission Control Protocol/ Internet Protocol
TFA	Transparent File Access
UNIX	Operating system based on Multics
US PRO	U.S. Product Data Association
UTA	University of Texas at Arlington
VME	Versatile Backplane Bus (IEC 821)
VMS	Serial Subsystem Bus of VME Bus (IEC 823)
VSB	Parallel Subsystem Bus of VME Bus (IEC 822)
XPG	X/Open Portability Guide

1.0 INTRODUCTION

The Next Generation Workstation/Machine Controller (NGC) program philosophy of systems that are interoperable, interchangeable, and portable is an outgrowth of a more general trend in all areas of systems development that stress "openness." Like many new ideas in engineering, the concept of an open system is one that has been much easier to describe qualitatively than it has been to establish rigorous design methodologies. In the arena of advanced manufacturing control systems, NGC and the Specification for an Open System Architecture Standard (SOSAS) are an attempt to merge the evolving methodology of open systems with a well-established technology base in machine tool, robotics, measurement, and process control. As in any emerging technology, growth and evolution do not come without some degree of controversy and argument. There is no lack of recognition on the part of the NGC development team that while the long-term benefits of an open approach to manufacturing controller design are obvious to virtually everyone, the burden of making this vision a reality will fall on a vendor community that is engaged in an intensive economic struggle and is faced, on a daily basis, with the necessity of looking at an unforgiving bottom line. The result is that the NGC development process has never been viewed as a "do it all over again" exercise, i.e., starts with a clean sheet of paper. Instead, it has been viewed as "given the space of open system solutions, pick the solution that is a closest fit to existing systems."

The NGC can be viewed in many different perspectives. While the control builder will ultimately be the one tasked with turning ideas and paper into working hardware and software, it is the end user of manufacturing control systems who will, in the long term, dictate the final form and structure of these evolving open systems. It is the end user who will apply the resulting systems in an attempt to produce products that are cheaper than the competitor and as a result, the end users that are successful in this process will, explicitly or implicitly, determine the marketplace forces that ultimately shape the technology. There are many complexities involved in defining what leads to a cheaper product from the standpoint of the end user. While the actual dollar cost of a manufacturing controller will always remain an important aspect of the overall metric, there are many other factors that influence the end user's manufacturing strategy.

Technology Infusion

The ability to rapidly integrate new technologies and practices has always been, and will always remain, a crucial factor in competitiveness. A new technology can appear in many different ways and impact practice, hardware, software, or all of these. A good example of this is the emerging trend towards machine tool controllers that makes use of temperature and/or vibration data at the spindle. This requires not only the ability to integrate new sensors into the system but also the ability to rapidly and efficiently modify existing software structures in order to get the right data to the right place in an appropriately timely fashion and implement newer and potentially more complex control laws that take advantage of the new data.

Maintenance

Key factors in the competitiveness of any system are the supporting elements necessary to keep the system running. In general, commonalty of system elements leads to lower system cost for a variety of reasons. These include a decrease in inventory size for spare components, the standardization of procedures for diagnosing and repairing systems, and the ability to use trained diagnostic and repair personnel across a wider variety of systems because of the decrease in specialization.

Operator Training

Commonalty of systems decreases the amount of additional training that is required to transition operators from one system to another. Major retraining can be supplanted by incremental training based upon knowledge of a core system structure. In this case, an open system architecture that utilizes standardized system features can achieve nearly the same desired commonalty as a mandate for all systems from one manufacturer. From the standpoint of the operator, common look and feel of interfaces is more important than the internal system that generates the look and feel.

While the issue of the specific attributes desired in a NGC controller were discussed at much greater length in both the Needs Analysis document and the Requirements Definition Document, it is obvious that the resulting NGC systems must help the end user in a number of ways; neither very high-technology nor extremely low-cost systems are in themselves the full answer.

Instead of attempting to mandate a specific point solution to the advanced manufacturing controller problem, a philosophy is embodied herein that leads to maximum flexibility with the respect to ability to modify both the hardware and the software in the system to achieve desired enhancements in capability as necessary.

Document Summary

Section 2.0 provides an overview of philosophy and structure of NGC. It describes the basic concepts that allow the freedom of action on the part of the designer to choose the proper mix of hardware and software to achieve an optimal solution to a given design problem.

To facilitate a standardized approach to developing the structural elements of a system, a component based approach has been adopted. The fundamental attributes of this approach are described including the use of a *reference architecture* that is comprised of *primitive components* that are used to delineate system requirements and structure in terms of generic "building blocks". From the reference architecture an *application architecture* is constructed that captures the functionality of the end system at an abstract level. While a level of abstraction above the final hardware and software system, the application architecture allows responsibilities and dependencies to be clearly established. Finally, through the selection of *implementation components* a final software structure is determined. The step from application architecture to implementation is a very difficult one because it involves the specifics of the system platform. The platform includes all system hardware as well other system software such as operating systems, communication software, etc.

Effectively dealing with platform issues requires dealing with the specifics of different types of potential hardware solutions as well as a large number of standards and conventions that have arisen with respect to communication, operating systems, device interfaces, graphics representations, etc. Because NGC has the goal of establishing a foundation for the introduction of open systems technology into the advanced manufacturing arena, it was felt that it would be a mistake for a small group to choose a relatively small set of possible standards and conventions that would then exclusively serve the NGC community. As a result, it was necessary to find a means of accommodating systems that could draw on a wide range of hardware and software solutions in arriving at a system implementation. This problem was dealt with through the use of a representation concept known as a *profile.* This concept is crucial to understanding how the NGC specification can achieve the open system objectives without overly constraining system designers. Profiles can be thought of as a means of classifying NGC-compliant systems. The concept of the profile was adopted from POSIX (Portable Operating System Interface) literature. (POSIX and it's role in NGC are discussed in greater depth in section 3.0.) The vendor will use profiles to succinctly and unambiguously describe the system structure. With a profile specified for a specific system, the end user will be able to determine whether or not the candidate system has the flexibility necessary for a given set of applications. A very simplified type of profile is now commonly used to describe software packages as being "Mac" or "PC" compliant. Similarly, the designation of "DOS" or "Windows" can be thought of as a subprofile under PC. This flexibility assessment could include the ability to operate with existing software packages, the ability to communicate with other factory systems with minor modifications, or the ability to expand hardware features such as input/output (I/O) or motor drivers. NGC does not preclude a vendor from building a system that is essentially non-open; it does, however, guarantee that a *system purchaser can be assured of understanding precisely what degree of openness is being provided.* Necessarily, standards play an important role in achieving the NGC goals. In addition to describing the basic architectural philosophy, Section 2.0 also provides a summary of the relevant standards that have been chosen for application to the NGC system and the rationale for their selection.

Section 3.0 includes both a more complete discussion of the component based approach to the design of the application architecture. Platform issues are discussed in depth from the standpoint of two basic elements; the *Application Program Interface* (API), and the *External Environment Interface* (EEI). Both are discussed in section 3.3. The full development process leading to implementation is discussed in section 3.4. Issues related to conformance are discussed in section 3.5.

A full design example illustrating all of the basic NGC concepts is presented in section 4.0 for a 5-axis machining center. The example includes the selection of requirements and primitive components leading to the application architecture. Implementation is considered from the standpoint of different realistic hardware and COTS software options.

Section 5.0 consists of a brief summary and further issues for the evolution of NGC. The primary issues cited with respect to the further evolution of NGC are the availability of libraries of NGC components (both primitive and implementation) that can be made widely available to the community at large for incorporation into emerging systems and the development and availability of tools that will facilitate the NGC development process. While an initial set of primitive components are provided for machining applications as an appendix to this report (see below), sufficient raw material for a robust initial implementation component library through the work of the National Institute of Standards and Technology (NIST) as an output of the Enhanced Machine Controller (EMC) program. Therefore, the key issue is the establishment of a repository for the initial library.

The issue of tools is more complex but not intractable. It is generally believed that the tool technology necessary for NGC currently exists through a variety of existing developmental and commercial packages. Ideally, a complete tool set could be integrated and made available to the general community. This would be the fastest way to spur the growth of NGC type systems. Unfortunately, the resources for such an activity have not yet been identified.

A set of appendices is provided that includes: Appendix A- Reference Requirements, Appendix B- Primitive Components, Appendix C- Architecture Description Rules, Appendix D- Domain Models, Appendix E- Domain Dictionary and Appendix F- Architecture Description Language.

2.0 NGC SYSTEM PHILOSOPHY AND STRUCTURE

2.1 Introduction

The open system concepts specified in this document establish a framework for the design and construction of a family of workstation/machine controllers for industrial machines. This framework addresses issues associated with both application software and the hardware software platforms on which this application software will run. This Next Generation Controller (NGC) specification is based where possible on de jure and de facto open standards in manufacturing, controls, and computing technology but provides a sufficiently robust structure such that evolving and new standards and technology can readily be incorporated. The NGC specification facilitates commonality of components across a complete range of machine controllers. These controllers are to be used in a wide variety of manufacturing operations that include machining, robotics, and inspection.

A NGC supports a wide range of processing and discrete part manufacturing applications, including machine tools of all types, robots, electronic assembly, material handling devices, inspection devices, and virtually all types of automated equipment in both manned and unmanned environments and networked and stand-alone configurations. Specifically, a NGC controls a manufacturing workstation, which is defined as a single material transformer and related material handling and inspection equipment. As shown in Figure 2.1, the manufacturing workstation is placed at the lowest level in the hierarchy of the overall manufacturing enterprise. Several manufacturing workstations, each controlled by an NGC, are managed by a cell. Multiple cells are managed by a center, and multiple centers are managed by a factory. At the top of the hierarchy, the enterprise manages multiple factories. While this specification focuses on the lowest level of the this hierarchy, the results and structure are immediately and easily extended to include the higher hierarchical levels on an evolutionary basis. Therefore, a foundation is established for much more complex Computer Integrated Manufacturing (CIM) systems that will retain the open systems structure described in this document.

The open systems concepts are evolving from requirements established by the NGC community of control builders, machine integrators, end users, members of standards organizations, and university researchers. It is flexible enough to cover the broad range of manufacturing practice, and it is extendible to absorb future advances in technology while accommodating existing manufacturing and controller practice.

To produce an enduring standard, manufactured products and manufacturing processes must be described independently of specific equipment, methods, or topologies. An architecture, calling for a specific topological arrangement and interconnection of components, does not provide enough flexibility to control every application in discrete part manufacturing, nor does it accommodate technological advances over a period of decades. Moreover, several topologies may be required in order to develop a system and understand its operation under a variety of situations. The overall view of the system architecture that is used in the NGC process is captured in the *Integration Architecture*.

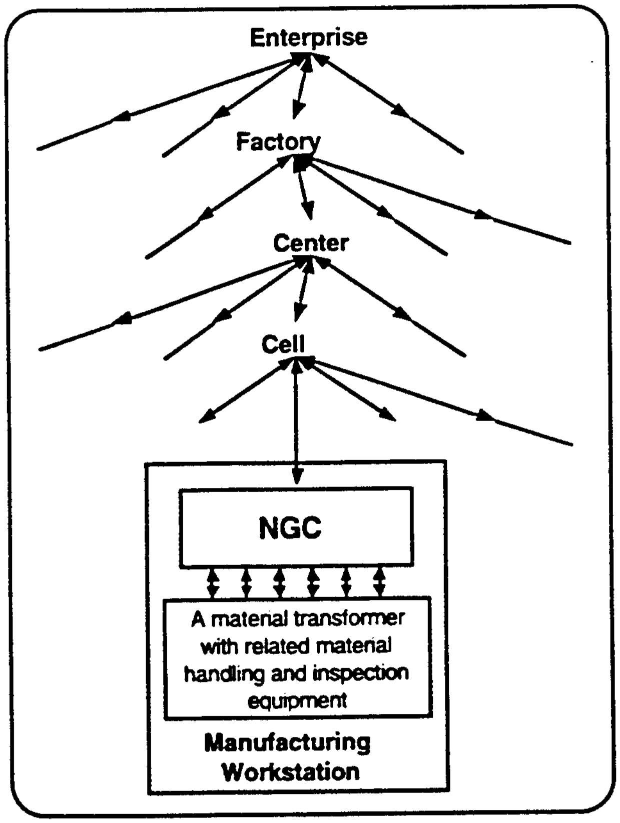

Enterprise
Factory
Center
Cell
NGC
A material transformer with related material handling and inspection equipment
Manufacturing Workstation

Figure 2.1a. Manufacturing Enterprise Hierarchy

The integration architecture, Figure 2.1b, combines manufacturing and control application components with a supporting infrastructure. It also acts as a framework for incorporating open and de facto standards. Applications components intercommunicate by messaging and can be adapted to a variety of component interconnection topologies.

Typical computing platform conventions, definitions, and capabilities are provided by the Open Systems Environment (OSE).

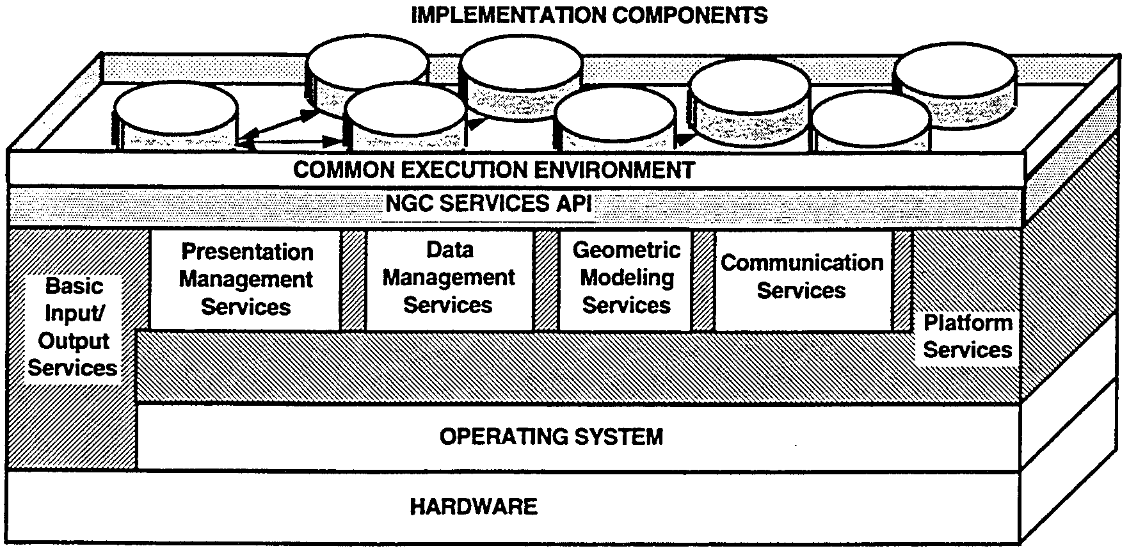

IMPLEMENTATION COMPONENTS
COMMON EXECUTION ENVIRONMENT
NGC SERVICES API
Basic Input/ Output Services
Presentation Management Services
Data Management Services
Geometric Modeling Services
Communication Services
Platform Services
OPERATING SYSTEM
HARDWARE

Figure 2.1b Integration Architecture

2.2 Industrial Control Domains

Manufacturing enterprises are complex, information-intensive environments. The practices associated with discrete part manufacturing can be grouped into three principal domains: (1) manufacturing practice, (2) controller practice, and (3) computing practice. The three domains and some associated concepts are shown in Figure 2.3. Terminology and representations from these domains are used in this specification. Existing and emerging standards, developed within the purview of these domains, are incorporated in this specification by reference. Thus the considerable investment and effort devoted to the three domains, especially computing practice, can be reused effectively to support manufacturing applications.

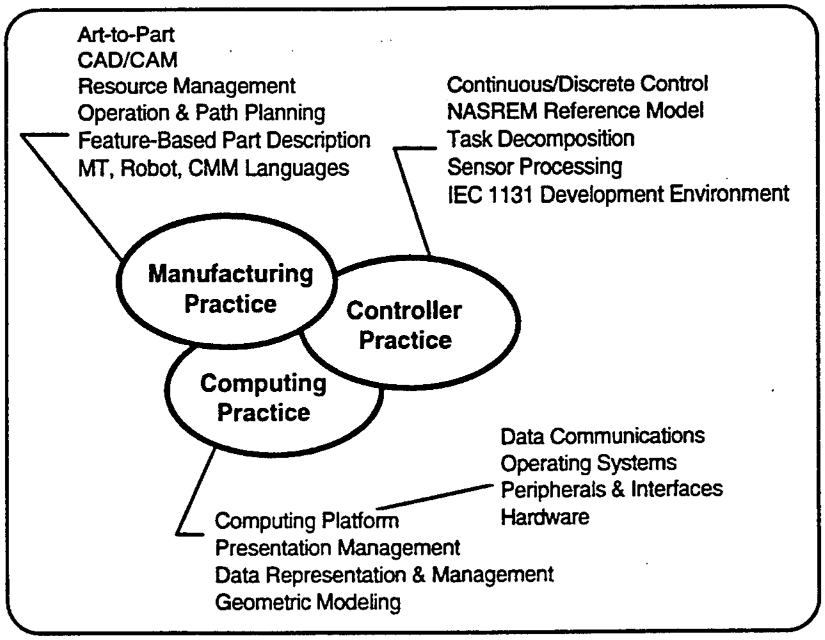


Figure 2.2. Domains of Practice for Industrial Control

Manufacturing practice covers the process of transforming raw materials into finished parts. The underlying concept for the discrete part manufacturing process is art-to-part. The process begins by capturing a part design using a computer-aided design (CAD) system. The resources for making the part are scheduled, and the machining, material handling, and measuring activities are planned using a computer-aided manufacturing (CAM) system. Motion paths are planned and used to drive the appropriate machinery, i.e., tool paths for machining, robot end-effector paths for manipulation, and probe paths for measurement. Some related manufacturing practice standards include those for feature-based part description, part programming, and measurement languages. Many of the steps in this process are done manually or disconnected. In the future, the art-to-part process is envisioned as a seamless information flow from the designer's concept to the finished part.

Controller practice covers the organization and control of manufacturing equipment. Equipment is controlled mostly by closed-loop controllers, which provide continuous (motion) control and discrete control.

Computing practice covers the computing and communication technologies required to support manufacturing and controller practice. The practice includes design and use of computer hardware and software. Computing practice is the target of intense open systems activity, which benefits NGC as it evolves and becomes sufficiently mature.

2.3 NGC Planning and Execution in the Manufacturing Context

The traditional manufacturing practice and general flow of numerically-controlled (NC) machine tool programming starts with a design review by a part programmer as shown in Figure 2.3-1. The part programmer's role has been to analyze the product design and determine how to make the part economically, the machining range of workpiece, the method of mounting the workpiece on the machine tool, the machining sequence of every operation, and the cutting tools and cutting conditions. The output of the part programmer has been a part programming manuscript that documents the logical order of machine operations. The part geometry and cutter location data are post processed to a specific machine-readable format and transferred to the machine control unit (MCU).

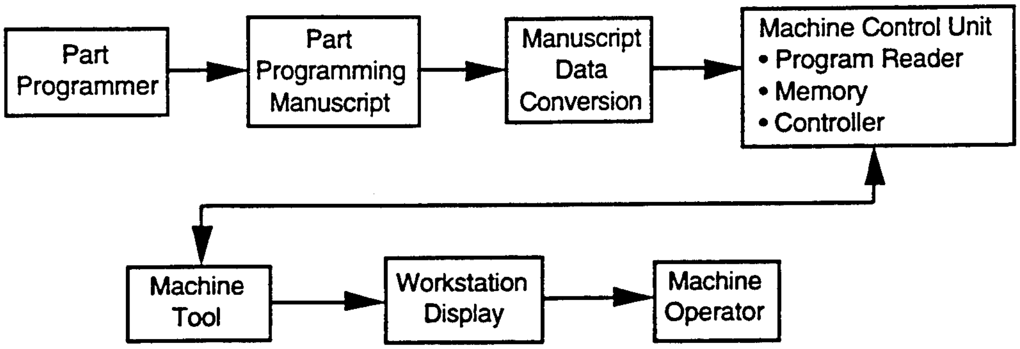


Figure 2.3-1. Traditional Flow of NC Machine Tool Programming

The MCU memory has an executive program, program reader, shop floor programming language, cutter line control, machine interface logic, program subroutines or canned cycles, and tool changer control. The program reader in the MCU converts the coded instructions and the MCU controller generates an output signal to servo mechanisms to drive and direct the machine tool. The machine tool resolver or feedback device determines the precise location of the workpiece relative to the cutter and returns this data to the machine controller and also to machine operator workstation display. The workstation display may display full operational and parametric data; display job setup instructions; and have the capability for program verification, editing and update, maintenance and troubleshooting, fault messages, and graphical representations of the workpiece, tools, and cutter paths. The responsibility of the machine operator is to monitor the machine operation, monitor workpiece loading and unloading, provide information feedback, and perform operator programming through manual data input.

The intent of a NGC is to provide the product designers, process planners, NC programmers, machine operators, and factory managers with more flexibility in planning and committing factory resources. This increased flexibility and efficiency will be feasible through industrial machine controllers which allow open exchange of planning data, feature-based part representation, cutter location data, tooling and fixture data, and operations sequence data among different types machine tools without rewriting the part programs source code, and eliminating programming and processing for specific machines.

The concept of NGC product design and production planning is shown in Figure 2.3-2. The product designer is able to generate a CAD that specifies the geometric features and considers material handling, assembly requirements, and manufacturability. Part geometry and features are interpreted through a CAM system to develop the process plan and generate the machining program. A computer-aided process planning (CAPP) system would evaluate economical production based upon a part family data base that has a library of part features, geometric data representations, and machining processing data. The process planning system determines the machine routing and operations sequence, methods, fixtures and tooling, and setups. From the process plan and machining program, a NC program is sent to a NGC workstation controller.

The concept of NGC planning and execution is shown in Figure 2.3-3. The part program could be assigned to any machine in the factory that has the process capability and control configuration to accept feature-based part model input, such as Product Data Exchange Specification (PDES), or NC legacy code such as binary cutter location/ASCII cutter location (BCL/ACL) part programs, RS-274 part programs, or automatically-programmed tools (APT) part programs. The NGC operations planner creates an operations plan that specifies the logical order of workstation operations among the different machine mechanisms for part handling, machining, inspection, setups, and fixturing. The part model input and operations plan provides input data instructions to a task planner and path planner. The task planner and path planner specifies for each mechanism the logical order of motion tasks, tool changes, feeds and speeds, detailed motion geometry, and obstacle avoidance. The output from NGC operations planner, task planner, and path planner directs each mechanism to obtain the resultant machine tool action.

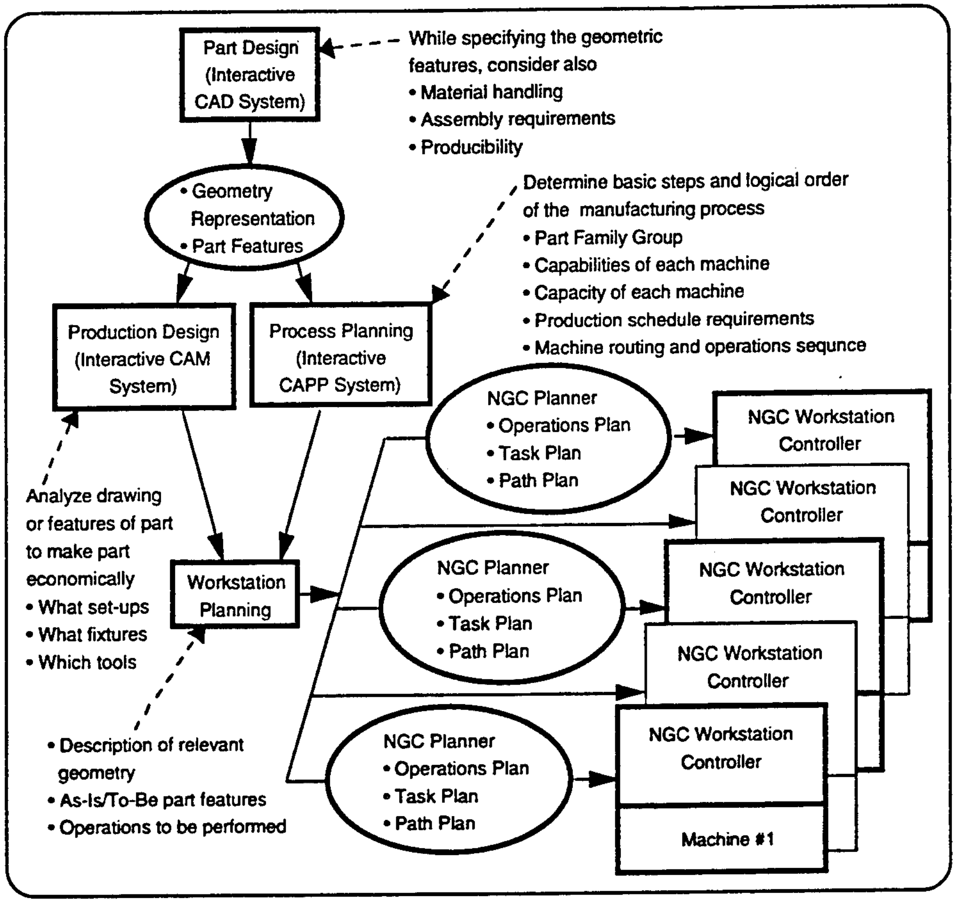

Part Design (Interactive CAD System)
While specifying the geometric features, consider also
• Material handling
• Assembly requirements
• Producibility
• Geometry Representation
• Part Features
Determine basic steps and logical order of the manufacturing process
• Part Family Group
• Capabilities of each machine
• Capacity of each machine
• Production schedule requirements
• Machine routing and operations sequnce
Production Design (Interactive CAM System)
Process Planning (Interactive CAPP System)
NGC Planner
• Operations Plan
• Task Plan
• Path Plan
NGC Workstation Controller
Analyze drawing or features of part to make part economically
• What set-ups
• What fixtures
• Which tools
Workstation Planning
• Description of relevant geometry
• As-Is/To-Be part features
• Operations to be performed
Machine #1

Figure 2.3-2. NGC Concept of Product Design and Production Planning

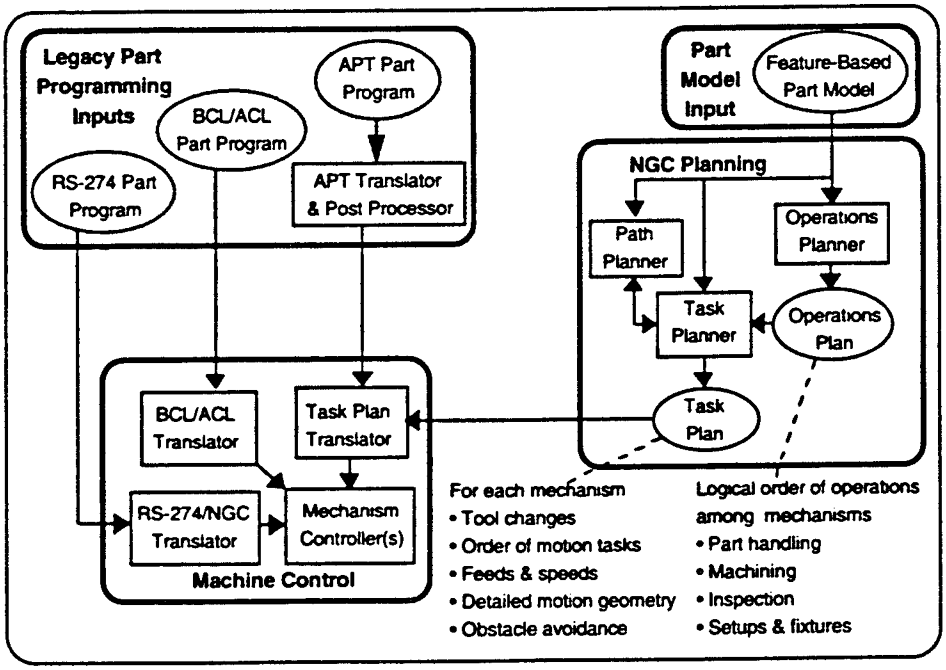


Figure 2.3-3. The Concept of NGC Planning and Execution

2.4 NGC Development Process

To achieve the goals of the NGC program, it is necessary that the SOSAS structure adequately addresses all of the primary issues associated with system structure, application software, and platform development. Without a roadmap that shows how all of these elements are accommodated by the SOSAS, it is difficult to appreciate how the SOSAS supports all of these areas: (1) the bottoms-up development of a fully integrated system, (2) development and integration of new system software, and (3) expansion of platform capabilities to accommodate new requirements. A key element of the NGC philosophy is the ability to achieve rapid and effective system modifications that address only the part that "needs to be fixed" and do not necessarily entail massive, expensive system re-design.

A roadmap for the SOSAS development process is shown in Figure 2.4, which illustrates the basic design and development paths for addressing system structural issues and platform issues. It is important to realize that any block on the diagram can be considered to be an entry point for the design process. Development of new system application software, for example, would entail only the "upper" path with the platform definition being considered fixed. Similarly, if only the platform structure is being considered for modification, then the application architecture elements can be considered as given from the standpoint of platform issues and therefore, definition of the new platform profile is the key issue.

The key to understanding Figure 2.4 is the realization that system structure is embodied in the specification of an **application architecture**. The "upper" design path results in the application architecture. To arrive at an application architecture, **reference requirements** are compared with **user needs** to derive a set of problem specific **application requirements**. By using a well-defined set of reference requirements as the basis for selecting the final system application requirements, a tremendous amount of consistency is brought to the entire requirements process, something not always achieved in ad hoc requirements derivation processes.

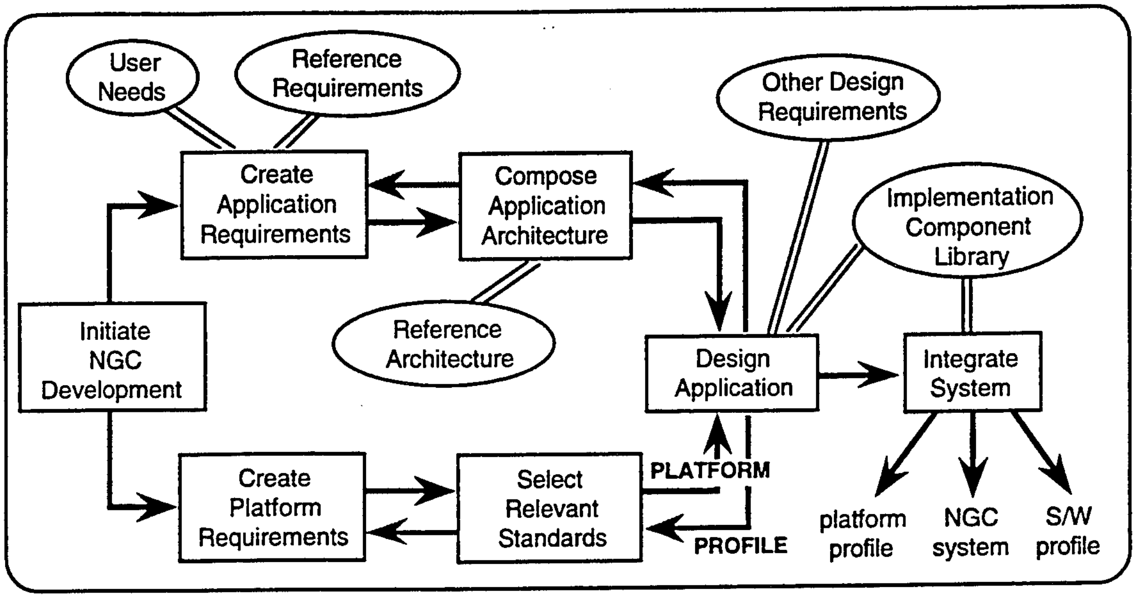

User Needs
Reference Requirements
Other Design Requirements
Create Application Requirements
Compose Application Architecture
Implementation Component Library
Initiate NGC Development
Reference Architecture
Design Application
Integrate System
Create Platform Requirements
Select Relevant Standards
PLATFORM
PROFILE
platform profile
NGC system
S/W profile

Figure 2.4. NGC Development Process

The **Reference Architecture** is the key to turning application requirements into the application architecture. The Reference Architecture consists of both **primitive** and **aggregate components**. Components are abstract building block elements that describe functionality and communication. The application architecture is built from these components. Although the Reference Architecture will necessarily be a "living" library that continually grows, the objective is to develop a set of components that "span the space" of desired functionality and communication for establishing a system structure. Therefore, in the system design process, when the Reference Architecture is found insufficient, new components are added to the reference architecture. The application architecture is a complete and consistent topology for the representation of the system.

The lower pathway of Figure 2.4 addresses the issue of how the system will be physically implemented from the standpoint of processors, buses, communication, I/O, etc. It is not an objective of the SOSAS to enforce a particular design philosophy with respect to the system platform structure. The NGC does not attempt to lead all system designers in the direction of a standard "box". What it does do, however, is to establish a methodology for accurately capturing what has been produced by a specific vendor.

Platform requirements are used to select the standards (or conventions) that the designer feels are necessary for a specific platform implementation based on heritage, cost, performance, etc. Once the basic platform structure has been established, the key elements of the design are documented in the form of a **platform profile.** The notion of the platform profile was derived from the realization that (1) because of diverse market forces there will never be a "one platform does it all" to satisfy everyone, and (2) from the standpoint of achieving system openness with all of the associated "-ilities" it is of crucial importance that the user is able to understand what is being purchased in such a way that system expansion and extension can readily be assessed. The platform profile, by documenting the key standards and conventions used to construct the system, provides this insight.

The application architecture and platform design (as reflected in the profile) are unified in the **design application** phase of the process. Here, the **implementation component library** is used to take advantage of off-the-shelf software elements to instantiate the application architecture and insure consistency with the platform profile. Abstract functionality is replaced by actual software that includes both the functional aspects as well as the practical man-specific aspects necessary to accommodate the chosen platform profile.

2.5 Structure of a Working NGC

The NGC open system is comprised of application software exchanging information via data communication mechanisms and connected to the services provided by operating systems and hardware through a common interface called an application program interface (API). The software is implemented in the form of components that interchange messages in the process of carrying out their responsibilities. Components run under a Common Execution Environment (CEE) that provides for transparent peer-to-peer message exchange between the components as well as other services. Figure 2.5 shows this concept.

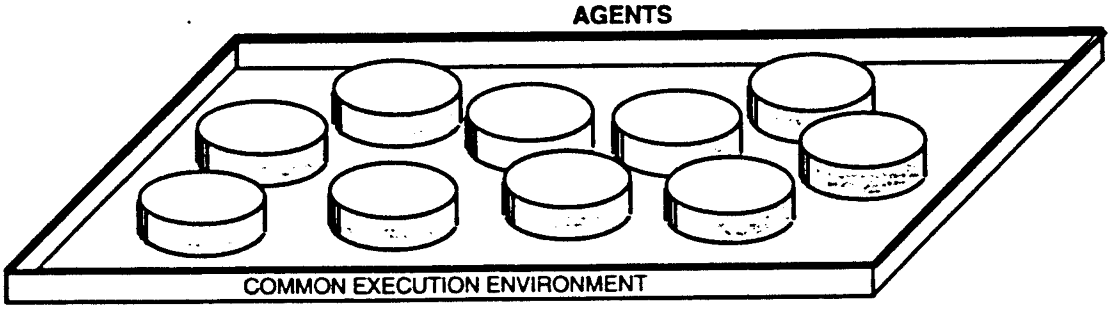


Figure 2.5. Components and the Common Execution Environment (CEE)

Each component is a single separate thread of execution within the CEE since a component must be able to send and receive messages independently from the other components. A component encapsulates the data and functionality it needs to carry out its assigned responsibility. This approach is extremely flexible compared with older hard-wired systems, allowing the NGC system software to be reconfigured dynamically during operation by activating or deactivating agents. Dynamic reconfiguration does not cover the addition or removal of hardware; connecting or disconnecting hardware will require a system shutdown.

Components have the following attributes:

- Responsibility—The role the component plays, distinguishing it from the other components in the CEE, in the successful overall operation of the NGC system;

- Peer-to-peer Relationship(s)—The collaborative relationships the component has with other components as required to carry out its responsibility;

- Behavior(s)—The specific functionality encapsulated by the component, where each behavior is expressed as one or more operations to be performed in response to a message;

- Message(s)—The complete set of specific instructions necessary for evoking all of the behaviors encapsulated by the component These messages must be defined in enough detail to guarantee interoperability.

- Application Program Interface(s)—The interface(s) a component uses specifically to access services provided by the Open Systems Environment (OSE).

For a specific controller application, a configuration is an information structure that provides a description of all required hardware and software elements and their interconnections. The configuration contains the information needed to maintain safe and reliable operation of the controller in carrying out all of its intended responsibilities. The configuration includes, as a minimum, the description of all of the required components. A directory service component or group of components has the responsibility for assuring that all of the components needed for reliable controller operation are available and working properly in the CEE. A configuration is used by the directory service component(s) to maintain a roster of active components and facilitate message interchange among them. The roster of active components will change dynamically, for example, with a shift in the operational requirements of the controller as it carries out its various responsibilities.

2.6 Timing Considerations

The NGC stands at the juncture between process planning for cell activities and controlling the equipment that actually fabricates parts or assembles products. Timing considerations are important, but not crucial, for the former types of activity. But controlling the equipment requires strict adherence to a time budget; failure to do so will often result in higher production costs due to flawed parts and/or damaged equipment.

The NGC spans three separate timing domains: non real time (non RT), real time (RT), and hard real time (hard RT). This partitioning of the timing requirements is shown in Figure 2.6. The non-RT domain of standard computing covers activities like compiling and linking software, reading data from a file, and reporting progress to the cell and other entities in the enterprise. The time required to complete an activity can be flexible, guided by the requirement to finish as early as possible. The standard computer domain covers a wide variety of environments and operating systems, e.g., UNIX, DOS, non-RT POSIX compliant systems, and Windows NT.

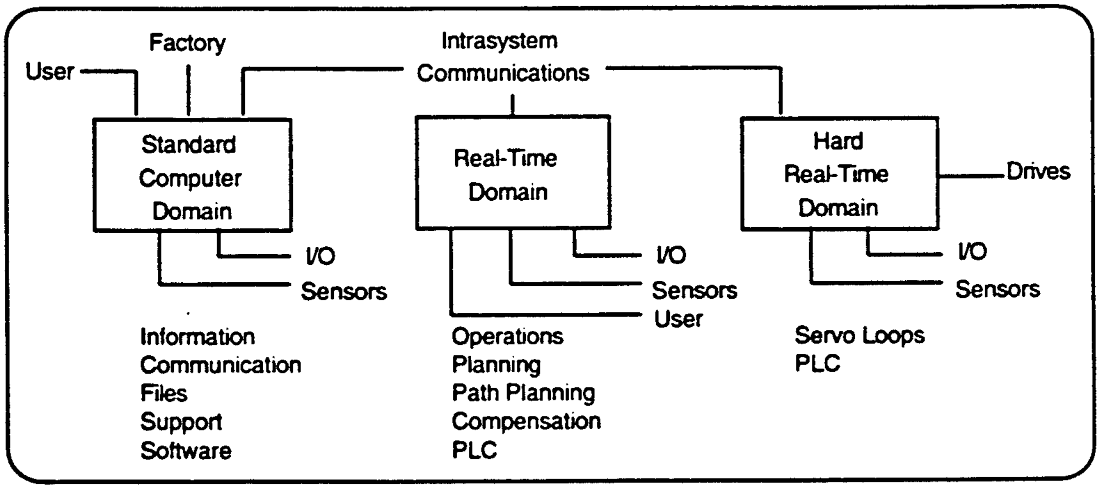


Figure 2.6. NGC Timing Domains

An activity in the RT domain is driven by a time budget, but the consequences of missing a deadline are not catastrophic, and the system can recover without serious loss. Some examples of RT activities are operations planning, task planning, path planning, cutter compensation, and a PLC's actuation of the coolant valve. Hard-RT activities have mandated timing deadlines; missing one of these deadlines can have serious consequences like a damaged part that must be reworked or discarded. Hard-RT constraints apply to servo loop closures and programmable logic controller (PLC) cycles that read values from sensors into memory and write values from memory to actuators.

The need to consider the three timing domains will persist in spite of the ever-improving performance of available computing. As better computers are made, the increased throughput makes more complicated algorithms feasible within a fixed time limit. When they are deployed in advanced manufacturing applications, greater machining precision at higher speeds will be made possible. The result is better quality products at faster production rates. Thus the RT envelope will be pushed continually, and the relevance of the three timing domains will remain as more capable processors emerge.

Several existing controller products can be aligned with the three timing domains. One vendor links a transputer-based, RT platform to a UNIX front end through transputer links. The transputers drive an Industry Standard Architecture (ISA) expansion bus, and all motion servo loops are closed in the transputer using digital-to-analog (D/A) converters and encoder feedback. Another vendor combines standard computing and less demanding RT operations in a single PC AT running the Lynx operating system. The demanding RT operations, that is, motion and high-speed discrete control, are implemented in a separate ISA card. (The use of a separate dedicated function card for demanding RT operations dominates current controller practice.) Yet another vendor hosts the non-RT operations on a DOS platform and distributes control via ARCNET to small proprietary nodes. All of these implementations fit within the timing partitions as long as the messages at the selected boundaries are conformant with the NGC messages.

NGC does not, at this time, directly address the issue of timing performance for resulting system implementations. In all likelihood, this will require dedicate tools that, in addition to insuring other NGC requirements are met, also evaluate the adequacy of the overall software/platform system with respect to performance/stability driven performance requirements. This issue is discussed further in sections 2.9 and 3.4.

2.7 Open System Architecture Specification

Controller designs are constrained from two aspects: the Reference Architecture and open standards (such as POSIX) and industry conventions (such as DOS). For a specific industrial application, controller primitive components are selected from the Reference

Architecture according to application-specific requirements and synthesized into an Application Architecture. The resulting Application Architecture constrains the design of the application system; that is, the system elements specifically required for manufacturing and control. The other constraining aspect supports the operation of the controller, which is based upon open and de facto standards of practice in manufacturing, controllers, computing, and data communication.

2.8 Components

All of the *primitive* components needed to cover the spectrum of machine controller applications over the NGC domain are maintained herein as the Reference Architecture. A component is an abstract encapsulation of functionality and information with an assigned responsibility and an abstract message interface definition. A component is specified as follows:

- Responsibility: The role the component plays, differentiating it from all other components in the Reference Architecture, in the successful overall operation of a controller;

- Message(s): The complete set of distinct instructions necessary for evoking all of the behaviors encapsulated by the component for carrying out its responsibilities. These messages are defined at an abstract level.

Depending on the application, the required primitive components are selected and arranged in an application architecture that will guide the actual design of the controller. An application architecture must contain, as a minimum, the components needed to fulfill all of the responsibilities of the application it is based on. Many such application architectures are possible over the NGC domain. Based on a single application architecture, a component may be implemented and delivered in a variety of ways. This activity will result in libraries of implementation components that will capture the functionality of the aggregated primitive components but contain the additional structure necessary for integration into specific platform implementations.

2.9 NGC Development and Validation Tools

The intense competitive pressures and risks associated with development and subsequent production of industry-fielded, open architecture, machine controllers necessitates the wise and prudent use of development and validation tools, such as knowledge-based tools, libraries, and object-modeling tools. Tools should address system design and integration support; configuration management; event timing; profile mapping and applications supported; and the taxonomy and hierarchy of events, applications, standards, agents, and components. Metrics and measurement methodologies need to be defined, developed, and tested for validity and significance and then be evaluated for the benefits, capabilities, and features of a specific implementation. Performance testing should consider the related open system specifications, data exchange and information handling protocols, external and internal interfaces, and the explicit and implicit features of openness, i.e., portability, interoperability, scaleability, interchangeability, commonality of components, etc. The development and validation tools, and documentation of lessons learned, should be mapped to the capabilities and features of an open architecture machine controller.

Many software tools that would fulfill requirements for the NGC tool suite are either available on the market today or are being developed in other ongoing programs. The ARPA Domain Specific Software Architecture (DSSA) program is in the process of establishing many of the fundamental tools that would be required in the NGC requirements definition, design, and validation and verification activities. As NGC continues to evolve and mature, it will be essential to find mechanisms for capturing the tools legacy that exists and effectively integrating it into the NGC structure.

2.10 Open System Environment Overview and Profiles

The NGC is based on open standards. Open standards help to achieve a level of portability and interoperability between multi-vendor controller products that does not exist in controllers today. This "openness" of the controller facilitates the addition of new controller features and innovative technology with a relative ease that is unavailable in controllers that are closed. The benefits are two-fold: (1) the end-user has a controller that is adaptable to market dynamics and can be easily modified to incorporate the latest cost-saving technologies, and (2) third-party vendors are encouraged by the opportunities to develop new technologies and market niches for a new generation of controllers.

Central to the open theme is the NGC Open Systems Environment (OSE) framework. The OSE framework leverages open standards, both de jure and de facto, to specify an infrastructure for open controllers. The OSE framework embodies three general concepts: the reference model, the taxonomy, and profiles. The OSE reference model is the context for NGC open standards, the taxonomy is the logical grouping of these standards, and the profiles are selections of standards from specific groups. Rather than attempting to mandate a limited set of standards, the concept of the profile allows the vendor and the user to communicate key structural aspects of a system via a "snapshot" that, through the shorthand afforded by defined standards, allows system structure to be quickly determined. This is very similar to the common practice today of indicating whether products are "Mac" or "PC" compatible. The OSE framework is essentially the organized menu for this selection. While the possible number of permutations and combinations of profiles is overwhelming at this point, it is expected that as more vendors produce NGC systems, a smaller set of accepted profiles will emerge. These concepts are again introduced and described in more detail in Section 3.3.

2.11 Conformance Overview

NGC conformance is determined by adherence to a specific profile of standards. The NGC SOSAS does not attempt to specify a standard set of profiles. The market will drive out the set of profiles that are most practical to both the controller developer and end-user.

A NGC-conformant product must include a “NGC Disclosure Statement” that specifies the profile, component interfaces, and other conformance claimers where applicable. Profiling offers the controller developer options for product conformance at a variety of levels to satisfy user requirements at a competitive cost, with varying degrees of openness. Section 3.4 describes the disclosure statement requirements, levels of conformance, language documentation, statements of intent, and associated claimers in more detail.

2.12 Growth and Evolution

This document should be viewed as the initial document in what will be a growing set of companion documents that govern the full NGC development process. In section 3.4 a document hierarchy is discussed that parallels the structure used by IEEE to document the POSIX standard. The structure of this family of documents is shown in Figure 2.7. Like the POSIX standard, there is an overall document that describes basic system structure and philosophy. Other documents (the .1 and .2 documents) address basic issues associated with system framework and implementation. Finally, the architecture document set deals with domain specific considerations. As shown, it is presumed that this would initially include CNC, robotics, process control and PLC applications.

The issue of the criticality of tools to a mature NGC process was discussed in an earlier section. In addition to tools, success of NGC also hinges on the emergence of libraries (or repositories) of effective and reliable implementation components that allow the system developer to quickly satisfy both functionality and platform requirements in the implementation process. As in the case of tools, other ongoing programs have already developed much of the foundation for these libraries. This issue is discussed at greater length in section 5.0.

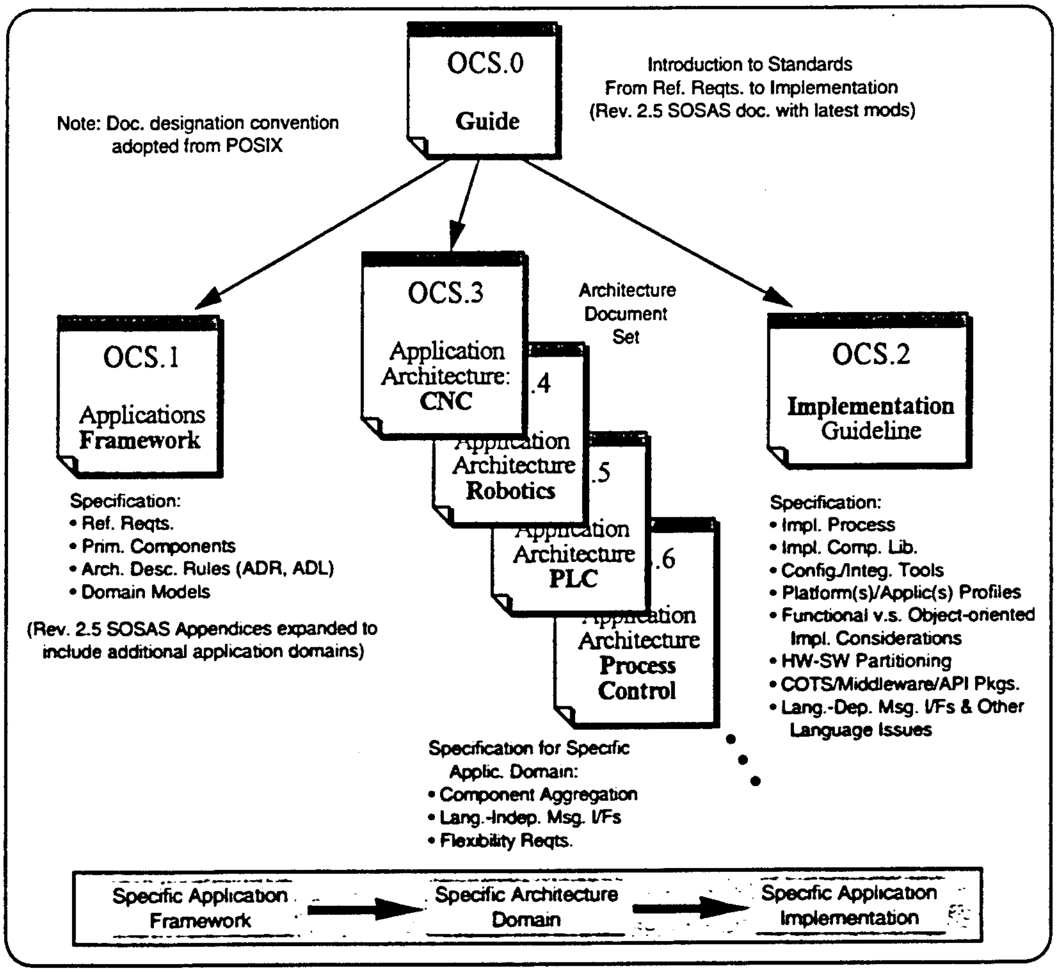


Figure 2.7 NGC Document Vision: Open Controller Standards

3.0 DETAILED ARCHITECTURE

This section specifies the details on the design, integration, and delivery of NGC conformant industrial controllers. Key elements of the NGC process, such as the Reference Architecture, Application Architecture, Open Systems Environment, etc. that were introduced in the previous section are described in greater detail with respect to the methodology involved in the NGC design and development process.

Section 3.1 addresses the concept of the Reference Architecture and the evolution of system structure. Emphasis is placed on the basic process that is used to move from Reference Architecture to Application Architecture to a final system implementation as an Application System. It is shown that this process is equivalent to proceeding from a level of high abstraction that deals with general responsibilities to a final system that is implemented via specific choices of hardware and software, defined by appropriate APIs and standards. The NGC levels of abstraction facilitate flexible and extensible software designs and emphasize software reuse.

The NGC integration architecture is presented in Section 3.2 as a framework for incorporating services identified and defined in open standards of computing and data communication. These standards are relevant to NGC because they are likely to be supported as commercially available products from vendors of hardware and operating systems. The integration architecture is crucial to the overall NGC process because it provides a means visualizing the interplay between application software, standard services and operating system functions, and the underlying hardware platform.

Section 3.3 presents the NGC Open Systems Environment (OSE) framework and its role with respect to NGC applications and the Common Execution Environment (CEE). The OSE is standards based, having a taxonomy with application program interfaces (APIs) and external environment interfaces (EEIs) as main branches. Also, the profiling concept is introduced, where, by specifying a profile, the NGC user can select from among a competing set of de jure and de facto standards to suit a particular implementation environment.

Section 3.4 addresses the issue of carrying controller development through to implementation. An application framework guides the design and implementation of the controller, based on available products. Key features of this process are reuse of hardware and software components, adaptation of such components to related applications, and separate development of interoperating components by independent vendors. A strategy for evolving a family of open controller standards is also presented. This strategy parallels that employed by the IEEE for the

POSIX standard. This approach allows for independent documentation of different aspects of the NGC process and the emergence of domain specific companion volumes for specialized domains such as machine tools, robotics, process control, etc.

Section 3.5 addresses the issue of conformance for NGC systems. Because of the mechanism of profiling, and the latitude it provides to the system designer, conformance becomes an issue of adequate documentation of all aspects of the final system. In addition to the profile suite for both API and EEI, the supplier will also be required to document component interface descriptions and language specifics.

3.1 Reference Architecture

This section describes the NGC Reference Architecture, the starting point for the design of a controller's application software, and its role in specifying the realization of an open controller. Stated in simplest terms, the Reference Architecture consists of a set of *primitive components*. The primitive components can be thought of as abstract building blocks for NGC applications. At the level of primitive components, the issues are those of function, responsibility, and generalized data flow rather than specifics of implementation via either hardware or software. Although it is somewhat of a simplification, primitive components can be thought of as a formalization of the elements normally used for generating system "block diagrams". The primitive components, as captured in the Reference Architecture form a language, of sorts, that can be used, later in the Application Architecture, for capturing overall system structure and requirements. The architectural levels of abstraction mark the stages in the process of implementing a controller from requirements analysis through implementation. Use of the Reference Architecture and a derivative Application Architecture constrains controller designs so that products developed independently by different suppliers can be expected to interoperate within a single controller implementation. Moreover, by publishing the architectural details of the controller design and its interfaces, a supplier is opening the system so that it can be readily extended in performance and/or capability. To realize a controller for a specific manufacturing application, an application architecture is synthesized from primitive components in the Reference Architecture, and the Application Architecture, in turn, is used to constrain the design of the application system.

3.1.1 Levels of Abstraction

For NGC, three levels of abstraction provide the flexibility to configure machines for a wide variety of discrete part manufacturing applications while enabling extensions of the standard into other industrial applications and/or controller implementation technologies. Figure 3.1.1 shows these levels of abstraction. In decreasing degree of abstraction, they are: domain level (description of the controller in manufacturing and controller terminology), technology level (broad structural partitioning that constrains implementation), and implementation level (actual elements of a working instance of a controller).

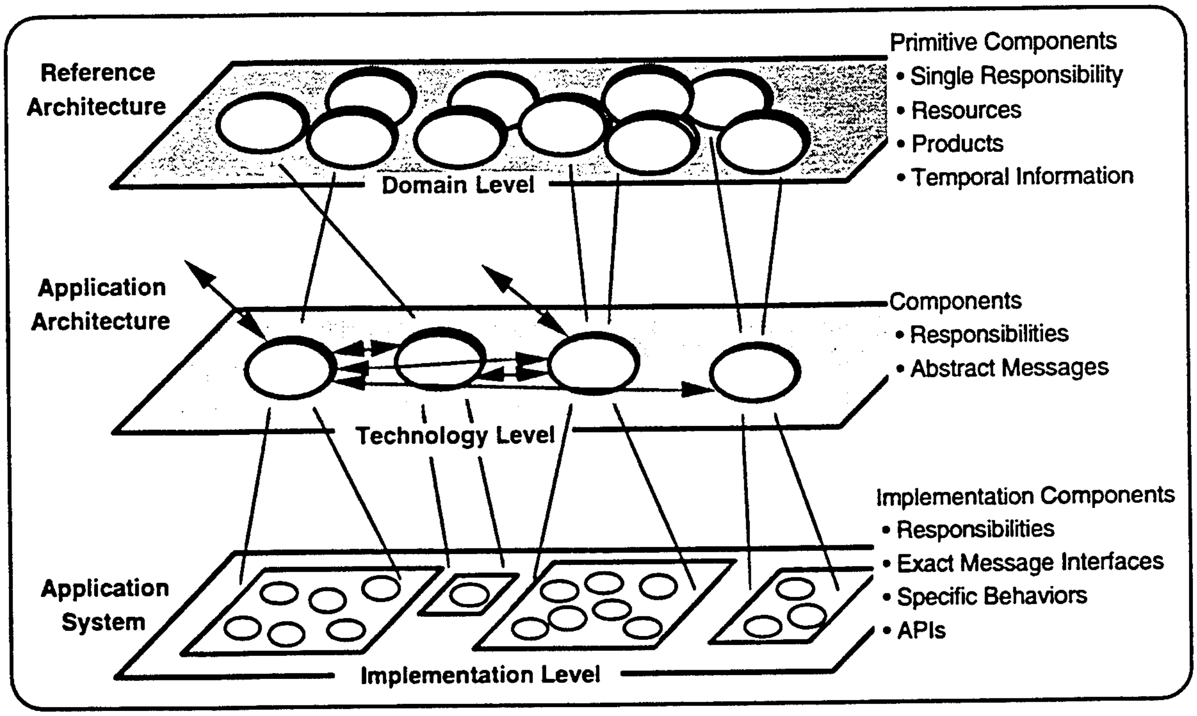


Figure 3.1.1 NGC Abstraction Levels

The domain level captures the NGC in the terminology of manufacturing and controller practice. Multiple application scenarios from differing expert viewpoints have been analyzed to produce a common set of reference requirements (see Appendix A), a set of domain models (see Appendix D), and a Reference Architecture. These domain-level elements are independent of a specific technology or implementation. Part fixturing, for example, could be performed manually by a human operator or automatically by a robot. As another example, a controller can be implemented using an analog (continuous variable) design, or as in the NGC case, the implementation technology is digitally-based using computing platforms and programmable logic controllers (PLCs). NGC covers a wide variety of applications, and the controller elements actually used will vary from application to application. For example, the controller for a three-axis milling machine will no doubt have a spindle, but a spindle is meaningless as an element of a centerless grinder.

The Reference Architecture has two parts: the complete set of primitive components across all NGC applications (see Appendix B) and some architecture description rules (see Appendix C). Controller responsibilities are distributed among primitive components in the Reference

Architecture. Each primitive component represents a single responsibility. Architecture description rules are used with application requirements and primitive components to compose an application architecture.

The technology level holds the components of an Application Architecture. The Application Architecture is synthesized out of primitive components following the analysis of the domain practices of a specific controller application. Interdependencies are made explicit through the definition of the component boundaries for the Application Architecture, and message interfaces, while still abstract, are specialized to those required by the specific application.

At the implementation level, the operational paradigm is *implementation components* exchanging messages within the Common Execution Environment (CEE) in order to fulfill their assigned responsibilities. An implementation component is an encapsulation of functions and data that interoperates concurrently with other implementation components within a computing environment by exchanging messages in a variety of contexts. A component from an application architecture represents one or more implementation components, but the actual number and composition of such implementation components are left as choices for an implementor. Consequently, an unlimited variety of designs can derive from a single application architecture component definition. This allows separate vendors to differentiate their products while retaining a correspondence to an application architecture at the technology level.

An Application Architecture component corresponds to an implementation component or a grouping of implementation components. Implementation components have non-overlapping responsibilities because they conform to the boundaries established by the Application Architecture. That is, no two Application Architecture components share an implementation component or group of implementation components, and no implementation component can be associated with more than one Application Architecture component. An implementation component is defined by its responsibilities, specific behaviors (functionality), exact message interfaces (messages that evoke behaviors), and APIs.

The NGC abstraction levels play an important role. During analysis, application requirements are selected from the reference requirements, and a range or a set of values are applied to them. A transition from the domain level to the technology level occurs when an application architecture is synthesized by applying the application requirements, domain models, and architecture description rules to selected primitive components. The Application Architecture and the integration architecture then guide the NGC design at the implementation level. A design is constrained by application architecture components' responsibilities and message interfaces.

The integration architecture facilitates portable software designs by defining standardized interfaces to services that are characteristic of a computing and data communications environment.

3.1.2 Components and Realization

The Reference Architecture contains all of the NGC primitive components (see Appendix B). A primitive component has only one assigned responsibility and the interface associated with that assigned responsibility; a primitive component's responsibility cannot be partitioned. A primitive component is an abstraction and, therefore, is not associated in the early system specification stage with either software or hardware. It is an abstract encapsulation of functionality and information with an assigned responsibility and an interface description. A primitive component in the Reference Architecture is specified as follows:

- Single Responsibility: The single indivisible role the primitive component plays, differentiating it from other primitive components, in the successful operation of an NGC system.

- Resources: The information the primitive component needs in order to carry out its responsibility.

- Products: The information the component, can provide.

- Temporal Information: The intended timing of the primitive component.

For a specific application, primitive components are selected from the NGC Reference Architecture and synthesized into the components of an Application Architecture. Such components may be composed into larger components, and in that sense, components of an Application Architecture are structurally recursive. Any combination of two or more components is called an aggregation. An aggregation carries the combined responsibilities of its constituents, and it must be able to respond to all of the constituents' abstract messages. A component of an Application Architecture is still an abstraction specifying functionality and responsibility, and it is specified as follows:

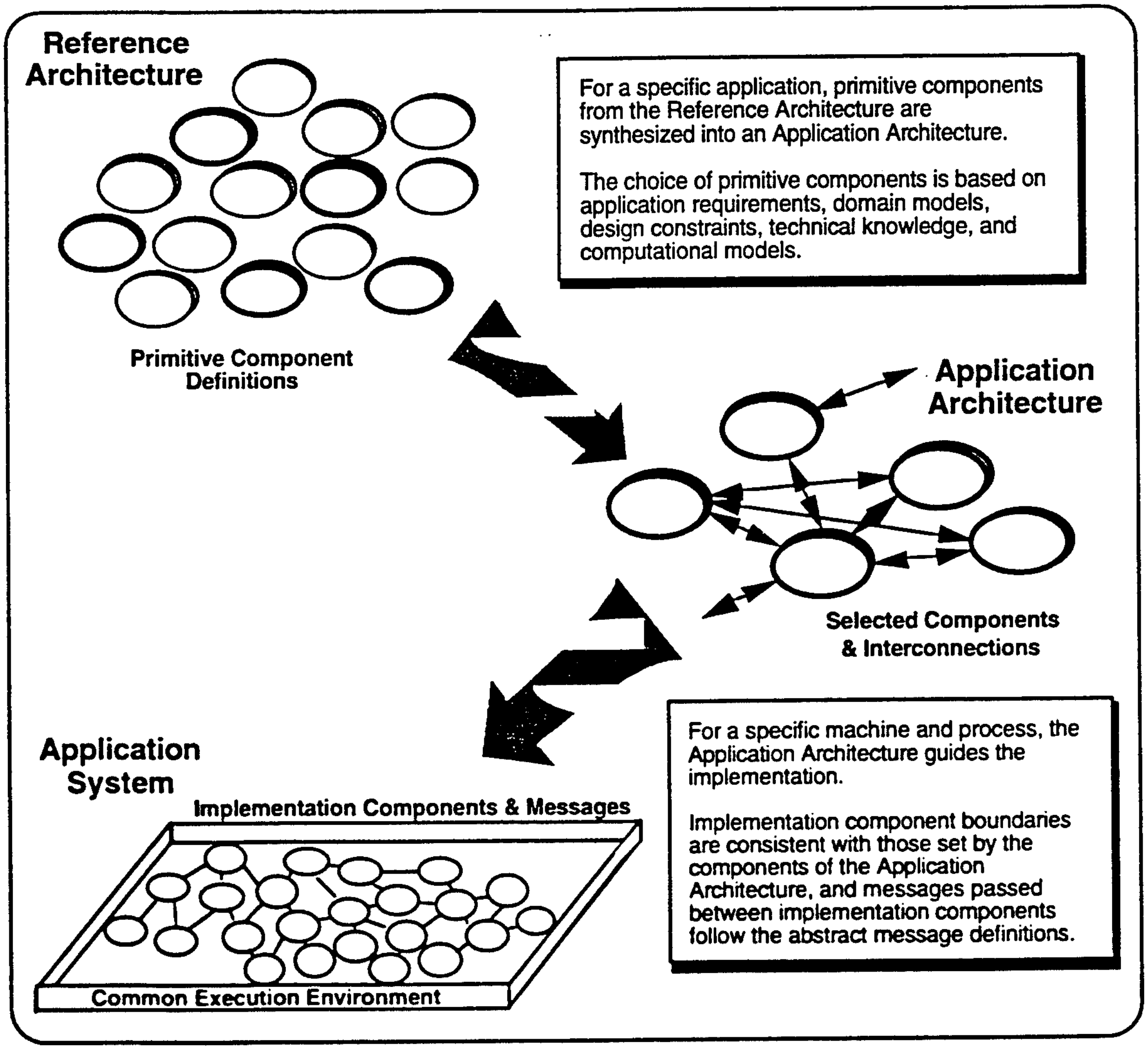
- Responsibility: The role the component plays, differentiating it from other components, in the successful operation of the overall controller application.

- Abstract Message(s): The complete set of instructions necessary for evoking all of the behaviors encapsulated by the component. These messages are defined at an abstract level.

The Application Architecture must satisfy all of the application requirements, and its components must interconnect properly, that is, the components' collective responsibilities must be sufficient to cover the application requirements, and abstract message definitions must align at component boundaries. Component interconnectivity is a necessary, but not sufficient condition to guarantee interoperability. Although Application Architecture component interconnectivity is verifiable at the abstract message interface, it will not assure that implementation components constrained by an application architecture are able to exchange messages cooperatively and unambiguously. Thus a component's abstract message definition must be specified exactly when the Application Architecture is realized as an implementation.

For a specific machine and process, the Application Architecture guides the implementation. One or more implementation components carry out an Application Architecture component's responsibilities. Implementation component boundaries conform to those set by the Application Architecture, and messages passed between implementation components correspond to the application architecture components' abstract message definitions. The process of realizing an application system from the Reference Architecture and a selected application architecture is shown conceptually in Figure 3.1.2.

There are as many different Application Architectures as needed to cover the wide range of NGC applications. Since this specification will evolve along with manufacturing and controller technology and be extended to new applications, the Reference Architecture is expected to grow and evolve continually. The variety of possible Application Architectures will also grow and evolve as the use of open industrial controllers becomes widespread.


Reference Architecture
Primitive Component Definitions
Application Architecture
Selected Components & Interconnections
Application System
Implementation Components & Messages
Common Execution Environment

For a specific application, primitive components from the Reference Architecture are synthesized into an Application Architecture.

The choice of primitive components is based on application requirements, domain models, design constraints, technical knowledge, and computational models.

For a specific machine and process, the Application Architecture guides the implementation.

Implementation component boundaries are consistent with those set by the components of the Application Architecture, and messages passed between implementation components follow the abstract message definitions.

Figure 3.1.2 NGC Realization Process

3.2 Integration Architecture

The integration architecture shown in Figure 3.2-1 is a framework that supports interoperable, portable, scaleable, and interchangeable NGC component implementations, designed and developed by competing vendors. The integration architecture is a detailed refinement (representation) of the application system (see Figures 3.1.1 and 3.1.2). An implementation is divided into domain-dependent and domain-independent parts. The domain-dependent part is made up of the implementation components needed by a specific application and the CEE. The domain-independent part consists of the computing and data communication services that represent the underlying platform. The domain-independent services provide support to the implementation components and to the CEE.

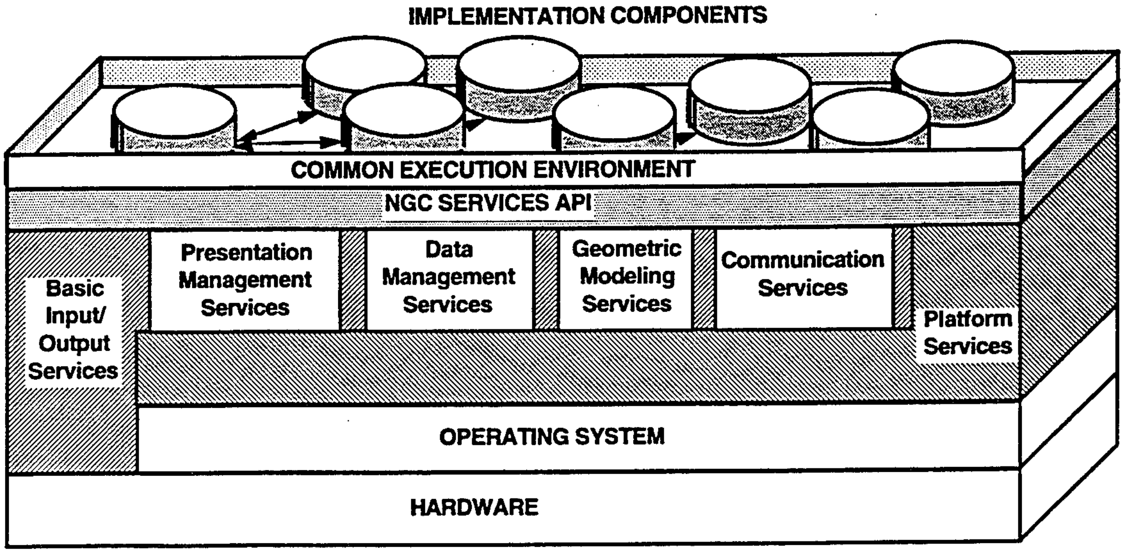

IMPLEMENTATION COMPONENTS
COMMON EXECUTION ENVIRONMENT
NGC SERVICES API
Basic Input/ Output Services
Presentation Management Services
Data Management Services
Geometric Modeling Services
Communication Services
Platform Services
OPERATING SYSTEM
HARDWARE

Figure 3.2-1 NGC Integration Architecture

Services are based on computing practice and have a broader applicability than NGC, so that they are termed "domain independent." The integration architecture conveniently associates relevant open standards of practice with the services as shown in Figure 3.2-2. Services are provided through a specified API, and they can be invoked by implementation components and called by other services. NGC services form six groups that act as an abstraction layer between the implementation components and the native operating system, computing hardware, and peripheral devices.

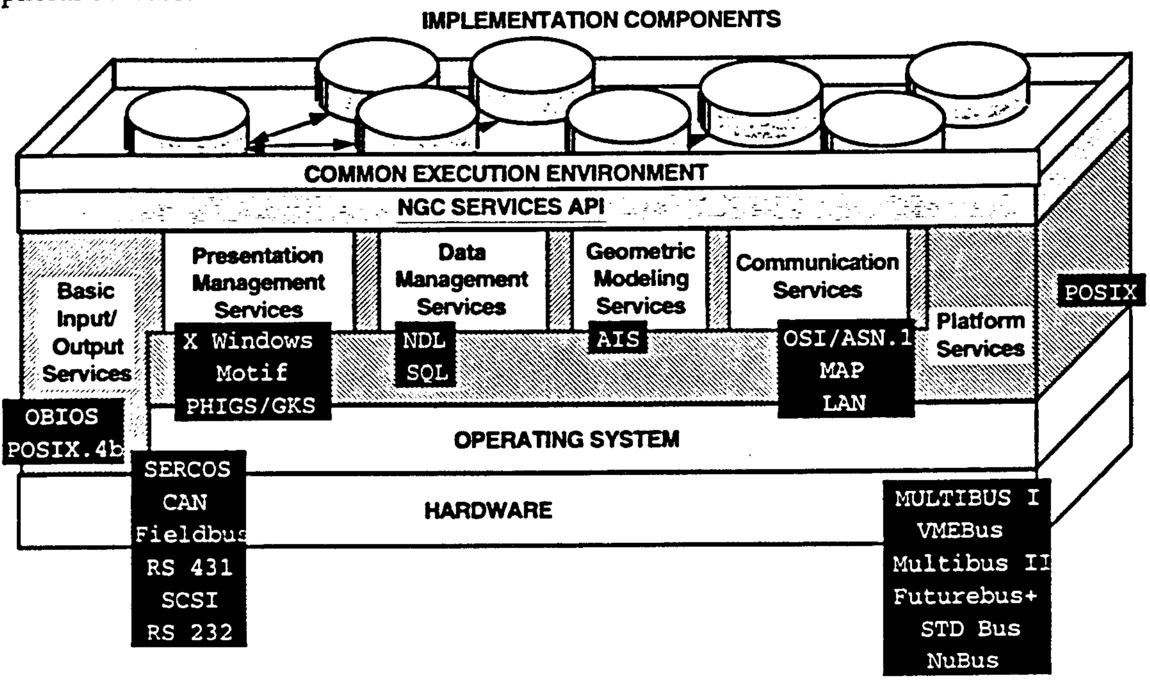


Figure 3.2-2 NGC Integration Architecture & Related Standards

3.2.1 Service Groups

The NGC integration architecture provides six groupings of domain-independent services, which include *Presentation Management, Data Management, Geometric Modeling, Communications, Platform, and Basic Input/Output (I/O) services.* The services are captured as a set of behaviors and an API to those behaviors. The responsibilities assigned to the service groups are described below.

Presentation Management services provide standardized means for multiple applications to present data to a user in a common display representation and to receive input from shared input devices. Its responsibilities include displaying data in a common display (such as a display window); receiving, coordinating, and managing data from the user (possibly from multiple, simultaneous input devices); and creating and managing consistent graphic attributes and user operations.

Data Management services are responsible for the information resources of a manufacturing workstation, allowing applications to share data using an implementation-transparent mechanism. Information resources appear to applications as a centralized repository of data called the information base. The information base may be implemented in a variety of centralized or distributed configurations using data bases, knowledge bases, shared memory, or other forms of information storage.

Geometric Modeling services have responsibility for interactive and non-interactive solid modeling and geometric operations. A solid modeling system unambiguously represents three-dimensional bodies using a combination of geometric and topological information. This capability is essential for NGC planning and execution utilizing feature-based part model inputs and geometric representation of part design.

Communication services are responsible for application-to-application communication via a protocol and mechanism independent interface. This facilitates interoperability among independently developed applications.

Platform services are a generic set of operating system and utility services that act as an abstraction layer over different operating system implementations, just like an operating system can be an abstraction layer over different hardware platforms. These services provide access to computing platform behaviors responsible for managing shared computing resources, and they encompass both RT and non-RT requirements.

The **Basic I/O** services implement the responsibility for interacting directly with a wide range of I/O devices. These services are used to initialize devices and transfer data to and from them.

3.2.2 Domain-Independent Components

For the most part there is an intentional correspondence between primitive components in the Reference Architecture at the domain level and application architecture components at the technology level. Occasionally however, a required component may not be identifiable at the domain level because its role in a controller is considered to be *domain independent.* The *Display Manager, Exception Handler, Directory Services, Safety,* and *Help* services are examples of domain-independent components that have required implementation counterparts. They are shown as implementation components in Figure 3.2.2.

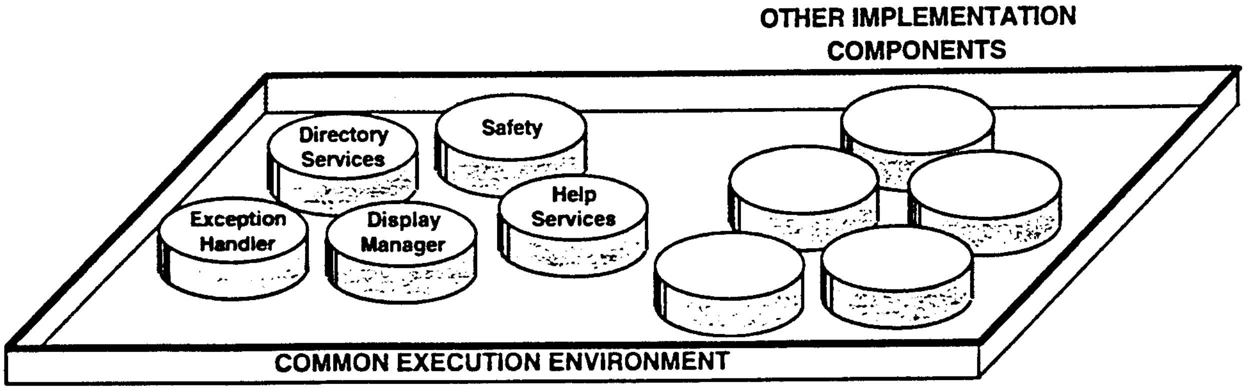


Figure 3.2.2 Domain-Independent Implementation Components

The Display Manager component illustrates the need for such domain-independent components. The Display Manager collects and organizes information intended for the video display of a digital controller implementation, but it is an artifact of computing practice, not manufacturing or controller practice. A Display Manager is obviously needed for the display area of a video monitor that must be shared among the independently-developed, components competing for display real estate during the operation of the controller.

3.3 NGC OSE Framework

The NGC is based on open system standards and conventions. While it would be preferable to deal only with official, documented standards endorsed by standards organizations, it is recognized that "conventions", such as DOS, are important technology elements in the advanced manufacturing community. The complete set of open standards and conventions is referred to as the Open Systems Environment (OSE) framework.

This section and subsequent subsections introduces the concepts of the OSE reference model, taxonomy, and profiles. The OSE framework embodies all of these concepts, providing a general foundation upon which a multitude of NGC OSEs may be derived. The OSE reference model is the context for NGC open standards, the taxonomy is the logical grouping of these standards, and the profiles are selections of standards from specific groups. Loosely stated, the OSE framework may be viewed as the organized menu for this selection. The mechanism of *profiles*, adopted from POSIX, provides a means for determining very quickly what the overall structure of a system employs with respect to standards and platform elements.

The OSE framework includes relevant de facto and de jure standards of practice in manufacturing, controls, computing, and data communications. Much of the OSE framework emphasis is on the Common Execution Environment (CEE), the environment for NGC application software execution. This emphasis is primarily due to the general-purpose nature of computing and communications and the availability of a large base of existing standards to draw from. Therefore, the CEE can be viewed as the physical instantiation of an OSE profile.

3.3.1 NGC OSE Reference Model

The NGC Open Systems Environment (OSE) reference model is the basic context for categorizing the standards within the OSE framework. It is rooted in the reference model used in POSIX but is intended to accommodate a much larger scope. POSIX, the Institute of Electrical and Electronics Engineers (IEEE) portable operating system interface, originated, in part, from the UNIX operating system. The X in POSIX denotes this UNIX origin. This standard is discussed in greater detail in Section 3.3.2.

Figure 3.3.1 illustrates the OSE reference model. It includes two important interfaces: an application program interface (API) between the application software and platform, and an external environment interface (EEI) that supports the interface of the controller platform to such external devices as displays, file servers, networks, etc. The platform is viewed as the general-purpose computing engine, typically commercial off-the-shelf (COTS) hardware and software, that includes processors, memory, clocks, input/output (I/O) boards, and buses, as well as operating system software and other related, general-purpose software packages.

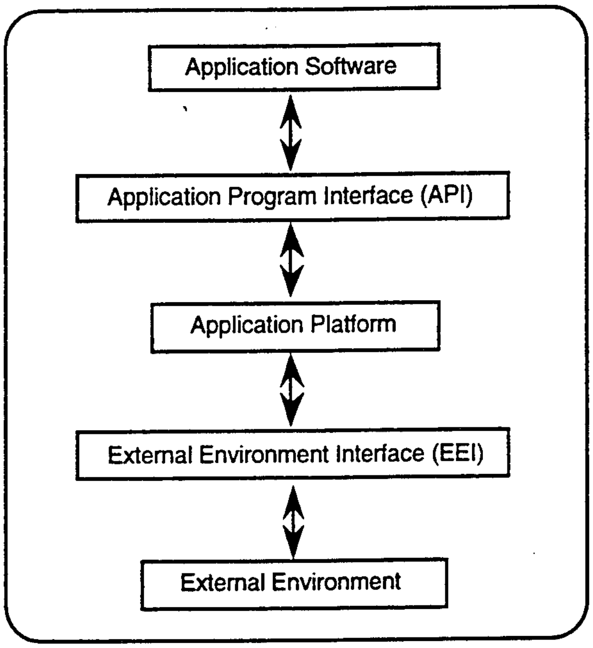

Application Software
Application Program Interface (API)
Application Platform
External Environment Interface (EEI)
External Environment

Figure 3.3.1 NGC OSE Reference Model

The focus of this specification is at the interface level. This allows for a variety of specialized controller platform implementations while still maintaining conformance with interfaces that provide the maximum benefit of open systems.

Open systems are characterized by the fundamental attributes of interoperability, portability, and user portability. A system and its components are interoperable if they are able to work properly together in accomplishing their responsibilities. Software is portable if it can be moved easily between computing platforms of different types with no more effort than recompiling. A system is user portable if its user interface is understandable, consistent in style, and tailorable.

The OSE framework supports system-level interoperability with the specification of EEIs. EEIs are essentially a collection of open standards in the areas of networking protocols, data interchange formats, device I/O, and distributed file systems. The OSE framework supports source-level portability through specification of the APIs. Call-level interface standards (C language bindings) are defined in the areas of general Platform, Device I/O, Communications, Data Management, Geometric Modeling, and Presentation Management services.

3.3.2 NGC OSE Taxonomy

The NGC Open Systems Environment (OSE) standards taxonomy categorizes the open standards referenced within the OSE framework. The NGC approach leverages existing standards in all applicable areas without "re-inventing the wheel". These standards, many of which are applications independent and of a general-purpose computing nature, play a significant role in opening up the controller.

The standards taxonomy is used in deriving NGC profiles and provides one level of guidance for NGC design. In previous sections, we specify the components that guide the design of the applications software, the system elements specifically required for manufacturing and control. The OSE taxonomy, related profiles, and component specifications together support the development of controller applications software that is both interoperable and portable.

A high-level inspection of the OSE taxonomy (Figure 3.3.2) reveals a close tie-in with the OSE reference model. The two major branches of the taxonomy map one-for-one with the two major interfaces identified in the reference model, specifically: application program interfaces (APIs) and external environment interfaces (EEIs). In review, the API branch deals with issues of source code portability and the EEI branch deals with system-level interoperability.

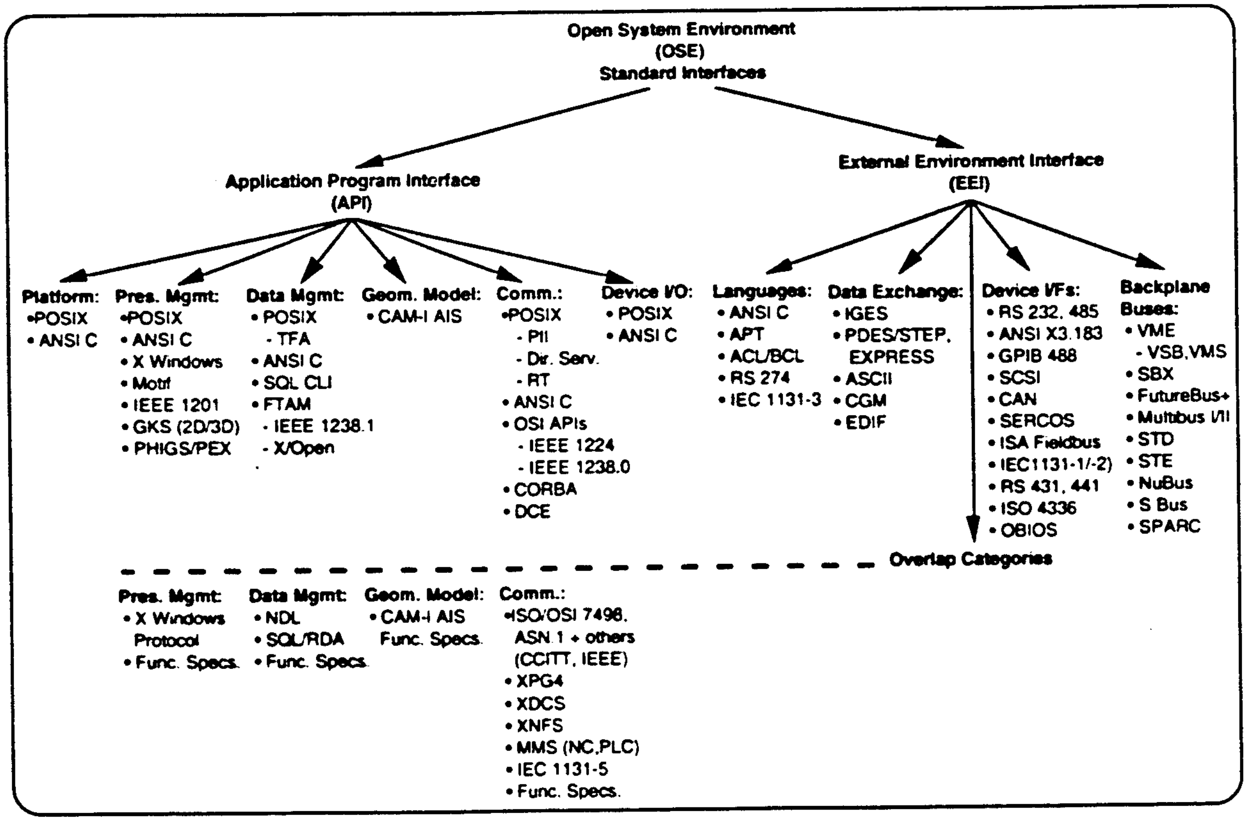


Figure 3.3.2 NGC OSE Taxonomy

It is unfortunate that today's available standards do not cover all the areas required for a fully open NGC. Many NGC application areas presently unsupported by existing standards are now being worked by standards committees and open systems consortiums.

This specification leverages such relevant draft standards work if it is sufficiently mature and promises to have market acceptance. This position is taken in lieu of the alternatives: either to define, by a much smaller committee, an independent solution, or to simply not address known gaps in the NGC open systems architecture. Existing standards and draft standards span most of the NGC open requirements space but still leave some gaps. These gaps are duly noted in the subsequent standards discussions.

3.3.2.1 Application Programming Interface (API)

API categories include: *platform, presentation management, data management, geometric modeling, communications,* and *device input/output (I/O).* In each category, a set of open standards are identified to facilitate source code level portability. The essential ingredient for this level of portability is a well-specified, language-dependent, call-level interface description.

The NGC API definitions specify C language bindings. It is left up to the discretion of the NGC application designer as to whether or not he may wish to implement C++ "wrappers" (or some other object-oriented language wrapper) around such C language calls to access API services using a message-passing paradigm.

Wrapping maps the language interface of an application to the object interface. It places an object-oriented interface in front of one that is not. This concept goes hand-in-hand with the concept of a "container" object. Standard "C" applications may be placed inside container objects so that they may send messages through an object-oriented interface while preserving it's appearance as a single application.

The Object Management Group's (OMG) Common Object Request Broker Architecture (CORBA), for example, embodies many of these concepts. It is an open question as to whether there are more CORBA implementations worldwide today than there are Open Software Foundation (OSF) Distributing Computing Environment (DCE) implementations. Both standards are primarily focused on writing applications that communicate with each other. DCE's Remote Procedure Call (RPC) communications method provides node transparency using a client-server methodology. CORBA's API specification is designed around the object-oriented methodology of message passing and supports the interoperability with other technologies via its "static" API interface, which compiles at runtime.

As evident in the NGC reference model and the standards taxonomy, a large part of the NGC API specification is rooted in the Institute of Electrical and Electronics Engineers, Inc., (IEEE) portable operating system interface (POSIX). Many of the NGC concepts draw from IEEE/TCOS POSIX work. However, it should be clear that the NGC extends the notion of profiles defined for POSIX, as well as that for International Organization for Standardization/Open Systems Interconnection (ISO/OSI), to include a number of non-POSIX API standards. Purists within the open systems standards communities may have some difficulty with this approach but certain de facto standards have such large market support that they can not be ignored.

A common misconception is that POSIX is an operating system. POSIX describes the contract between the applications and the operating system. It defines the interface between applications and their libraries and says little about how to write those applications programs or how to write the operating system. The importance of adopting the interface is to enable portability of software applications across a variety of platforms. Any specific implementation of a NGC platform is acceptable provided the standard interface API, services, protocols, and associated behaviors are followed.

Therefore, workstation operating systems such as Digital Equipment Corporation (DEC) Ultrix, SMI SunOS, and Hewlett-Packard HP-UX; PC-based operating systems such as Santa Cruz Operation, Inc. (SCO) Xenix and MS Windows NT; micro-kernel-based systems such as Mach and Chorus; and finally real-time operating systems (RTOSs) such as Integrated Systems pSOS, Ready Systems VRTX, and Lynx LynxOS are all acceptable operating system implementations provided that they support at least one of the NGC platform API profiles. In terms of the computer hardware, platform implementations may include processors such as Intel, Motorola and Inmos; backplane busses such as VME, ISA/EISA, and NuBus; and displays such as ASCII-based terminals and X-terminals. NGC platform implementations comprised of multiple processors and multiple operating systems are also viable solutions.

Many of the COTS operating system products already provide support for the NGC platform API. Others need to be adapted to support one or more of the NGC API profiles. This is especially true for those custom or proprietary operating systems that are not presently based on existing standards. The process of adapting COTS and proprietary products to the desired NGC API naturally leads to the notion of middleware.

Middleware is a term associated with distributed systems and communications. Within a client/server distributed database environment, middleware allows transparent client access of information over wide area networks. In the context of communications, it is a layer of code that sits between the operating system and the application. Middleware is designed to ease the development of peer-to-peer and client/server distributed systems. Middleware implements an API layer that shields distributed system and network designers from the details of specific programming environments. In other words, with middleware developers need not have to write to vendor-specific API; instead they may develop to a more generic, globally compatible standard interface.

The NGC extends the concept of middleware to support generic APIs for computer numerically controlled (CNC) device I/O, fieldbus and distributed control, window-based display management, etc. With this view, implementation of the NGC CEE may simply be considered a selection of computer hardware, a selection of COTS operating system and networking software, and a specific implementation of middleware to support one or several NGC API profiles. Figure 3.3.2.1 illustrates this perspective. The NGC support services layer is the middleware implementation layer.

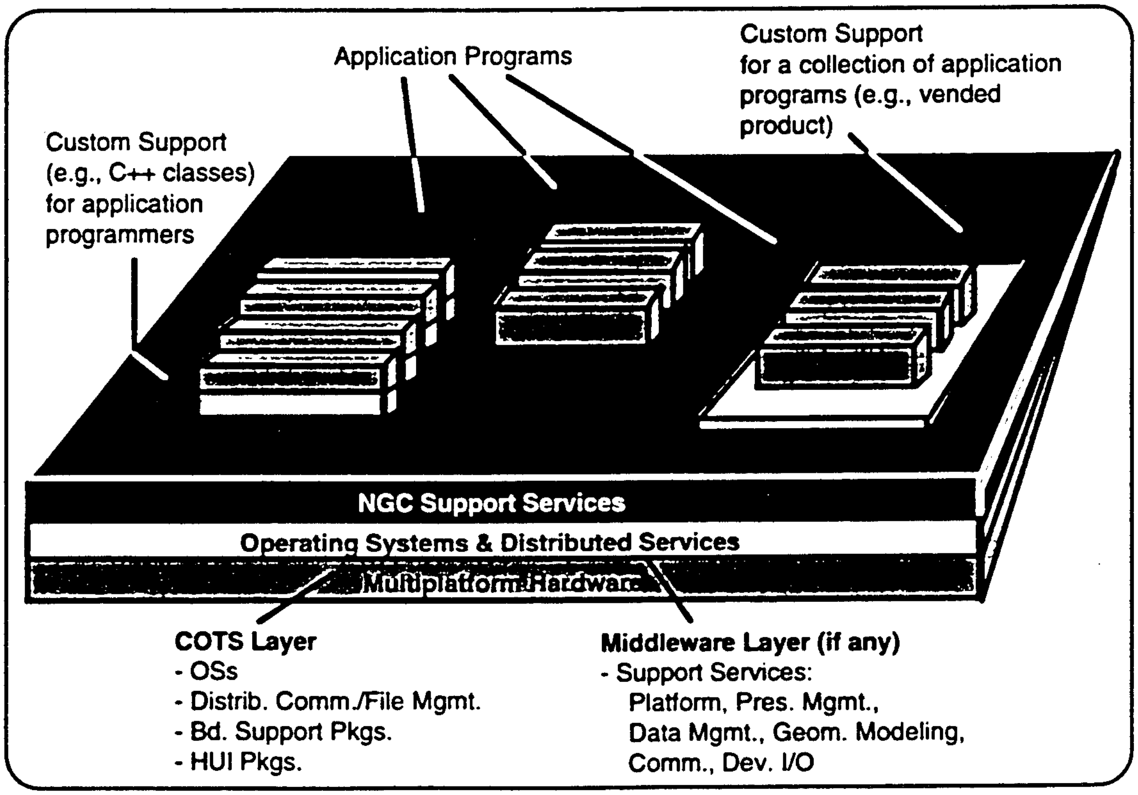


Figure 3.3.2.1 NGC Architecture Perspective

The following discusses relevant standards within each major API category. POSIX is first introduced as a general background. Explanatory information and several examples are also provided to help clarify and bring into focus some of the key NGC concepts. A general background in computing, communications, and software is assumed.

- 3.3.2.1.1 POSIX and ANSI C: An Introduction

Growing in popularity is the commitment among operating system vendors, especially among the real-time operating system (RTOS) vendors, to support the POSIX interface and real-time (RT) extensions to POSIX. Since POSIX specifies an interface, and does not describe how the functions are implemented, it can be applied across a number of underlying COTS operating systems.

Some operating system vendors have leveraged their existing operating system products by simply adding to them a POSIX-compliant interface. To accomplish this, a translation element must be designed, typically in the form of a runtime library, that takes POSIX system calls into native operating system calls. Digital's OpenVMS with its POSIX C-language runtime library and added POSIX system services kernel is an example of this. This approach provides a migration path toward open solutions in the future while maintaining a level of legacy support for existing applications and their dependencies on proprietary interfaces. The burden of immediate conversion to a fully open system is somewhat eased, and can be achieved in a period of time to spread the cost of changeover.

Alternatively, a controller developer may internally develop a POSIX API middleware layer on top of a COTS operating system product or on top of a proprietary kernel for reasons of product discrimination such as improved controller performance, cost competitiveness, etc. The choice of the specific POSIX implementation is up to the developer.

Table 3.3.2.1.1 lists many of the POSIX related documents that are either current standards or at some level of draft release. This table provides a context for subsequent discussions of API services, profiles, and conformant implementations. Only the POSIX documents most applicable to this specification are identified.

The versions of designated draft standards are the latest available as of March 1993. It is recognized that this specification must be updated periodically if it is to accommodate the most recent standards developments. This is actually less of a problem than it may sound since many operating system vendors have already announced their intent to support the evolving POSIX draft standards, especially in the area of RT POSIX extensions (POSIX.4). In that sense, modifications to the SOSAS in the area of POSIX will merely track known direction for commercial operating system products.

Table 3.3.2.1.1 POSIX Documents

Designation	Title	Relevant NGC API
IEEE 1003.0 D15	Guide to POSIX Based Open System Architecture	All
none - guide	IEEE Standard Interpretations of IEEE Standard Portable Operating System Interface for Computer Environments (IEEE Std 1003.1-1988) Prod. No. SH15313-PVB ISBN: 1-55937-216-8	All
ANSI/IEEE 1003.1-1990	Portable Operating System for Computer Environments (same as ISO/ IEC 9945-1:1990)	All
ISO/IEC 9945-1:1990	Information Technology–Portable Operating System Interface (POSIX)-Part 1: System Application Program Interface (API) [C language] Ed. 1 356p. JTC 1	All
IEEE 1003.1a D7	Draft Revision to POSIX.1	All
IEEE 1003.1 LIS D3	System Application Program Interface	All
IEEE 1003.2a D8	POSIX--Part II: Shell Utilities, User Portability Extensions	Pres.
IEEE 1003.2b D4	POSIX--Part II: Shell and Utilities, Amendment 2	Pres.
ANSI/IEEE 1003.3-1991	Information Technology-Test Methods for Measuring Conformance to POSIX	All
IEEE 1003.3-1991	IEEE Standard for Information Technology- Test Methods for Measuring Conformance to POSIX Prod. No. SH14068-PVB ISBN: 1-55937-104-8 (same as ANSI/IEEE 1003.3-1991)	All
IEEE 1003.3.1 D14	Test Methods for Measuring Conformance to POSIX–System Interfaces	All
IEEE 1003.3.2 D8	Test Methods for Measuring Conformance to POSIX–Shell & Utilities Interface	Pres.
IEEE 1003.4 D13	POSIX--Part 1: Real Time & Related System API	All except Pres.
IEEE 1003.4a D6	Standards for Threads Interface to POSIX	Platform, Pres, Data
IEEE 1003.4b D4	Feb 92 POSIX--Part 1: Real Time System API Extensions	Platform, Comm., Device I/O
IEEE 1003.8 D6	POSIX–Part 1: Network-Transparent File Access	Data
IEEE 1003.12 D1.2	POSIX–Protocol Independent Interfaces	Comm
IEEE 1003.13 D5	POSIX Standardized Profile	Profiles-All
IEEE 1003.16 D3	POSIX C Language Bindings-Part 1	All
IEEE 1003.17 D4	POSIX Directory Service API	Comm., Data
IEEE 1003.21 D?	POSIX Real-Time Communications	Comm.
IEEE 2003.1	IEEE Standard for Information Technology--Test Methods for Measuring Conformance to POSIX SH15826	All
FIPSPUB151-2	Portable Operating System Interface (POSIX)–System Application Program Interface [C Language] 93 May 12	All

The Federal Information Processing Standards Publications document, FIPSPUB-151, is a procurement profile specified by the government based on the POSIX.1 (ANSI/IEEE 1003.1) standard. This document is proof of government support of the POSIX standard, and is included here as a basis for defining future NGC government/military profiles.

Not included in the table are POSIX documents in the areas of system administration, language bindings for FORTRAN and ADA, system security, supercomputing, profiles for transaction processing and multi-process platforms, and batch processing. Although these subjects may have some NGC relevance, they are considered secondary to the fundamental requirements for open system controllers. As open controller philosophies materialize in open controller prototypes and NGC products, these additional documents need to be revisited for SOSAS applicability.

POSIX.1 covers the basic operating system services. It had originated as an Institute of Electrical and Electronics Engineers, Inc., (IEEE) effort, became an American National Standards Institute (ANSI) standard (ANSI/IEEE 1003.1) in 1990, and is also an international standard (ISO/IEC 9945-1:1990). It is language-dependent, meaning the APIs are specified for C bindings (a call-level interface description for the C language) and is perhaps the most essential document within the standard set. Current standards efforts include the re-write of this document as a language independent specification (LIS). The C language bindings are to be specified as a separate document, as for bindings to other languages such as FORTRAN or ADA.

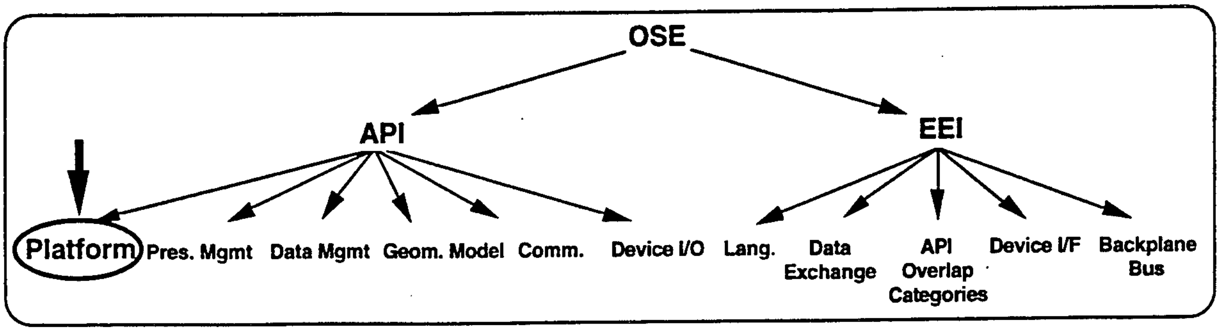
The NGC POSIX profiles are based on POSIX.1 and are strongly dependent on ANSI C, officially designated ANSI/ISO 9899. This 1990 version of the C language standard is an international standard (ISO/IEC 9899:1990). It is a revision and redesignation of ANSI X3.159-1989. The corresponding government procurement profile to this standard is FIPSPUB-160.

Many of the NGC API services are supported through the adoption of the ANSI C standard libraries. POSIX.1 explicitly deals with C language-dependent services and defines two types of C language conformance: (1) C standard, and (2) common usage C. This emphasizes the strong connection between POSIX and ANSI C.

The adoption of ANSI C for use in non-POSIX operating system environments also satisfies a level of POSIX compliance. In such implementations, it is only when services are requested outside of the support ordinarily provided through standard ANSI C libraries that the applications are no longer conformant with POSIX. The designer's choices for NGC applications are then to use either the API extras and RT extensions defined within the POSIX standard and implement a layer of corresponding middleware, or to rely on the support of vendor-specific API libraries and de facto standards.

For the case where the developer chooses not to use POSIX, a NGC profile set is provided for using other system implementations and corresponding support libraries. This approach satisfies a need to support legacy systems and eases the migration toward future POSIX implementations. The developer may always choose, instead, to either select a commercial POSIX operating system platform or develop a POSIX middleware layer on top of a proprietary operating system.

3.3.2.1.2 Platform Services API



Platform services support many of the features commonly provided by commercial operating systems today. These services include, for example, process/task control, synchronization and scheduling, event and interrupt handling, time services, memory, and other resource management.

Table 3.3.2.1.2 identifies the relevant de jure standards in the area of Platform services. De facto standards in this area and their relevance to the NGC are discussed in Section 3.3.3, Profiles.

Table 3.3.2.1.2 Platform Services—Relevant Open Standards

ISO/IEC 9945-1:1990	Portable Operating System for Computer Environments (same as ISO/ IEC 9945-1:1990)
IEEE 1003.4 D13	POSIX--Part 1: Real Time & Related System API
IEEE 1003.4a D6	Standards for Threads Interface to POSIX
IEEE 1003.4b D4	Feb 92 POSIX--Part 1: Real Time System API Extensions
ISO/IEC 9899:1990	Programming Language--C

ANSI C, now an international standard, is identified in the table as a standard separate from POSIX but it should be recognized that it is also an integral part of the POSIX.1 specification. Most of the standard ANSI C library services fall under the umbrella of NGC platform services. This includes C library functions and corresponding API in the areas of character and string handling, localization, mathematics, non-local jumps, input/output, date and time, diagnostics, and general utilities. It is only in the I/O area that some of the C library functions relate to other service categories such as presentation management, data management, and device I/O.

POSIX.1 Platform services API include functionality for single and multi-processing, signals, and user groups. RT Platform services API, such as RT signals, semaphores, memory management, priority scheduling, and timers, are defined in POSIX.4. Thread services and reentrant functions API are covered in POSIX.4a. Spawning, central processing unit (CPU) time management, and sporadic server services API are specified in POSIX.4b. Note that the services mentioned are not the full breadth of the POSIX services, but only those services that relate specifically to the platform category. POSIX also spans the areas of communications, device I/O, data management (file management), etc.

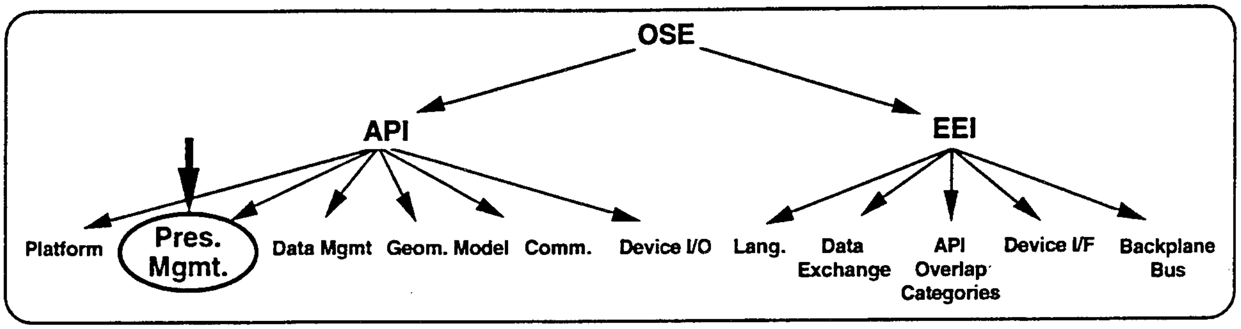
The draft nature of the RT API definition is not viewed as a major problem due to the current support of draft levels of POSIX.4 in a variety of RTOS products. For many NGC controllers, these COTS operating system products may form an integral part of their total controller implementation solution.

The operating system industry acceptance and momentum towards open systems is evident in Microsoft's latest operating system product, Windows NT, which supports POSIX. The list of RTOS products with varying levels of in-progress and available POSIX.1 and RT support includes:

- Encore Computer (UMAX V R/T)
- Eyring (PDOS)
- Harris Computer Systems (CX/UX)
- Industrial Programming (MTOS)
- Integrated Systems (pSOS)
- JMI Software Systems (C Executive)
- Lynx Real-Time Systems (Lynx operating system)
- Microware Systems (OS-9000)
- Modular Computer Systems (8xxx)
- Precise Software Technologies (Precise/MQX, MPX)
- QNX Software Systems (QNX), RTMX-UniFLEX (RTMX/RN/RX)
- Spectron Microsystems (SPOX)
- Wind River Systems (VxWorks, MicroWorks)

Many other operating system vendors have announced their product plans to support POSIX.

3.3.2.1.3 Presentation Management Services (PM API)



Presentation Management services support the man-machine aspects of the NGC typically associated with a standard computer terminal. Supported hardware may include a display, keyboard, and trackball or mouse. Presentation Management services may be used to support control panel functionality, provided such capability is emulated on the NGC terminal and is not a hardware panel implementation. For example, one NGC controller product line may use a graphics windowing environment with specialized NGC menus, dialog boxes, and icons that implement (in software) many of the control panel features commonly supported today using hardware dials and buttons.

To clarify this point, Presentation Management services do not interface physical buttons dedicated to jog, feedhold, pause, and E-stop functions, nor do they manage jog handwheels, mode-select switches, and programmable function keys. However, API services have been identified to support these types of hardware interfaces and are described under Device I/O services API.

Table 3.3.2.1.3 lists the major presentation management standards. Not all the standards listed need to be adopted for NGC-conformant controllers and many of the standards shown overlap in functionality. NGC profiles for presentation management define the standards subsets as well as applicable de facto standards. In designing the controller, satisfying many of the standards listed in the table is easily accomplished through the use of existing COTS products.

In selecting relevant standards for this specification, the following general guideline has been applied: if the selection of a standard for a specific application involves a choice between the national (e.g., ANSI) standard and the corresponding international standard (e.g., ISO/IEC), the NGC selection is usually the international standard. Note the NGC selection for the ANSI C language is ISO/IEC 9899. Correspondingly, ISO/IEC standards are identified for Programmer's

Hierarchical Interactive Graphics System (PHIGS) and Graphical Kernel System (GKS) instead of existing ANSI standards, e.g., ANSI X3.124-1985 (R1991) GKS Functional Description.

Table 3.3.2.1.3 Presentation Management Services—Relevant Open Standards

ISO/IEC 9945-1:1990	Portable Operating System for Computer Environments (same as ISO/IEC 9945-1:1990)
IEEE 1003.2a D8	POSIX--Part II: Shell Utilities, User Portability Extensions
IEEE 1003.2b D4	POSIX-Part II: Shell and Utilities, Amendment 2
IEEE 1003.4a D6	Standards for Threads Interface to POSIX
ISO/IEC 9899:1990	Programming Language - C
MIT X Windows (X11R5)	MIT X Window System Version 11, Release 5
MIT X Windows PEX extension	MIT X Window System PHIGS EXtension
OSF/Motif 1.2	Open Software Foundation - Motif
ISO/IEC 8651-4:1991	Graphical Kernel System (GKS) language bindings - Part 4: C
ISO/IEC 8806-4:1991	Graphical Kernel System for Three Dimensions (GKS-3D) language bindings - Part 4: C
ISO/IEC 9593-4:1991	Programmer's Hierarchical Interactive Graphics System (PHIGS) language bindings - Part 4: C
IEEE 1201.1	Uniform Applications Program Interface Graphical User, Rev. 3, DS01719 (Windowing Toolkit API)

Relevant portions of POSIX.1 deal with terminal identification and device/class specific APIs. POSIX.4a specifies reentrant functions for generating a terminal path name and determining terminal device names. Character and string based keyboard and display I/O functions defined in the C language (e.g., getchar(), printf(), etc.) are also applicable.

The POSIX shell utilities (POSIX.2x) are that part of the specification that deals with what is commonly referred to on most systems as the command line interpreter (CLI). It is the CLI that people identify with most when referring to a particular type of computer system and the expected behavior of that system from a user's perspective. The well-known "C, colon, backslash" (C:\) syntax is the CLI of the popular MS-DOS operating system found on many IBM PC compatible, Intel-based platforms. The POSIX shell is modeled after the user interface commonly found on many UNIX-based platforms.

A number of API open standards exist for graphics. The PHIGS and GKS international standards define 2-D and 3-D graphic programming interfaces. PHIGS is also the interface supported by the Massachusetts Institute of Technology (MIT) X Consortium for using the PEX 3-D graphics extension. It is interesting to note that the X Consortium is currently in the process of spinning out of MIT into a nonprofit, more market-driven organization. OpenLook, the chief graphical user interface (GUI) rival of Motif, is absent from the list due to the recent agreement by leading parties in the UNIX market to settle on the OSF/Motif interface for the Common Open Software Environment (COSE).

Other API strategies include X/Open's favorable intention to adopt the Public Windows Interface (PWI) in a future release of the Portability Guide (XPG). The PWI, developed by several COSE sponsors, is a specification of the proprietary Microsoft Windows interface. The plan is to pass the specification off to a vendor-neutral standards body (most likely X/Open) so that Windows-based technologies would no longer have to pay royalties to Microsoft. This is important for SunSelect's new "Wabi" interface product, which allows users to run Windows 3.1 applications on reduced instruction set computer (RISC) workstations running UNIX and X Windows. Should the PWI document survive the potential mitigation and become public domain, future revisions of this specification may include this standard as part of its OSE specification.

Also in this area is a current IEEE effort to standardize on a windowing toolkit API. This standard, IEEE 1201.1, is currently in draft form. It is premature to assess the resulting product support but it appears to carry enough momentum to qualify its candidacy in the NGC SOSAS.

Figure 3.3.2.1.3 illustrates some of the possibilities for implementing Presentation Management services at varying levels of capability and windowing support. Several different combinations of COTS products and/or middleware developed solutions are implied. As described through profiles, not all levels need be supported by any one specific implementation. This subject is expanded in the profile section.

The API standards identified provide for portable operator interface component source code. In addition, they also address issues related with user portability; that is, they provide a foundation for one of the major NGC benefits, common look-and-feel. Although the standards pave a direction, they are not, in themselves, the complete solution.

A display style guide to be followed in the development of a NGC operator interface component, and the user interface capability to customize and tailor the interface, are also essential ingredients. This allows one shop floor, for example, to tailor all their NGC controllers to suit their specific needs, yet also support a common user interface to minimize training and maintenance throughout their shop.

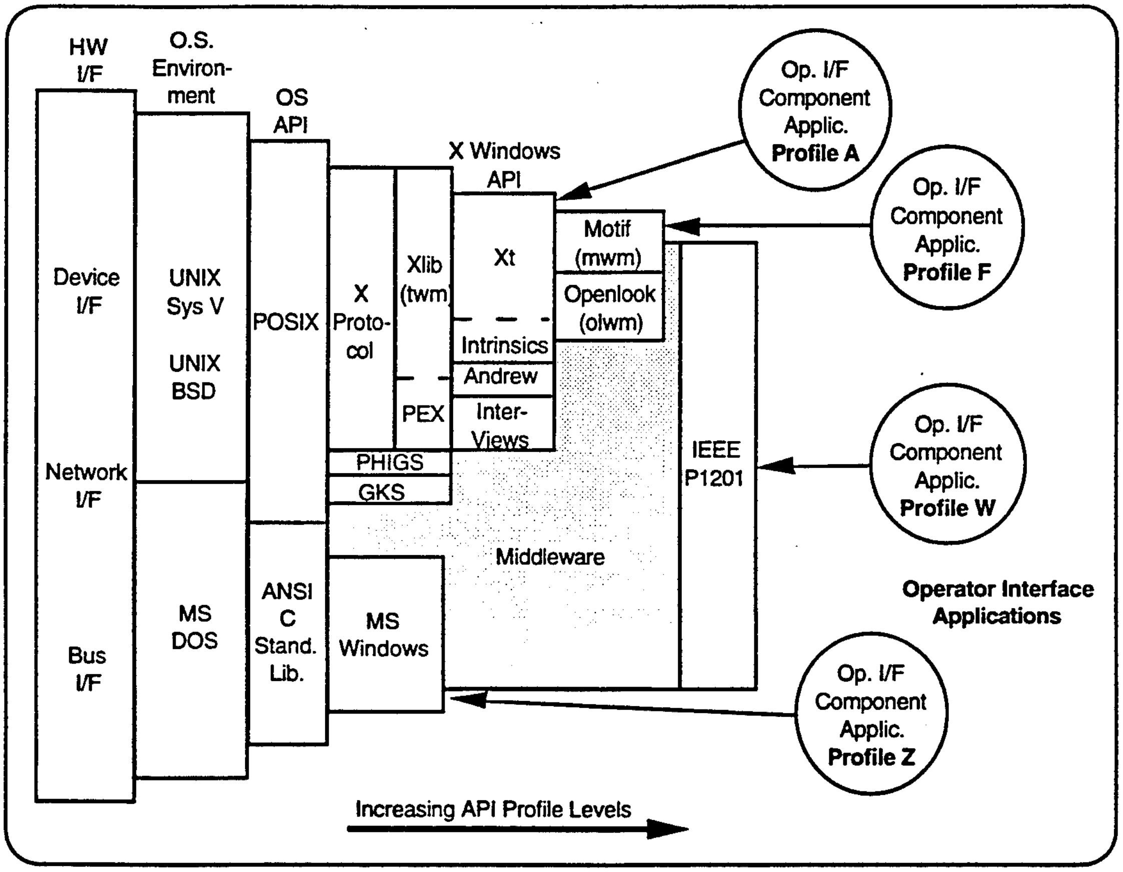
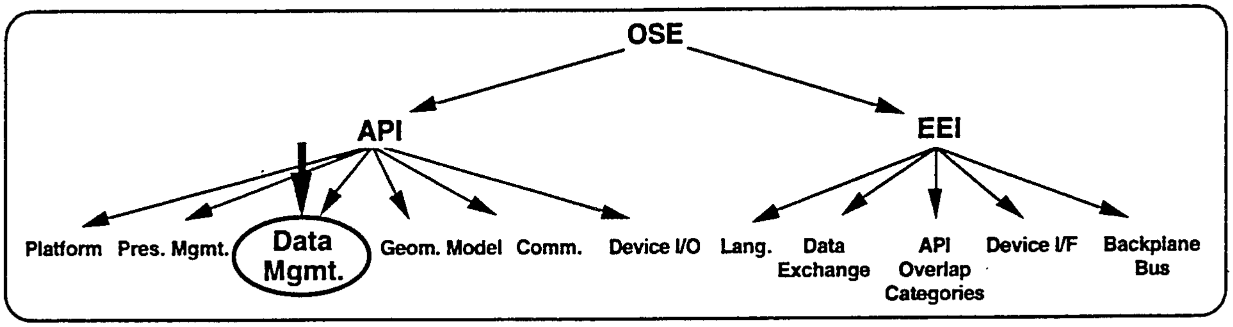


Figure 3.3.2.1.3 Presentation Management Services—API Levels and Implementation

3.3.2.1.4 Data Management Services (DM API)



Data Management services refer to the file management capabilities of the system. The commercial products available for data management today cover a broad spectrum from independent proprietary protocol implementations on a single host environment to open, transparent file access solutions for multi-processor configurations. The latter provides the most portability advantage for NGC applications.

From the perspective of the NGC as a system, Data Management services are concerned with both external as well as internal system file support. External file support may simply involve the use of a utility such as file transfer protocol (FTP) across a local area network (LAN) or Internet to a remote host file system or network. In terms of an ISO/OSI open system alternative, it may involve the use of file, transfer, access, and manipulation (FTAM) for remote file access. Both approaches are closely tied to the NGC communications services as well as Data Management services, and both are supported through NGC profiles. Internal to the NGC, Data Management services may also provide data storage access across loosely and tightly coupled multi-processor controller implementations as well as simple file access within a single processor environment.

The choice of a data management implementation is based on the intended application of the specific controller product line. It is conceivable that some low-cost NGC families may not feature a cell-level interface for external file support. The channel to the outside world for these controllers could be a simple serial interface for RS-274 input. In contrast, high-ended NGC controllers may not only support remote host file access across a shop floor network, but also support "lights-out" operation for a 24-hour period. This, of course, is the vision of the future.

Transparent remote data access (RDA), such as that provided from Sun's Network File System (NFS) product is the current trend and a key focus within open systems communities. In the UNIX environment, NFS makes remote file systems look the same as local ones. OSF's

Distributed Computing Environment (DCE) takes this a step further with the Distributed File Service (DFS) by providing all users a single view of all files (both UNIX and non-UNIX based) within an organization. Transparency deals with making the file resource more visible than the computer host system that supplies the resource.

Table 3.3.2.1.4 lists the standards most relevant to Data Management services. Relevant portions of the POSIX.1 standard include files and directory services, system database access services, and services defined for file descriptor manipulation and control operations on files. POSIX.4 covers file truncation, synchronization, and real-time files. POSIX.4a identifies reentrant functions for group/user database access, file lock/unlock synchronization, and thread-specific data key/data management. Standard C defines file input/output functions, e.g. fread(), fwrite(), fseek(), etc.

Table 3.3.2.1.4 Data Management Services—Relevant Open Standards

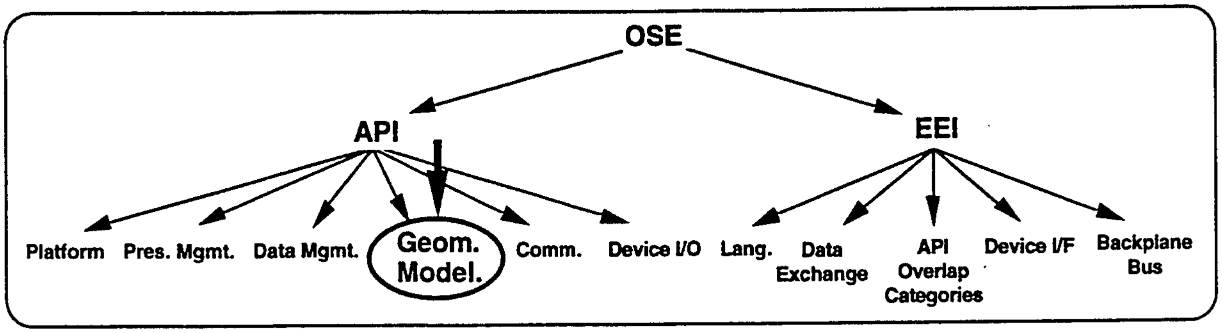
ISO/IEC 9945-1:1990	Portable Operating System for Computer Environments (same as ISO/IEC 9945-1:1990)
IEEE 1003.4 D13	POSIX--Part 1: Real Time & Related System API
IEEE 1003.4a D6	Standards for Threads Interface to POSIX
IEEE 1003.8 D6	POSIX--Part 1: Network-Transparent File Access
ISO/IEC 9899:1990	Programming Language - C
X/Open S203	Data Management: SQL Call Level Interface (CLI)
IEEE 1238.1	File Transfer Access & Management Applications Interface (Rev 2), DS02345
X/Open P206	FTAM High-level API (XFTAM)

The most interesting standard in the list is POSIX.8, Transparent File Access (TFA). TFA refines the file system services specified in POSIX.1. It defines full TFA and core TFA in a manner similar to the general NGC profiling approach. Full TFA provides all the file services defined in POSIX.1. Core TFA defines a minimum set so that TFA may be used with other non-POSIX file systems. Functional extensions to the core services are also defined for the flexibility of adding capability on an individual file basis. POSIX.8, Annex F includes informative profiles for several different file systems (e.g., Sun NFS, FTAM and PC/DOS, etc.).

Note the Structured Query Language (SQL) international standard is not referenced in the table, although it is quite relevant to data management. Many popular database management systems today (e.g., Oracle, Sybase, Ingress, Informix, etc.) support the SQL client/server protocol. It should be clear that the standards listed in the table focus on portability and relevant API. SQL is a protocol standard and, by definition, addresses interoperability concerns. Consequently, this standard falls under the external environment interface (EEI) branch of the Open Systems Environment (OSE) taxonomy and is discussed there.

Of lesser importance, X/Open's SQL call-level interface standard is included for completeness to allow portability of controller applications that have built-in remote file management using SQL. This may be an alternative to using a COTS proprietary data base API but may require a middleware bridge between the NGC applications and the leveraged commercial data base product. The other OSI/FTAM API standards are included for analogous reasons.

3.3.2.1.5 Geometric Modeling Services (GM API)



Geometric Modeling services define a standard programming interface to product modeling systems for extracting and operating on product definition data (PDD). The Geometric Modeling services category is unique in that it tends to be more application specific than the other API categories. However, the existence of open standards work in geometric modeling justifies its place as an API category. Since this specification is an anticipatory standard, it is hoped that other application-specific API categories will emerge in areas such as motion control, programmable logic control, etc.

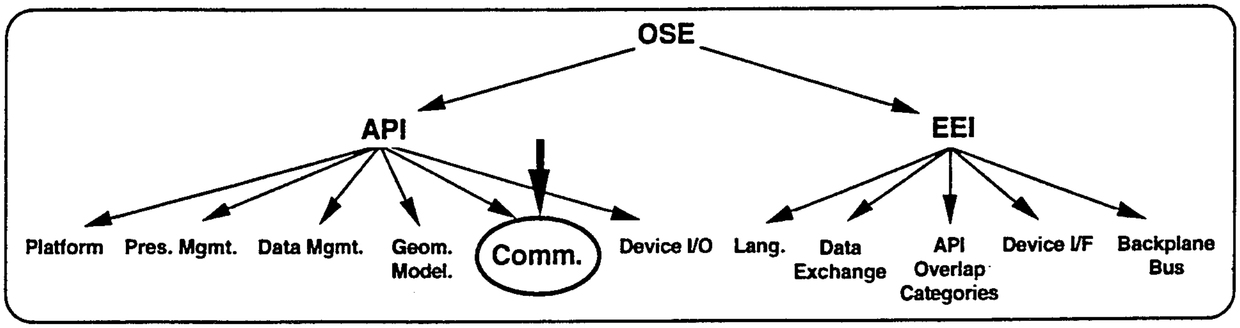
Table 3.3.2.1.5 shows the only relevant open standard in the area of geometric modeling, the Consortium for Advanced Manufacturing International (CAM-I) Application Interface Specification (AIS). CAM-I is committed to maintaining compatibility with the PDES/STEP standards. The significance of the AIS is that it goes beyond data exchange by supporting operations on product data definition.

Table 3.3.2.1.5 Geometric Modeling Services—Relevant Open Standards

CAM-I AIS 2.0 Draft Standard Vol II	CAM-I Application Interface Specification (AIS) Revised 1991 Document Number R-90-PM-03 AIS 2.0 C Language Binding

The PDES/STEP product data exchange standards have an important future relevance to NGC and there will probably be a wealth of new products entering the market based on these standards as they continue to mature. They may be the key to the NGC art-to-part factory of the future by providing an industry standard that completely describes all part characteristics. As data standards, their NGC role is one of ensuring system and applications level interoperability and fall under the EEI branch of the NGC taxonomy.

3.3.2.1.6 Communications Services (Comm API)



Communications services API are perhaps the most important set of services of the NGC OSE. Communications API not only provides for portability of source code that interfaces the NGC to other external systems (e.g., networks), but also defines the set of callable interfaces that support peer-to-peer communications between the internal NGC component agents.

The transparency of the communications system, that is, the ability of the communications system to make resources more visible than the computer host that supplies the resource, is a key focus of the NGC. The Data Management services discuss the notion of transparency in the context of remote data access. The objective for file transparency is to make remote file systems resident on heterogeneous hosts look the same as local ones. Architecturally, the inherent transparency of a distributed file system is often a function of its underlying distributed communications system and directory services.

Table 3.3.2.1.6 lists the open standards most relevant to communications. Inter-process communications is achieved at various levels. One implementation may use a pipe to create an inter-process channel and the general read(), write() functions defined in POSIX.1. Other implementations may rely on POSIX.4 message queue functions and the corresponding blocking timeout services specified in POSIX.4b.

POSIX.12, protocol independent interface (PII), is perhaps the most important open standard of the list. It defines two networking interfaces for protocol-independent, process-to-process communication: the simple networking interface (SNI) and the detailed network interface (DNI). SNI is a simple applications interface that provides for co-operating process intercommunication without requiring details about underlying protocols. DNI offers protocol-independent mechanisms for manipulating protocol specific features of the underlying network. DNI will be specified with C language bindings for Berkeley Software Distributions (BSD) sockets and X/Open XTI. Both connection-oriented, byte- or record-stream virtual circuits and connectionless channel datagrams are supported. Relevant OSI APIs are also referenced for completeness.

Table 3.3.2.1.6 Communications Services—Relevant Open Standards

ISO/IEC 9945-1:1990	Portable Operating System for Computer Environments (same as ISO/IEC 9945-1:1990)
IEEE 1003.4 D13	POSIX--Part 1: Real Time & Related System API
IEEE 1003.4b D4	Feb 92 POSIX--Part 1: Real Time System API Extensions
IEEE 1003.12 D1.2	POSIX - Protocol Independent Interface
IEEE 1003.17 D4	POSIX Directory Service API
IEEE 1003.21 D	POSIX Real-Time Communications
ISO/IEC 9899:1990	Programming Language - C
IEEE 1238.0	Draft Common OSI Connection Management (Rev 4), DS01339 (API to Common OSI Connection Management and Support Functions Project)
IEEE 1224	OSI Applications Programmer Interface. Jan 93 (Rev 8) DS 01123
IEEE 1224.1	X.400 Electronic Messaging API (Rev 6) DS01362
IEEE 1224.2	Directory Services Applications Programmer Interface (Rev 5) DS02428
X/Open T003	X.400 (APIsand EDI Messaging) Vols. 1-6
X/Open P203	ASCE/Presentation Services API
X/Open C196	X/Open Transport Interface (XTI)
X/Open P210	The Common Object Request Broker: Architecture and Specification - X/Open in conjunction with Object Management Group (OMG)
OSF DCE 1.0.2	Open Software Foundation Distributed Computing Environment Version 1.0.2

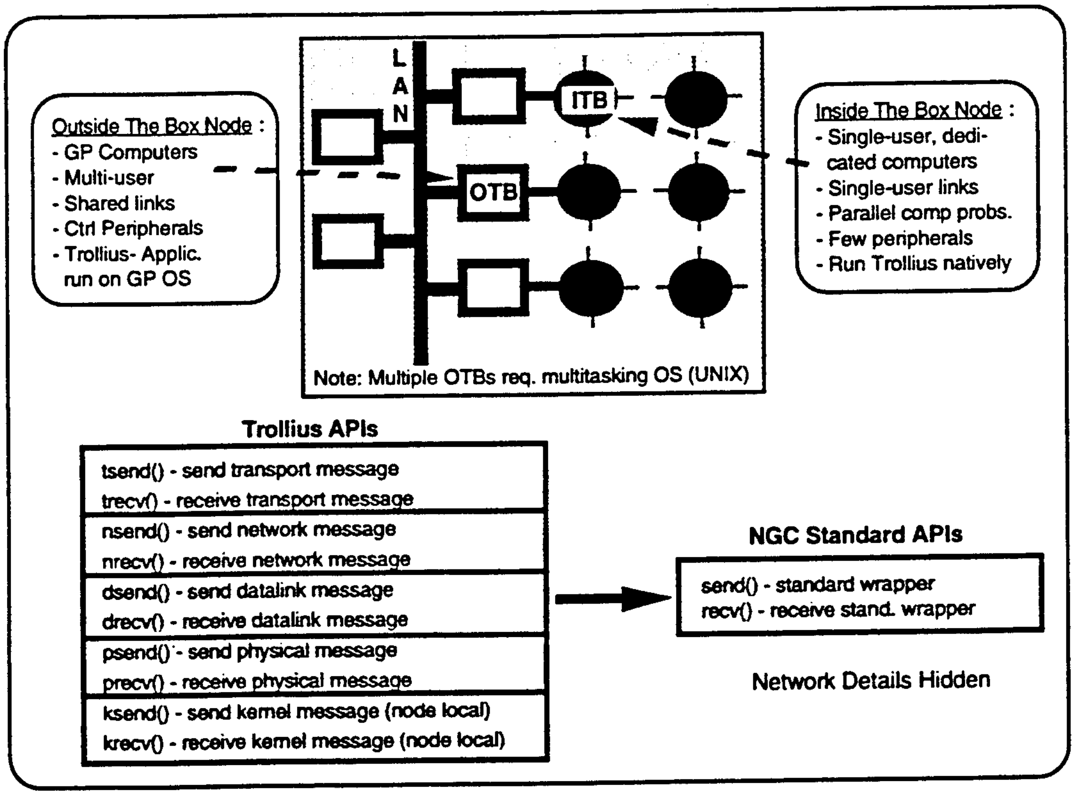
Directory Services (POSIX.17) is the transparency glue for resolving the physical location of objects, given globally defined logical names. This standard was mainly intended as an International Consultative Committee on Telegraphy and Telephony (CCITT) X.500 API but may also be used to access directory systems in existing practice. It offers one open system approach to NGC for network communications and physical node transparency.

A Directory services component is also defined for NGC to provide a level of communications transparency for peer-to-peer communications between agents, the processes and threads, within a NGC. This component hides the communications details (sockets, process ids, message queue descriptors, etc.) for single processor, multiple process environments. It also hides node id details for distributed, multi-processor NGC implementations.

Like a telephone book, the Directory services component is responsible for managing all the address information to globally named system entities. Every NGC component implementation must have at least one agent for sending and receiving messages. Each communications agent globally registers to the system symbolically, by its agent and corresponding component name set. Node residency and all other required information to communicate between agents may be obtained dynamically, during system operation, through the Directory services.

Several existing products attempt to provide a simple and efficient communications interface in a loosely and tightly coupled multi-processor environment. Most of these products fall short in hiding node details or the underlying communications mechanisms.

For example, the Ohio State Trollius is an open architecture operating system for a concurrent message passing machine and attached UNIX machines. This product, although it is quite powerful by today's standards, defines APIs that are network dependent. In addition, nodes are not transparent at an applications programming level, and node ids as well as event synchronization information must be supplied as arguments to the communications API service calls. Figure 3.3.2.1.6-1 illustrates the Trollius distributed memory multiple instruction, multiple data (MIMD) architecture and the dependency of the Trollius communications API on the four lower layers of the ISO/OSI model: physical, datalink, network, and transport layers.



tsend() - send transport message
trecv() - receive transport message

nsend() - send network message
nrecv() - receive network message

dsend() - send datalink message
drecv() - receive datalink message

psend() - send physical message
precv() - receive physical message

ksend() - send kernel message (node local)
krecv() - receive kernel message (node local)

send() - standard wrapper
recv() - receive stand. wrapper

Figure 3.3.2.1.6-1 Communications Services—Trollius Example APIs Network Dependent

An applications-level API for NGC must shield the programmer from these networking details, yet allow the programmer to selectively configure the implementation of the lower layers. This can be achieved with programming options during the link process of the application, so the desired communications implementation levels are built-in with the selection of the required software component libraries.

As Figure 3.3.2.1.6-1 illustrates, higher level send() and recv() API services would make network details transparent to the applications programmer, yet he may still have the option during the applications build process to select the level of protocol required to meet his specific performance needs. Link options to a variety of object libraries during applications build could facilitate such a flexible mechanism.

Two competing paradigms for communications are message passing and remote procedure calls (RPCs). Two consortiums, Open Software Foundation (OSF) and Object Management Group (OMG) are developing "middleware" in support of these different communications strategies. OSF's Distributed Computing Environment (DCE) implements RPCs. OMG's Common Object Request Broker Architecture (CORBA) standardizes distributed object-oriented computing using message passing, which may also use RPC mechanisms as its underlying implementation.

From an architectural perspective, the interrelationship between POSIX, DCE, and CORBA is somewhat hierarchical. This hierarchy is illustrated in Figure 3.3.2.1.6-2. This view is quite a simplified view of the relationships of the standards. The whole story is, in fact, quite complicated but the hierarchical view helps to understand some of the NGC communications options and the levels of transparency that are available or missing when a particular option is selected. Although CORBA and DCE both provide mechanisms for remote procedure calls (RPCs), CORBA is shown at the highest level of the hierarchy because it is also an object-oriented standard. DCE is function-oriented but there is nothing to preclude "wrapping" DCE functionality into an object-oriented implementation.

DCE has evolved through the integration and standardization of a number of vendor products, many of which were once separate stand-alone or bundled products. Although it is incorrect to make the claim that all of DCE is built around POSIX, there are portions of DCE that are truly based on POSIX. The DCE "threads" and the concept of "lightweight processes" to improve application performance through parallelism are based on POSIX threads. However, some DCE services are not layered on POSIX and potentially need not be supported by commercially available OS products, as the architectural hierarchy diagram also graphically depicts.

Within the set of POSIX standards there is also an implicit hierarchy of interrelated standards. The POSIX hierarchy shown in Figure 3.3.2.1.6-2 is somewhat arbitrary in that it is not the only hierarchy possible. Some of these standards are still in draft form. The resulting products will establish the true interrelationships when these standards mature.

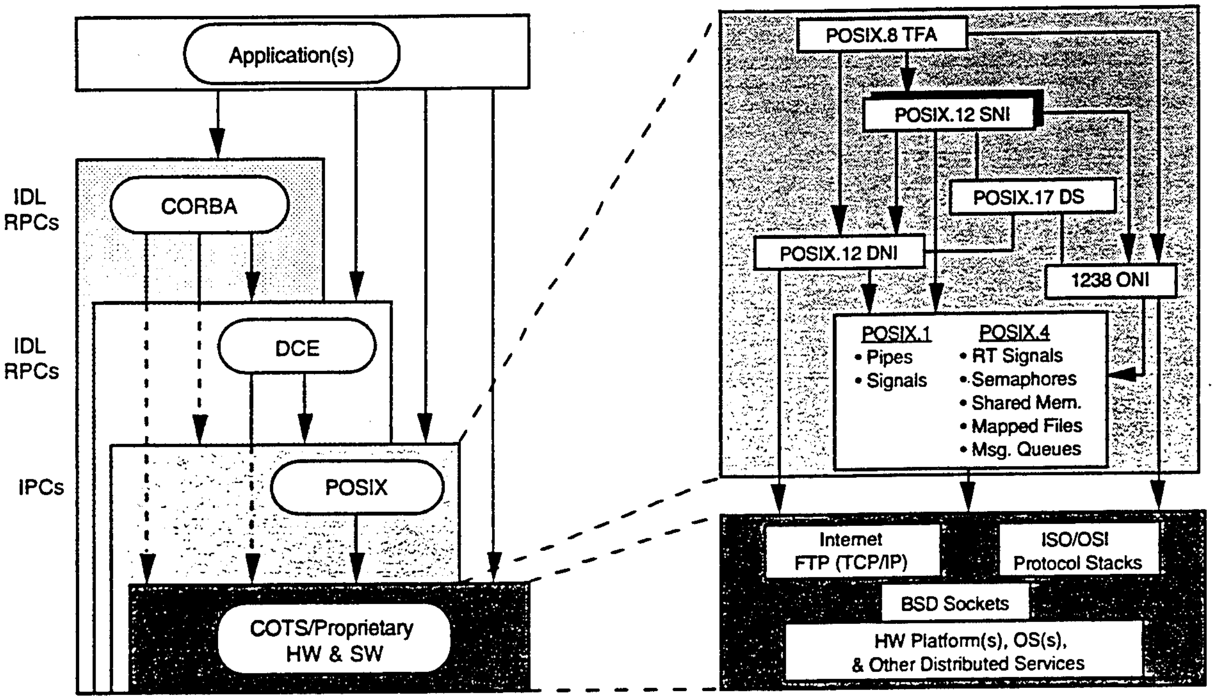


Figure 3.3.2.1.6-2 Communications Standards Architectural Hierarchy

POSIX.8 is the POSIX standard for transparent file access (TFA). This standard, referenced in the DM services section and earlier in this section by inference in the discussion of remote file access, is shown at the top of the POSIX hierarchy. The rationale for its placement at the top is due to its inherent dependence on the underlying network and communications mechanisms for its implementation.

Potential communications mechanisms appear as the POSIX.12 SNI (Simple Network Interface) and POSIX.12 DNI (Detailed Network Interface). Both of these standards are specified in the POSIX.12 PII (Protocol Independent Interface) document and were briefly described earlier in this section. POSIX.12 SNI appears at a higher level than POSIX.12 DNI to illustrate that an "implementation" of a simple network interface may involve access to detailed network services.

The distributed services, illustrated as POSIX.17 DS in the hierarchy diagram, are accessed by both POSIX.12 SNI and DNI. In addition, IEEE 1238.0, the Open Network Interface (ONI) makes use of these same distributed services. The POSIX standards include not only those standards designated by IEEE 1003.x, but also a number of other standards such as the 1238 ONI standard. The IEEE 1238 standard defines API at an applications level to the ISO/OSI protocol stacks. As illustrated in the hierarchy, the 1238 services may be used to support network or file access functions required of the higher level POSIX standards.

To complete the POSIX hierarchy, POSIX.1 and POSIX.4 may provide all the service support required of the other POSIX standards at a local host level. Local communications services may range from pipes and signals to real-time semaphores, shared memory, mapped files, or message queues.

The POSIX.12 SNI standard is highlighted (shadowed) in the hierarchy diagram because of its importance in specifying two communications API: sni_send() and sni_recv(). These two APIs approach a level of simplicity and communications transparency previously described as a necessary NGC feature yet is not available in many real-time communications products today. At an applications program level, these simple API may be used. The decision to use POSIX.4 shared memory or message queues for communications need not be made during applications development, but instead could be made during the applications build by selection from specific object libraries.

Another major point of distinction between the POSIX communications standards and the CORBA and DCE standards is that the POSIX standards deal with communications at the level of inter-process communications (IPCs). Both CORBA and DCE provide mechanisms for remote procedure calls (RPCs) which may be viewed to be at a higher level of abstraction than IPCs.

Figure 3.3.2.1.6-3 illustrates the development mechanisms for implementing CORBA or DCE RPCs. Within both the CORBA and DCE standards, an interface definition language (IDL) is specified. The applications developer defines his interfaces using IDL and then "translates" these interface definitions using an IDL compiler. The compiler generates header files to be included within the applications source code as well as client stubs and server skeletons which are "linked" into the executable client and server applications programs during the build process.

Both standards, CORBA and DCE, are currently viewed as most applicable at the external interface or workstation level of NGC communications, although there is nothing to prevent their use for peer-to-peer communications internal to NGC. The only restriction for their use in real-time applications is the current lack of real-time CORBA and DCE COTS products. Today's CORBA and DCE products are transaction-based and are oriented toward workstation level business applications. OMG's CORBA may provide the foundation for one implementation of an NGC external communications profile as well as perhaps OSF's DCE RPCs or even the TCP/IP capability of the underlying operating system. A real-time CORBA product, if one existed today, could have a tremendous impact on the approach to inter-process peer-to-peer communications within an NGC.

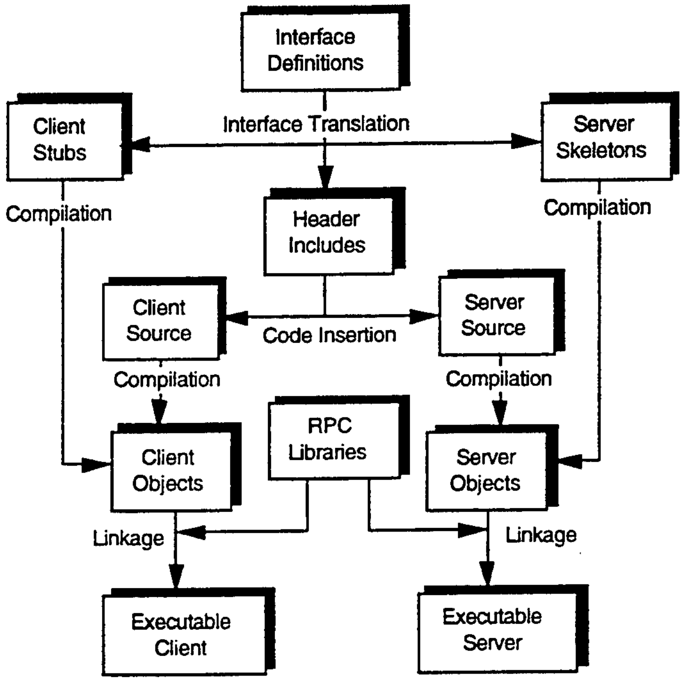
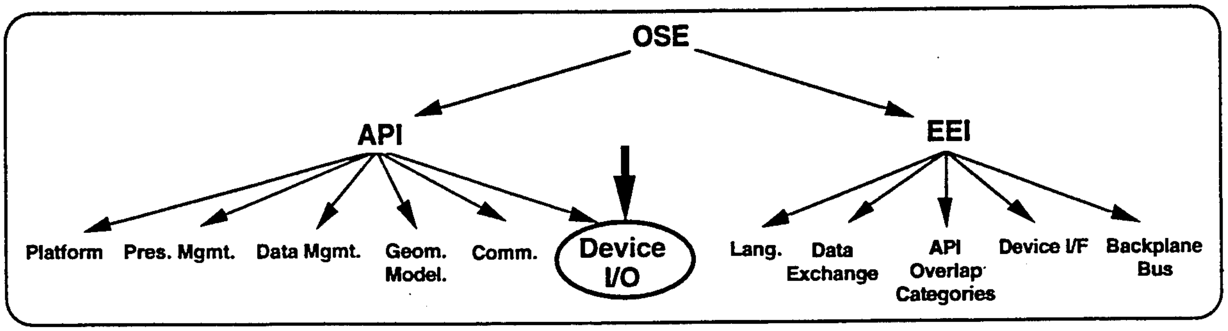


Figure 3.3.2.1.6-3 CORBA-DCE IDL & RPCs

The mechanism for message passing in NGC is Communications services. As mentioned in Section 3.3.2.1 (general API section) it is left up to the discretion of the NGC application designer as to whether or not he may wish to implement object-oriented wrappers around function-oriented calls to access API services using a message passing paradigm. Through the use of Communications services, NGC components may be implemented that embody the container object concepts defined for OMG's CORBA.

In summary, a set of communications standards and profiles are defined to enable the selection of a communications strategy most suitable for the specific application and real-time requirements of the NGC. As with all other API services, the Communications API focuses on NGC applications source code portability and does not necessarily define how the underlying services are to be implemented.

3.3.2.1.7 Device Input/Output Services (Device I/O API)



The Device I/O services are the peripheral management functions of the system. These services perform the logical-to-physical and physical-to-logical mappings between the device requests and the corresponding electrical signals to the device. The devices supported cover a broad spectrum from dedicated operator control panel buttons, handwheels and switches, to specialized sensor and actuator drive signals for motion control and tool change. Other devices may include communications devices, auxiliary storage devices, analog-to-digital (A/D) and digital-to-analog (D/A) converters, Centronics parallel interfaces, etc.

Table 3.3.2.1.7 identifies relevant device I/O open API standards. POSIX.1 covers general I/O primitives. These general open(), read(), write(), and close() primitives apply not only to files but also to devices. POSIX treats all I/O devices in the same general manner. File descriptors are general-purpose descriptors used for all device management. POSIX.4 defines asynchronous I/O primitives, e.g., aio_read(), aio_write(). Memory mapping functions may provide the required interface to memory mapped I/O devices. POSIX.4b specifies how to associate a user-written interrupt service routine (ISR) with an interrupt and also defines a control device function, devctl(), for a more direct path to the device driver.

Table 3.3.2.1.7. Device I/O Services—Relevant Open Standards

ISO/IEC 9945-1:1990	Portable Operating System for Computer Environments (same as ISO/ IEC 9945-1:1990)
IEEE 1003.4 D13	POSIX--Part 1: Real Time & Related System API
IEEE 1003.4b D4	Feb 92 POSIX--Part 1: Real Time System API Extensions
ISO/IEC 9899:1990	Programming Language - C

In 1991, The Real Time Consortium had been working on the Open Basic Input Output System (OBIOS) standard. The standardization effort, representing a collaborative effort of several key hardware and system software vendors in the real-time and embedded systems marketplace, was focused on developing a call interface and associated client/device interaction model that provides a common abstract interface for a wide range of I/O devices.

The OBIOS is an ideal candidate for inclusion in the NGC SOSAS. Unfortunately, the consortium dissolved while the standard was still in the draft stage. This was the only standard of its kind and identifies a standard gap in the NGC OSE. It may be of little value to include the contents from the OBIOS draft specification in the SOSAS since it appears at present to have little market interest. It is listed as a placeholder in the EEI section.

3.3.2.2 External Environment Interface (EEI)

The EEI includes API categories where interoperability issues are concerned and includes the following additional categories: *language, data exchange, device interface, and backplane bus.* In each category a set of relevant open standards are identified to facilitate both system-level interoperability and the interoperability of component-level implementations (applications software) as well.

Interoperability considerations include networking protocols, data interchange formats, distributed file systems, legacy support, and device interface. Each EEI category is concerned with addressing one or more of these considerations. The following summarizes the NGC approach to interoperability.

Networking protocols are defined in open standards such as IEEE 802.3 CSMA/CD, 802.4 token-passing bus, 802.5 token-passing ring, etc. These standards are supported by NGC profile implementations using de facto standards and products such as Ethernet and ARCNET, as well as the commonly used TCP/IP internet protocol found on many UNIX-based operating systems.

Standard data interchange formats tackle the problem of handling heterogeneous data, such as differences in byte ordering, data formats, or the padding of data items in heterogeneous host environments. The IEEE floating-point standard and OSF's DCE marshaling capability are examples in this area. Standard product data formats such as PDES/STEP are also key.

Distributed file systems support file exchange or shared file access in distributed environments requiring such access. Applications range from transferring files across a cell-level interface to the transfer of legacy IGES or RS274 files across a CAD/CAM network or RS-232 interface.

In the area of device interface, there is a wealth of de facto and de jure standards that deal with device interoperability.

Tables 3.3.2.2-1 through 3.3.2.2-5 list the EEI standards by category. The list of standards for EEI can be quite large, especially in the area of ISO/OSI standards which are not enumerated here for that reason. However, it must be remembered that subsets of these standards are selected through NGC profiling and that most of these standards are already accommodated with the use of commercial products.

There are some obvious overlaps of certain API standards within the EEI domain. For example, X Windows standards not only specify the X library and Xt intrinsics APIs, but also define an X protocol for client-server interoperability between the client application and an X terminal. Therefore, the X Windows standard also appears under the EEI branch of the taxonomy. In the area of data management, SQL is another protocol standard that has a corresponding API standard.

Even within the EEI categories there are functional overlaps. No matter what view is taken, there will probably be the potential for overlap. CAN bus, for example, may be classified as having fieldbus characteristics, yet may also be used as a remote keyboard interface. The primary focus for the NGC is on relevant standards for profiling. The NGC taxonomy organizes standards to facilitate the profiling process. A statement need only be supplied for use of a standard outside of its originally intended scope.

Figures 3.3.2.2-1 and 3.3.2.2-3 correlate some of the EEI standards into a more physical view of the NGC system context. At the highest level, the NGC has only three major physical interface types: a human-user interface, a cell or CAD/CAM interface, and a plethora of device interfaces. Relative to these physical interfaces, a number of standards are mapped that define protocols and/or data formats. Data format standards are grouped under the process heading in the diagram. These data standards could apply to all major physical interfaces.

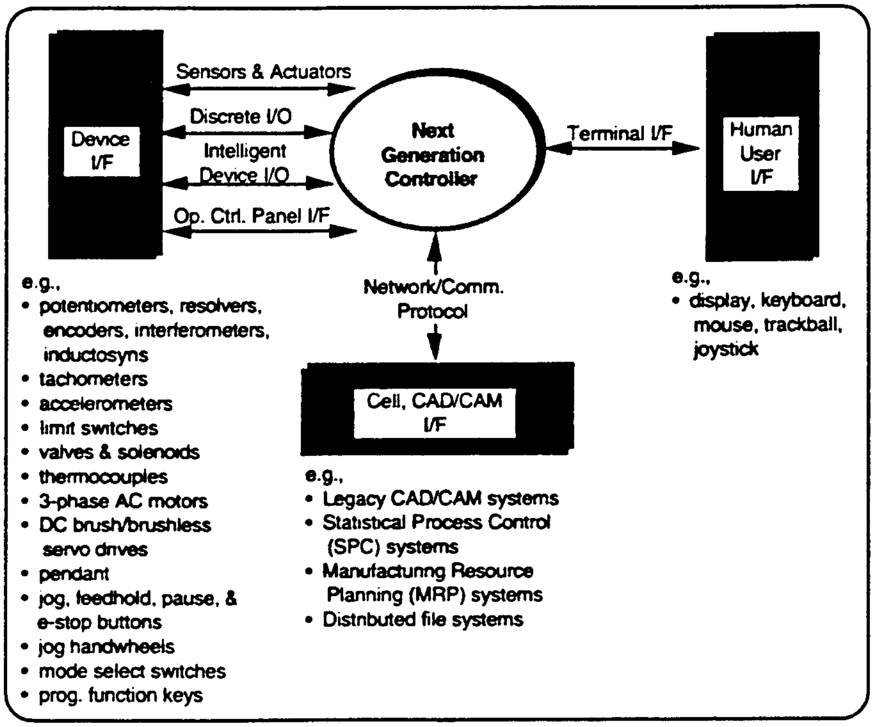


Figure 3.3.2.2-1. NGC EEI System Context—Physical View

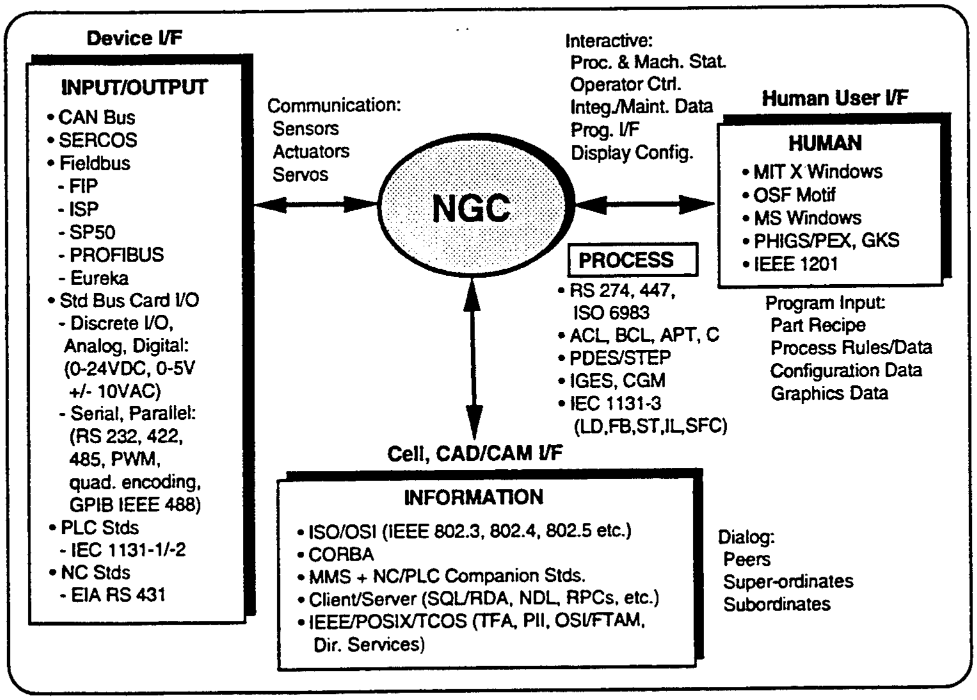


Figure 3.3.2.2-2. NGC EEI Interoperability Standards—Physical Mapping

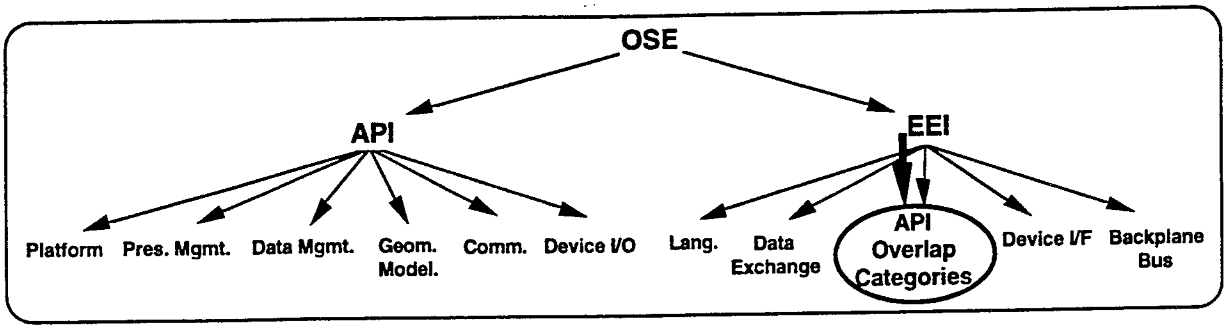


Table 3.3.2.2-1. EEI Relevant Open Standards—Overlapping API Categories

Presentation Mgmt EEI	
MIT X Windows (X11R5)	MIT X Window System Version 11, Release 5
MIT X Windows PEX extension	MIT X Window System PHIGS EXtension
OSF/Motif 1.2	Open Software Foundation - Motif
IEEE 1201.1	Uniform Applications Program Interface Graphical User, Rev. 3, DS01719 (Windowing Toolkit API)
ANSI X3.124-1985 (R1991)	Graphical Kernel System (GKS) Functional Description (includes ANSI X3.124.1-1985)
FIPSPUB120-1	Graphical Kernel System (GKS) 91 Jan 08
ISO 7942:1985	Information Processing Systems - Computer Graphics - Graphical Kernel System (GKS) functional description
ISO 8805:1988	Information Processing Systems - Computer Graphics - Graphical Kernel System for Three Dimensions (GKS-3D) functional description
ANSI/ISO 9592.1-1989	Information Processing Systems - Computer Graphics - Programmer's Hierarchical Interactive Graphics System (PHIGS) - Part 1: Functional Description
FIPSPUB153	Programmer's Hierarchical Interactive Graphics System 88 Oct 14
ISO/IEC 9592-1:1989	Information Processing Systems - Computer Graphics - Programmer's Hierarchical Interactive Graphics System (PHIGS) - Part 1: Functional Description
Data Management EEI	
IEEE 1003.8 D6	POSIX--Part 1: Network-Transparent File Access
ISO 8907:1987	Information Processing Systems - Database Languages - NDL
ANSI X3.133-1986	Database Language - NDL
FIPSPUB126	Database Language NDL 87 Mar 10
ISO/IEC 9075:1992	Information Technology - Database Languages - SQL
ISO/IEC 9579-1	Information Technology - Database Languages - Remote Data Access - Part 1: Generic Model, Service and Protocol
ANSI X3.135-1989	Information Systems - Database Language - SQL with Integrity Enhancement
ANSI X3.168-1989	Information Systems - Database Language - Embedded SQL
FIPSPUB127-1	Database Language SQL
X/Open C201	Structured Query Language (SQL)
X/Open P205	SQL Remote Database Access
X/Open J301	RDA Mapping for TCP/IP
Geometric Modeling EEI	
CAM-I AIS 2.0 Draft Standard Vol I	CAM-I Application Interface Specification (AIS) Revised 1991 Document Number R-90-PM-03 AIS 2.0 Functional Specification

Table 3.3.2.2-1 (Continued)

Communications EEI	
IEEE 1003.12 D1.2	POSIX - Protocol Independent Interface
IEEE 1003.17 D4	POSIX Directory Service API
ISO 7498:1984	Information Processing Systems - Open Systems Interconnect - Basic Ref. Model
X/Open T904	X/Open Portability Guide 4 (C202, C203, C204, G204)
X/Open G212	X/Open Distributed Computing Services (XDCS) Framework
X/Open P210	The Common Object Request Broker: Architecture and Specification - X/Open in conjunction with Object Management Group (OMG)
OSF DCE 1.0.2	Open Software Foundation Distributed Computing Environment Version 1.0.2
X/Open C210	Protocols for X/Open Interworking: XNFS, Issue 4
ANSI/EIA 511-1989	Manufacturing Message Specification - Service Definition and Protocol
ISO/IEC 9506-1:1990	Industrial automation systems - Manufacturing Message Specification - Part 1: Service Definition
ISO/IEC 9506-2:1990	Industrial automation systems - Manufacturing Message Specification - Part 2: Protocol Specification
ISO/IEC 9506-4:1992	Industrial automation systems - Manufacturing Message Specification - Part 4: Companion Standard for Numerical Control
ISO/IEC 9506-5:1990 (IEC 65A/ 65B(Secretariat)111/138 Sept. '90)	Industrial automation systems - Manufacturing Message Specification - Part 5: Companion Standard for Programmable Controllers
IEC DIS 1131-5 (IEC 1131-5 PC Comm. Draft 1/15/93)	Programmable Controllers - Part 5: PC Communications

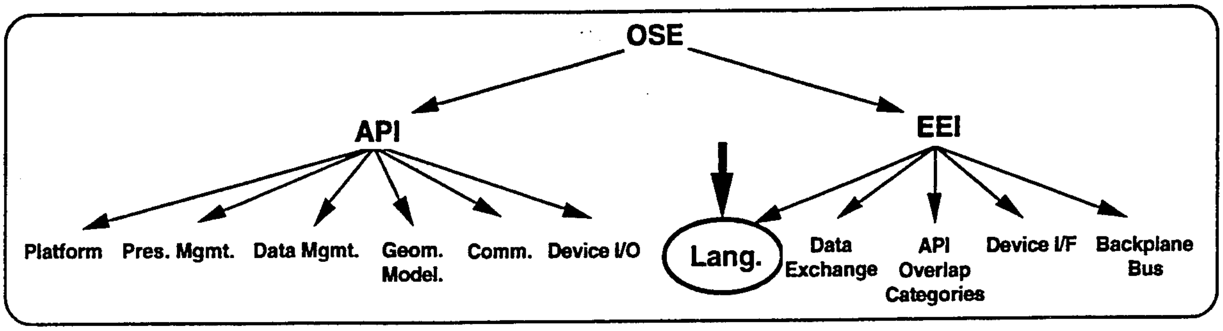


Table 3.3.2.2-2. Language EEI—Relevant Open Standards

ISO/IEC 9899:1990	Programming Languages C
FIPSPUB160	C 91 Mar 13
ANSI X3.37-1987	Programming Language APT
ANSI/EIA 494-B-1992	32-Bit Binary (BCL) and 7-Bit ASCII (ACL) Input Format for NCM
ANSI/EIA 274-D-1980 (1988)	Interchangeable Variable Block Data Format for Positioning, Contouring, and Contouring/Positioning Numerically Controlled Machines
ANSI/EIA 267-C-1990	Axis and Motion Nomenclature for Numerically Controlled Machines
ISO 841:1974	Numerical control of machines - Axis and motion nomenclature
IEC 1131-3 (1993)	Programmable controllers - Part 3: Programming languages
ISO 3592:1978	Numerical control of machines - NC processor output - Logical structure (and major words)
ISO 4342:1985	Numerical control of machines - NC processor input - Basic part program reference language
ISO 4343:1978	Numerical control of machines - NC processor output - Minor elements of 2000-type records (post-processor commands)
ISO 6983-1:1982	Numerical control of machines - Program format and definition of address words - Part 1: Data format for positioning, line motion and contouring control systems

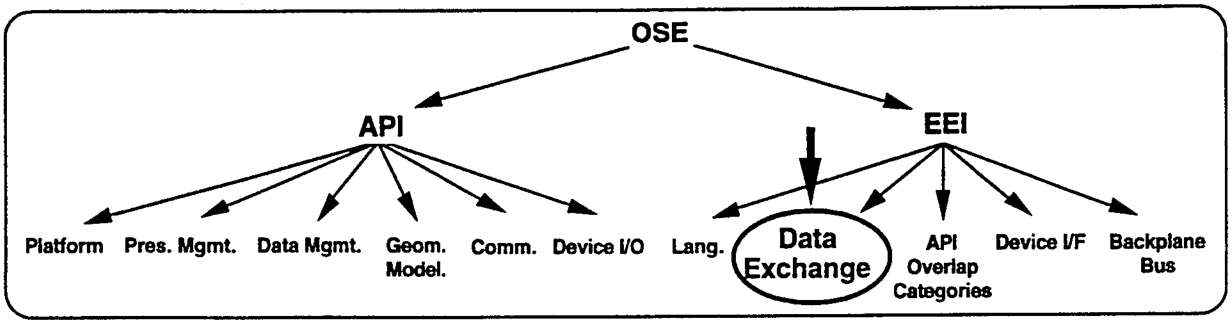


Table 3.3.2.2-3. Data Exchange EEI—Relevant Open Standards

IGES 5.1	Initial Graphics Exchange Specification 5.1, October 1991 (US PRO P15.1)
ANSI/ASME Y14.26M-1989	Digital Representation for Communication of Product Definition Data (IGES 4.0)
FIPSPUB177	Initial Graphics Exchange Specification (IGES) 92 November 30
ISO 10303 Initial Release	Standard for the Exchange of Product Model Data (STEP) - Product Data Representation and Exchange (US PRO P10303.IR - incl. parts 1, 11, 21, 31, 41 - 44, 46, 101, 201, and 203)
ANSI X3.4-1986 (R1992)	Coded Character Set - 7-Bit American National Standard Code for Information Interchange (ASCII)
ANSI X3.41-1990	Code Extension Techniques for Use with the 7-byte Coded Character Set of ASCII
ANSI X3.42-1990	Representation of Numeric Values in Character Strings for Information Interchange
ANSI X3.64-1979 (R1990)	Additional Controls for Use with the ASCII
FIPSPUB1-2	Code for Information Interchange, its Reps., Subsets, and Extensions 84 Nov 14
FIPSPUB86	Additional Controls for Use with the ASCII 81 Jan 29
ISO/IEC 646:1991	ISO 7-bit coded character set for information interchange
ISO 2022:1986	Information processing - ISO 7-bit and 8-bit coded character sets - Coded extension techniques
ISO 6093:1985	Information processing - Representation of numerical values in character strings for information interchange
ISO/IEC 6429:1992	Information technology - Control functions for coded character sets
ANSI/EIA 227-A-1978 (1988)	One-inch Perforated Tape
ANSI/EIA 358-B-1980 (R1990)	Subset of ASCII for Numerical Machine Control Perforated Tape
FIPSPUB2-1	Perforated Tape Code for Information Interchange 84 Nov 14
FIPSPUB26	One-inch Perforated Paper Tape for Information Interchange 73 June 30
ANSI/ISO 8632-1990	Information Processing Systems - Computer Graphics - Metafile for the Storage and Transfer of Picture Description Information (rev. & redesig. of X3.122-1986)
FIPSPUB128-1	Computer Graphic Metafile, 93 May 11
ISO/IEC 8632-1/2/3/4	Information Technology - Computer Graphics - Metafile for the storage and transfer of picture description information - Part 1: Functional Specification, Part 2: Character encoding, Part 3: Binary encoding, Part 4: Clear text encoding
ANSI/EIA-548-1988	Electronic Design Interchange Format (EDIF)
FIPSPUB161	Electronic Design Interchange (EDI) 91 Mar 29

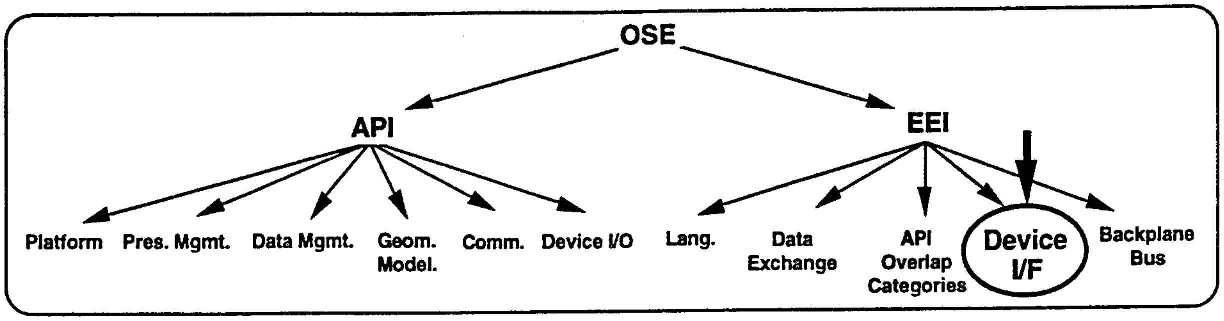


Table 3.3.2.2-4. Device Interface EEI—Relevant Open Standards

ANSI/EIA/TIA 232-E-1991	Interface between Data Terminal Equipment and Data Circuit-Terminating Equipment Employing Serial Binary Data Interchange
EIA 485	Standard for Electrical Characteristics of Generators and Receivers for Use in Balanced Digital Multipoint Systems
ANSI/IEEE 488.1-1987	Digital Interface for Programmable Instrumentation
ANSI/IEEE 488.2-1987	ICodes, Formats, Protocols, and Common Commands
ANSI X3.183-1991	Information Systems - High Performance Parallel Interface
ANSI X3.131-1986	Small Computer Systems Interface (SCSI)
ISO 9316:1989	Information processing systems - Small Computer System Interface (SCSI)
SAE J1583	Controller Area Network (CAN) An In-vehicle Serial Communications Protocol
IEC 44(secretariat)148	Serial Data Link for Real Time Communication Between Controls and Drives (SERCOS)
ISA-S50.02-1992	Fieldbus Standard for Use In Industrial Controls Systems - Part 2: Physical Layer Specification and Service Definition
ANSI/EIA 431-1992	Electrical Interface Between a Numerical Control and Machine Tools
IEC 1131-1/2 (1992)	Programmable Controllers. Part 1: General information, Part 2: Equipment requirements and tests
EIA 441-1979 (R1992)	Operator Interface Functions of Numerical Controls
ISO 4336:1981	Numerical control of machines - Specification of interface signals
OBIOS 1.15	OBIOS: Open Basic Input Output System Draft Specification - Real-Time Consortium

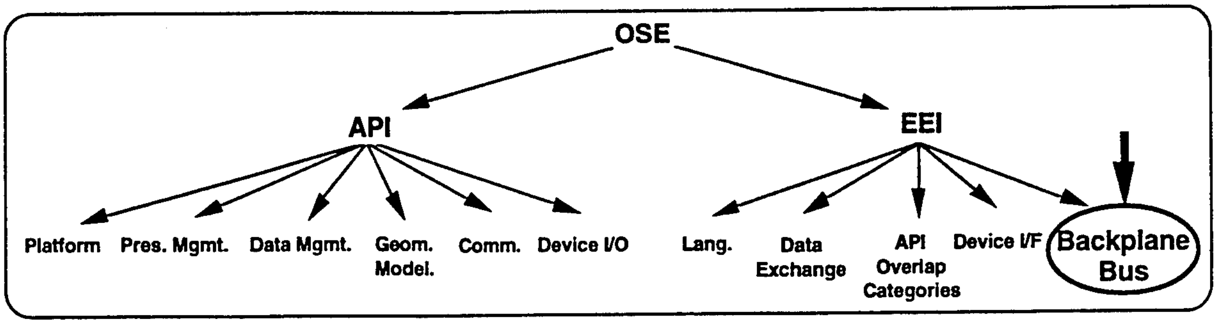


Table 3.3.2.2-5. Backplane Bus EEI—Relevant Open Standards

ANSI/IEEE 896.1-1992	Backplane Bus Specification for Multiprocessor Architectures: Futurebus+, Logical Layer Specification
ANSI/IEEE 1296-1987	High performance Synchronous 32-Bit Bus: Multibus II
IEC 796-1/2/3	Microprocessor System Bus - 8-bit and 16-bit data (MULTIBUS I) Parts 1-3:
ANSI/IEEE 1196-1987	NuBus - A simple 32-Bit Backplane Bus
ANSI/IEEE 959-1988	Specification for an I/O Expansion Bus: SBX Bus
ANSI/IEEE 961-1987	8-bit Microcomputer Bus System (STD Bus)
ANSI/IEEE 1000-1987	Specification for a Standard 8-Bit Backplane Interface (STE Bus)
IEC 821 (1991)	VMEBus - Microprocessor system bus for 1 byte to 4 byte data
IEC 822 (1988)	VSB - Parallel Subsystem Bus of the IEC 821 VME bus
IEC 823 (1990)	Microprocessor System Bus (VMSbus) - Serial Subsystem bus of the IEC 821 bus
ANSI/IEEE 1014-1987	Versatile Backplane Bus: VMEbus
ANSI/IEEE 1096-1988	Multiplexed High-Performance Bus Structure (VSB)
IEEE P1496	SBus
IEEE P1754	SPARC

3.3.3 Profiles

The NGC standard supports varying levels of vendor-neutral open system conformance through the concept of profiles. Simply stated, a profile specifies a set of de jure and de facto standards that are adhered in the implementation of a specific controller. Thus, a profile characterizes a controller by defining its unique level of openness and functional capability.

Profiles may include all the open standards specified in the OSE taxonomy. In addition, profiles tie into accepted industry practice with provisions for including "de facto" standards. De facto standards, especially those that are commonly used in controller products today, can not be ignored. These standards are not always considered "open" in the purest sense. Many de facto standards are proprietary and they all lack formal specification by a recognized standards body. However, some de facto standards have had a tremendous impact on industry. One example of this is the PC ISA backplane bus which not only has gained wide popularity in the personal computer industry but is also growing in use in industrial controls applications.

Profiles provide a mechanism for "apples-to-apples" comparison of NGC applications software and platform environments to assess their relative compatibility. Through the use of profiles, the selection of controller software for execution on a specific platform implementation is a simple matter of comparing profiled capabilities. Compatible implementations may be interchanged since they support equivalent levels of system interoperability, component interchangeability, and portability. These "-abilities", facilitated through the use of profiles, enable the realization of the major benefits of NGC.

It is therefore the application of profiling that facilitates a new generation of controllers. These NGC standards-based controllers are distinguished by their relative ease in accommodating:

(1) new features,

(2) scaleable and upgradeable features,

(3) interchangeable controller components,

(4) portability of components to different platform implementations,

(5) adaptable interfaces to a variety of hardware devices,

(6) flexible peer and factory-level communications and networks,

(7) enhancements to performance, and

(8) consistent and tailorable user interfaces.

NGC profiles support both existing and emerging open standards. Many POSIX draft standards are referenced where mature standards are unavailable. As described earlier, de facto public and proprietary standards are also adopted where current market trends support such inclusion. Some NGC implementations will inevitably evolve based on profiles that specify only vendor-neutral open standards. It is hoped that completely vendor-neutral profiles will ultimately dominate the marketplace. NGC conformance implies the adoption of one or more NGC profiles in a controller implementation (see Section 3.5, Conformance).

NGC relevant standards are grouped into dimensions of similar purpose or functionality, as defined by the Open Systems Environment (OSE) taxonomy. For example, the Backplane Bus hardware dimension includes Multibus I/II, VME, Futurebus+, STD Bus, and NuBus. Backplane Bus profiles also include proprietary standards that are widely used, e.g., Industry Standard Architecture (ISA) and Extended Industry Standard Architecture (EISA) bus, when popularity of these standards merit their inclusion as de facto standards. Selection of one or more standards from each of the profile dimensions constitutes a specific profile. For example, the Backplane Bus dimension of an implementation profile can include the selection of both the VMEBus and the EISA bus standards.

Figure 3.3.3-1 illustrates the full set of NGC profile templates and their hierarchical relationship. The template organizational structure is strongly influenced by the OSE taxonomy with some subtle differences. One important difference is the introduction of de facto standards that are not identified in the OSE standards taxonomy.

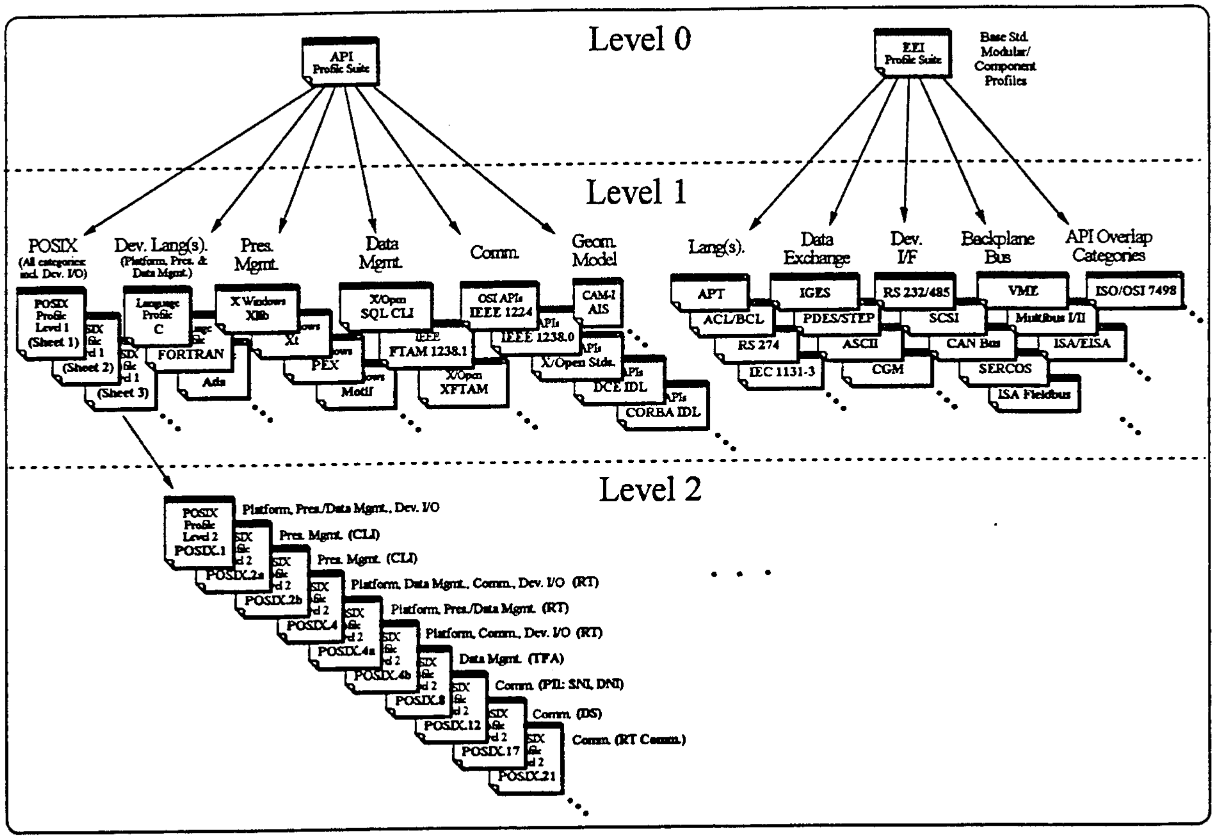


Figure 3.3.3-1 NGC Profiles—Template Structure

Standards "levels" in the profile template structure is another departure from the OSE taxonomy organization. Template levels provide the obvious benefit of helping to organize and represent the tremendous amount of profile information on limited template space. However, this is only part of the rationale for specifying the POSIX profile templates at two levels of detail. Level 1 supports profiling of POSIX API options at a feature or API group level. This higher level satisfies the operating system developer's need to represent the compiler options used for a specific operating system product. Compiler options are essentially packaging/bundling options for tailored products that meet specific applications requirements and targeted hardware environments. Level 2 supports profiling of POSIX at an individual API level so that an applications developer may accurately reflect the service requirements of his controller component by reference to specific API calls.

A complete specification of all the profile templates represented by Figure 3.3.3-1 is beyond the scope of this specification. Figures 3.3.3-2 through 3.3.3-9, located at the end of this section, are sample profile templates used in defining implementation profiles. As the template headings suggest, the templates are dual-use and may be used to profile either a specific NGC platform or a specific NGC software application component or aggregate. To define a specific profile, one needs only to check mark the boxes corresponding to the desired template items.

Selection of some items may either preclude or mandate the inclusion of other items in a profile. The real-time (RT) or non-RT nature of the specific platform or application profile may also have a potential impact on the applicability of certain template items. (Note that hard-RT applications are not profiled since such applications are typically embedded or built-in hardware and fall outside the scope of general platform and applications portability issues.)

The profiling process described, and the impact of one profile selection on the use of another template, is best illustrated by example. The POSIX.13/D5 draft is a "Standardized Application Environment Profile--POSIX Realtime Application Support (AEP)". This draft specifies four POSIX RT profiles: MINimal embedded, CONtroller, DEDicated, and MULtipurpose. These profiles appear as selection items on the API Profile Suite template (Figure 3.3.3-2, upper right-hand corner). In specifying a platform profile, selection of one of these items negates the need to fill out the POSIX Profile Level 1 template (Figure 3.3.3-3) and the relevant POSIX Level 2 templates (Figures 3.3.3-4 thru 3.3.3-7) since these items are already predefined by the RT AEP selection. However, RT POSIX custom platforms, or extensions to one of the POSIX RT AEPs, will require the use of all the relevant POSIX profile templates.

The POSIX profiling example above illustrates the benefits to be gained from a user-friendly, interactive set of tools for profiling. Intelligent profiling tools can simplify the profiling task so that an NGC developer is presented with only the applicable options based on his previous profiling selections. These tools do not exist today. The availability of automated tools for NGC development and integration is considered a key enabling technology for bringing the NGC concept to a practical reality.

It is interesting to note that POSIX.13/D5 is based directly on existing small and/or RT (typically non-UNIX) kernel practice. Table 1-1 of the POSIX.13 draft specification offers a comparison of five products (VRTX32, pSOS, C Exec, MTOS, and VxWorks) based on small RT kernel features. The general approach of the standard specifies POSIX interfaces sufficient to deliver the functionality typical of current RT systems.

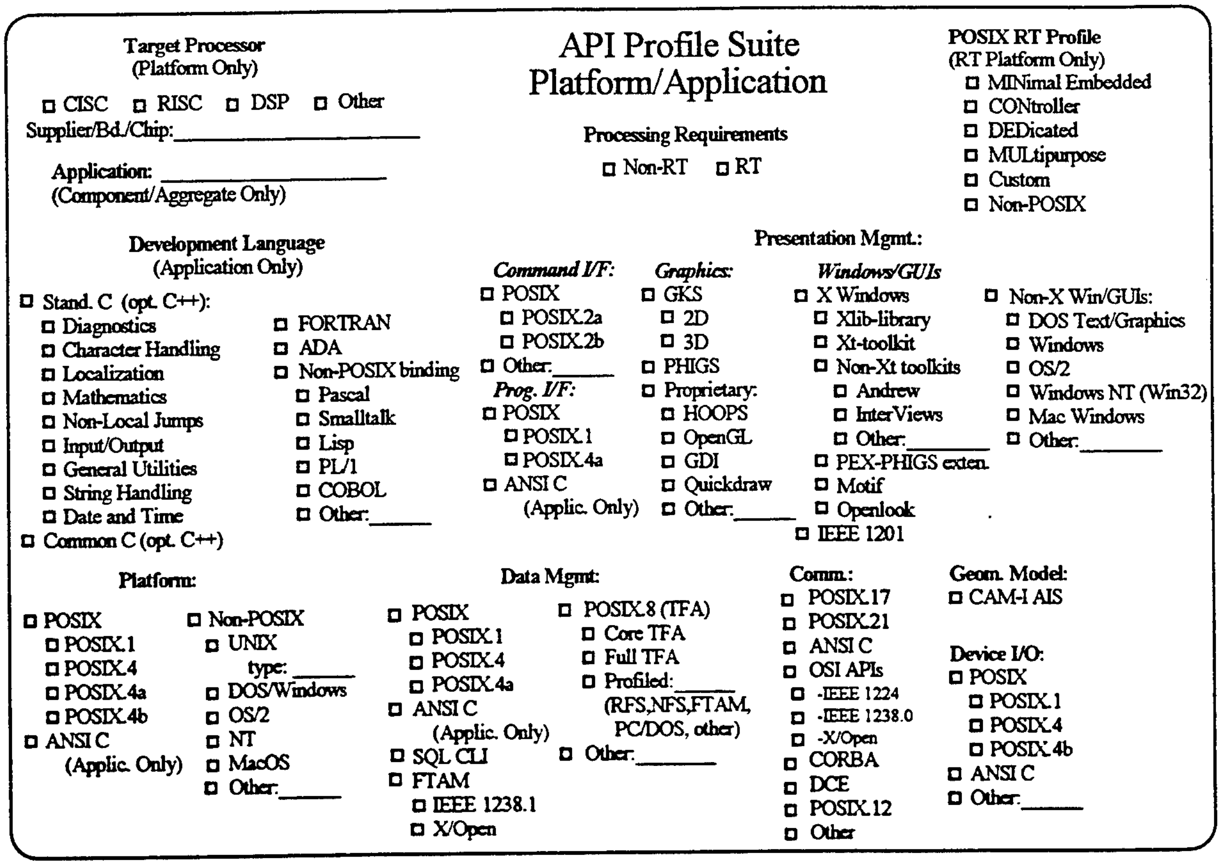
Ongoing and future SOSAS work involves the tailoring and refinement of the profile suites. Standard AEPs that are more encompassing than the POSIX RT AEPs, that span all OSE dimensions, should evolve based on market-driven factors. It is premature (and unnecessary) to anticipate or guess at what these standard AEPs should be. Documented profiles and the benefit of time will illuminate the demand for certain profiles. Additional work is also required to identify component profiles or NGC-relevant subsets of the base standards that are selectable items in the profile suites. As with AEP definition, component profiling must also be sensitive to legacy and the projected product market.

Also related, but of a lower priority, is the feasibility of specifying a government/military AEP based on the Federal Information Processing Standards (FIPS) procurement standards (made available through the National Technical Information Service). Many of the PUBS documents refer to open standards that are identified in the NGC OSE. Each of these documents contain a "Specifications" section that is essentially a component profile of the referenced base standard. All additions to, modifications of, and deletions to the referenced standard are specified.

It is obvious that the POSIX standard has been leveraged to a large degree for its value-added standards in addition to its contribution to the fundamental philosophies of NGC and the OSE reference model. The adoption of POSIX.1 implies the adoption of the C language. Less obvious, perhaps, is that the adoption of ANSI C implies the adoption of POSIX.1. This connection to POSIX via the standard ANSI C library is often overlooked. It implies that much of the legacy software developed using the C language may be leveraged with little or no modification for POSIX compliance. POSIX-conformance is violated only if source-level API system calls exist within the C source code that fall outside of the POSIX specification. Therefore, when defining profiles for legacy C code, one should not be so quick to rule out its compliance to POSIX.

The IEEE/TCOS committee is also currently developing language-independent specification (LIS) versions of the POSIX standards. The LIS version of POSIX.1 is intended to replace the current C language-dependent version of the specification. FORTRAN and Ada language bindings to the POSIX.1 standard are also at various stages of draft release. As these standards mature and there is evidence of supporting products, they need to be integrated into the NGC OSE profile strategy to fill some the present language gaps that exist with the current NGC platform and POSIX approach.

Profiling offers complete flexibility in design to the developer. Many platform profiles will evolve by leveraging existing COTS products. It is possible to implement each profile dimension with a different product. Even the adherence to POSIX is optionally selectable by dimension. Little profiling constraints are placed on the developer other than to specify his unique profile. End user requirements and market demand will establish de facto standard profiles that will have the most impact on the next generation of controllers.



API Profile Suite
Platform/Application

Target Processor
(Platform Only)

□ CISC □ RISC □ DSP □ Other

Supplier/Bd./Chip:____________________

Application: ____________________
(Component/Aggregate Only)

Processing Requirements

□ Non-RT □ RT

POSIX RT Profile
(RT Platform Only)

□ MINimal Embedded
□ CONtroller
□ DEDicated
□ MULtipurpose
□ Custom
□ Non-POSIX

Development Language
(Application Only)

□ Stand. C (opt. C++):
- □ Diagnostics
- □ Character Handling
- □ Localization
- □ Mathematics
- □ Non-Local Jumps
- □ Input/Output
- □ General Utilities
- □ String Handling
- □ Date and Time

□ Common C (opt. C++)

□ FORTRAN
□ ADA
□ Non-POSIX binding
- □ Pascal
- □ Smalltalk
- □ Lisp
- □ PL/1
- □ COBOL
- □ Other:_____

Presentation Mgmt.:

Command I/F:
□ POSIX
- □ POSIX.2a
- □ POSIX.2b

□ Other:_____

Prog. I/F:
□ POSIX
- □ POSIX.1
- □ POSIX.4a

□ ANSI C
(Applic. Only)

Graphics:
□ GKS
- □ 2D
- □ 3D

□ PHIGS
□ Proprietary:
- □ HOOPS
- □ OpenGL
- □ GDI
- □ Quickdraw
- □ Other:_____

Windows/GUIs
□ X Windows
- □ Xlib-library
- □ Xt-toolkit
- □ Non-Xt toolkits
  - □ Andrew
  - □ InterViews
  - □ Other:_______
- □ PEX-PHIGS exten.
- □ Motif
- □ Openlook

□ IEEE 1201

□ Non-X Win/GUIs:
- □ DOS Text/Graphics
- □ Windows
- □ OS/2
- □ Windows NT (Win32)
- □ Mac Windows
- □ Other:_______

Platform:

□ POSIX
- □ POSIX.1
- □ POSIX.4
- □ POSIX.4a
- □ POSIX.4b

□ ANSI C
(Applic. Only)

□ Non-POSIX
- □ UNIX
  type: _____
- □ DOS/Windows
- □ OS/2
- □ NT
- □ MacOS
- □ Other:_____

Data Mgmt:

□ POSIX
- □ POSIX.1
- □ POSIX.4
- □ POSIX.4a

□ ANSI C
(Applic. Only)

□ SQL CLI
□ FTAM
- □ IEEE 1238.1
- □ X/Open

□ POSIX.8 (TFA)
- □ Core TFA
- □ Full TFA
- □ Profiled:_____
  (RFS,NFS,FTAM,
  PC/DOS, other)

□ Other:_______

Comm.:

□ POSIX.17
□ POSIX.21
□ ANSI C
□ OSI APIs
- □ -IEEE 1224
- □ -IEEE 1238.0
- □ -X/Open

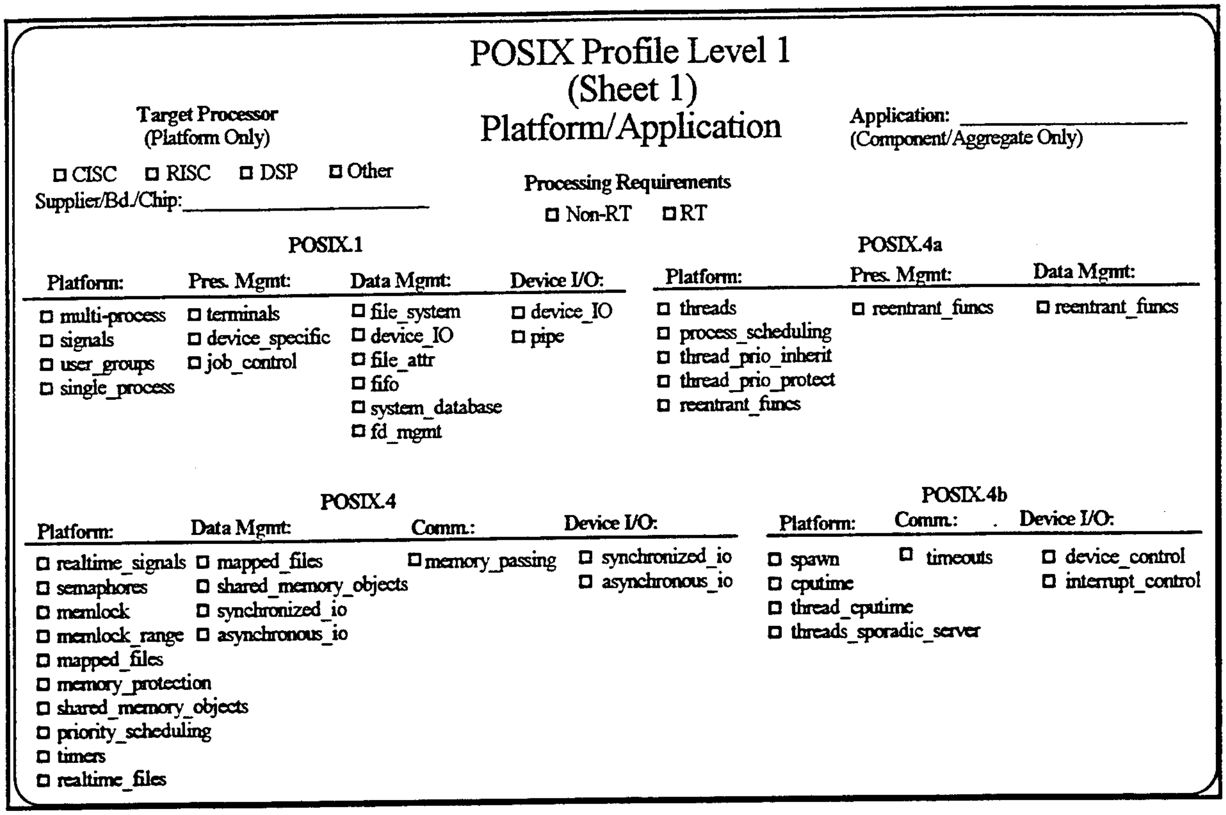
□ CORBA
□ DCE
□ POSIX.12
□ Other

Geom. Model:
□ CAM-I AIS

Device I/O:
□ POSIX
- □ POSIX.1
- □ POSIX.4
- □ POSIX.4b

□ ANSI C
□ Other:_____

Figure 3.3.3-2 API Profile Suite



POSIX Profile Level 1
(Sheet 1)
Platform/Application

Target Processor
(Platform Only)

□ CISC □ RISC □ DSP □ Other

Supplier/Bd./Chip:____________________

Processing Requirements

□ Non-RT □ RT

Application: ____________________
(Component/Aggregate Only)

POSIX.1

Platform:
- □ multi-process
- □ signals
- □ user_groups
- □ single_process

Pres. Mgmt:
- □ terminals
- □ device_specific
- □ job_control

Data Mgmt:
- □ file_system
- □ device_IO
- □ file_attr
- □ fifo
- □ system_database
- □ fd_mgmt

Device I/O:
- □ device_IO
- □ pipe

POSIX.4a

Platform:
- □ threads
- □ process_scheduling
- □ thread_prio_inherit
- □ thread_prio_protect
- □ reentrant_funcs

Pres. Mgmt:
- □ reentrant_funcs

Data Mgmt:
- □ reentrant_funcs

POSIX.4

Platform:
- □ realtime_signals
- □ semaphores
- □ memlock
- □ memlock_range
- □ mapped_files
- □ memory_protection
- □ shared_memory_objects
- □ priority_scheduling
- □ timers
- □ realtime_files

Data Mgmt:
- □ mapped_files
- □ shared_memory_objects
- □ synchronized_io
- □ asynchronous_io

Comm.:
- □ memory_passing

Device I/O:
- □ synchronized_io
- □ asynchronous_io

POSIX.4b

Platform:
- □ spawn
- □ cputime
- □ thread_cputime
- □ threads_sporadic_server

Comm.:
- □ timeouts

Device I/O:
- □ device_control
- □ interrupt_control

Figure 3.3.3-3 POSIX Profile Level 1

POSIX.1 Profile Level 2
Platform/Application

Target Processor
(Platform Only)

□ CISC □ RISC □ DSP □Other

Supplier/Bd./Chip:____________________

Processing Requirements

□ Non-RT □ RT

Application: ____________________
(Component/Aggregate Only)

□ single_process

□ uname()
□ sysconf()
□ main()

□ multi_process

□ execl()
□ execv()
□ execle()
□ execve()
□ execlp()
□ execvp()
□ getpid()
□ getppid()
□ times()
□ fork()
□ wait()
□ waitpid()
□ _exit()
□ time()
□ getenv()
□ sleep()
□ alarm()
□ pause()

□ job_control

□ setpgid()
□ tcgetpgrp()
□ tcsetpgrp()

□ signals

□ sigemptyset()
□ sigfillset()
□ sigaddset()
□ sigdelset()
□ sigismember()
□ kill()
□ sigaction()
□ sigprocmask()
□ sigpending()
□ sigsuspend()

□ user_groups

□ getuid()
□ geteuid()
□ getgid()
□ getegid()
□ setuid()
□ setsid()
□ setgid()
□ getgroups()
□ getlogin()
□ cuserid()
□ getpgrp()

□ terminals

□ ctermid()
□ ttyname()
□ isatty()

□ file_system

□ opendir()
□ readdir()
□ rewinddir()
□ closedir()
□ chdir()
□ getcwd()
□ mkdir()
□ creat()
□ unlink()
□ rmdir()
□ rename()
□ stat()
□ fstat()
□ access()
□ utime()
□ pathconf()
□ fpathconf()
□ link()

□ file_attr

□ umask()
□ chmod()
□ chown()

□ fd_mgmt

□ dup()
□ dup2()
□ lseek()
□ fcntl()

□ device_IO

□ open()
□ close()
□ read()
□ write()

□ device_specific

□ cfgetispeed()
□ cfgetospeed()
□ cfsetispeed()
□ cfsetospeed()
□ tcgetattr()
□ tcsetattr()
□ tcsendbreak()
□ tcdrain()
□ tcflush()
□ tcflow()

□ system_database

□ getgrgid()
□ getgrnam()
□ getpwuid()
□ getpwnam()

□ pipe

□ pipe()

□ FIFO

□ mkfifo()

Figure 3.3.3-4 POSIX.1 Profile Level 2

POSIX.4 Profile Level 2
Platform/Application

Target Processor
(Platform Only)

□ CISC □ RISC □ DSP □ Other

Supplier/Bd./Chip:____________________

Processing Requirements

□ Non-RT □ RT

Application: ____________________
(Component/Aggregate Only)

□ realtime_signals

□ sigwaitrt()
□ sigtimedwait()
□ sigqueue()

□ semaphores

□ sem_init()
□ sem_destroy()
□ sem_open()
□ sem_close()
□ sem_unlink()
□ sem_wait()
□ sem_trywait()
□ sem_post()
□ sem_getvalue()

□ memlock

□ mlockall()
□ munlockall()

□ memlock_range

□ mlock()
□ munlock()

□ mapped_files

□ ftruncate()
□ mmap()
□ munmap()

□ fsync

□ fsync()

□ shared_memory_objects

□ shm_open()
□ shm_unlink()

□ memory_protection

□ mprotect()

□ priority_scheduling

□ sched_setparam()
□ sched_getparam()
□ sched_setscheduler()
□ sched_getscheduler()
□ sched_yield()
□ sched_get_priority_max()
□ sched_get_priority_min()
□ sched_rr_get_interval()

□ timers

□ clock_settime()
□ clock_gettime()
□ clock_getres()
□ timer_create()
□ timer_delete()
□ timer_settime()
□ timer_gettime()
□ timer_getoverrun()
□ nanosleep()

□ realtime_files

□ rf_create()
□ rf_getattr()
□ rf_setattr()
□ rf_getalloccap()
□ rf_getcachecap()
□ rf_getbiocap()
□ rf_getaiocap()
□ rf_getdiocap()
□ rf_getallocincr()
□ rf_getincr()
□ rf_getbuf()
□ rf_freebuf()

□ synchronized_io

□ fdatasync()
□ msync()

□ asynchronous_io

□ aio_read()
□ aio_write
□ aio_listio()
□ aio_error()
□ aio_return()
□ aio_cancel()
□ aio_suspend()
□ aio_fsync()

□ message_passing

□ mq_open()
□ mq_close()
□ mq_unlink()
□ mq_send()
□ mq_receive()
□ mq_notify()
□ mq_setattr()
□ mq_getattr()

Figure 3.3.3-5 POSIX.4 Profile Level 2

POSIX.4a Profile Level 2
Platform/Application

Target Processor
(Platform Only)

□ CISC □ RISC □ DSP □ Other
Supplier/Bd./Chip:____________________

Processing Requirements
□ Non-RT □ RT

Application: ____________________
(Component/Aggregate Only)

□ threads

□ pthread_attr_init()
□ pthread_attr_destroy()
□ pthread_attr_setstacksize()
□ pthread_attr_getstacksize()
□ pthread_attr_setdetachstate()

□ pthread_self()
□ pthread_equal()
□ pthread_once()
□ pthread_mutexattr_init()
□ pthread_mutexattr_destroy()

□ pthread_mutex_trylock()
□ pthread_mutex_unlock()
□ pthread_condattr_init()
□ pthread_condattr_destroy()
□ pthread_condattr_getpshared()

□ pthread_cond_wait()
□ pthread_cond_timedwait()
□ pthread_key_create()
□ pthread_setspecific()
□ pthread_getspecific()

□ pthread_attr_getsched()
□ ptherad_attr_setprio()
□ pthread_attr_getprio()
□ pthread_getschedattr()
□ pthread_setschedattr()

□ pthread_attr_getdetachstate()
□ pthread_create()
□ pthread_join()
□ pthread_detach()
□ pthread_exit()

□ pthread_mutexattr_getpshared()
□ pthread_mutexattr_setpshared()
□ pthread_mutex_init()
□ pthread_mutex_destroy()
□ pthread_mutex_lock()

□ pthread_condattr_setpshared()
□ pthread_cond_init()
□ pthread_cond_destroy()
□ pthread_cond_signal()
□ pthread_cond_broadcast()

□ pthread_attr_setscope()
□ pthread_attr_getscope()
□ pthread_attr_setinheritsched()
□ pthread_attr_getinheritsched()
□ pthread_attr_setsched()

□ pthread_yield()
□ sigwait()
□ pthread_kill()
□ pthread_cancel()
□ pthread_setintr()

□ pthread_setintrtype()
□ pthread_testintr()
□ pthread_cleanup_push()
□ pthread_cleanup_pop()

□ process_scheduling
- □ setprio()
- □ getprio()
- □ setscheduler()
- □ getscheduler()
- □ yield()

□ thread_prio_inherit
- □ pthread_mutexattr_setprotocol()
- □ pthread_mutexattr_getprotocol()
- □ pthread_mutexattr_setprio_ceiling()
- □ pthread_mutexattr_getprio_ceiling()

□ thread_prio_protect
- □ pthread_mutex_getprio_ceiling()
- □ pthread_mutex_setprio_ceiling()

□ reentrant_funcs

□ readdir_r(): file_system
□ localtime_r(): C date and time
□ gmtime_r(): C date and time
□ asctime_r(): C date and time
□ ctermid_r(): terminals

□ ctime_r(): C date and time
□ getlogin_r(): user_groups
□ ttyname_r(): terminals
□ getgrgid_r(): system_database
□ getgrnam_r(): system_database

□ getpwnam_r(): system_database
□ getpwuid_r(): system_database
□ strtok_r(): C string handling
□ flockfile():
□ funlockfile()

□ getc_unlocked(): C I/O - PM
□ getchar_unlocked(): C I/O - PM
□ putc_unlocked(): C I/O - PM
□ putchar_unlocked(): C I/O - PM
□ tmpnam(): C I/O - DM

□ rand_r(): C general utilities

1. Figure 3.3.3-6 POSIX.4a Profile Level 2

POSIX.4b Profile Level 2
Platform/Application

Target Processor
(Platform Only)

□ CISC □ RISC □ DSP □ Other
Supplier/Bd./Chip:____________________

Processing Requirements
□ Non-RT □ RT

Application: ____________________
(Component'Aggregate Only)

□ spawn
- □ spawn()
- □ spawnp()

□ timeouts
- □ sem_timedwait()
- □ mq_timedreceive()
- □ mq_timedsend()
- □ pthread_mutex_timedlock()

□ cputime
- □ clock_getcpuclockid()
- □ clock_setenable_attr()
- □ clock_getenable_attr()

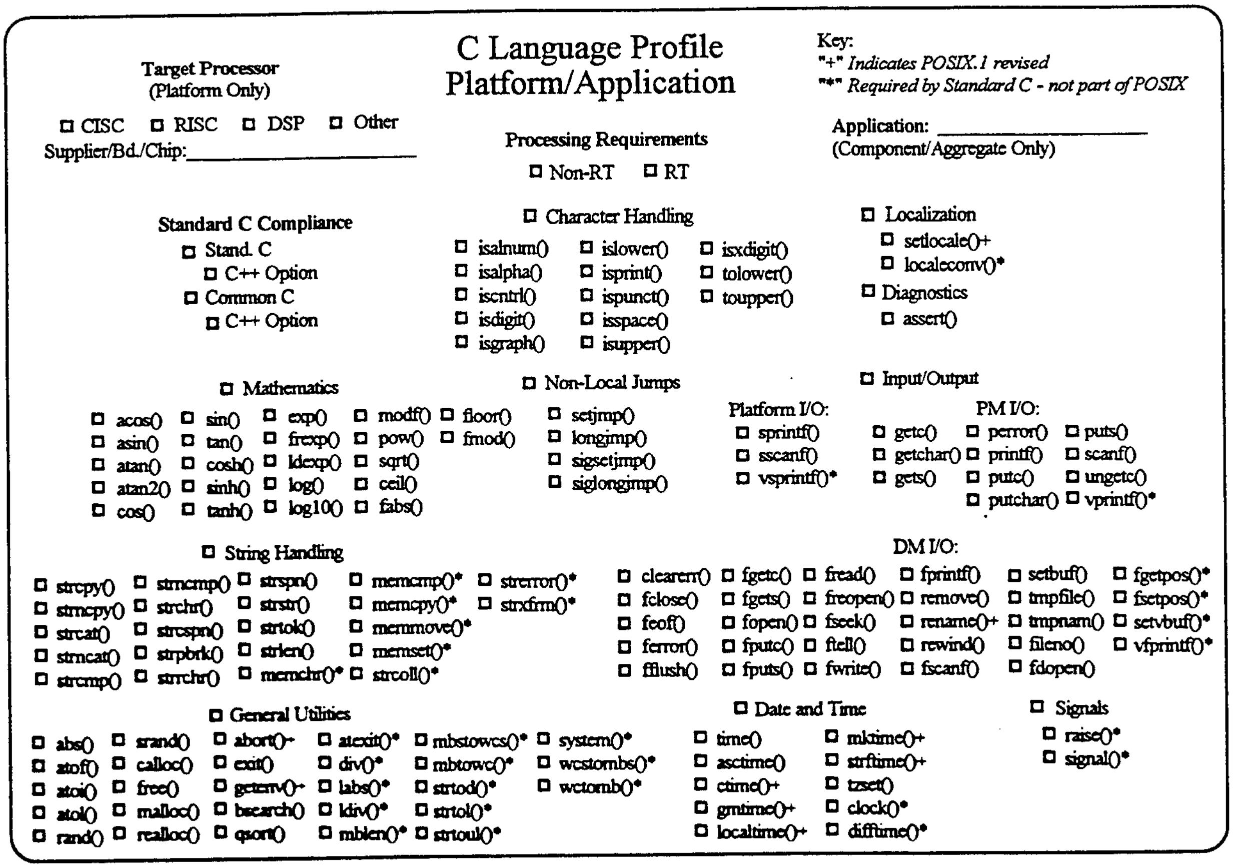
□ thread_cputime
- □ pthread_getcpuclockid()
- □ pthread_attr_setcputime()
- □ pthread_attr_getcputime()

□ threads & thread_priority_scheduling & threads_sporadic_server
- □ pthread_attr_setssparams()
- □ pthread_attr_getssparams()

□ device_control
- □ devctl()

□ interrupt_control
- □ intr_capture()
- □ intr_lock()
- □ intr_unlock()
- □ intr_timed_wait()

Figure 3.3.3-7 POSIX.4b Profile Level 2



C Language Profile
Platform/Application

Target Processor
(Platform Only)
□ CISC □ RISC □ DSP □ Other
Supplier/Bd./Chip:_______________________

Processing Requirements
□ Non-RT □ RT

Key:
"+" *Indicates POSIX.1 revised*
"*" *Required by Standard C - not part of POSIX*

Application: ___________________
(Component/Aggregate Only)

Standard C Compliance
- □ Stand. C
  - □ C++ Option
- □ Common C
  - □ C++ Option

□ Character Handling
□ isalnum() □ isalpha() □ iscntrl() □ isdigit() □ isgraph()
□ islower() □ isprint() □ ispunct() □ isspace() □ isupper()
□ isxdigit() □ tolower() □ toupper()

□ Localization
- □ setlocale()+
- □ localeconv()*

□ Diagnostics
- □ assert()

□ Mathematics
□ acos() □ asin() □ atan() □ atan2() □ cos()
□ sin() □ tan() □ cosh() □ sinh() □ tanh()
□ exp() □ frexp() □ ldexp() □ log() □ log10()
□ modf() □ pow() □ sqrt() □ ceil() □ fabs()
□ floor() □ fmod()

□ Non-Local Jumps
□ setjmp() □ longjmp() □ sigsetjmp() □ siglongjmp()

□ Input/Output
Platform I/O:
□ sprintf() □ sscanf() □ vsprintf()*

PM I/O:
□ getc() □ getchar() □ gets()
□ perror() □ printf() □ putc() □ putchar()
□ puts() □ scanf() □ ungetc() □ vprintf()*

DM I/O:
□ clearerr() □ fclose() □ feof() □ ferror() □ fflush()
□ fgetc() □ fgets() □ fopen() □ fputc() □ fputs()
□ fread() □ freopen() □ fseek() □ ftell() □ fwrite()
□ fprintf() □ remove() □ rename()+ □ rewind() □ fscanf()
□ setbuf() □ tmpfile() □ tmpnam() □ fileno() □ fdopen()
□ fgetpos()* □ fsetpos()* □ setvbuf()* □ vfprintf()*

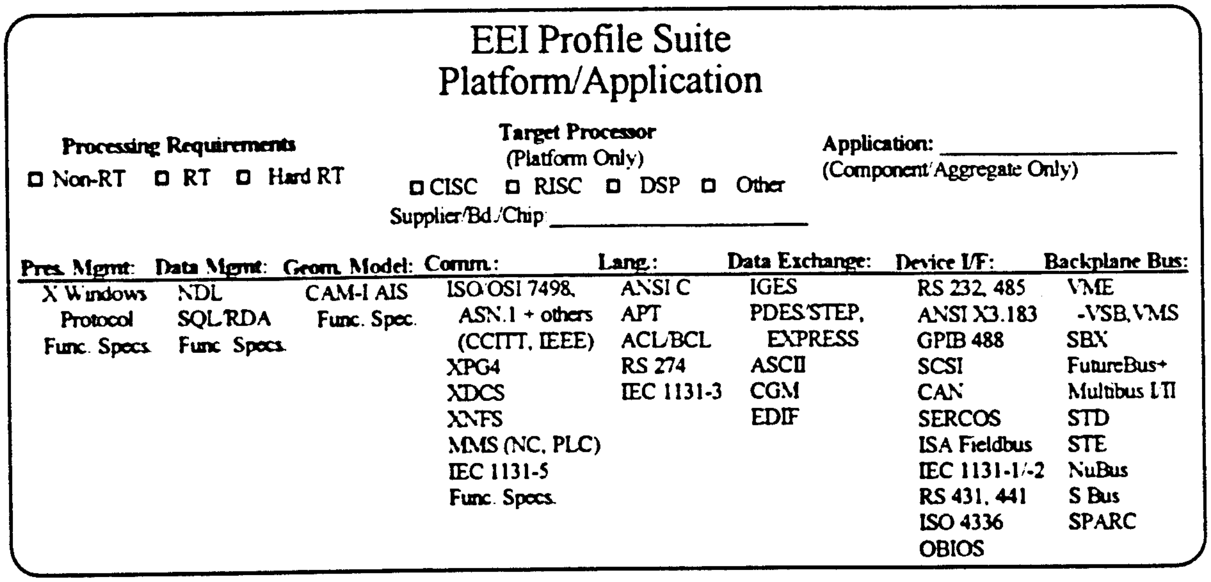
□ String Handling
□ strcpy() □ strncpy() □ strcat() □ strncat() □ strcmp()
□ strncmp() □ strchr() □ strcspn() □ strpbrk() □ strrchr()
□ strspn() □ strstr() □ strtok() □ strlen() □ memchr()*
□ memcmp()* □ memcpy()* □ memmove()* □ memset()* □ strcoll()*
□ strerror()* □ strxfrm()*

□ General Utilities
□ abs() □ atof() □ atoi() □ atol() □ rand()
□ srand() □ calloc() □ free() □ malloc() □ realloc()
□ abort()+ □ exit() □ getenv()+ □ bsearch() □ qsort()
□ atexit()* □ div()* □ labs()* □ ldiv()* □ mblen()*
□ mbstowcs()* □ mbtowc()* □ strtod()* □ strtol()* □ strtoul()*
□ system()* □ wcstombs()* □ wctomb()*

□ Date and Time
□ time() □ asctime() □ ctime()+ □ gmtime()+ □ localtime()+
□ mktime()+ □ strftime()+ □ tzset() □ clock()* □ difftime()*

□ Signals
- □ raise()*
- □ signal()*

Figure 3.3.3-8 C Language Profile



EEI Profile Suite
Platform/Application

Processing Requirements
□ Non-RT □ RT □ Hard RT

Target Processor
(Platform Only)
□ CISC □ RISC □ DSP □ Other
Supplier/Bd./Chip ______________________

Application: ___________________
(Component/Aggregate Only)

Pres. Mgmt:	Data Mgmt:	Geom. Model:	Comm.:	Lang.:	Data Exchange:	Device I/F:	Backplane Bus:
X Windows	NDL	CAM-I AIS	ISO/OSI 7498,	ANSI C	IGES	RS 232, 485	VME
Protocol	SQL/RDA	Func. Spec.	ASN.1 + others	APT	PDES/STEP,	ANSI X3.183	-VSB,VMS
Func. Specs.	Func. Specs.		(CCITT, IEEE)	ACL/BCL	EXPRESS	GPIB 488	SBX
			XPG4	RS 274	ASCII	SCSI	FutureBus+
			XDCS	IEC 1131-3	CGM	CAN	Multibus I/II
			XNFS		EDIF	SERCOS	STD
			MMS (NC, PLC)			ISA Fieldbus	STE
			IEC 1131-5			IEC 1131-1/-2	NuBus
			Func. Specs.			RS 431, 441	S Bus
						ISO 4336	SPARC
						OBIOS	

Figure 3.3.3-9 EEI Profile Suite—Platform/Application

3.4 Development Process and Implementation

This section describes the NGC development process and illustrates the path from an architecture specification to an implementation solution. It is perhaps one of the most important sections for guidance to the NGC controller builder. Emphasis is placed on implementation, where the "rubber meets the road" in terms of NGC product development.

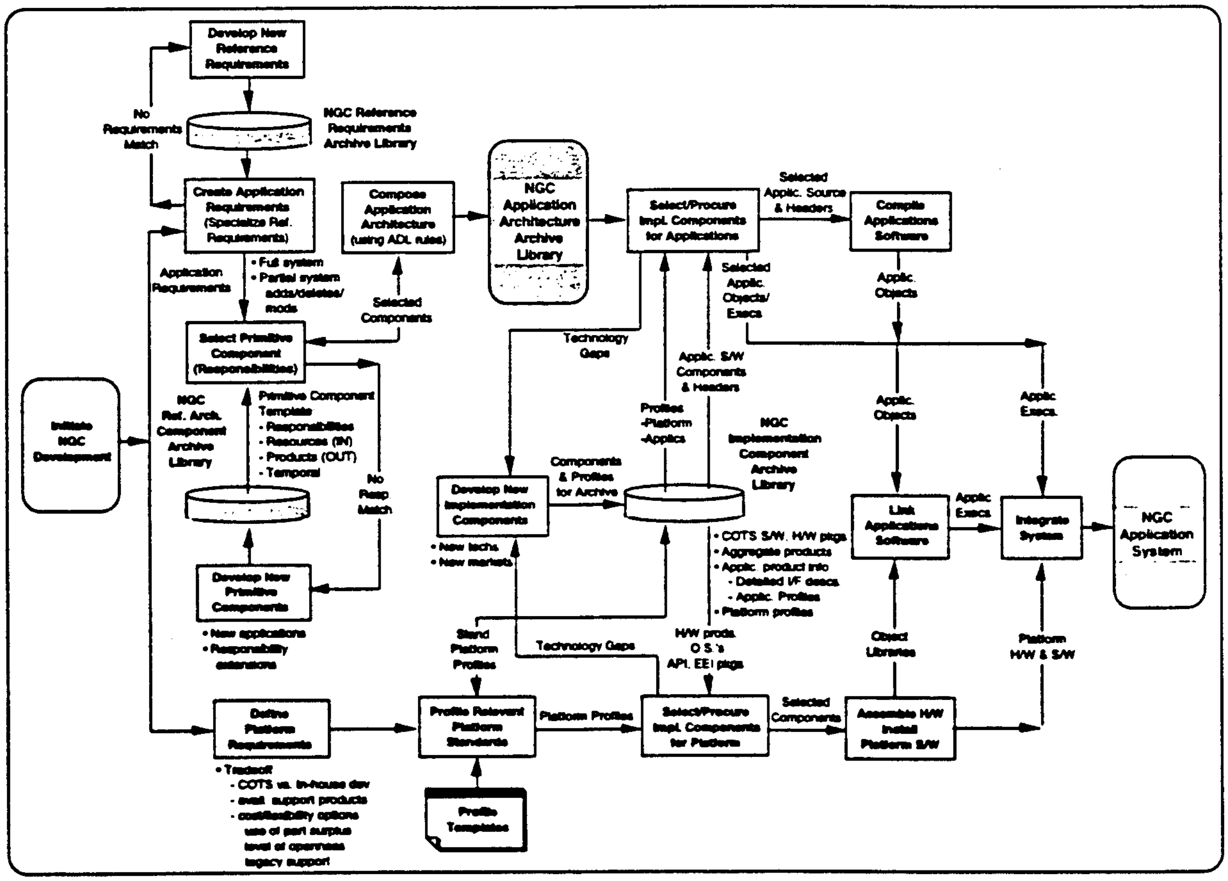
Up to this point, little guidance on "how to implement an NGC" is provided. The preceding sections describe a context, architectural framework, and supporting environment for a NGC. In some respect, they introduce the necessary background and terminology to better understand the NGC specification in preparation for development.

The remainder of this section focuses on those implementation aspects that will evolve a next generation of flexible controller products, bringing specification to reality. After all, the NGC vision will only be realized if there is a large base of existing NGC products available.

Open controller characteristics such as component interchangeability and system flexibility have fundamental significance at an implementation level. These characteristics bring the ultimate benefits to the end user. The ability to add new features, scale performance, and interconnect physical hardware and controller components are natural by-products of open controllers. It is the implementation approach to component development and the resulting interoperability of hardware and software products that enables the desired system flexibility. The NGC implementation approach is therefore key to bringing the controller flexibility desired by end users today into the capabilities of controllers tomorrow.

3.4.1 Integration Process

Figure 3.4.1 illustrates the NGC integration process with focus and greater detail on the design and implementation phase. The term "integration" is used in the title to highlight the notion that if controller development begins with a mature and robust implementation component archive, it is possible to construct a working controller system from existing COTS products without having to create any new components. Early NGC development work must naturally begin by populating this component repository with newly developed components and interactive tools for its access. A plan is currently underway to establish the controlling organization for this repository and develop a strategy for its required maintenance.



Develop New Reference Requirements
No Requirements Match
NGC Reference Requirements Archive Library
Create Application Requirements (Specialize Ref. Requirements)
Compose Application Architecture (using ADL rules)
NGC Application Architecture Archive Library
Select/Procure Impl. Components for Applications
Selected Applic. Source & Headers
Compile Applications Software
Application Requirements
• Full system
• Partial system adds/deletes/ mods
Selected Components
Selected Applic. Objects/ Execs
Applic. Objects
Select Primitive Component (Responsibilities)
Technology Gaps
Applic. S/W Components & Headers
Applic. Objects
Applic. Execs.
Initiate NGC Development
NGC Ref. Arch. Component Archive Library
Primitive Component Template
- Responsibilities
- Resources (IN)
- Products (OUT)
- Temporal
No Resp Match
Profiles -Platform -Applics
NGC Implementation Component Archive Library
Develop New Implementation Components
Components & Profiles for Archive
Link Applications Software
Applic. Execs.
Integrate System
NGC Application System
• New techs.
• New markets
• COTS S/W, H/W pkgs
• Aggregate products
• Applic. product info
- Detailed I/F descs.
- Applic. Profiles
• Platform profiles
Develop New Primitive Components
• New applications
• Responsibility extensions
Stand Platform Profiles
Technology Gaps
H/W prods. O.S.'s API, EEI pkgs.
Object Libraries
Platform H/W & S/W
Define Platform Requirements
Profile Relevant Platform Standards
Platform Profiles
Select/Procure Impl. Components for Platform
Selected Components
Assemble H/W Install Platform S/W
• Tradeoff
- COTS vs. in-house dev
- avail. support products
- cost/flexibility options
use of part surplus
level of openness
legacy support
Profile Templates

Figure 3.4.1 NGC Integration Process

The integration process diagram captures the general flow and process dependencies and is not necessarily a prescription for any one single approach to development. The developer may begin development starting from any stage in the process. For example, one organization may decide to follow the NGC integration process using concurrent engineering practices, assigning engineering work groups to various stages of the process simultaneously. All feedback paths are implicit and do not appear on the diagram to avoid unnecessary complexity in illustration.

The integration process flow consists of two main parallel paths: applications development and platform development. The applications development path is illustrated as the upper path in the process diagram. The lower path addresses platform development. Both paths compliment and support each other and are interrelated in ways not easily captured in the process flow diagram.

For example, the selection of applications software is driven by the compatibility of the available software components and the platform selected for a specific controller implementation. Profiles provide the mechanism for contrasting software component implementations against selected platform implementations so their compatibility may be ascertained. The availability or lack of existing applications components for a specific platform implementation may drive the need to select an alternate platform implementation. Trades such as these, the "make-or-buy" choices, are implicit in the process flow.

The platform implementation tends to solve more of the general-purpose computation and communications requirements. These requirements are referred to as "domain-independent" requirements. Conversely, application component implementations address more of the application-specific or "domain-dependent" requirements to control and manufacturing. The dependence of the platform on existing open standards makes available to the developer a variety of implementation solutions and supporting COTS products. Every NGC platform implementation must be profiled. With every new NGC applications software product developed, a corresponding profile and interface description must be supplied.

To complete the integration story, selected and/or developed source-level software components are compiled and linked together with object-level component libraries to generate the executable controller applications for the target controller platform. Platform profiles and profiles supplied with applications software are used in the selection process so that only compatible components are selected. All hardware and executable software is then integrated into a fully functional NGC application system, or simply the controller.

3.4.2 Application Framework, Architecture, and Implementation

The architectural concept of a "framework" forms the underlying infrastructure and foundation of NGC. At the platform level, the NGC Open System Environment (OSE) framework embodies 3 concepts: (1) a "reference model", which is based on and extends the POSIX model; (2) a "taxonomy" of de jure standards; and (3) "profiles" of de facto and de jure standards. It defines a set of language-dependent, function-oriented API callable services and facilitates the development of object-oriented wrappers.

In addition, at the application level, the NGC specification facilitates the generation of a variety of frameworks for specific application architectures, e.g., 5-axis machining center. Much of the information contained in this specification may be used to compose a "framework" for the development of an application architecture. An application framework is characterized by a selected set of primitive components and an implicit topology (resource-product associations). A framework is "composed" by selecting and specializing reference requirements with a specific application in mind, and by applying architecture description rules (ADRs) for primitive component selection and interconnection. Specification of the reference requirements, primitive components, and architecture description rules (ADRs) used for composing an application framework is contained in the appendices of this document.

With this background, the process of creating an "application architecture" is better understood. This process is lightly touched upon in the upper left hand portion of the integration process diagram (Figure 3.4.1) and is further expanded in Figure 3.4.2. An application architecture is distinct from its corresponding framework in that it specifies the first major level of design detail.

Starting from an application framework, architectural components are defined consisting of primitive components and aggregates of primitive components that compose the framework. Language-independent message interfaces are specified between architectural components and between components and the external environment. The architecture specification must also take into consideration design requirements for system flexibility. By design, the architecture must accommodate the system's ability to integrate legacy motion controller boards, add new sensors, or increase the number of axis of control for a specific application requirement. The architecture may also be the first level for considering timing requirements. The "aggregation" or grouping of primitive components into larger architectural components for product packaging may be driven in-part by commonly shared timing requirements. The application architecture is therefore a top-level design manifestation of an application framework that supports all the desired requirements for system flexibility.

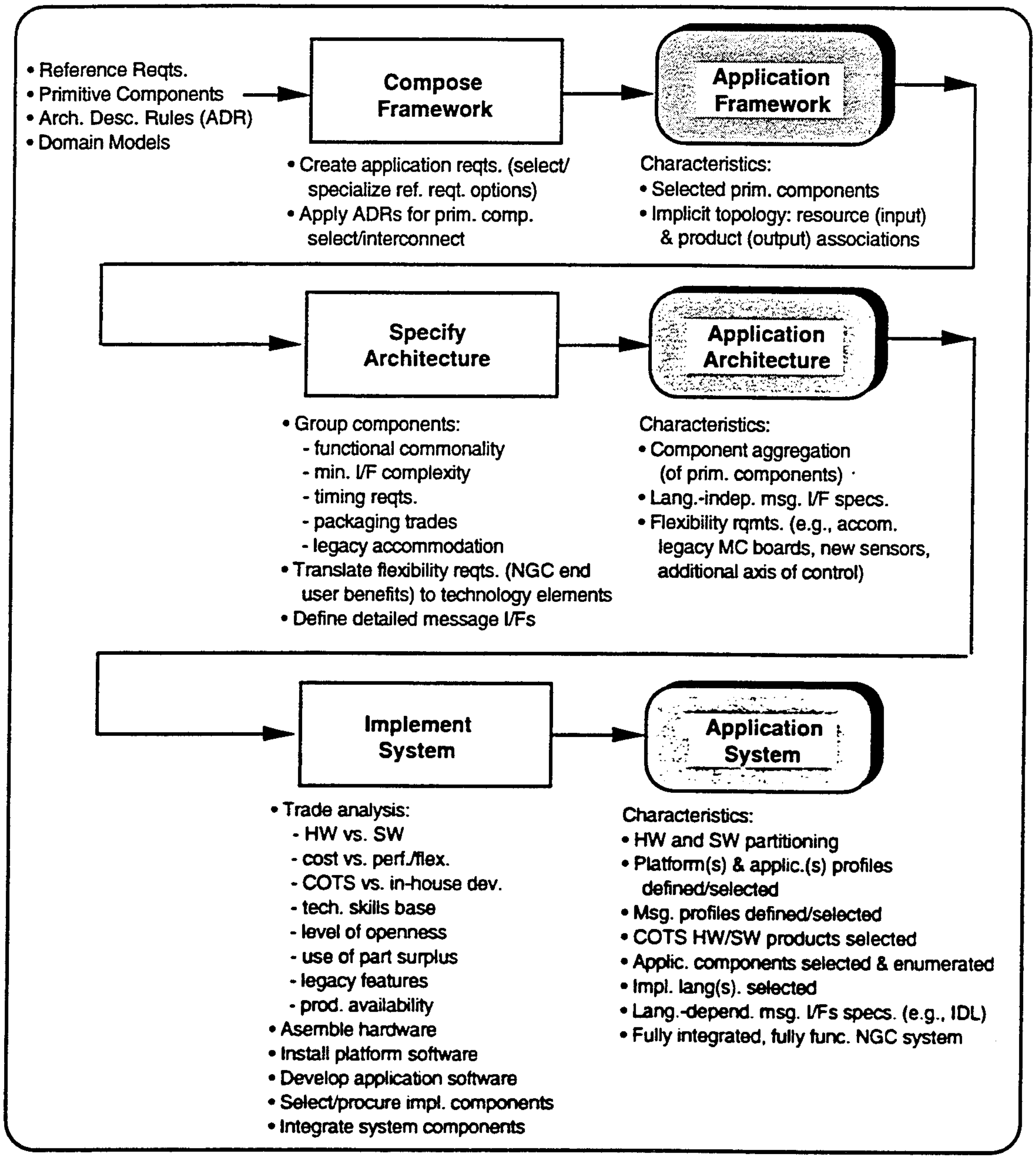

• Reference Reqts.
• Primitive Components
• Arch. Desc. Rules (ADR)
• Domain Models
Compose Framework
Application Framework
• Create application reqts. (select/ specialize ref. reqt. options)
• Apply ADRs for prim. comp. select/interconnect
Characteristics:
• Selected prim. components
• Implicit topology: resource (input) & product (output) associations
Specify Architecture
Application Architecture
• Group components:
- functional commonality
- min. I/F complexity
- timing reqts.
- packaging trades
- legacy accommodation
• Translate flexibility reqts. (NGC end user benefits) to technology elements
• Define detailed message I/Fs
Characteristics:
• Component aggregation (of prim. components)
• Lang.-indep. msg. I/F specs.
• Flexibility rqmts. (e.g., accom. legacy MC boards, new sensors, additional axis of control)
Implement System
Application System
• Trade analysis:
- HW vs. SW
- cost vs. perf./flex.
- COTS vs. in-house dev.
- tech. skills base
- level of openness
- use of part surplus
- legacy features
- prod. availability
• Asemble hardware
• Install platform software
• Develop application software
• Select/procure impl. components
• Integrate system components
Characteristics:
• HW and SW partitioning
• Platform(s) & applic.(s) profiles defined/selected
• Msg. profiles defined/selected
• COTS HW/SW products selected
• Applic. components selected & enumerated
• Impl. lang(s). selected
• Lang.-depend. msg. I/Fs specs. (e.g., IDL)
• Fully integrated, fully func. NGC system

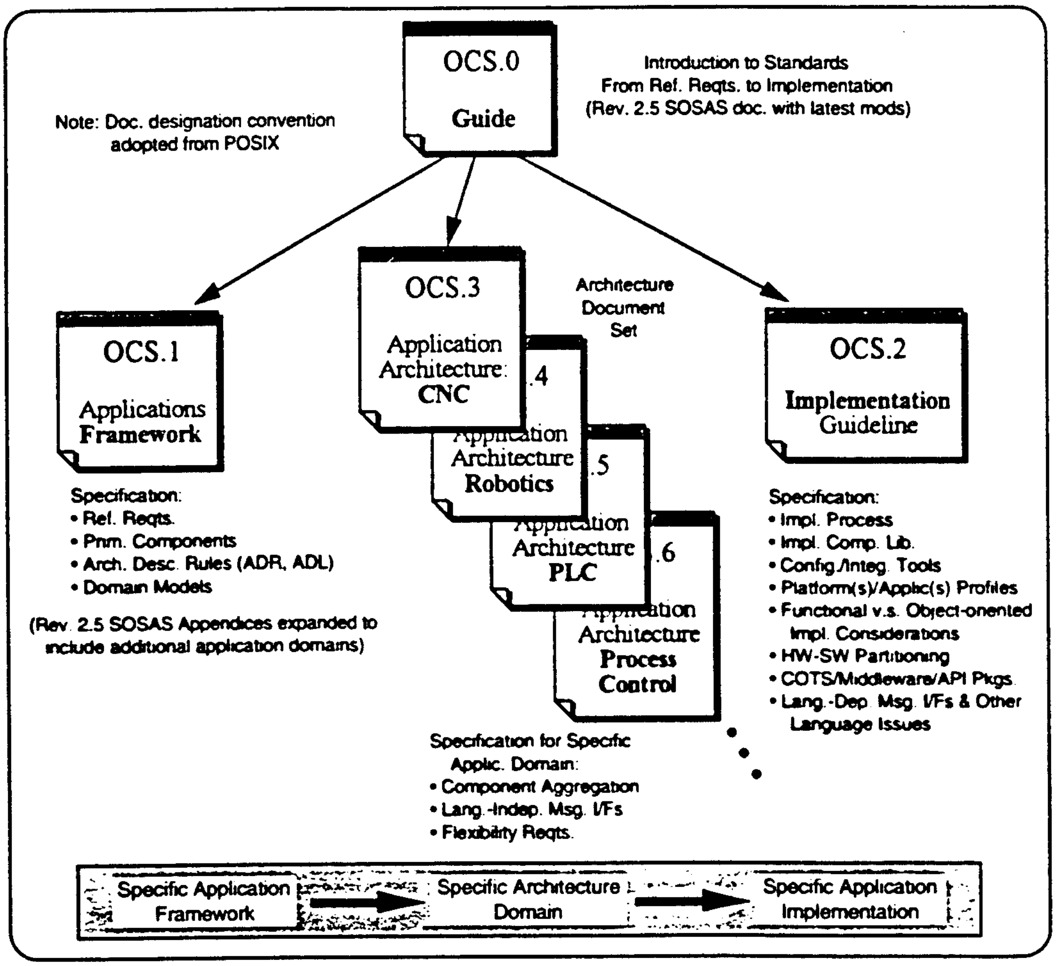
Figure 3.4.2 From Reference Requirements to Implementation

Figure 3.4.2 illustrates the process of developing an implementation from reference requirements, showing the characteristic differences between an application framework, architecture, and the final system. Note that profiles and language issues are implementation concepts and are not architectural concepts. This reinforces the previous discussion that component interchangeability, facilitated through the use of profiles, is fundamental at the implementation level.

Different architectures may potentially be spawned from one framework. Consistent with the profiling philosophy of this specification, the number of different architectures that evolve from one framework and the number of specific frameworks that require standardization must be market-driven. The subsequent discussion on standardization and the document vision provides additional supportive rationale for separating the application framework from its corresponding architecture.

3.4.3 Standardization and Document Vision

Figure 3.4.3-1 depicts the NGC document vision for open controller standardization. This SOSAS specification, including its latest modifications, with the exception of its many appendices, may be viewed essentially as the "Guide to Open Controller Standards (OCS.0)". It introduces the standards and philosophies and provides a context and terminology for understanding by example the other standards documents. The adopted convention for designating the documents is similar to that used for the POSIX standards. For example, POSIX.0 is the guide to the POSIX standards.



OCS.0
Guide
Introduction to Standards
From Ref. Reqts. to Implementation
(Rev. 2.5 SOSAS doc. with latest mods)
Note: Doc. designation convention adopted from POSIX
OCS.1
Applications Framework
Specification:
• Ref. Reqts.
• Prim. Components
• Arch. Desc. Rules (ADR, ADL)
• Domain Models
(Rev. 2.5 SOSAS Appendices expanded to include additional application domains)
OCS.3
Application Architecture: CNC
Architecture Document Set
.4
Application Architecture Robotics
.5
Application Architecture PLC
.6
Application Architecture Process Control
Specification for Specific Applic. Domain:
• Component Aggregation
• Lang.-Indep. Msg. I/Fs
• Flexibility Reqts.
OCS.2
Implementation Guideline
Specification:
• Impl. Process
• Impl. Comp. Lib.
• Config./Integ. Tools
• Platform(s)/Applic(s) Profiles
• Functional v.s. Object-oriented Impl. Considerations
• HW-SW Partitioning
• COTS/Middleware/API Pkgs.
• Lang.-Dep. Msg. I/Fs & Other Language Issues
Specific Application Framework
Specific Architecture Domain
Specific Application Implementation

Figure 3.4.3-1 NGC Document Vision: Open Controller Standards

The “Applications Framework Open Controller Standard (OSC.1)”, is envisioned to contain most of the appendices from this SOSAS specification. In addition, it will contain an expanded set of reference requirements, primitive components, architecture description rules, and domain models to compose frameworks for application domains outside of its present focus on CNCs. Other application domains include robotics, programmable logic controllers (PLCs), process control, semiconductor manufacturing, etc.

Architecture specifications (OCS.3, OCS.4, etc.) are envisioned to evolve for specific applications based on market-driven factors. For traceability, these documents must specify the application framework upon which they are based. Each framework must be individually derivable from the OCS.1 document which will be expanded to accommodate additional application domains as market forces dictate.

Different documents satisfy different needs of the community. For example, from a CNC developer’s perspective, the “CNC Application Architecture (OCS.3)” document and the “Implementation Guideline (OCS.2)” may be the only two specifications of interest. OSC.2 is envisioned to be an expansion of this SOSAS implementation section with detailed information on profiling, language binding, the component library, interactive development tools, and other system development and integration information. Standards bodies, on the otherhand, may have much more of an interest in the framework document, OCS.1, and its role in governing the evolving set of application architecture specifications. The developer of an NGC product may or may not care about the OCS.1 document. However, if a controller developer has a vested interest in standardizing his application architecture, he will not only have an interest in OCS.1 but must also show compliance to a specific application framework that is derivable from the OCS.1 specification.

As a final point in understanding the NGC document vision and the document interrelationships, consider the ISO/OSI 7 layer reference model specification (ISO 7498) and its relationship to the Ethernet level 2 specification (IEEE 802.3). Full interoperability is only guaranteed when a specific physical implementation is determined (e.g. twisted pair, coax, etc.). Even the Ethernet specification is incomplete in assuring interoperability of physical components, but it certainly specifies a detailed level of protocols that is not specified by its parent ISO 7498 specification. Token ring and token bus are also defined level 2 protocols under the ISO/OSI reference model - yet neither will interoperate without converting bridges or gateways.

By analogy, the NGC applications framework is to a specific architecture as the ISO/OSI reference model is to Ethernet. Ethernet is to token ring as one NGC application architecture is to another. Neither will interoperate. Just as physical characteristics are important to ensure interoperable Ethernet implementations, NGC component profiles and language binding considerations are of equal importance to ensure the interoperability and interchangeability of application architecture component implementations. These implementation issues must be considered to ensure interoperability yet, like the Ethernet specification, are not specified within the NGC architecture documents. The criteria for ensuring NGC implementation interoperability is discussed in the OCS.2 Implementation Guideline.

A formal standardization approach is required to bring the NGC documentation vision to reality. The following 3 methods are recognized by ANSI as establishing evidence of consensus: (1) Accredited sponsor using the canvas method, (2) Accredited standards committee method (ASC), and (3) Accredited standards developing organization method (ASDO). A full explanation of these methods is beyond the scope of this specification. Figure 3.4.3-2 is a graphic depiction of the process model for the second method, the ASC method. This process model provides some insight into the standardization process. All methods focus on capturing consensus and adequate representation (due process) and are equivalent in their final results. Therefore, any of these methods are acceptable for NGC document standardization.

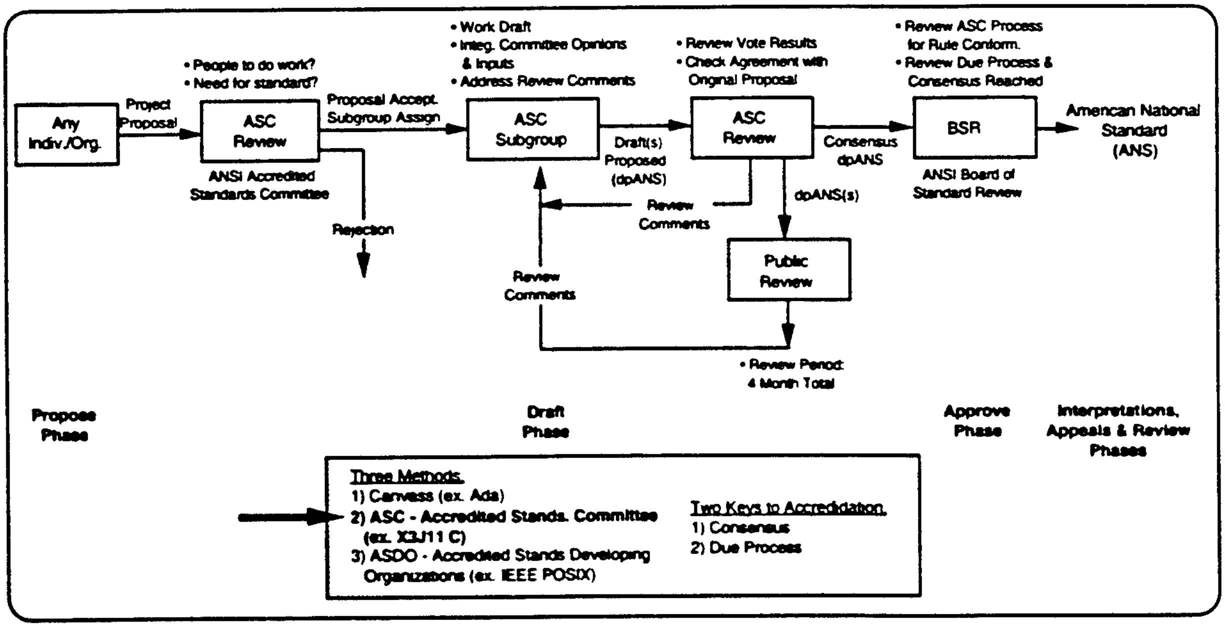

Any Indiv./Org.
Project Proposal
• People to do work?
• Need for standard?
ASC Review
ANSI Accredited Standards Committee
Rejection
Proposal Accept. Subgroup Assign
• Work Draft
• Integ. Committee Opinions & Inputs
• Address Review Comments
ASC Subgroup
Draft(s) Proposed (dpANS)
Review Comments
Review Comments
• Review Vote Results
• Check Agreement with Original Proposal
ASC Review
dpANS(s)
Public Review
• Review Period: 4 Month Total
Consensus dpANS
• Review ASC Process for Rule Conform.
• Review Due Process & Consensus Reached
BSR
ANSI Board of Standard Review
American National Standard (ANS)
Propose Phase
Draft Phase
Approve Phase
Interpretations, Appeals & Review Phases
Three Methods.
1) Canvass (ex. Ada)
2) ASC - Accredited Stands. Committee (ex. X3J11 C)
3) ASDO - Accredited Stands Developing Organizations (ex. IEEE POSIX)
Two Keys to Accredidation.
1) Consensus
2) Due Process

Figure 3.4.3-2 ANSI Standardization Process Model—ACS Method

3.4.4 Implementation Component Library & Tools

Section 3.4.1, Implementation Process, introduced the notion of an implementation component archive that may be used to create a working controller system from an existing repository of controller parts. This, of course, is currently a vision and early NGC development work must naturally begin by populating this component repository with newly developed components and interactive tools for its access.

The NGC implementation component library is envisioned to be a shopping list of COTS software and hardware packages as well as a central archive of reuseable, publicly available (perhaps at a cost) aggregate software products. It is the repository for storage and retrieval of applications/platform product information, detailed interface descriptions, and profiles. To support this vision, interactive tools for creating, maintaining and browsing the library are needed that do not currently exist. Figure 3.4.4 presents a conceptual view of the component library structure and contents.

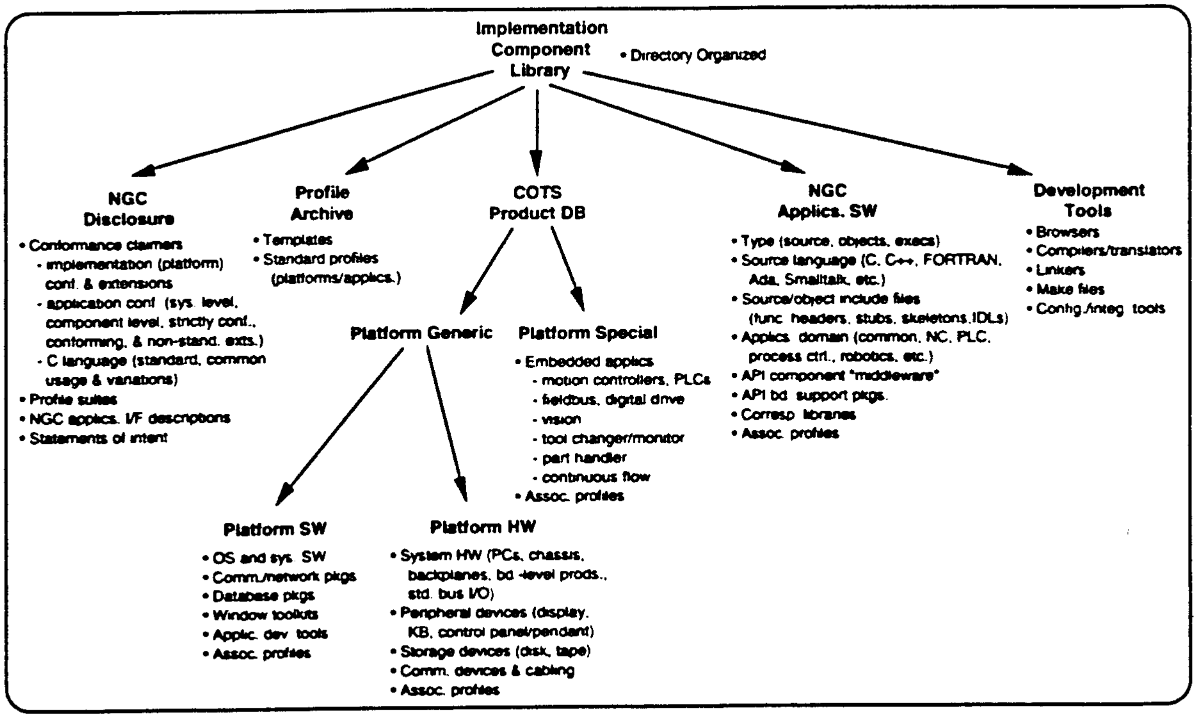

Implementation Component Library
• Directory Organized
NGC Disclosure
• Conformance claimers
- implementation (platform) conf. & extensions
- application conf (sys. level, component level, strictly conf., conforming, & non-stand. exts.)
- C language (standard, common usage & variations)
• Profile suites
• NGC applics. I/F descriptions
• Statements of intent
Profile Archive
• Templates
• Standard profiles (platforms/applics.)
COTS Product DB
Platform Generic
Platform Special
• Embedded applics
- motion controllers, PLCs
- fieldbus, digital drive
- vision
- tool changer/monitor
- part handler
- continuous flow
• Assoc. profiles
Platform SW
• OS and sys. SW
• Comm./network pkgs
• Database pkgs
• Window toolkits
• Applic. dev. tools
• Assoc. profiles
Platform HW
• System HW (PCs, chassis, backplanes, bd.-level prods., std. bus I/O)
• Peripheral devices (display, KB, control panel/pendant)
• Storage devices (disk, tape)
• Comm. devices & cabling
• Assoc. profiles
NGC Applics. SW
• Type (source, objects, execs)
• Source language (C, C++, FORTRAN, Ada, Smalltalk, etc.)
• Source/object include files (func. headers, stubs, skeletons, IDLs)
• Applics. domain (common, NC, PLC, process ctrl., robotics, etc.)
• API component "middleware"
• API bd. support pkgs.
• Corresp. libraries
• Assoc. profiles
Development Tools
• Browsers
• Compilers/translators
• Linkers
• Make files
• Config./integ. tools

Figure 3.4.4 Implementation Component Library & Tools—Structure & Contents

Having user interactive tools to peruse available profile information on specific products facilitates "apples-to-apples" comparison of products. Product profiles may be used to assess if a specific software component will run on a selected platform implementation. Profiles may also determine the relative compatibility of two different vendor applications for replacing one with the other to upgrade an existing NGC system. Creating a profile for a platform that was conceived with a specific set of vendor products in mind has the added potential benefit of exposing alternative implementation solutions that may not have been considered during the initial trade studies.

The unavailability of interactive library tools today implies that the first NGC controllers must break new ground. Early implementations will define platform and component profiles from clean templates. Much of the controller-specific applications software will have to be developed either from scratch or through the reuse and modification of legacy software. With every new NGC software product developed, a corresponding profile and interface description must be supplied.

The mature existence of an implementation library and interactive development tools for its use is believed to be the single, most critical enabling technology for NGC.

3.4.5 Implementation

Previous sections describe the process flow leading to NGC implementations. Sections 3.4.1 and 3.4.2 illustrate the integration process and provide a context for the implementation process. The NGC concepts of application framework and application architecture are reinforced and related to the implementation process. Section 3.4.3 ties specification to implementation and section 3.4.4 provides a deeper understanding of the structure and contents of the implementation component library.

Section 3.4.3, Standardization and Document Vision, discusses the need for an "Implementation Guideline" specification (OCS.2). Its contents must include all of the information described thus far in section 3.4 with more detail. It must address language issues such as language-dependent message interface specification. In addition, it must cover issues related with profiling, hardware-software partitioning, functional vs. object-oriented implementations, and the development and integration of various "types" of component packages from COTS to middleware to board support packages to specialized API service development. In depth coverage of these subjects are beyond the scope of this specification but is a requirement for interoperable, interchangeable component implementations.

It should now be apparent that an applications architecture specification forms the basis for an implementation but falls short of specifying the requirements for interoperable implementation components. Figure 3.4.5-1 illustrates some of the new design considerations in transitioning from an application architecture to an implementation. An application architecture is a high-level design that takes form in an implementation in terms of hardware and software components. Implementation software components support applications or platform functions, all of which execute on the hardware platform. Similarly, the platform hardware is composed of a number of hardware implementation components.

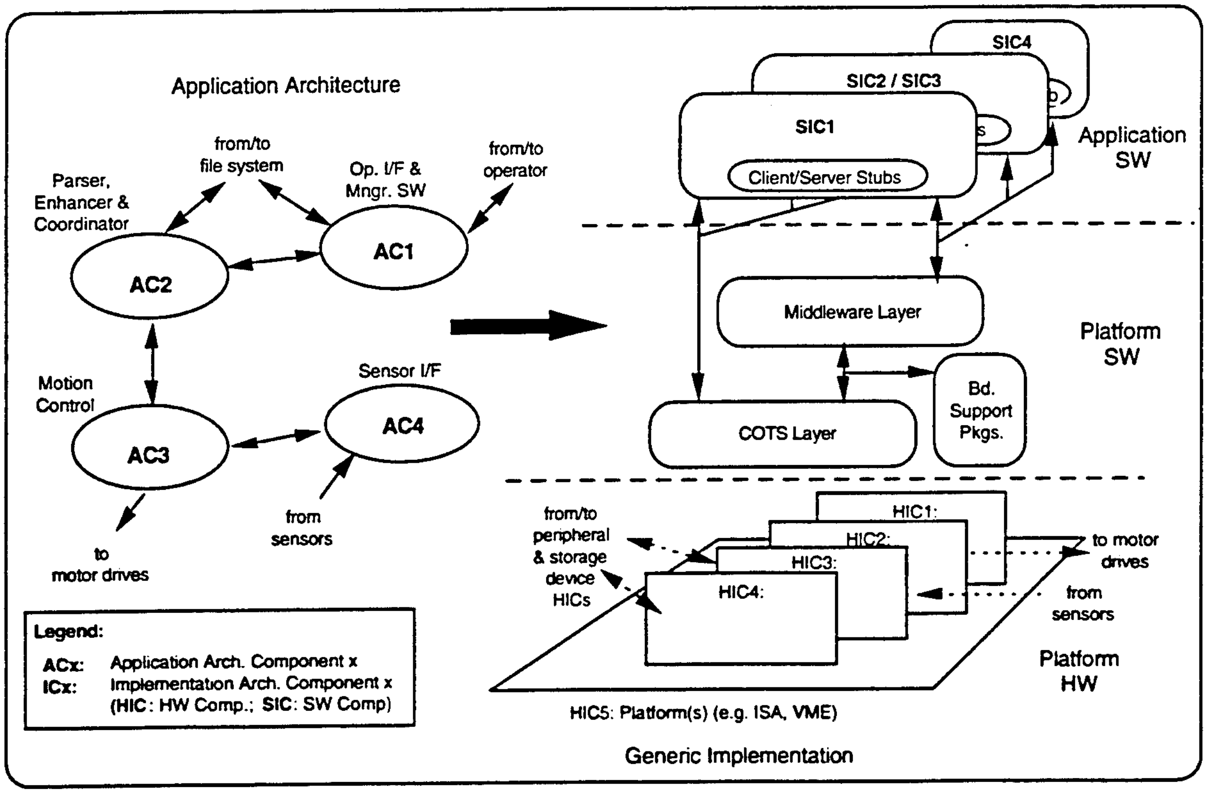


Figure 3.4.5-1 From Application Architecture to Implementation

Platform software may simply involve the installation of standards-based COTS packages. Where COTS packages fall short of the selected platform profile standards, middleware components may be developed or reused (if available) to bridge the API gap. Middleware may also provide the required access layer to board-level and embedded controller products.

At an applications software level, client/server stubs provide the required mechanisms for transparent communications between local and/or remote implementation components. These components are natural by-products of CORBA and DCE based implementations. They are generated by the interface definition language (IDL) compiler and implement remote procedure calls (RPCs).

Figures 3.4.5-2 and 3.4.5-3 expand the simplified architecture and generic implementation example of Figure 3.4.5-1 to introduce timing and multi-platform considerations. Component aggregation is illustrated in the expanded application architecture example. The primitive components of a specific application component aggregate are numerically identified. The intercomponent message sets are also numerically designated and directly correlate between the application architecture and the resulting generic implementation.

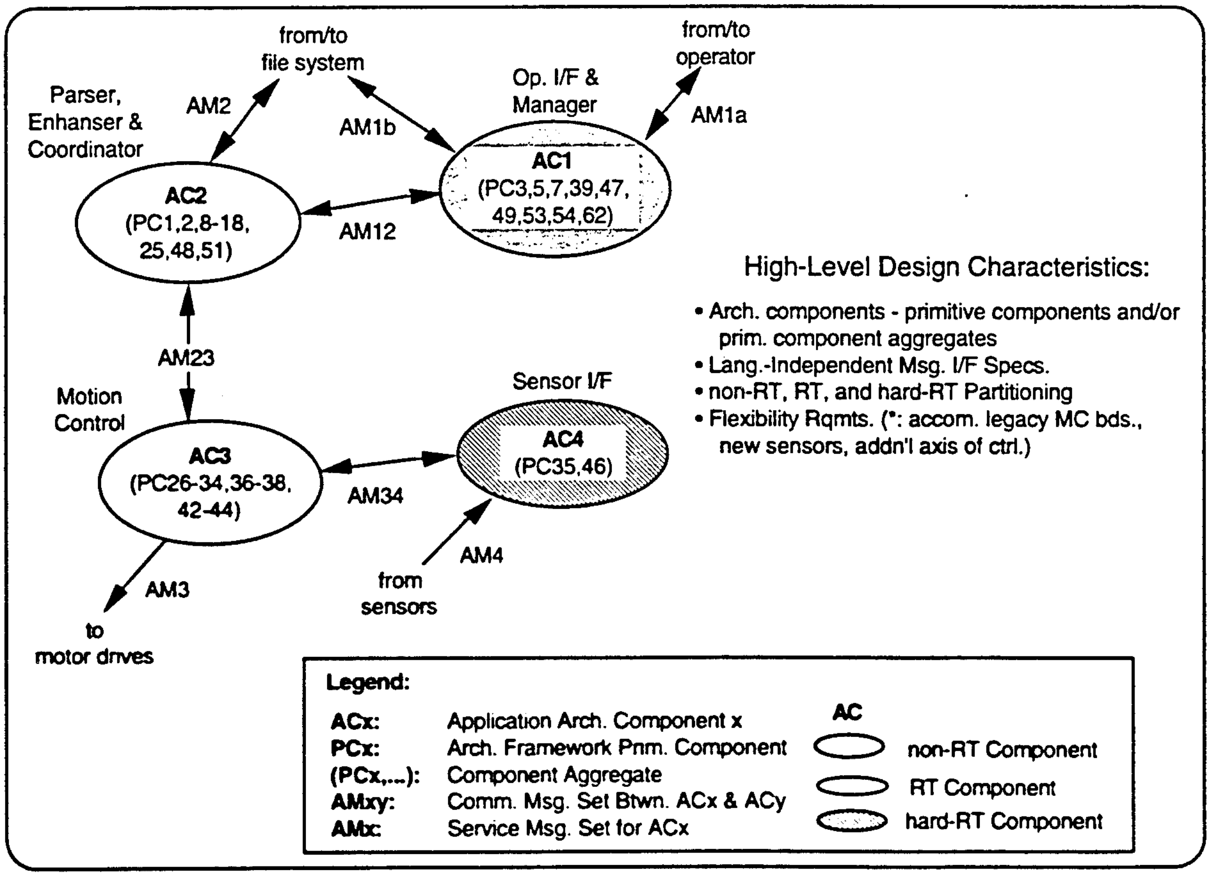

from/to
file system
from/to
operator
Parser,
Enhanser &
Coordinator
AM2
AM1b
Op. I/F &
Manager
AM1a
AC1
(PC3,5,7,39,47,
49,53,54,62)
AC2
(PC1,2,8-18,
25,48,51)
AM12
High-Level Design Characteristics:
• Arch. components - primitive components and/or prim. component aggregates
• Lang.-Independent Msg. I/F Specs.
• non-RT, RT, and hard-RT Partitioning
• Flexibility Rqmts. (*: accom. legacy MC bds., new sensors, addn'l axis of ctrl.)
AM23
Motion
Control
Sensor I/F
AC3
(PC26-34,36-38,
42-44)
AC4
(PC35,46)
AM34
AM4
from
sensors
AM3
to
motor drives
Legend:
ACx: Application Arch. Component x
PCx: Arch. Framework Prim. Component
(PCx,...): Component Aggregate
AMxy: Comm. Msg. Set Btwn. ACx & ACy
AMx: Service Msg. Set for ACx
AC
non-RT Component
RT Component
hard-RT Component

Figure 3.4.5-2 Application Architecture

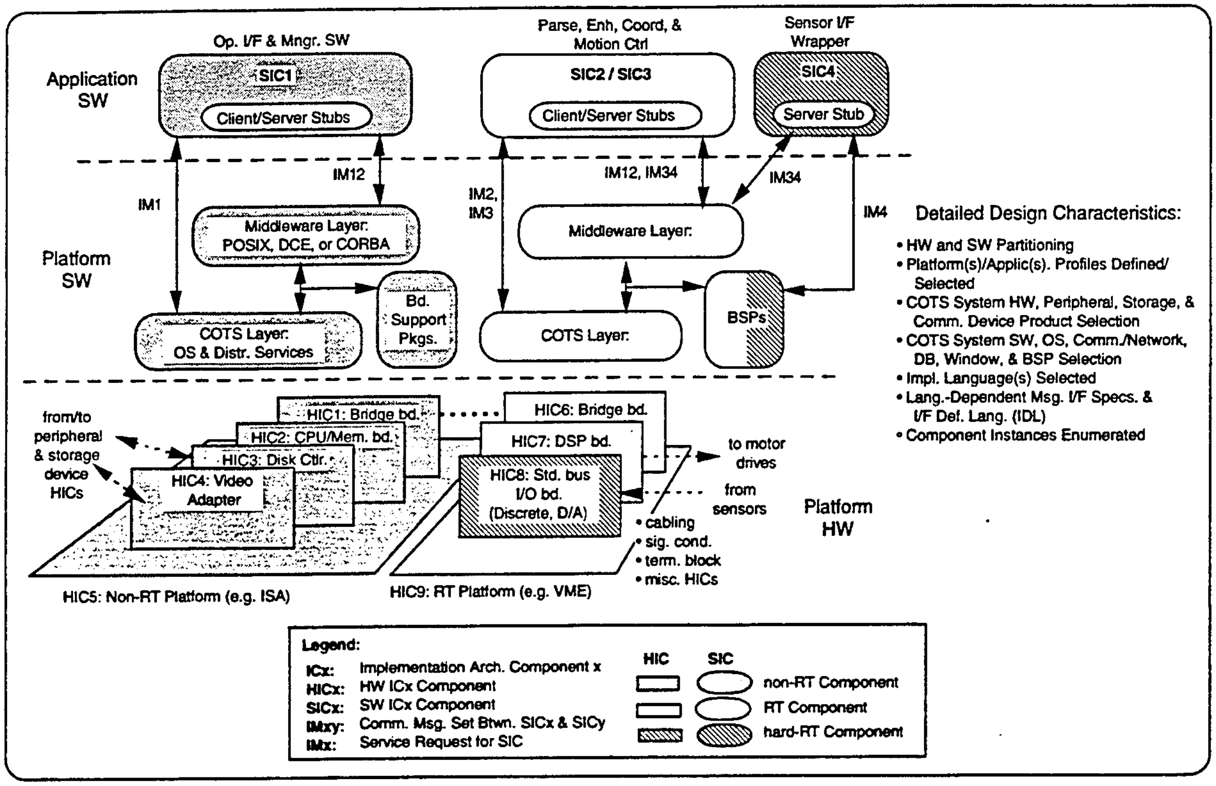

Application SW
Op. I/F & Mngr. SW
SIC1
Client/Server Stubs
Parse, Enh, Coord, & Motion Ctrl
SIC2 / SIC3
Client/Server Stubs
Sensor I/F Wrapper
SIC4
Server Stub
IM1
IM12
IM2, IM3
IM12, IM34
IM34
IM4
Platform SW
Middleware Layer: POSIX, DCE, or CORBA
Middleware Layer:
Bd. Support Pkgs.
COTS Layer: OS & Distr. Services
COTS Layer:
BSPs
Detailed Design Characteristics:
• HW and SW Partitioning
• Platform(s)/Applic(s). Profiles Defined/ Selected
• COTS System HW, Peripheral, Storage, & Comm. Device Product Selection
• COTS System SW, OS, Comm./Network, DB, Window, & BSP Selection
• Impl. Language(s) Selected
• Lang.-Dependent Msg. I/F Specs. & I/F Def. Lang. (IDL)
• Component Instances Enumerated
from/to peripheral & storage device HICs
HIC1: Bridge bd.
HIC2: CPU/Mem. bd.
HIC3: Disk Ctlr.
HIC4: Video Adapter
HIC6: Bridge bd.
HIC7: DSP bd.
HIC8: Std. bus I/O bd. (Discrete, D/A)
to motor drives
from sensors
• cabling
• sig. cond.
• term. block
• misc. HICs
Platform HW
HIC5: Non-RT Platform (e.g. ISA)
HIC9: RT Platform (e.g. VME)
Legend:
ICx: Implementation Arch. Component x
HICx: HW ICx Component
SICx: SW ICx Component
IMxy: Comm. Msg. Set Btwn. SICx & SICy
IMx: Service Request for SIC
HIC
SIC
non-RT Component
RT Component
hard-RT Component

Figure 3.4.5-3 Generic Implementation

3.5 NGC Conformance

NGC conformance is defined at several levels. At the system level, conformance is determined by following at least one of several NGC-specified profiles of EEI and API standards sets. An application is NGC-conformant with respect to a specific profile if that application always uses the standard API defined for that profile whenever it accesses functionality that is supported by that API.

The OSE reference model described in Section 3.3.1 explains the OSE framework and its impact on applications portability and systems-level interoperability. It is important to note that the OSE framework does not specifically deal with component-level interoperability issues other than to specify several applicable data standards for message interface. As described in Section 3.1, component messages and behaviors are specified to facilitate software interoperability, but component interfaces are not specified at a level that guarantees fully interoperable, fully interchangeable component implementations. However, specification of components, a reference topology, and definition of high-level message classes puts a natural limitation on the total design space of controller applications.

To ensure that independent component developers may develop complimentary products that will interoperate with other controller components, the following SOSAS requirement is added: a detailed description of all component interfaces shall be supplied with any controller or component that claims component-level interoperability.

This specification is a guide for an anticipatory standard. The fact that it is not a specification that simply documents existing practice presents an inherent obstacle to its acceptance by the controller industry. This partially explains the adopted position on component interoperability, that is, to keep the component interface descriptions at an abstract, implementation-independent level. In an ideal world, the SOSAS could specify component interfaces at a sufficient level of detail that would ensure full software interoperability. If such a detailed specification is to be adopted by the controller builder community, it would have to be sensitive to market drivers and de facto standards for component interfaces that may potentially evolve. To ensure the acceptance and future applicability of the SOSAS, these interfaces have been deliberately specified at a high level.

This NGC standards approach is analogous to the approach used for the ISO/OSI communications standard. The ISO/OSI reference model standard (ISO 7498) specifies a multi-level framework for communications at a sufficient enough level of abstraction to facilitate the evolution of several different network implementations (e.g., ethernet, token ring, token bus, etc.), yet it constrains the implementations to a specific hierarchy and specific sets of services and protocols. A possible scenario for the NGC is that supporting SOSAS documents may evolve that define the component interfaces at a level of detail that will facilitate fully interoperable implementations.

For a full NGC controller product, there are two basic levels of conformance: system level and component level. System-level conformance implies the disclosure of all profiling information specific to the platform (for all API/EEI categories). This same profiling information must also be disclosed for the applications software as a whole. Component-level conformance subsumes system-level conformance requirements and additionally requires the disclosure of detailed component interface specifications.

The SOSAS extends POSIX philosophies with regard to conformance. All conformance requirements within each individual POSIX.n specification shall apply if that specification is selected for inclusion in the developer's profile. Implementation conformance refers to the NGC system platform and associated services, or the execution environment for applications-level NGC software components. An NGC platform is implementation conformant if it conforms to a specific and complete profile in all dimensions of API and EEI standards, from communications protocols and services to device interfaces and human-user interface (HUI) API. The NGC platform may include nonstandard extensions, provided such extensions are identified in the system documentation. For such platforms, an environment must be defined and documented in which applications may run with the expected behaviors specified by the corresponding standards of the profile suite.

Application conformance applies to an application component of the NGC claiming conformance to a specific platform implementation. A "strictly conforming" NGC application is one that requires only the facilities defined by the specific platform implementation for which it conforms. Additional documentation shall be supplied for "conforming" NGC applications that use non-standard extensions of a NGC platform implementation.

Software applications claiming conformance with the C language shall claim conformance with either "C Standard Language-Dependent System Support" or "Common Usage C Language-Dependent System Support". The former refers to conformance with the C programming language international standard, ISO/IEC 9899:1990. Applications conforming instead with the latter, common usage C, shall clearly document all variations from the international C standard. It is expected that much of this information can be acquired from the vendor of the specific C compiler product used for development.

POSIX.3, formally IEEE Standard 1003.3-1991: Test Methods for Measuring Conformance to POSIX, will be the NGC adopted standard for test suite validation of POSIX-compliant products. Since COTS products claiming POSIX compliance must follow this standard, the adoption of POSIX.3 for the NGC is simply a statement of current and expected future practice. Note the term "compliance" is used instead of the term "conformance". Compliance has stronger meaning and implies the existence of a test suite for measuring the conformance of an implementation. The application and extension of POSIX.3 test methods (beyond its intended POSIX scope) into other NGC standards domains is not a requirement. Other supporting test standards will apply as the market dictates.

Many POSIX draft standards are identified in the NGC OSE. Since there is an obvious lag of commercially available products for such standards, the NGC developer need only disclose a "statement of intent" for inclusion of such standards in future revisions of his product. This statement of intent applies especially to those implementations that may accommodate some level of the draft specification, but not necessarily the SOSAS referenced version, due to the development lag of supporting products.

For example, Wind River Systems' VxWorks 5.1 product literature advertises full ANSI C compliance. Several features from the real-time extensions proposed in the POSIX.4 draft were added to the existing POSIX.1 file and input/output system. Wind River's product literature further documents their commitment to full compliance with POSIX.1 and their offering of incremental compliance with POSIX.4 and .4a drafts, with full support once these standards are approved.

The specification of a profile suite, component interface descriptions, language documentation, statements of intent, and associated conformance claimers shall be supplied as applicable with each NGC-conformant product in the form of a "**disclosure statement**". An NGC product may include:

- Fully implemented platform;
- Platform product;
- Software applications (primitive component, component aggregates, module, and library implementations);
- NGC system.

4.0 NGC DEVELOPMENT EXAMPLE

The 5-axis horizontal machining center shown in Figure 4.0 requires a controller. This section addresses how to provide an open controller for this application, and it serves as an example of the open controller design process. We begin with a brief overview of the design and implementation process for this example. Salient parts of this process are elaborated in subsections.

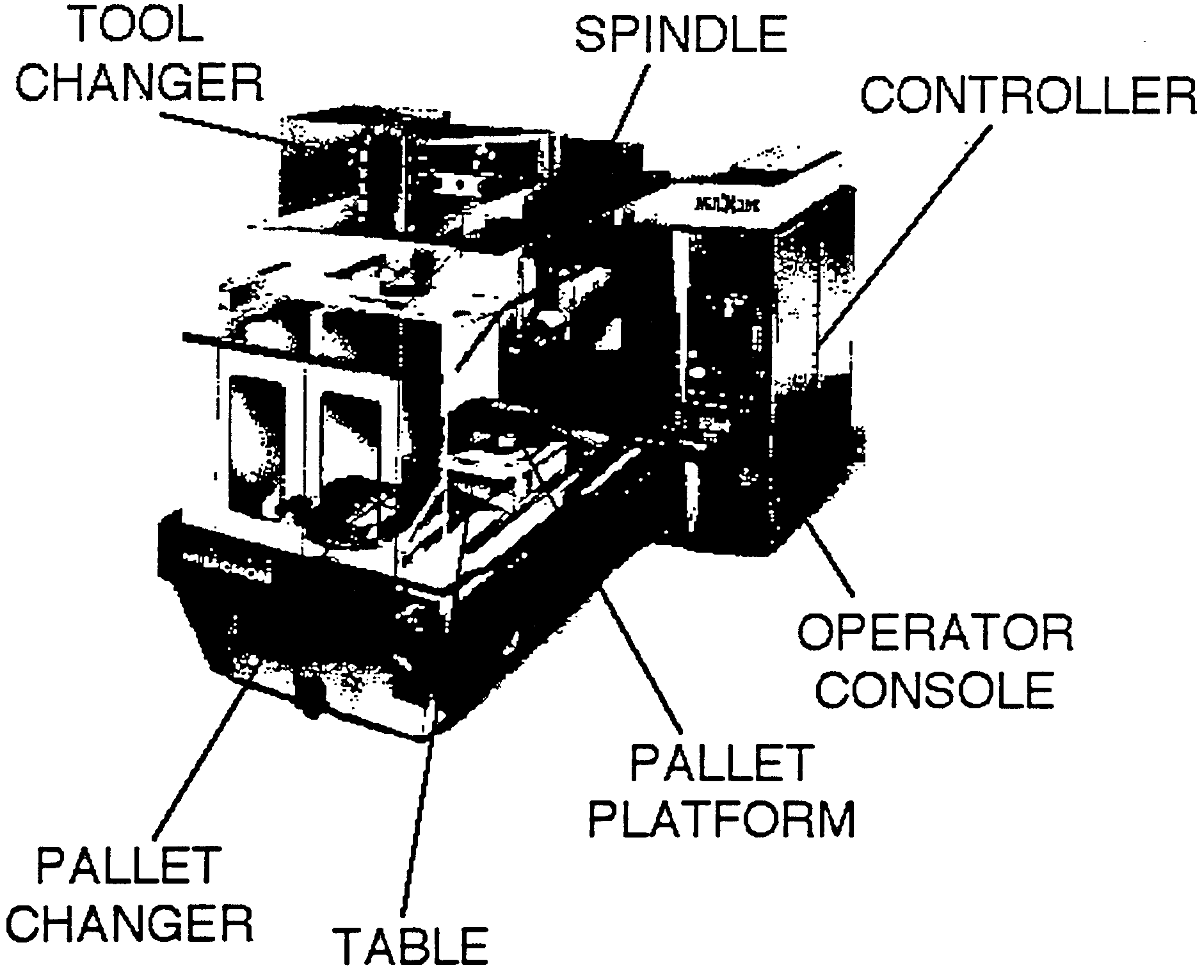
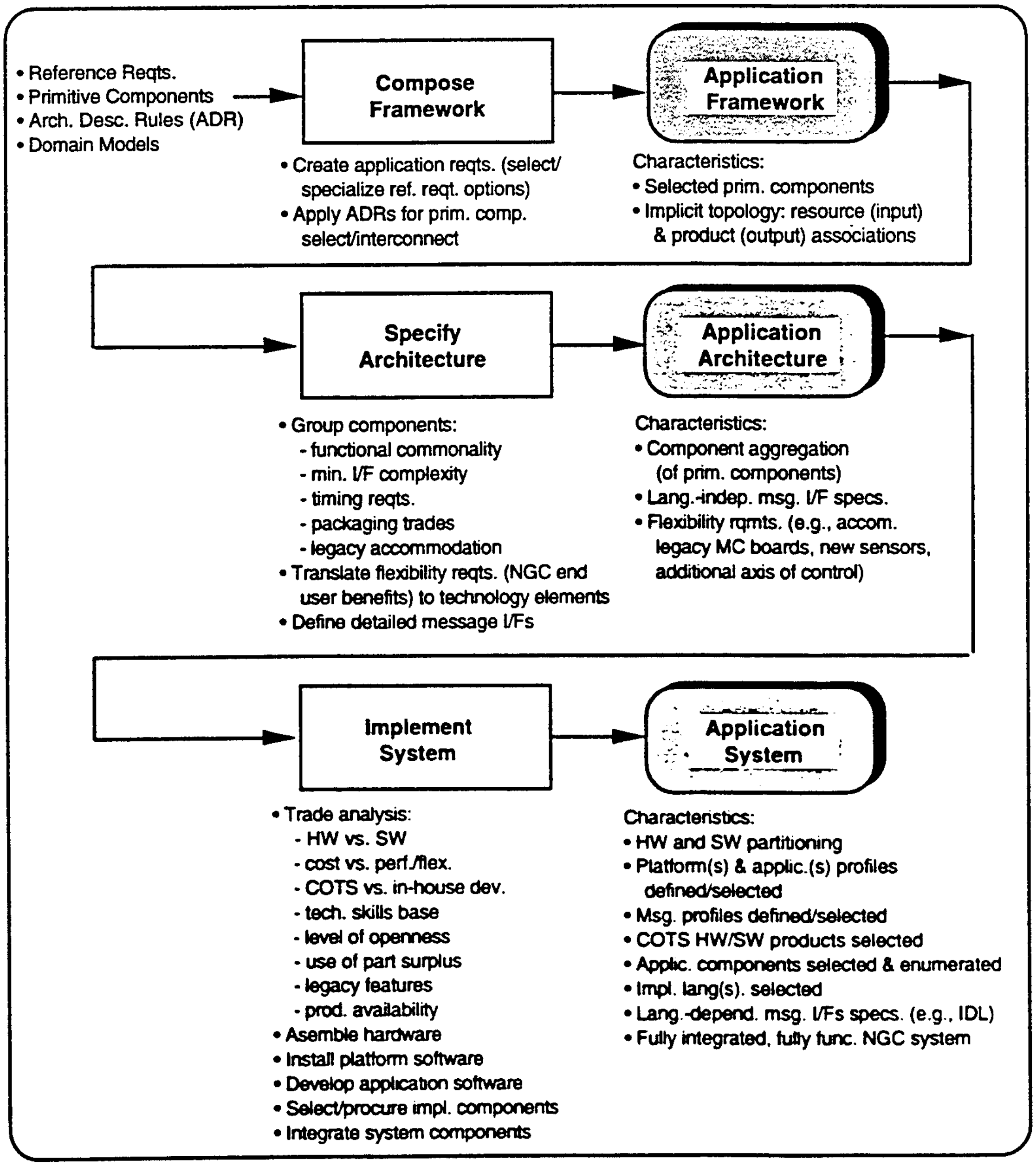


Figure 4.0. Horizontal Machining Center

The first major step in the process is to create the application requirements for 5-axis machining by selecting all relevant Reference Requirements (see Appendix A) and making them specific to the application. Application requirements help to establish an application framework for 5-axis machining. These requirements drive the selection of the Reference Architecture primitive components (see Appendix B) that constitute the application framework. Collectively, the primitive component responsibilities must cover the application requirements. If no set of primitive components from the Reference Architecture can cover the requirements, then existing primitive components are modified and/or new ones are added until the revised set of primitive components entirely matches the requirements. If any of the new or modified primitive components prove to be generally useful and if they have unique responsibilities, they can be added to the Reference Architecture and used to compose future application frameworks.


• Reference Reqts.
• Primitive Components
• Arch. Desc. Rules (ADR)
• Domain Models
Compose Framework
• Create application reqts. (select/ specialize ref. reqt. options)
• Apply ADRs for prim. comp. select/interconnect
Application Framework
Characteristics:
• Selected prim. components
• Implicit topology: resource (input) & product (output) associations
Specify Architecture
• Group components:
- functional commonality
- min. I/F complexity
- timing reqts.
- packaging trades
- legacy accommodation
• Translate flexibility reqts. (NGC end user benefits) to technology elements
• Define detailed message I/Fs
Application Architecture
Characteristics:
• Component aggregation (of prim. components)
• Lang.-indep. msg. I/F specs.
• Flexibility rqmts. (e.g., accom. legacy MC boards, new sensors, additional axis of control)
Implement System
• Trade analysis:
- HW vs. SW
- cost vs. perf./flex.
- COTS vs. in-house dev.
- tech. skills base
- level of openness
- use of part surplus
- legacy features
- prod. availability
• Asemble hardware
• Install platform software
• Develop application software
• Select/procure impl. components
• Integrate system components
Application System
Characteristics:
• HW and SW partitioning
• Platform(s) & applic.(s) profiles defined/selected
• Msg. profiles defined/selected
• COTS HW/SW products selected
• Applic. components selected & enumerated
• Impl. lang(s). selected
• Lang.-depend. msg. I/Fs specs. (e.g., IDL)
• Fully integrated, fully func. NGC system

During the next step, we group primitive components from the application framework into the components of the Machining Center Application Architecture, which is an open specification of components and message interfaces. Suppliers will base their interoperable controller products on this published, freely available specification. These primitive components groupings are influenced by Architecture Description Rules (see Appendix C), by Domain Models (see Appendix D), and by design considerations and implementation technology, e.g., availability of off-the-shelf implementation components, timing capabilities, and information flow. The previous step focused on analysis, but during this step, design considerations become important. The behaviors of an application architecture component derive from the collective responsibilities of its grouped primitive components. A component's message set derives from its behaviors and the resources and products of its primitive components.

Then we try to match existing software implementation components from the library, in this example the Acme™ software library, to the components of the application architecture. Again, wherever no implementation components match the component specifications of the application architecture, existing software must be modified and/or new implementation components must be added to the Acme™ software library. After any needed modifications or additions, the resulting set of implementation components comprise an application system, which is one instantiation of the application architecture. The Acme™ library can be updated by adding the new or modified software.

A controller for this 5-axis machining center becomes open when it is accompanied by a complete architectural description of its application system. This description includes, but is not limited to: the standards profile, the set of APIs used, the application architecture, the hardware configuration, the application software configuration, and the detailed definitions of the implementation components their message interfaces.

4.1 Application Architecture for Machining

Application requirements (see Table 4.1) are used in conjunction with the Architecture Description Rules and Domain Models to assemble the components of an application architecture for controlling the 5-axis machining center as shown in the example of Figure 4.1. Based on requirements, relevant primitive components are selected from the Reference Architecture, and their responsibilities and interfaces are aggregated into the components of this example. In Figure 4.1, each component is given a unique name descriptive of the role it plays in the application architecture. A component's constituent primitive components are indicated by number (see Appendix B). The lines connecting the components show the message paths.

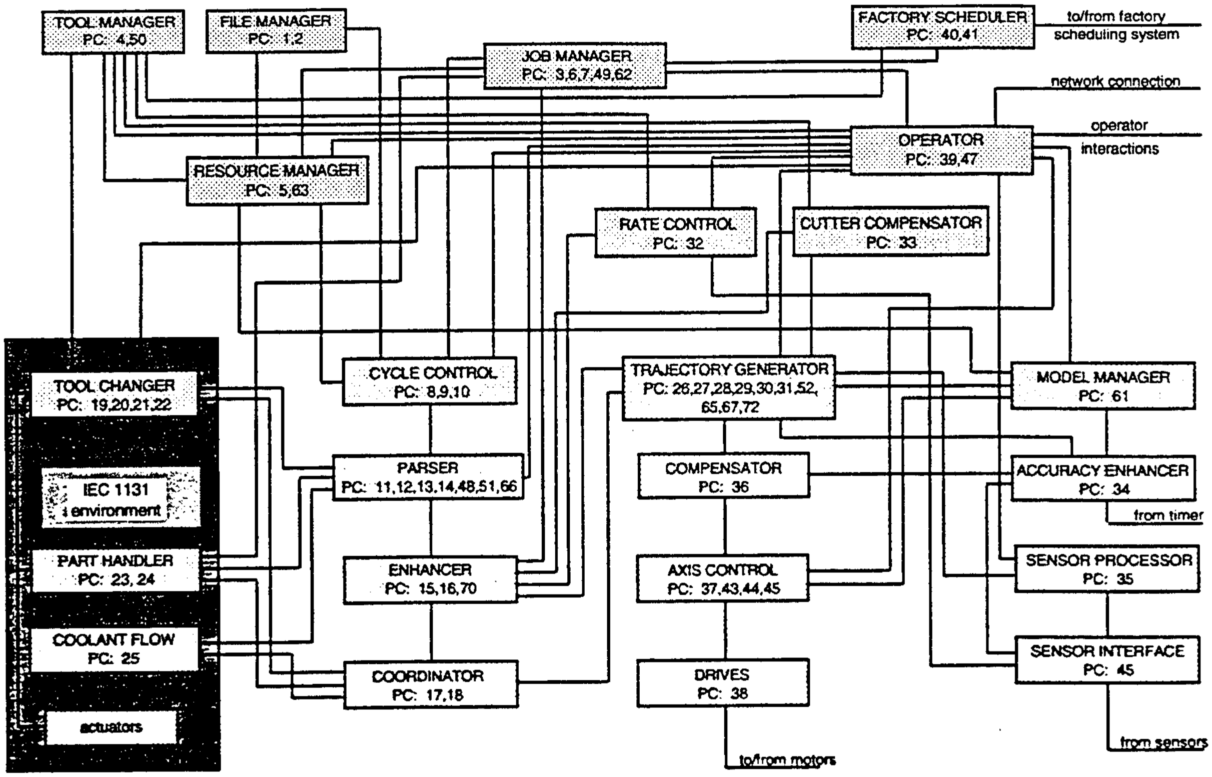


Figure 4.1. Example of an Application Architecture for a Machining Center

Table 4.1. Application Requirements for Machining Example

Reference Number	Application Requirement
A.1.3.1	The controller shall interface with a supervisory computer system.
A.1.3.2	The supervisory computer system shall exert complete control over the controller.
A.1.7.1	Block processing time shall be fast enough to prevent data starvation.
A.1.7.2	The controller shall automatically decelerate axes movement when the block execution time is less than the average block processing time.
A.2.1.1	The controller shall provide continuous path control of tool motion.
A.2.1.2	The controller shall provide point-to-point programming.
A.2.2.1	The controller shall perform linear interpolation in simultaneous motion of all axis drives.
A.2.2.2	The controller shall perform circular interpolation with 3-D motion control.
A.2.2.3	The controller shall perform helical interpolation.
A.2.7.1	The controller shall automatically control axis acceleration/deceleration.
A.2.8.1	The controller shall provide automatic compensation of lead screw errors (pitch error compensation).
A.2.8.2	The controller shall provide automatic compensation of backlash.
A.2.8.3	Lead screw error compensation shall be reprogrammable for wear compensation.
A.2.8.4	The controller shall perform cross compensation for axis alignment, beam sag and axis motion geometry errors.
A.2.9.1	The controller shall perform temperature growth compensation.
A.2.12.1	The controller shall support programmable feedrates.
A.3.1.1	Tool management shall have storage for the maximum number of tool data sets.
A.3.1.4	The tool shall be referenced by the tool identification code.
A.3.1.5	The tool shall be referenced by the tool location identification code.
A.3.1.6	The tool magazine shall provide random storage of tools with preassignment.
A.3.1.7	Tool slots, compartments, or other storage elements shall not have sensors for identification and location of each tool.
A.3.1.8	The tool magazine shall hold the complement of tools and spares for specific jobs.
A.3.1.9	The tool magazine shall not support removable tool cartridges.
A.3.1.10	The tool magazine shall not support sensors for identification of tool cartridges and their job associations.
A.3.1.15	Tool change sequence shall be initiated by a T code.
A.3.1.16	Tool change sequence shall not be initiated manually.
A.3.1.18	Tool length shall not be verified as part of the tool change procedure.
A.3.1.19	Tool diameter shall not be verified as part of the tool change procedure.
A.3.1.20	Tool form shall not be verified as part of the tool change procedure.
A.3.2.1	The controller shall identify pallets by non-contact sensors.
A.3.2.4	Pallet identification shall be used to automatically select part programs.
A.3.2.5	Pallet identification shall be used to automatically select fixture offsets.
A.3.2.6	Pallet identification shall be used to automatically select pallet offsets.
A.3.2.7	Multiple part programs shall be selected per pallet.
A.3.3.9	The controller shall measure engineering force levels on the spindle bearings.
A.3.3.10	The controller shall measure temperature of the spindle bearings.
A.3.3.14	The controller shall adjust feedrate to maintain constant cutting force.

(Continued)

Table 4.1. Application Requirements for Machining Example (Cont.)

Reference No.	Application Requirement
A.4.1.5	Programming shall include feedrate programming: direct feedrate, inverse time feedrate, per revolution unit feed for turning controls.
A.4.1.11	Programming shall include fixture offsets for multiple pallet and table machines.
A.4.1.12	Programming shall include multiple program storage and management.
A.4.1.16	Programming shall include programmed tool change.
A.4.1.17	Programming shall include programming for cutting process control.
A.4.1.19	Programming shall include programmable adaptive control parameters.
A.4.1.23	Programming shall include preprogrammed (canned) cycles for drilling, boring, and tapping.
A.4.1.24	Programming shall include preprogrammed cycles for area milling, rectangular pocket milling, circular pocket milling, and bolt hole circles.
A.4.1.25	Programming shall include automatic chamfering and corner radiusing.
A.4.1.26	Programming shall include look-ahead cutter compensation.
A.4.1.27	Programming shall include coordinate system rotation.
A.4.1.28	Programming shall include work coordinate system setting.
A.4.1.30	Programming shall include constant surface speed programming.
A.4.1.31	Programming shall include tool center point programming for 5-axis machining.
A.4.2.2	Manual programming shall include automatic selection of cutting speed and feed.
A.4.2.3	Manual programming shall include cutter offset.
A.4.2.5	Manual programming shall include pallet or table change cycles.

4.2 Machining Scenario

The application architecture for a machining center controller facilitates its design and implementation. In this section we show how the application architecture, which consists of components and message interfaces, is also useful for verifying the operation of the controller in an example scenario. The example scenario covers mass production of parts, and it is better understood by referring to Figure 4.2 where the message interfaces of the application architecture in Figure 4.1 are shown in greater detail.

This is a partial example of an application architecture. For clarity, the functionality of the components and the number of messages have been limited intentionally just to cover the scenario. This cannot reflect the complete set of messages needed for all of the operating scenarios of a machining center, but it gives a flavor of the level of message specification for an application architecture.

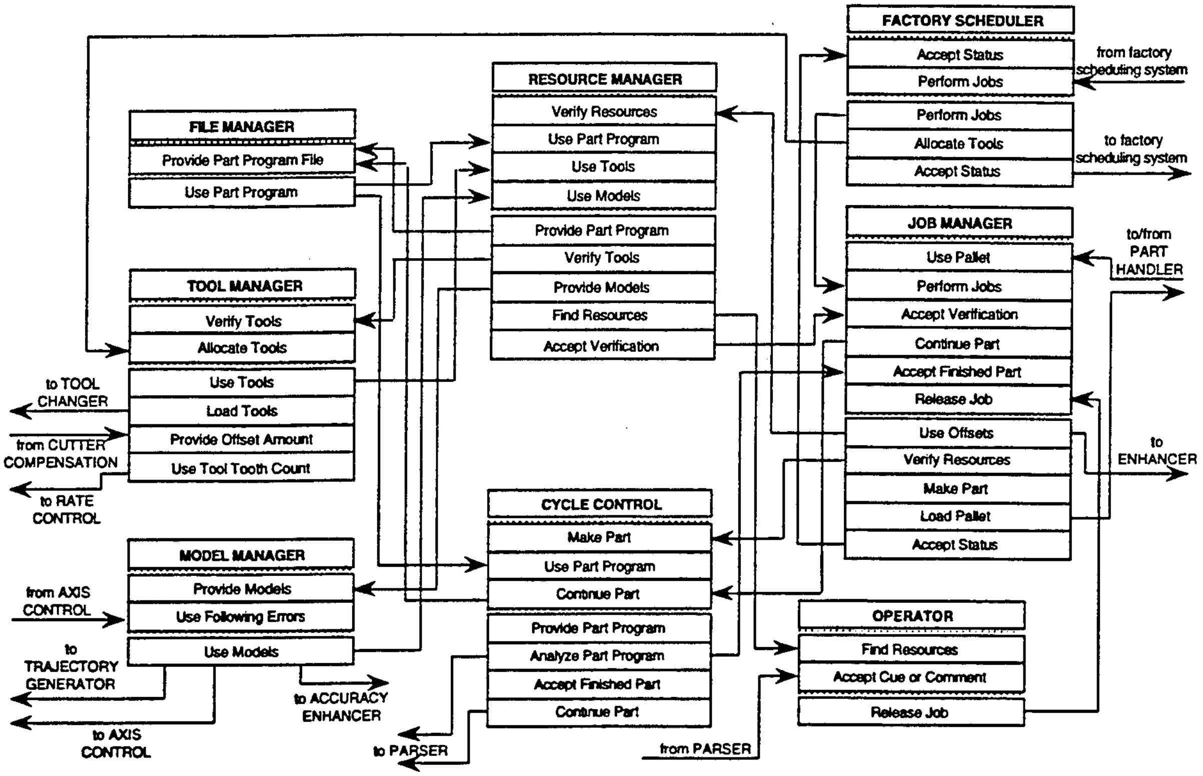


Figure 4.2a Application Architecture Components and Messages—Part 1

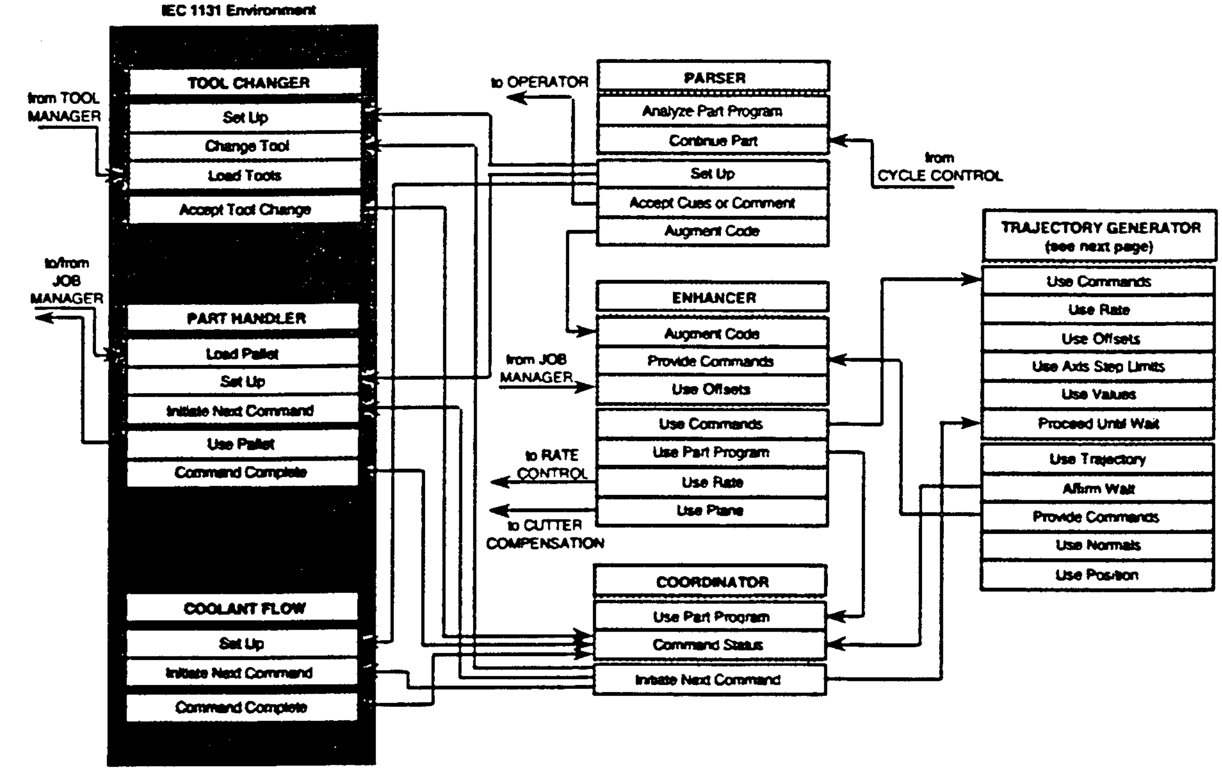


Figure 4.2b Application Architecture Components and Messages—Part 2

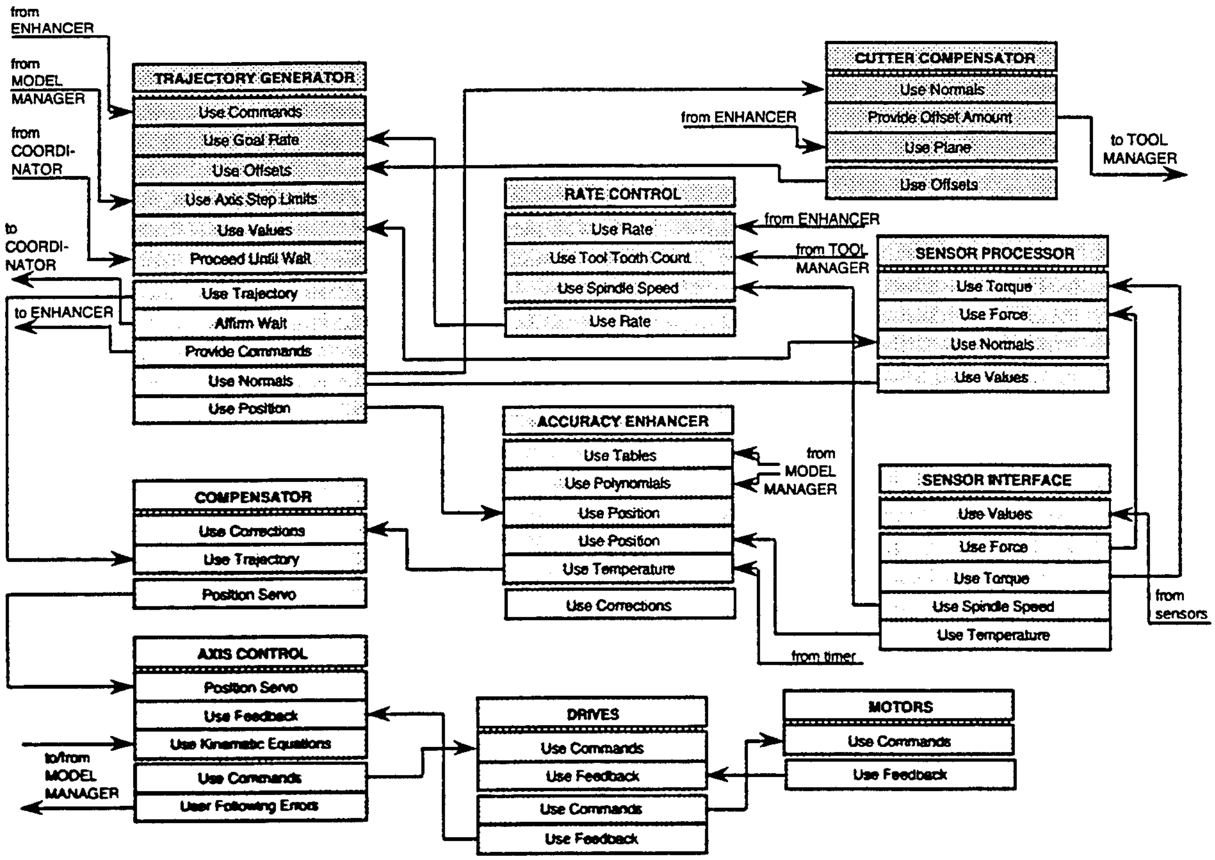


Figure 4.2c Application Architecture Components and Messages—Part 3

In an application architecture, the components embody the responsibilities of the primitive components selected for the application, and the messages represent the flow of resources and products explicitly. However, messages between application architecture components may not be as specific as the interface definitions required for implementation components, e.g., function prototypes in C-language header files. This example will compare representative application architecture messages with their corresponding implementation messages.

In the following discussion, "the controller" refers to an open controller. Component names are all capitalized, e.g., JOB MANAGER. Message names, e.g., Allocate Tools, and the names of information types and structures in the messages, e.g., Tool Information, have leading caps. System elements outside the controller have no capitals, e.g., factory scheduling system.

4.2.1 Factory Interaction

The factory scheduling system resides above the shop floor and sends job instructions to the controller. These instructions are contained in the Perform Jobs message, which consists of a Job

List specifying what is to be done and a Tool List enumerating the tools required from the tool room. The Job List contains a sequence of Job Descriptions designated by job id. Each Job Description identifies a series of pallets to be processed, each having a pallet id, a pallet offset, and a list of parts. A Job Description is organized as follows:

```
job id
    pallet id, pallet offset
            part 1, stock id, part program id, part offsets, beta offset
            part 2, stock id, part program id, part offsets, beta offset
            part 3, stock id, part program id, part offsets, beta offset
    pallet id, pallet offset
            part 1, stock id, part program id, part offsets, beta offset
            part 2, stock id, part program id, part offsets, beta offset
            part 3, stock id, part program id, part offsets, beta offset
    etc.
```

factory scheduling system ' Perform Jobs (FACTORY SCHEDULER, Job List, Tool List)

FACTORY SCHEDULER ' Accept Status (factory scheduling system, Job Status)

The FACTORY SCHEDULER, a component of the controller, is responsible for communicating with the factory scheduling system. It passes the Job List to the JOB MANAGER. The Tool List is passed to the TOOL MANAGER, and the TOOL MANAGER sends the Tool IDs to the TOOL CHANGER. Through the FACTORY SCHEDULER, information about the status of jobs can be shared with the factory scheduling system because the JOB MANAGER maintains an awareness of the jobs that have been released and those which are still queued.

FACTORY SCHEDULER ' Perform Jobs (JOB MANAGER, Job List)

FACTORY SCHEDULER ' Allocate Tools (TOOL MANAGER, Tool List)

TOOL MANAGER ' Load Tools (TOOL CHANGER, Tool IDs)

JOB MANAGER ' Accept Status (FACTORY SCHEDULER, Job Status)

4.2.2 Resource Verification

When the JOB MANAGER receives the Job List, it asks the RESOURCE MANAGER to verify the availability of the needed resources . For each job in the Job List, the JOB MANAGER sends the RESOURCE MANAGER a Job Description, which contains the part program ids and stock ids.

JOB MANAGER ’ Verify Resources (RESOURCE MANAGER, Job Description)

To verify that proper resources are available, the RESOURCE MANAGER asks other managers for applicable information. First, the FILE MANAGER is asked to send the RESOURCE MANAGER a Part Program. The FILE MANAGER responds with the Part Program File, or a Missing Program Error.

RESOURCE MANAGER ’ Provide Part Program (FILE MANAGER, Part Program ID)

FILE MANAGER ’ Use Part Program (RESOURCE MANAGER, Part Program File)

or Accept Error (RESOURCE MANAGER, Missing Program)

After scanning the Part Program File for tool information, the RESOURCE MANAGER asks the TOOL MANAGER if the necessary tools (or adequate substitutes) are loaded. The TOOL MANAGER responds with Tool IDs or a Missing Tool Error.

RESOURCE MANAGER ’ Verify Tools (TOOL MANAGER, Tool Information)

TOOL MANAGER ’ Use Tools (RESOURCE MANAGER, Tool IDs)

or Accept Error (RESOURCE MANAGER, Missing Tool)

The RESOURCE MANAGER may need to further verify the ability of the machine to perform the intended job. It asks the MODEL MANAGER for the Machine Model, Stock Models, Part Models, etc. to compare them to the part program.

RESOURCE MANAGER ’ Provide Models (MODEL MANAGER, Machine Model, Stock Models, Part Models)

MODEL MANAGER ’ Use Models (RESOURCE MANAGER, Model Information)

Should any of these resources be missing, the RESOURCE MANAGER informs the OPERATOR which resources are missing, and it informs the JOB MANAGER that this job did not pass. The JOB MANAGER then holds this job until the OPERATOR indicates something has been done to correct the problem. The JOB MANAGER then resubmits the job for verification to the RESOURCE MANAGER. This cycle continues until the RESOURCE MANAGER indicates to the JOB MANAGER that the required resources are present.

RESOURCE MANAGER ’ Find Resources (OPERATOR, Job Description, Missing Resources)

RESOURCE MANAGER ’ Accept Verification (JOB MANAGER, Resources)

or Hold for (JOB MANAGER, Resource Verification)

OPERATOR ’ Release Job (JOB MANAGER, Job Description)

4.2.3 Job Release

After the OPERATOR has released the job for execution, the JOB MANAGER asks the PART HANDLER to load the first pallet with stock. The PART HANDLER loads a pallet and tells the JOB MANAGER the Pallet ID. The JOB MANAGER now knows which part to do first; it tells the CYCLE CONTROL to start running the first program; and it tells the ENHANCER the offsets (Pallet Offset, Part Offsets, Beta Offset) necessary to locate the part on the pallet. The JOB MANAGER marks the job as "in-progress" and waits for the CYCLE CONTROL to request the next program.

JOB MANAGER ’ Load Pallet (PART HANDLER, Stock IDs)

PART HANDLER ’ Use Pallet (JOB MANAGER, Pallet ID)

JOB MANAGER ’ Make Part (CYCLE CONTROL, Pallet ID, Part ID, Stock ID, Part Program ID)

JOB MANAGER ’ Use Offsets (ENHANCER, Pallet Offset, Part Offsets, Beta Offset)

4.2.4 Part Program Setup

The CYCLE CONTROL has been told to run the next part program, so it asks the FILE MANAGER to provide the program. If the CYCLE CONTROL can't hold the entire program, it periodically asks the FILE MANAGER to provide more code, until the entire program has been processed. The JOB MANAGER is informed when the end of the program has been reached so it can start processing the next part on the current pallet. If the next part uses the same program as the last part, then the file may not need to be accessed from the FILE MANAGER. If there are no more jobs on the current pallet, the JOB MANAGER asks the PART HANDLER to change the pallet, and processing starts on the new pallet.

CYCLE CONTROL ’ **Provide Part Program (FILE MANAGER, Part Program ID)**

FILE MANAGER ’ **Use Part Program (CYCLE CONTROL, Part Program File)**

CYCLE CONTROL ’ **Accept Finished Part (JOB MANAGER, Part ID)**

To continue, CYCLE CONTROL feeds the Part Program to the PARSER, which analyzes each block of code to identify commands pertinent to the attached mechanisms. In our example, the mechanisms are the TOOL CHANGER, the PART HANDLER, and a coolant control device called COOLANT FLOW. The PARSER tells these devices which commands they will be expected to perform when given the appropriate messages. This gives the mechanisms a chance to do any necessary preliminary setups before the actual command is given.

CYCLE CONTROL ’ **Analyze Part Program(PARSER, Part Program)**

PARSER ’ **Setup (TOOL CHANGER, Part Program Commands)**

PARSER ’ **Setup (PART HANDLER, Part Program Commands)**

PARSER ’ **Setup (COOLANT FLOW, Part Program Commands)**

Any comments or operator cues for manual tasks are sent to the OPERATOR from the PARSER. When the OPERATOR completes the task, it informs the JOB MANAGER through the same pathway it used when adding missing resources. The JOB MANAGER tells the CYCLE CONTROL to continue with this cycle, and the CYCLE CONTROL passes the continue message on to the PARSER.

PARSER ’ **Accept Comment (OPERATOR, Explicit Comment)**

PARSER ’ **Accept Cue (OPERATOR, Manual Task Description)**

OPERATOR ’ **Release Job (JOB MANAGER, Job Description)**

JOB MANAGER ’ **Continue Part (CYCLE CONTROL, Part ID)**

CYCLE CONTROL ’ **Continue Part (PARSER, Part ID)**

4.2.5 Code Augmentation

When the interaction with the OPERATOR is concluded, the PARSER feeds the ENHANCER the Part Program, inserting the recipient or destination into each command. The ENHANCER augments motion command code from the PARSER. During the augmentation process, detail blocks or commands are inserted in place of canned cycles. The ENHANCER also translates all coordinate system data since it can derive the exact location of the part through the series of offsets provided by the JOB MANAGER.

PARSER ' Augment Code (ENHANCER, Part Program with Recipients)

Final augmentation done by the ENHANCER relates to the coordinated activities between mechanisms and motion. The ENHANCER inserts coordination points into the command sequence. Say a tool change needs to occur at some point. The ENHANCER would replace the tool change command in the program with a series of commands in the motion stream. These inserted commands might first bring the tool to a chosen location and wait for a continue message. Following the wait command would come commands to move the spindle to another location and wait again for a continue message. Finally, a rapid traverse is inserted after the second wait command.

This augmentation performed by the ENHANCER is highly dependent on the types of mechanisms and their functionality. In some cases, the system startup procedures may require the ENHANCER to ask the mechanisms for specialized augmentations for motion to perform whenever one of their commands is encountered.

ENHANCER ' Provide Augmentations (TOOL CHANGER)

ENHANCER ' Provide Augmentations (PART HANDLER)

ENHANCER ' Provide Augmentations (COOLANT FLOW)

After the ENHANCER inserts coordination points into the command sequence, it gives the COORDINATOR a complete copy of the augmented code, but it gives the TRAJECTORY GENERATOR just the Motion Command Sequence. This Motion Command Sequence consists of a series of points along with Feedrate, Spindle Speed, Acceleration, Deceleration, and any other part program information relevant to motion. The ENHANCER feeds the Motion Command Sequence to the TRAJECTORY GENERATOR in blocks and waits until the TRAJECTORY GENERATOR requests more.

ENHANCER ’ Use Part Program (COORDINATOR, Augmented Part Program)

ENHANCER ’ Use Commands (TRAJECTORY GENERATOR, Motion Command Sequence)

TRAJECTORY GENERATOR ’ Provide Commands (ENHANCER)

4.2.6 Coordination Between Mechanisms

Before covering the details of motion, we consider coordination between mechanisms. The TRAJECTORY GENERATOR does not distinguish between the motion commands from the original Part Program and those commands inserted into the Motion Command Sequence by the ENHANCER to effect mechanism coordination. So, for example, if a tool change is the next operation, the next command the TRAJECTORY GENERATOR sees in the Motion Command Sequence is Wait. The TRAJECTORY GENERATOR informs the COORDINATOR, and then it waits.

COORDINATOR ’ Proceed Until Wait (TRAJECTORY GENERATOR)

TRAJECTORY GENERATOR ’ Affirm Wait (COORDINATOR, Trajectory Generator)

Since the COORDINATOR is in possession of the entire Augmented Part Program, it can understand and coordinate the activities of the mechanisms under control. The COORDINATOR knows that a tool change is next when the TRAJECTORY GENERATOR sends it a Trajectory Generator Waiting message. The COORDINATOR tells the TOOL CHANGER that it is time to initiate a tool change. When the TOOL CHANGER receives Change Tool, it verifies that the new tool is indeed the proper tool change, performs the change, and informs the COORDINATOR that the tool change is complete. The COORDINATOR directs the TRAJECTORY GENERATOR to continue, and the TRAJECTORY GENERATOR proceeds until it encounters the next wait.

COORDINATOR ’ Change Tool (TOOL CHANGER, Tool ID)

TOOL CHANGER ’ Accept Tool Change (COORDINATOR, Tool ID)

4.2.7 Tool Motion

To begin a description of the motion operations, the TRAJECTORY GENERATOR sends the Path Normal Vectors to the CUTTER COMPENSATOR and the SENSOR PROCESSOR. The RATE CONTROL and CUTTER COMPENSATOR components receive messages from the ENHANCER: the RATE CONTROL needs the Rate required by the program; the CUTTER COMPENSATOR needs the Plane Selection.

TRAJECTORY GENERATOR ’ Use Normals (CUTTER COMPENSATOR, Path Normal Vectors)

TRAJECTORY GENERATOR ’ Use Normals (SENSOR PROCESSOR, Path Normal Vectors)

ENHANCER ’ Use Rate (RATE CONTROL, Rate)

ENHANCER ’ Use Plane (CUTTER COMPENSATOR, Plane Selection)

The RATE CONTROL determines the Goal Rate by comparing the program requested Rate to the Tool Tooth Count from the TOOL MANAGER and the Spindle Speed from the SENSOR INTERFACE. (The SENSOR INTERFACE gets its information from the sensors.) The RATE CONTROL supplies the Goal Rate to the TRAJECTORY GENERATOR. The TRAJECTORY GENERATOR would prefer to operate at the Goal Rate, but the machine may be unable to handle the Goal Rate. The TRAJECTORY GENERATOR asks the MODEL MANAGER for the Axis Step Limits, and modifies the rate accordingly.

TOOL MANAGER ’ Use Tool Tooth Count (RATE CONTROL, Tool Tooth Count)

SENSOR INTERFACE ’ Use Spindle Speed (RATE CONTROL, Spindle Speed)

RATE CONTROL ’ Use Rate (TRAJECTORY GENERATOR, Goal Rate)

MODEL MANAGER ’ Use Axis Step Limits(TRAJECTORY GENERATOR, Axis Step Limits)

Since the CUTTER COMPENSATOR must adjust the path according to the specific dimensions and characteristics of the tool, it requests the tool's Offset Amount from the TOOL MANAGER, and calculates the Cutter Offsets and passes them to the TRAJECTORY GENERATOR.

CUTTER COMPENSATOR ’ Provide Offset Amount (TOOL MANAGER, Tool ID)

CUTTER COMPENSATOR ’ Use Offsets (TRAJECTORY GENERATOR, Cutter Offsets)

The last information TRAJECTORY GENERATOR requires is the corrections based on sensor information. The SENSOR PROCESSOR gets the Force and Torque from the SENSOR INTERFACE, which, of course, receives it from the sensors. The SENSOR PROCESSOR uses Force, Torque, and the path normal vectors to correct Position, Rate, and Spindle Speed for the TRAJECTORY GENERATOR.

SENSOR INTERFACE ’ Use Force (SENSOR PROCESSOR, Force)

SENSOR INTERFACE ’ Use Torque (SENSOR PROCESSOR, Torque)

sensors ’ Use Values (SENSOR INTERFACE, Sensor Values)

SENSOR PROCESSOR ’ Use Values (TRAJECTORY GENERATOR, Position Corrections, Rate Corrections, Spindle Speed Corrections)

The TRAJECTORY GENERATOR generates a Normalized Trajectory. It is normalized in the sense of precise time segments. The TRAJECTORY GENERATOR gives the Position to the ACCURACY ENHANCER.

TRAJECTORY GENERATOR ’ Use Position(ACCURACY ENHANCER, Position)

The ACCURACY ENHANCER corrects the trajectory to compensate for predictable variations, which include thermal effects, axis non-linearities, etc. Thermal compensation is governed by accessing the Temperature from the SENSOR INTERFACE which is updated from a temperature sensor. Timing is provided by the system clock or by an internal timer. Tables and Polynomials of precalculated corrections are accessed from the MODEL MANAGER. From all of these information sources, the ACCURACY ENHANCER computes the corrections and passes them on to the COMPENSATOR.

SENSOR INTERFACE ’ Use Temperature (ACCURACY ENHANCER, Temperature)

system clock or timer ’ Use Time (ACCURACY ENHANCER, Timing)

MODEL MANAGER ’ Use Tables (ACCURACY ENHANCER, Tables)

MODEL MANAGER ’ Use Polynomials(ACCURACY ENHANCER, Polynomials)

ACCURACY ENHANCER ’ Use Corrections (COMPENSATOR, Corrections)

The COMPENSATOR receives the Normalized Trajectory from TRAJECTORY GENERATOR, makes corrections, and sends the Servo Command Positions to the AXIS CONTROL. Timing concerns will separate the COMPENSATOR from the ACCURACY ENHANCER: the former must operate continuously to direct tool motion; the latter is brought in when compensation or adjustments to the basic commanded motion path are needed.

TRAJECTORY GENERATOR ’ Use Trajectory (COMPENSATOR, Normalized Trajectory)

COMPENSATOR ’ Position Servo(AXIS CONTROL, Servo Command Positions)

The AXIS CONTROL converts the Servo Command Positions to messages commanding the DRIVES. It accesses the Kinematic Equations from the MODEL MANAGER and stores the Following Errors. If the Following Errors exceed the Following Error Limits, the AXIS CONTROL will shut down the machine and request maintenance. The interfaces from the AXIS CONTROL to the DRIVES and from the DRIVES to the MOTORS are simple position and velocity messages.

MODEL MANAGER ’ Use Kinematic Equations (AXIS CONTROL, Equations)

AXIS CONTROL ’ Use Following Errors (MODEL MANAGER, Following Errors)

AXIS CONTROL ’ Use Commands (DRIVES, Drive Commands)

DRIVES ’ Use Commands (motors, Motor Commands)

motors ’ Use Feedback (DRIVES, Motor Feedback)

DRIVES ’ Use Feedback (AXIS CONTROL, Drive Feedback)

4.2.8 IEC 1131 Environment

The IEC 1131 graphical programming and execution environment provides for creating, debugging, and managing real-time programs that control the TOOL CHANGER, the PART HANDLER, the COOLANT CONTROL, and other discrete devices like the spindle. The environment supports control software coded either in C or in the programmable controller languages defined by IEC-1131 Standard for Programmable Controllers Part 3 - Programming Languages, which include:

- Sequential Function Chart,
- Ladder Diagram,
- Function Block Diagram,
- Instruction List, and
- Structured Text.

4.2.9 Implementation Component Messages

The application architecture components exchange abstract messages, but when a controller implementation is realized from the application architecture, the abstract messages must be translated into an exact specification. For this example the exact message specification takes the form of a C function prototype similar to those found in C header files. The following exact message specifications are taken from the header files produced by the NIST Enhanced Machine Controller project.

<<<<More to go here --- Specific interface definitions from the NIST implementation are matched with selected component messages as shown below.>>>>

The abstract message from the COORDINATOR directing a tool change by the TOOL CHANGER is repeated below:

COORDINATOR ’ Change Tool (TOOL CHANGER, Tool ID)

The NIST header file defines a message type `(nml_io_change_tool_msg_t)` and a function prototype of an interface `(nml_io_change_tool)` to the behavior that implements the tool change. This is a C-language implementation of the message interface to the TOOL CHANGER component. The first parameter of the message structure carries an integer, which is the identifier of the recipient of the message, TOOL CHANGER. The second message parameter is the integer Tool ID. The function prototype specifies the interface to the function that effects the tool change.

```
/* Message ID 313 */

typedef struct {
 int id;   /* Message Recipient's ID */
 int i;    /* Tool ID        */
} nml_io_change_tool_msg_t;
extern int nml_io_change_tool(int i);
```

Note that the application architecture message is made specific when a particular controller is implemented. The implementation interface will vary depending on the language chosen. In this case, ladder code can be used as easily as a C code, but the interface definitions will differ due to the differences in the languages. Also the way implementation components are identified may vary among implementations. However, interoperability can be assured among implementation components only if the interfaces are consistent.

Another example compares the application architecture message interface to the COOLANT FLOW with the NIST interface definition.

COORDINATOR ’ Initiate Next Command (COOLANT FLOW, Tool ID)

The Initiate Next Command message to the COOLANT FLOW component expands into four interface definitions for the NIST implementation component

```
/* Message ID 314 */

typedef struct {
 int id;   /* Message Recipient's ID */
} nml_io_mist_on_msg_t;
extern int nml_io_mist_on();

/* Message ID 315 */

typedef struct {
 int id;   /* Message Recipient's ID */
} nml_io_mist_off_msg_t;
extern int nml_io_mist_off();

/* Message ID 316 */

typedef struct {
 int id;   /* Message Recipient's ID */
} nml_io_flood_on_msg_t;
extern int nml_io_flood_on();

/* Message ID 317 */

typedef struct {
 int id;   /* Message Recipient's ID */
} nml_io_flood_off_msg_t;
extern int nml_io_flood_off();
```

4.3 Implementation

The scenario described in the previous section walks through an example of a specific NGC application architecture for machining. It is important to emphasize that the example is merely one of many realizable controller architectures. This section continues this discussion with several examples to illustrate the process of how the NGC developer implements an abstract application architecture to generate a fully integrated NGC application system.

4.3.1 Integration Process Review

Figure 4.3.1 on the following page illustrates the NGC integration process and emphasizes the design and implementation phases. The integration process flow is first introduced in Section 3.4.1 and is repeated in this example section because of its importance in understanding the general flow and process dependencies. This figure should be referenced throughout the subsequent implementation discussions to provide an overall context for specific steps described in the examples. Only the general process flow is captured. The developer may start from any stage in the process and all feedback paths are intentionally omitted for the sake of simplicity, but these additional entry points and paths may be inferred.

In brief review, two basic process paths are illustrated. Applications development appears as the upper path in the diagram and platform development is the lower path. Although each path may be traversed independently and concurrently, both paths are strongly interrelated and each path potentially feeds the other throughout the integration process. The two paths merge at the final development stages to produce a complete NGC system.

The unifying concept that joins the platform and applications paths is profiles. All NGC implementation components, whether they support platform or applications related functions of the system, are profiled. It is through the application of profiles that many of the implementation decisions are made.

Profiles provide a convenient mechanism for assessing the relative compatibility of complimentary products and for making "apples-to-apples" comparisons of similar products. They may be used to assess if a specific application software component will run on a selected platform implementation. They provide the developer with the information he needs to select the optimal mix of COTS products based on the availability of compatible applications and platform products. Profiles also clarify the "make or buy" choices that are of fundamental importance to developer organizations when they embark on the development of a new product line.

Frequently, the selection of certain implementation options are predetermined. The inclusion of a specific product in a system is often mandated by an organization for a variety of strategic reasons. In such cases, profiles provide the mechanism to compare and contrast available products and options based on previous selections. They expose the alternatives for component interchange and simplify the task of system upgrade.

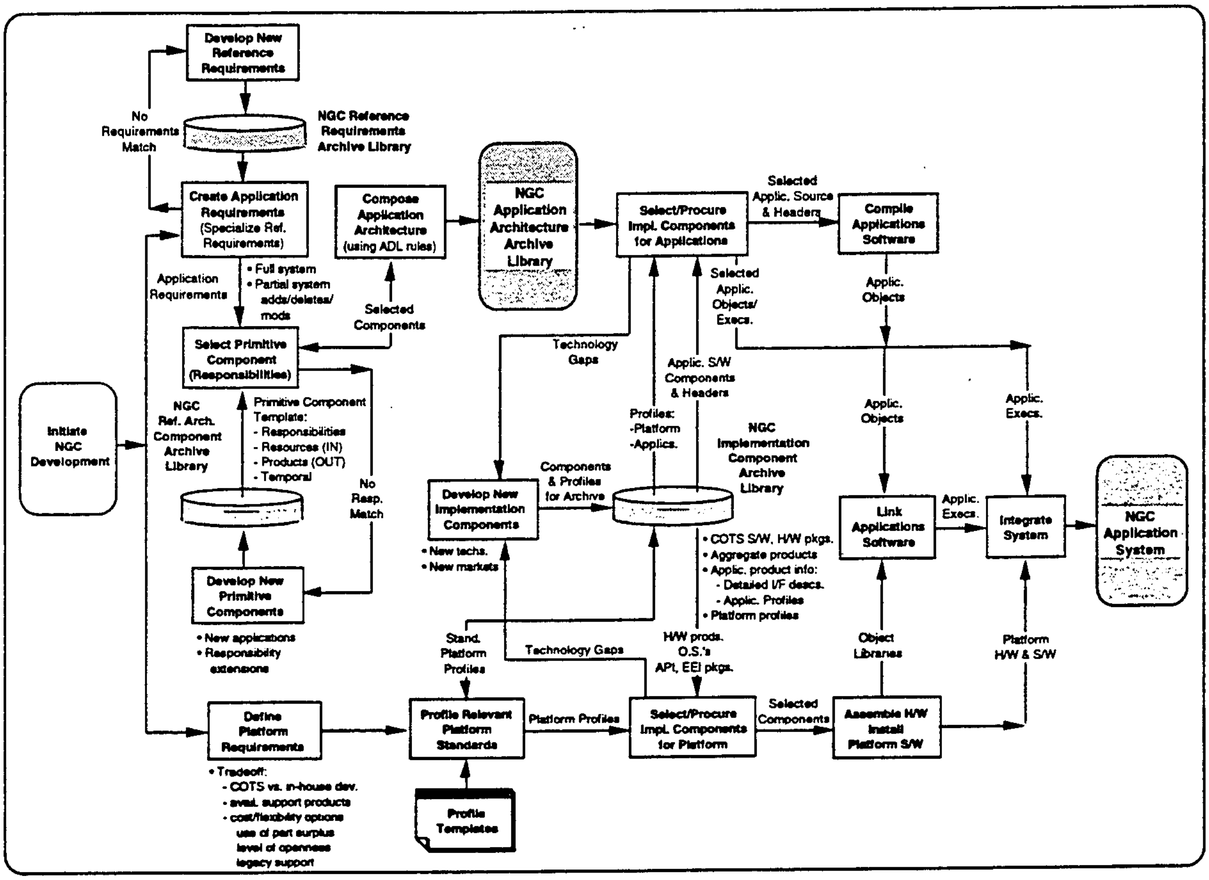

Develop New Reference Requirements
No Requirements Match
NGC Reference Requirements Archive Library
Create Application Requirements (Specialize Ref. Requirements)
Compose Application Architecture (using ADL rules)
NGC Application Architecture Archive Library
Select/Procure Impl. Components for Applications
Selected Applic. Source & Headers
Compile Applications Software
Application Requirements
• Full system
• Partial system adds/deletes/ mods
Selected Components
Selected Applic. Objects/ Execs.
Applic. Objects
Select Primitive Component (Responsibilities)
Technology Gaps
Applic. S/W Components & Headers
Initiate NGC Development
NGC Ref. Arch. Component Archive Library
Primitive Component Template:
- Responsibilities
- Resources (IN)
- Products (OUT)
- Temporal
No Resp. Match
Profiles:
-Platform
-Applics.
NGC Implementation Component Archive Library
Applic. Objects
Applic. Execs.
Develop New Implementation Components
Components & Profiles for Archive
Link Applications Software
Applic. Execs.
Integrate System
NGC Application System
• New techs.
• New markets
• COTS S/W, H/W pkgs.
• Aggregate products
• Applic. product info:
- Detailed I/F descs.
- Applic. Profiles
• Platform profiles
Develop New Primitive Components
• New applications
• Responsibility extensions
Stand. Platform Profiles
Technology Gaps
H/W prods. O.S.'s APt, EEI pkgs.
Object Libraries
Platform H/W & S/W
Define Platform Requirements
Profile Relevant Platform Standards
Platform Profiles
Select/Procure Impl. Components for Platform
Selected Components
Assemble H/W Install Platform S/W
• Tradeoff:
- COTS vs. in-house dev.
- avail. support products
- cost/flexibility options
use of part surplus
level of openness
legacy support
Profile Templates

Figure 4.3.1. NGC Integration Process

Creating a profile for a platform conceived with a specific set of vendor products in mind has the added benefit of surfacing new product solutions. Such solutions may not have been evident during the initial survey or trade studies yet they may offer comparable or even better alternatives than the original solution. Platform implementations that share a common set of profiles support the same set of services and equivalent levels of portability and interoperability by inference. These "compatible" implementations may offer significant cost saving alternatives that are not apparent prior to profiling. Products organized by profile category offer a powerful new dimension to the access of product information for both controller developers and end-users.

It may be useful to assume the existence of a mature NGC implementation component library while walking through the subsequent examples. This library, referenced in Section 3.4.1 and described in further detail in Section 3.4.4, is envisioned to be an on-line catalog of COTS hardware and software and an archive for publicly available NGC software products, all with a supporting set of interactive tools for creation, maintenance, and perusal. It is an information repository for the storage and retrieval of detailed component interface descriptions, standard profiles and templates, and product disclosure information used for NGC-conformant product development.

Early NGC development work must populate this component repository with newly developed components and interactive tools for its access. NGC implementation may be accomplished without the existence of the component library. This is how the first NGC controllers must be built. However, the integration process flow can not be fully appreciated without consideration of the library as an integral part of the process. The component library brings the full advantages of open controllers to the end-users by making available a complete variety of off-the-shelf interchangeable, replaceable components for system upgrade and flexibility. Component "integration" and "reuse" become a major implementation emphasis. Less development effort need be spent on "reinventing the wheel" and efforts may be directed toward developing value-added features and enabling technology areas. The availability of a mature library, fully stocked with NGC-compliant products and user friendly interactive tools for product integration, will bring the major benefits of NGC to the end user. It is therefore critical that early NGC activities focus on developing the component library.

4.3.2 Implementation Example - Machining Center

The integration process is best understood by looking at specific implementations of the application architecture example for a machining center. To illustrate some of the key points, two specific controller implementations are described. Figure 4.3.2-1 portrays the real time (RT) and non-RT platforms, the major communications links, and profiling options of each system implementation. For lack of formal identification of these NGC controllers, they will be referred to as simply Sys-A and Sys-B. It should be recognized that there are many different possible implementations of the architecture example.

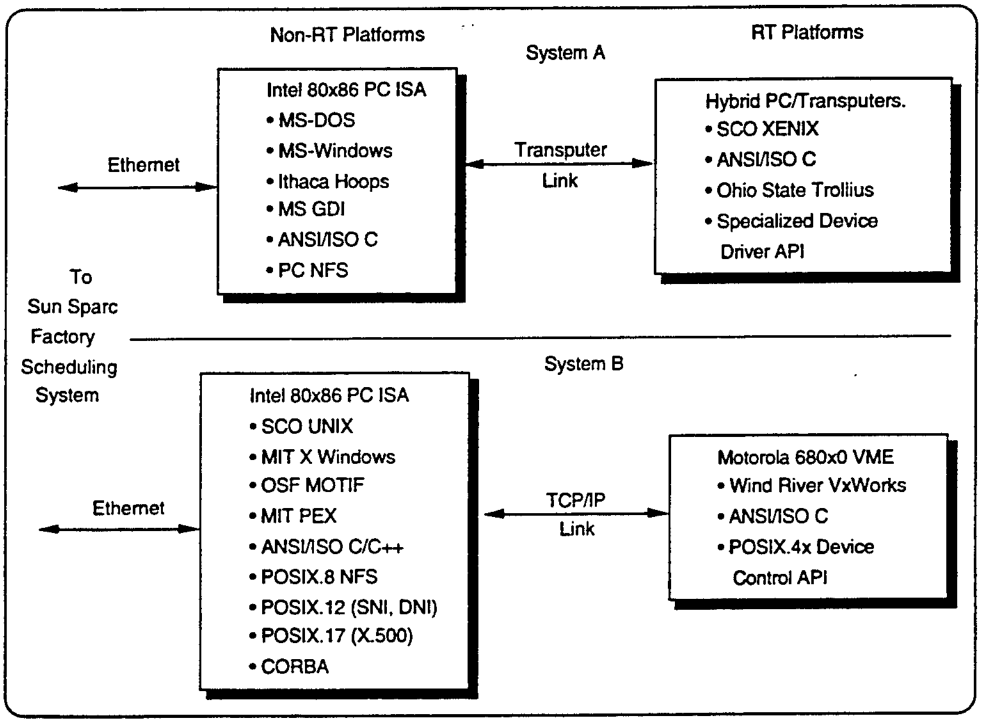


Figure 4.3.2-1 Platform Implementation Examples

Figures 4.3.2-2 and 4.3.2-3 contrast the platform and presentation management dimensions of the API profiles of the two platforms. Both systems are bi-polar, whereby the applications software is physically partitioned onto RT and non-RT system platforms, and both systems use a PC ISA-based (Intel 386/486) front end to support all non-RT functions. However, the operating system and graphical user interface (GUI) for the non-RT platform of Sys-A is a combination of MS DOS and Windows.

The Sys-A developer also opted to include Hoops as a strategic move. It was heard that AutoDesk, the maker of AutoCAD and one of Microsoft's key software vendors for Windows NT, recently acquired Ithaca Software, the maker of Hoops. Hoops also works with Microsoft's Graphical Device Interface (GDI).

The non-RTOS for the Sys-B platform is SCO UNIX running X Windows and Motif. SCO UNIX happens to be POSIX compliant and an XPG3 (X/Open Portability Guide 3) branded product. The X Windows PEX option was also purchased for three-dimensional, PHIGS-compatible graphics.

Non-RT Platforms

Sys-A

Platform:

- □ POSIX
  - □ POSIX.1
  - □ POSIX.4
  - □ POSIX.4a
  - □ POSIX.4b
- □ ANSI C (Applic. Only)
- ☑ Non-POSIX
  - □ UNIX
    - type: ______
  - ☑ DOS/Windows
  - □ OS/2
  - □ NT
  - □ MacOS
  - □ Other:______

Sys-B

Platform:

- ☑ POSIX *(XPG3)*
  - ☑ POSIX.1
  - □ POSIX.4
  - □ POSIX.4a
  - □ POSIX.4b
- □ ANSI C (Applic. Only)
- □ Non-POSIX
  - □ UNIX
    - type: ______
  - □ DOS/Windows
  - □ OS/2
  - □ NT
  - □ MacOS
  - □ Other:______

RT Platforms

Sys-A

Platform:

- □ POSIX
  - □ POSIX.1
  - □ POSIX.4
  - □ POSIX.4a
  - □ POSIX.4b
- □ ANSI C (Applic. Only)
- ☑ Non-POSIX
  - ☑ UNIX
    - type: *SCO XENIX*
  - □ DOS/Windows
  - □ OS/2
  - □ NT
  - □ MacOS
  - □ Other:______

Sys-B

Platform:

- ☑ POSIX
  - ☑ POSIX.1
  - ☑ POSIX.4
  - ☑ POSIX.4a
  - □ POSIX.4b
- □ ANSI C (Applic. Only)
- □ Non-POSIX
  - □ UNIX
    - type: ______
  - □ DOS/Windows
  - □ OS/2
  - □ NT
  - □ MacOS
  - □ Other:______

incr. draft compli-ance

Figure 4.3.2-2 Profile: Platform Dimension

Non-RT Platforms

System-A

Presentation Mgmt.:

Command I/F:

- □ POSIX
  - □ POSIX.2a
  - □ POSIX.2b
- □ Other ______

Prog. I/F:

- □ POSIX
  - □ POSIX.1
  - □ POSIX.4a
- □ ANSI C (Applic. Only)

Graphics:

- □ GKS
  - □ 2D
  - □ 3D
- □ PHIGS
- □ Proprietary
  - ☑ HOOPS
  - □ OpenGL
  - ☑ GDI
  - □ Quickdraw
  - □ Other ______

Windows/GUIs

- □ X Windows
  - □ Xlib-library
  - □ Xt-toolkit
  - □ Non-Xt toolkits
    - □ Andrew
    - □ InterViews
    - □ Other ________
  - □ PEX-PHIGS exten.
  - □ Motif
  - □ Openlook
- □ IEEE 1201
- ☑ Non-X Win/GUIs:
  - □ DOS Text/Graphics
  - ☑ Windows
  - □ OS/2
  - □ Windows NT (Win32)
  - □ Mac Windows
  - □ Other:________

System-B

Presentation Mgmt.:

Command I/F:

- □ POSIX
  - □ POSIX.2a
  - □ POSIX.2b
- □ Other ______

Prog. I/F:

- ☑ POSIX
  - ☑ POSIX.1
  - □ POSIX.4a
- □ ANSI C (Applic. Only)

Graphics:

- □ GKS
  - □ 2D
  - □ 3D
- ☑ PHIGS
- □ Proprietary
  - □ HOOPS
  - □ OpenGL
  - □ GDI
  - □ Quickdraw
  - □ Other ______

Windows/GUIs

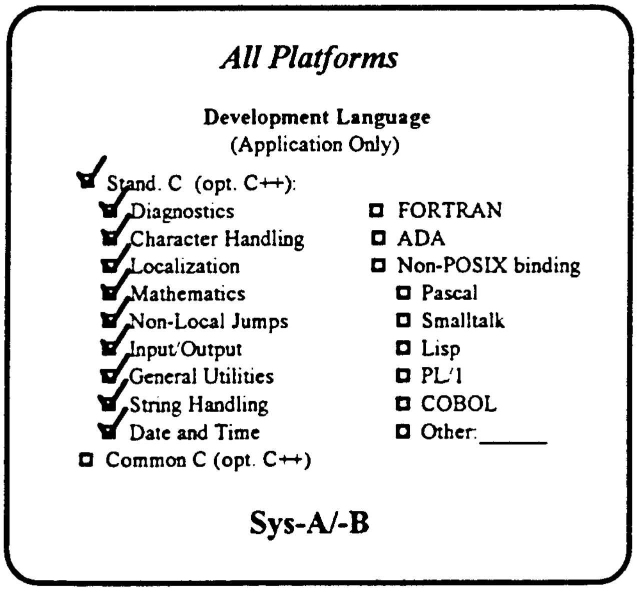
- ☑ X Windows
  - ☑ Xlib-library
  - ☑ Xt-toolkit
  - □ Non-Xt toolkits
    - □ Andrew
    - □ InterViews
    - □ Other ________
  - ☑ PEX-PHIGS exten.
  - ☑ Motif
  - □ Openlook
- □ IEEE 1201
- □ Non-X Win/GUIs:
  - □ DOS Text/Graphics
  - □ Windows
  - □ OS/2
  - □ Windows NT (Win32)
  - □ Mac Windows
  - □ Other ________

Figure 4.3.2-3 Profile: Presentation Management Dimension

Sys-A uses a hybrid PC/transputer-based platform for its real-time functionality and runs SCO XENIX. Although this system is not POSIX compliant, it is a multi-user, multitasking UNIX system optimized for personal computers. Sys-B is VME Motorola 68030/40-based and uses the VxWorks operating system. VxWorks is POSIX.1 compliant and will incrementally support the POSIX.4 real-time extensions to POSIX that are currently at various stages of draft release.

Both systems support full ANSI C. The non-RT platform for Sys-B also supports C++. The Development Language profile is used to show language support (Figure 4.3.2-4).

Files are accessed on the MS-DOS/Windows platform (Sys-A) using ANSI C. The factory scheduling system external to the NGC is a SunSparc. Job lists are accessed using PC NFS which provides access to files on the Sun server transparently, as if they were stored locally on the PC disk (refer to Figure 4.3.2-5). System B, on the other hand, is much more elaborate and leverages the full capability of the POSIX file system. It also conforms with the POSIX.8 Sun NFS profile.



All Platforms

Development Language
(Application Only)

☑ Stand. C (opt. C++):
- ☑ Diagnostics
- ☑ Character Handling
- ☑ Localization
- ☑ Mathematics
- ☑ Non-Local Jumps
- ☑ Input/Output
- ☑ General Utilities
- ☑ String Handling
- ☑ Date and Time

□ Common C (opt. C++)

□ FORTRAN

□ ADA

□ Non-POSIX binding
- □ Pascal
- □ Smalltalk
- □ Lisp
- □ PL/1
- □ COBOL
- □ Other:______

Sys-A/-B

Figure 4.3.2-4 Profile: Development Language Dimension

Non-RT Platforms

Sys-A

□ POSIX
  □ POSIX.1
  □ POSIX.4
  □ POSIX.4a
☑ ANSI C
□ SQL CLI
□ FTAM
  □ IEEE 1238.1
  □ X/Open

Data Mgmt:

□ POSIX.8 (TFA)
  □ Core TFA
  □ Full TFA
  □ Profiled:______
    (RFS,NFS,FTAM,
    PC/DOS, other)
☑ Other: PC NFS

Sys-B

☑ POSIX
  ☑ POSIX.1
  □ POSIX.4
  □ POSIX.4a
☑ ANSI C
□ SQL CLI
□ FTAM
  □ IEEE 1238.1
  □ X/Open

Data Mgmt:

☑ POSIX.8 (TFA)
  □ Core TFA
  □ Full TFA
  ☑ Profiled: *Sun NFS*
    (RFS,NFS,FTAM,
    PC/DOS, other)
□ Other:________

Figure 4.3.2-5 Profile: Development Language Dimension

Figure 4.3.2-6 illustrates the communications and device I/O dimensions of the API profile suite. A data communications link ties the two platforms of each system together. Sys-A makes use of the on-chip communications capability of transputers and an Ohio State University package called Trollius to achieve this inter-platform link. Sys-B uses the TCP/IP network capability of the two UNIX environments and specialized software to transfer data between the two environments.

Combined Platform Summaries (External & Internal Comm.)

Sys-A

Comm.:
□ POSIX.17
□ POSIX.21
□ ANSI C
□ OSI APIs
  □ IEEE 1224
  □ IEEE 1238 0
  □ X/Open
□ CORBA
□ DCE
□ POSIX.12
☑ Other *(Trollius)*

Sys-B

Comm.:
☑ POSIX.17
□ POSIX.21
☑ ANSI C
□ OSI APIs
  □ IEEE 1224
  □ IEEE 1238.0
  □ X/Open
☑ CORBA
□ DCE
☑ POSIX.12
□ Other

RT Platforms

Sys-A

Device I/O:
□ POSIX
  □ POSIX.1
  □ POSIX.4
  □ POSIX.4b
□ ANSI C
☑ Other: *User Drivers*

Sys-B

Device I/O:
☑ POSIX
  ☑ POSIX.1
  ☑ POSIX.4
  ☑ POSIX.4b
□ ANSI C
□ Other:______

Figure 4.3.2-6 Profiles: Communications and Device I/O Dimension

Standard networks are used to link the NGCs to the factory scheduling system. System B has a sophisticated set of communications alternatives. It has both simple and detailed network interfaces (SNI, DNI) for protocol independent interface (PII) that is fully conformant with POSIX.12. To facilitate global naming and remote access over the Internet, POSIX.17-

compliant Directory Services (X.500) software is installed. In addition, in preparation for the wave of soon-to-be-released object-oriented products, a C++ CORBA package, Expersoft's XShell, has been recently installed.

Sys-A uses specialized user-defined drivers to interface with NGC sensors, mechanisms, motor drives, etc. Sys-B makes use of the available POSIX.4x device control API for standard I/O device communications (some of the latter features of Sys-B are exaggerated for illustrative purposes and do not necessarily reflect the present capabilities of the underlying operating system).

Having defined the profiles of the system platform in all dimensions, for both the API and EEI branches of the OSE taxonomy, the developer then begins to select/develop application components that implement various portions of the application architecture.

Figure 4.3.2-7 shows a possible partitioning of the application architecture primitive and aggregate components. The shaded boxes are non-RT components and the unshaded boxes represent RT components. The RT or non-RT nature of the components determine whether they operate on the RT or non-RT platforms of our two system example. Profiles for the software implementations of the architectural components are then compared against profiles of their respective platforms to determine their relative compatibility. Compatible implementations are then integrated into a final NGC system.

Figure 4.3.2-8 is a variation on the possible partitioning of the architecture components. In this example, the cross-hatched boxes indicate hard-RT components. In terms of the platform, this may signify that a specialized COTS motion controller board is incorporated into the system and subsumes all of the component functionality in firmware or hardware. It should be obvious from the numerous examples in this section that the number of implementation options is enormous.

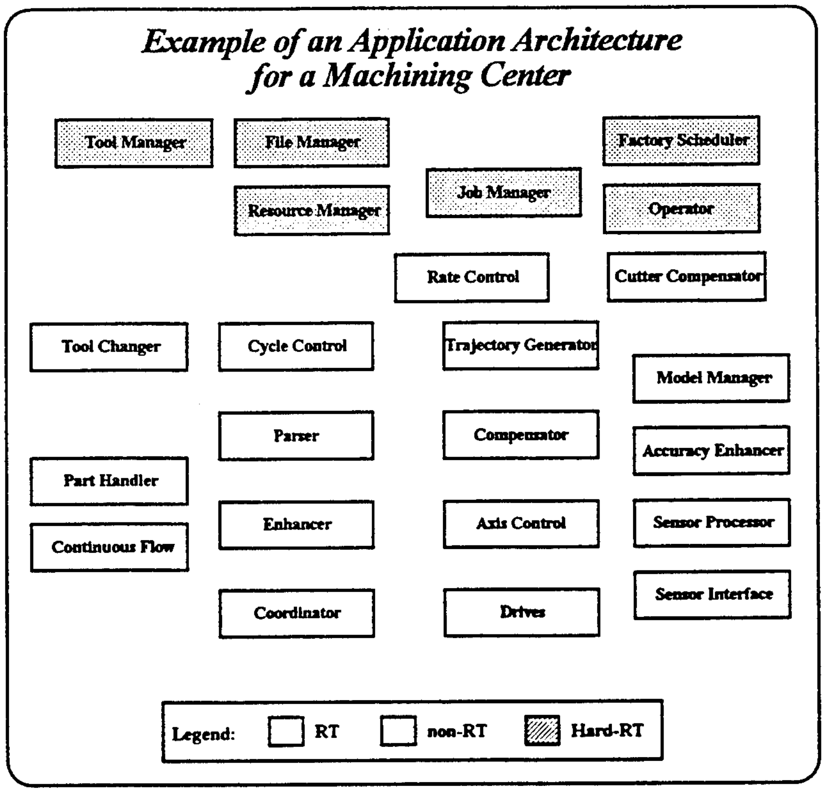


Figure 4.3.2-7 Platform Partitioning, Example 1

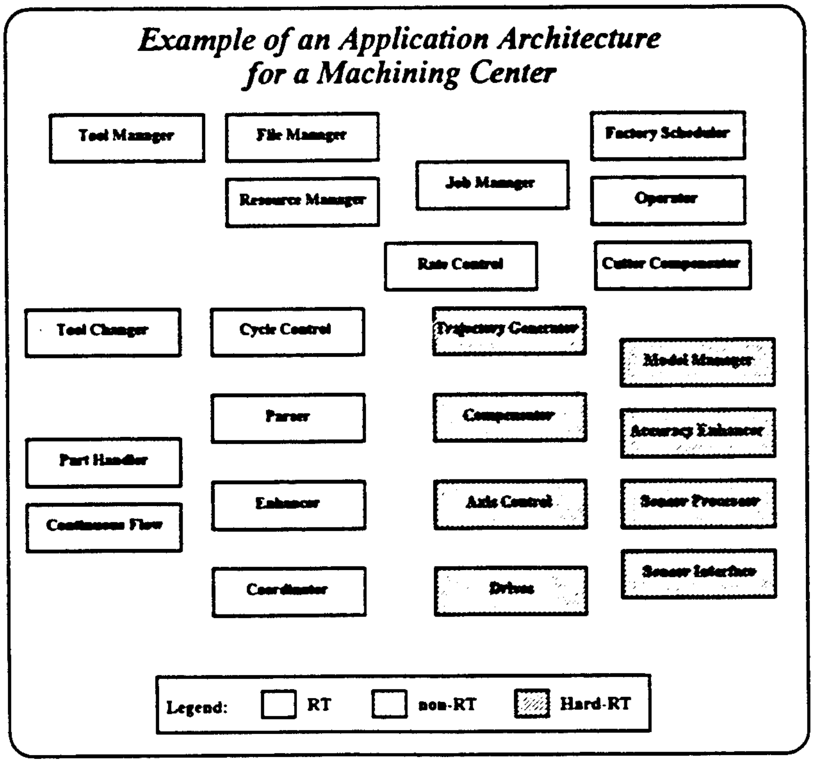


Figure 4.3.2-8 Platform Partitioning, Example 2

4.3.3 Implementation Example—Low End Controller

The previous example describes the profiling process for two platform implementations. It focuses on platform development, the lower branch of the integration process diagram (Figure 4.3.1). The developer then selects or develops application components that are compatible with the profiled platforms to implement various portions of the application architecture. Similarly, the platform is implemented from a selection of off-the-shelf products and/or developed components. This section provides an expanded view of this implementation process.

Many of the implementation concepts presented herein have been introduced in subsections of Section 3.4, Development Process and Implementation. Figure 3.4.2.1 illustrates characteristic differences between architecture development and implementation. For example, profiling and language selection are concepts of implementation, and are not considerations during architectural development and specification. Section 3.4.3 introduces the notion of a specification of "implementation guidelines" to ensure the development of implementation components that are truly portable, interoperable, and interchangeable. Some of these guiding philosophies are described in this section. The "implementation component library", described in Section 3.4.4, offers an on-line central archive of reuseable, publicly available products and information. The existence of this library is viewed to be the enabling technology that will bring NGC from a conceptual vision to a reality.

The implementation process transitions from an architectural specification on paper to a set of implementation components that are integrated into a physical system. An example used for illustrating the process is introduced in Section 3.4.5. This section is a more thorough treatment of the use of profiles, interrelationships with the component library, timing considerations, and other language issues as they relate to the implementation process. Profiling is discussed as a logical continuation of the previous example and is followed with a detailed description of the example of Section 3.4.5. Relevant figures are repeated in this section for convenience.

To complete the profiling story of the previous example, it is useful to consider some of the more subtle implications of platform profiling. Given a mature implementation component library, standard profiles are made available. One possible scenario for platform implementation is to first choose a standard profile from the implementation component library and then select specific COTS products that, when packaged together, meet all the standards requirements of that profile. Platform profiling should not be viewed as that necessary step in the process that is performed after all the implementation choices are made, and specific COTS products are selected. Although this second scenario is certainly a possibility, it does not leverage many of the inherent advantages of profiling.

Profiling is a selection from a large list of relevant standards that identifies the unique level of openness of a controller product. It is a statement of the commitment of the developer to support specific standards in his product lines. In that sense, platform profiles are the fundamental requirements for platform implementation. Many developer organizations will make decisions to commit to a particular set of standards prior to making any specific implementation decisions. In such circumstances, platform profiling naturally precedes any consideration to implementation.

In practice, it is probably more realistic to assume that the platform profile for a specific product will evolve based on a number of mitigating factors. The integration process described in Section 3.4.1 and reviewed in Section 4.3.1 illustrates how profiles "unify" platform and applications development. Profiles clarify "make or buy" choices and provide the developer with the information he needs to select the optimal mix of COTS products based on the availability of compatible applications and platform products. Implicit in this statement is the advantage of profile flexibility in the development process. Profiles that are tailorable based on product availability help to achieve the best solution. Such a solution leverages all available products, meets cost, and satisfies many if not all of the initial objectives for controller openness with little compromise.

Figures 4.3.3-1 thru 4.3.3-3 illustrate the process of implementing a specific application architecture. Figure 4.3.3-1a is a simplification of the detailed architecture and generic implementation example of Figure 4.3.3-1b. For the sake of readability, Figure 4.3.3-1b is enlarged and divided into two diagrams, Figures 4.3.3-2 and 4.3.3-3, to better illustrate the detailed architecture and generic implementation examples respectively. These figures were introduced in Section 3.4.5.

Figure 4.3.3-1a shows the basic design considerations in transitioning from an application architecture to an implementation. In review, an application architecture is a high-level design that is implemented with hardware and software components. Hardware components support platform functions while software components may support either platform or applications level functions. Platform functions are the general-purpose service oriented functions that support software applications.

Implementation components fall into three general categories: platform hardware, platform software, and application software. As evident in figure 4.3.3-1a, these categories may be further subdivided. Although a complete list of categories is beyond the scope of this specification, many of the categories may be inferred from the implementation component library figure of Section 3.4.4. The implementation component library is envisioned to have a directory structure to accommodate all of the various types of implementation components and associated information.

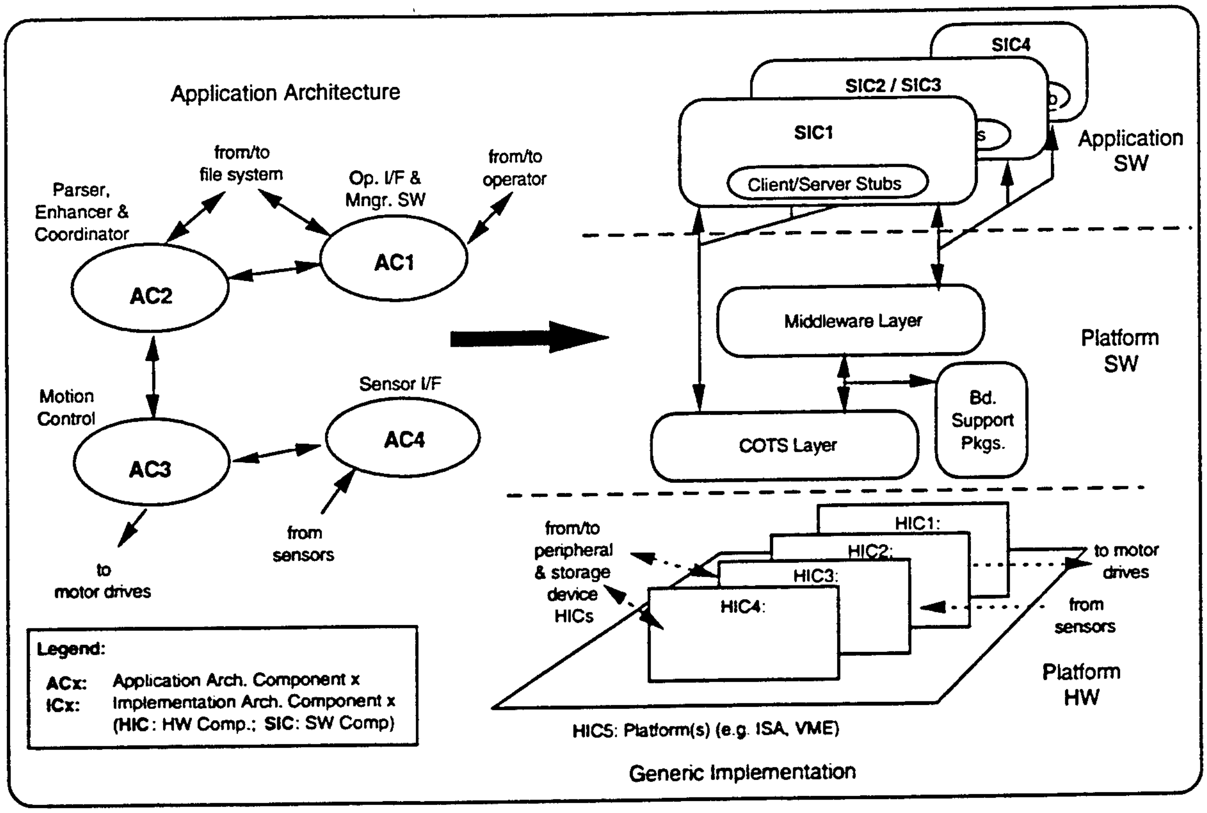


Figure 4.3.3-1a From Application Architecture to Implementation (Simplified)

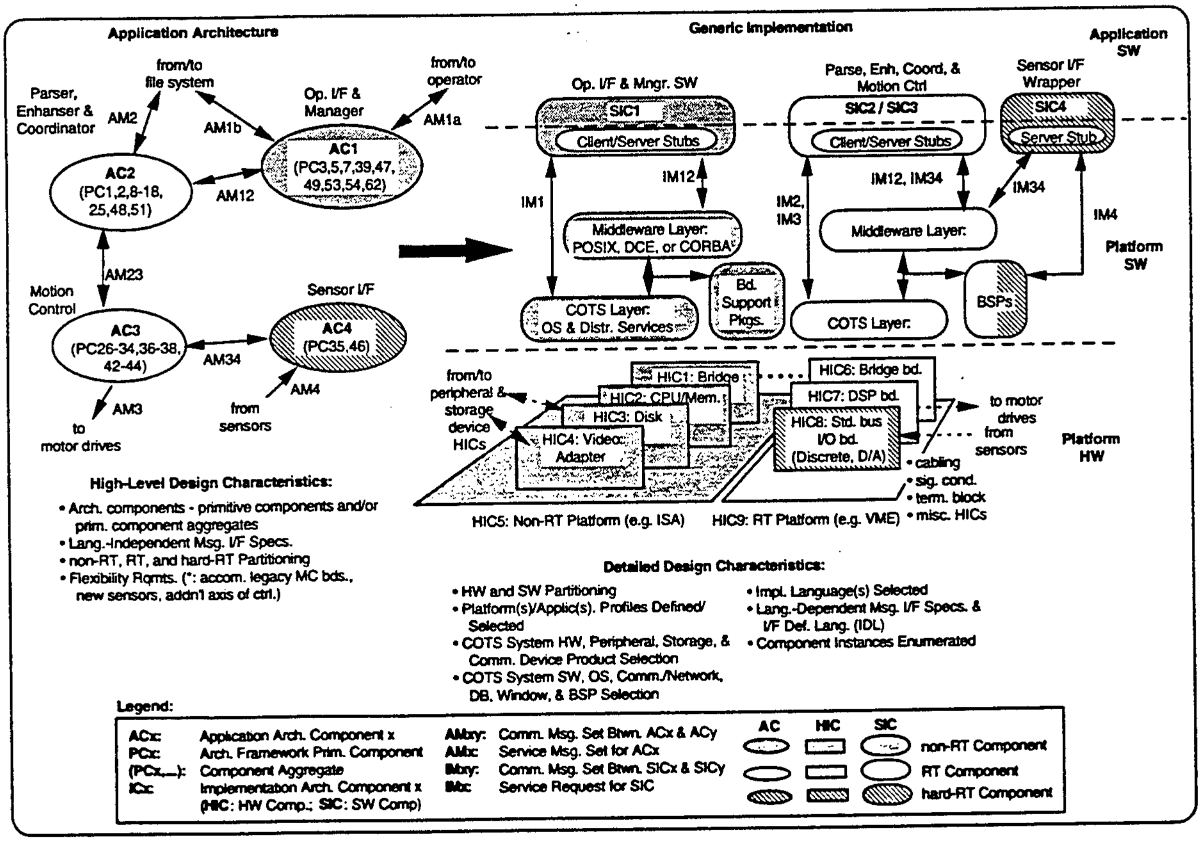

Application Architecture
Generic Implementation
Application SW
Parser, Enhanser & Coordinator
from/to file system
from/to operator
Op. I/F & Manager
AM2
AM1b
AM1a
AC1 (PC3,5,7,39,47, 49,53,54,62)
AC2 (PC1,2,8-18, 25,48,51)
AM12
AM23
Motion Control
Sensor I/F
AC3 (PC26-34,36-38, 42-44)
AC4 (PC35,46)
AM34
AM4
AM3
to motor drives
from sensors
Op. I/F & Mngr. SW
SIC1
Client/Server Stubs
Parse, Enh, Coord, & Motion Ctrl
SIC2 / SIC3
Client/Server Stubs
Sensor I/F Wrapper
SIC4
Server Stub
IM12
IM1
IM12, IM34
IM34
IM2, IM3
IM4
Middleware Layer: POSIX, DCE, or CORBA
Middleware Layer:
Platform SW
Bd. Support Pkgs.
COTS Layer: OS & Distr. Services
COTS Layer:
BSPs
from/to peripheral & storage device HICs
HIC1: Bridge
HIC2: CPU/Mem.
HIC3: Disk
HIC4: Video Adapter
HIC6: Bridge bd.
HIC7: DSP bd.
HIC8: Std. bus I/O bd. (Discrete, D/A)
to motor drives
from sensors
Platform HW
• cabling
• sig. cond.
• term. block
• misc. HICs
HIC5: Non-RT Platform (e.g. ISA)
HIC9: RT Platform (e.g. VME)
High-Level Design Characteristics:
• Arch. components - primitive components and/or prim. component aggregates
• Lang.-Independent Msg. I/F Specs.
• non-RT, RT, and hard-RT Partitioning
• Flexibility Rqmts. (*: accom. legacy MC bds., new sensors, addn'l axis of ctrl.)
Detailed Design Characteristics:
• HW and SW Partitioning
• Platform(s)/Applic(s). Profiles Defined/ Selected
• COTS System HW, Peripheral, Storage, & Comm. Device Product Selection
• COTS System SW, OS, Comm./Network, DB, Window, & BSP Selection
• Impl. Language(s) Selected
• Lang.-Dependent Msg. I/F Specs. & I/F Def. Lang. (IDL)
• Component Instances Enumerated
Legend:
ACx: Application Arch. Component x
PCx: Arch. Framework Prim. Component
(PCx,...): Component Aggregate
ICx: Implementation Arch. Component x (HIC: HW Comp.; SIC: SW Comp)
AMxy: Comm. Msg. Set Btwn. ACx & ACy
AMx: Service Msg. Set for ACx
IMxy: Comm. Msg. Set Btwn. SICx & SICy
IMx: Service Request for SIC
AC
HIC
SIC
non-RT Component
RT Component
hard-RT Component

Figure 4.3.3-1b From Application Architecture to Implementation (Detailed)

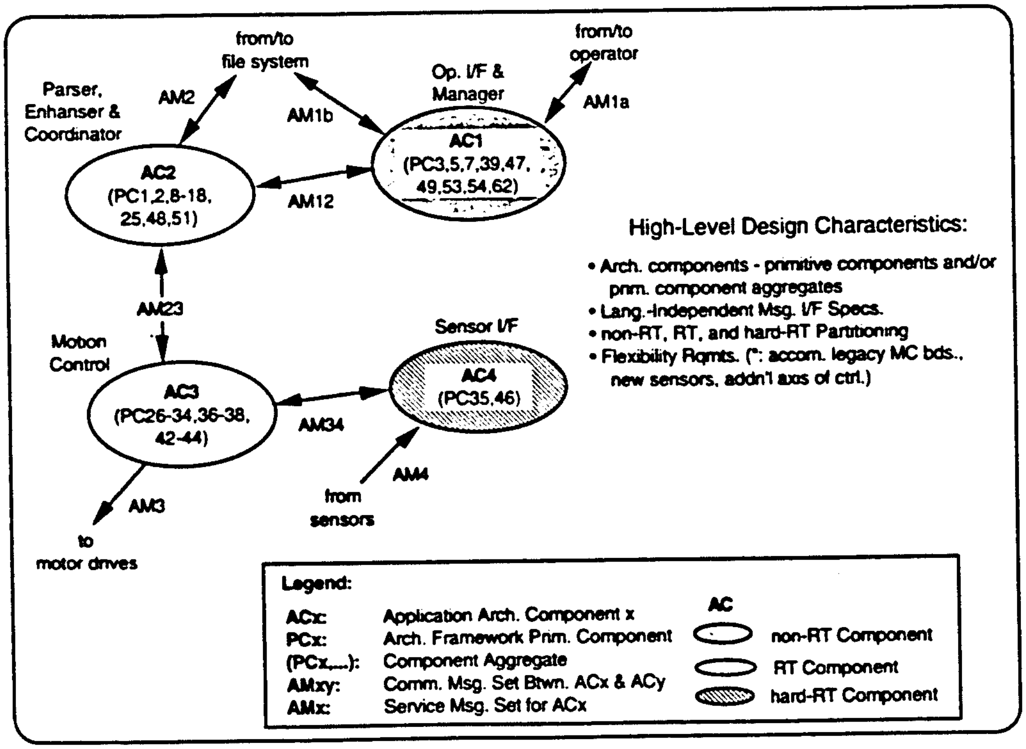

from/to file system
from/to operator
Parser, Enhanser & Coordinator
AM2
AM1b
Op. I/F & Manager
AM1a
AC1 (PC3,5,7,39,47, 49,53,54,62)
AC2 (PC1,2,8-18, 25,48,51)
AM12
AM23
Motion Control
Sensor I/F
AC3 (PC26-34,36-38, 42-44)
AC4 (PC35,46)
AM34
AM4
from sensors
AM3
to motor drives
High-Level Design Characteristics:
• Arch. components - primitive components and/or prim. component aggregates
• Lang.-Independent Msg. I/F Specs.
• non-RT, RT, and hard-RT Partitioning
• Flexibility Rqmts. (*: accom. legacy MC bds., new sensors, addn'l axis of ctrl.)
Legend:
ACx: Application Arch. Component x
PCx: Arch. Framework Prim. Component
(PCx,...): Component Aggregate
AMxy: Comm. Msg. Set Btwn. ACx & ACy
AMx: Service Msg. Set for ACx
AC
non-RT Component
RT Component
hard-RT Component

Figure 4.3.3-2 Application Architecture

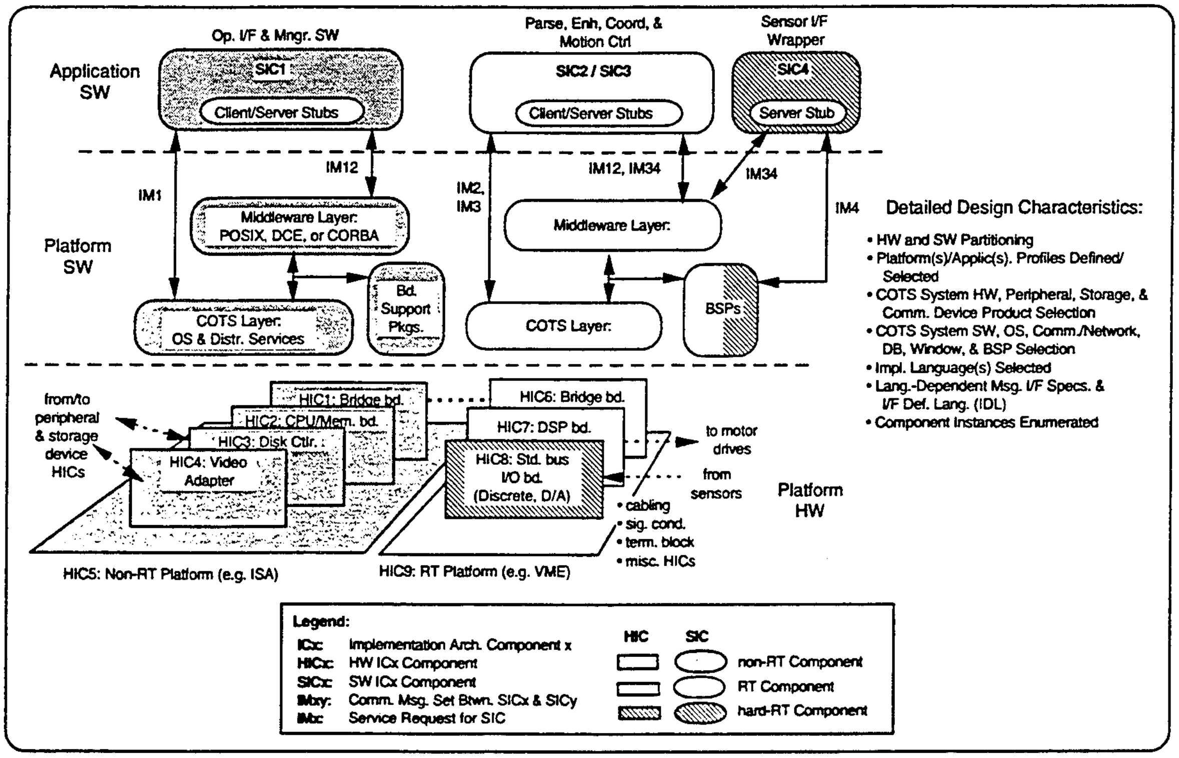

Op. I/F & Mngr. SW
Parse, Enh, Coord, & Motion Ctrl
Sensor I/F Wrapper
Application SW
SIC1
Client/Server Stubs
SIC2 / SIC3
Client/Server Stubs
SIC4
Server Stub
IM12
IM12, IM34
IM34
IM1
IM2, IM3
IM4
Platform SW
Middleware Layer: POSIX, DCE, or CORBA
Middleware Layer:
Bd. Support Pkgs.
COTS Layer: OS & Distr. Services
COTS Layer:
BSPs
Detailed Design Characteristics:
• HW and SW Partitioning
• Platform(s)/Applic(s). Profiles Defined/ Selected
• COTS System HW, Peripheral, Storage, & Comm. Device Product Selection
• COTS System SW, OS, Comm./Network, DB, Window, & BSP Selection
• Impl. Language(s) Selected
• Lang.-Dependent Msg. I/F Specs. & I/F Def. Lang. (IDL)
• Component Instances Enumerated
from/to peripheral & storage device HICs
HIC1: Bridge bd.
HIC2: CPU/Mem. bd.
HIC3: Disk Ctlr.
HIC4: Video Adapter
HIC6: Bridge bd.
HIC7: DSP bd.
HIC8: Std. bus I/O bd. (Discrete, D/A)
to motor drives
from sensors
Platform HW
• cabling
• sig. cond.
• term. block
• misc. HICs
HIC5: Non-RT Platform (e.g. ISA)
HIC9: RT Platform (e.g. VME)
Legend:
ICx: Implementation Arch. Component x
HICx: HW ICx Component
SICx: SW ICx Component
IMxy: Comm. Msg. Set Btwn. SICx & SICy
IMx: Service Request for SIC
HIC
SIC
non-RT Component
RT Component
hard-RT Component

Figure 4.3.3-3 Generic Implementation

Platform hardware categories include system hardware, peripheral devices, storage devices, communications devices, etc. Each of these categories are further subdivided. For example, the system hardware category includes PCs, chassis, backplanes, board-level products, and standard bus I/O. Platform software categories closely resemble those that appear in the OSE taxonomy. Operating systems and system software, communications/network packages, database packages, and window toolkits are example platform software categories.

The commercial availability of a product, or lack thereof, brings another level of dimensionality to the categorization of implementation components. Many platform profiles may be implemented using COTS products. However, where there is a deficiency in satisfying profile requirements with commercially available products, middleware must be developed. Middleware is strictly defined as a term associated with distributed systems and communications and is a layer of code that sits between the O.S. and the application. Middleware is more loosely defined in the context of NGC to represent software components that are developed expressly for bridging COTS interfaces to standard APIs referenced by specific profiles.

COTS hardware products include special-function boards, many of which provide services in support of embedded applications. Board support packages for board-level products and library packages that provide software interfaces for cross-development environments are often purchase options made available through third party sources. These support products are designed to minimize the difficulty of interfacing special-function boards and simplify their integration into a system. Such products today offer a variety of levels of software support and many do not necessarily adhere to open standards. They may be leveraged as with all other COTS platform products but may require a middleware component layer to satisfy profile API requirements.

At the applications level, Figure 4.3.3-1a illustrates a correlation between architecture components and implementation components that is not a one-to-one mapping. The SIC2/SIC3 component implements all the responsibilities that are defined for two components, AC2 and AC3, at the architecture level. SIC2/SIC3 is an example implementation component that represents the packaging or bundling implementation strategy of one developer. The client/server stubs are used for implementing transparent peer-to-peer communications and are typically object components that only have meaning to a link editor during the applications build process. Therefore, they are shown as an integral part of application component implementations. Client/server stubs have significance in CORBA and DCE implementations. They are generated by the interface definition language (IDL) compiler and implement remote procedure call mechanisms (RPCs). Refer to the discussion on CORBA and DCE IDL and RPCs in Section 3.3.2.1.6, Communications Services, for further details on client/server stub generation and the link process.

Client/server stubs, COTS products, board support packages and the anticipated set of component middleware products all have a place in the implementation component directory structure. From this viewpoint, they may all be considered implementation components. To keep the notation simple, these components are not numerically designated in the example implementation figures.

External message interfaces are reflected in the application architecture example as arrows that connect to a component from a single end with the other end disconnected. These messages are numerically designated and correlate with platform service requests at the implementation level. Intercomponent message sets at the architecture level are also numerically designated and translate into corresponding communications messages in the resulting implementation.

Figure 4.3.3-1b and its corresponding enlargements, Figures 4.3.3-2 and 4.3.3-3, expand the simplified example with an introduction to timing and multi-platform considerations. Also, the expanded application architecture example illustrates component aggregation. The primitive components of a specific application component aggregate are numerically identified.

The controller responsibilities, as defined by the primitive components included in this example, capture many of the basic requirements of the NGC controller at the low-end. This example concentrates on low-end requirements since it is anticipated that early NGC implementations will naturally have this focus. As technological advancements make higher-end capabilities more readily available in off-the-shelf products, it is believed that these capabilities will then be incorporated into future NGC products.

To illustrate the low-end capabilities of this architecture, consider the operator interface and manager component, designated as AC1 in Figure 4.3.3-2. The operator has responsibility for determining and sequencing each part program and its coordinate offsets for a given job (PC3) and ensures the availability of resources required by the jobs (PC5). Job sequences are initiated manually (PC62). Status is obtained interactively (PC7), and startup/shutdown procedures are also manually initiated (PC39,47). Unlike the the first implementation example, there is no support in this example for a factory scheduling system interface. Part programming and customization (PC53,54) is performed interactively using the manual data interface of the controller console. These operator responsibilities could be automated, as described by the first implementation example, but are instead handled by the operator interactively in this example. This type of stand-alone functionality, with perhaps a serial port interface for part program download, is typical of low-end controllers today.

The file access (and other service support) implicit in the operator interface and management functions is reflected abstractly in the application architecture as AM1b service message set. Display and keyboard management to/from the physical operator console is depicted as AM1a. These translate into IM1 service requests in the generic implementation example (Figure 4.3.3-3). Similarly, the AM12 communications message set between the operator interface component (AC1) and the parser, enhancer & coordinator component (AC2) at an architecture level maps to the IM12 message set of the implementation example.

Analogous to the display, keyboard and file management services that are required to support the operator I/F component, similar I/O services are required for implementing interfaces to motor drives and sensors. These are reflected as IM3 and IM4 service requests in the generic implementation and directly correlate with AM3 and AM4 message sets shown in the application architecture example. It is left as an exercise to the reader to use the appendices contained in this specification to research the definition of the primitive components identified in each of the other application architecture component aggregates and correlate message sets in the application architecture with specific communications messages and services in the generic implementation.

Applying the hard knock experience of implementing systems is crucial in specifying an architecture that may be feasibly implemented. System prototyping should play an essential role in the architectural specification process for its validation. This is especially important when the practicality of a new architectural approach remains unproven due to the lack of existing, working implementations.

Timing considerations required for implementation will naturally influence the aggregation of components specified at an architectural level. An architecture is a high-level design and can not be divorced from timing issues involved during implementation. The implementation of the architecture must be anticipated. Primitive components will be grouped into architectural aggregates such that their respective responsibilities will have "like" timing requirements in their implementation.

However, with the exception of considerations to the grouping of primitive components, timing issues need not be addressed at the architectural specification level. Most considerations to timing should be left to the details of implementation. This approach leverages the most benefit from the portability and interoperability characteristics inherent in NGC-compliant implementation component software. The timing of portable applications software will often only be limited by the capabilities of its underlying platform and supporting services. The timing characteristics of the applications software will also vary dramatically from platform to platform.

Implementation component aggregates having similar timing requirements may ultimately co-reside on the same platform when they are implemented. The generic implementation example represents a multi-platform solution that has both RT and non-RT hardware platform components that are interconnected via a bridge. The two platforms have the effect of partitioning the applications software components such that SIC1 resides on the non-RT platform and the SIC2/SIC3 and SIC4 components reside on the RT platform. This is a realistic partitioning since most operator interface functions are easily managed by a non-RT support environment. For example, a one second response time is an acceptable response time to an operator (though possibly annoying), but is much too slow and potentially disastrous for sensor update to a motion control servo while cutting a part.

The RT platform also integrates a special-purpose sensor I/O board that is interfaced to the other components of the system via the SIC4 sensor I/F wrapper component. This particular component of the system takes on hard-RT timing characteristics because of the unmodifiable, applications-specific logic embedded in firmware on the product wrapped by SIC4.

5.0 SUMMARY AND RECOMMENDATIONS

5.1 Summary

This document has described a structure for the development of open architecture systems for advanced manufacturing applications. This structure accommodates both a wide range of software applications while at the same time permitting the use of a variety of potential platform solutions. The admissible platform solutions allow for a wide range of variability with respect to processor selection, buss selection, operating systems, and device interface and communication devices and standards.

The process of NGC system development has been formalized with respect to development of the overall system structure that satisfies system requirements through the use of a component based approach that allows the design process to begin at a level of abstraction that focuses on responsibilities and information flow within the system. This flow from the selection of primitive components from a reference architecture to the construction of an application architecture was chosen because of it's consistency with what is becoming common practice in the software community. This follows the same basic practices set forth in the Advanced Research Project Agency (ARPA) Domain Specific Software Architecture (DSSA) program.

The equally challenging issue of the system platform has been addressed through the concept of profiles, a concept adopted from the IEEE Portable Operating System Interface (POSIX) standard. The profile is essentially a means of declaring a specific implementation of a system by selecting standards and options from a large set of possible choices. The strategy of profiles was chosen as an alternative to selecting a small set of standards and conventions from a very large set and attempting to force this on the overall community. As stated in the introduction of this document, it will be the inevitable market forces that will result in a narrowing of design options for NGC systems and the emergence of a small set of standard implementations such as the PC and the Macintosh in the personal computer arena. Realistically, any attempt to guess which set of standards will win with respect to processors, busses, operating systems, device interfaces, etc. is futile. The profiling concept necessarily leads to the possibility of an extremely large number of NGC systems, at least initially. In some ways this can be viewed as no different from the current situation. There is, however, a crucial difference.

Profiling allows users to determine, with little effort, the degree of openness in the system that is being purchased. By declaring the fundamental structure of the system, the path is opened for other vendors to produce system elements that can easily be integrated into the resulting standardized systems. The ultimate benefit is to the user who, using the system profile, can draw on a wide range of related products for system expansion and improvement.

The profile system that has been developed addresses both the Application Program Interface (API) as well as the External Environment Interface (EEI). (Again, this is similar to POSIX). Taken together, the application architecture and the profiling structure form a firm foundation for NGC system development; they do not, however, represent a complete methodology that would support immediate development of a national, commercial NGC product base. There are two elements that are missing from the overall system that would facilitate the adoption of the NGC approach; *fully populated and widely accessible implementation component libraries* and a family of *tools for system design, integration, and validation.* Unfortunately, the difficulty lies in developing the initial libraries and tool set. Once initiated, it seems clear that they will themselves become important commercial products.

At the implementation component level (see section 3.3 and 3.4) abstraction is finally lost and issues of system interoperability are directly addressed. The implementation library, (or repository in DSSA terms), consists of elements of software that are actually linked and compiled into the final system. From the standpoint of the end user the truest manifestation of NGC is in the availability of implementation components, not in the abstract but necessary process that results in these components. It is not an over simplification to say that the implementation components will be the "currency" of the evolutionary NGC system.

Even with the availability of libraries of implementation components, computer based tools will be essential to the propagation and success of this technology. The tool set is what will be responsible for institutionalizing the full NGC process because it will provide a quantum leap in the way systems are developed and modified in the same way Windows and Macintosh interfaces represented a quantum leap with respect to humans interfacing with computers.

5.2 Synergistic Activities

There are a large number of ongoing activities that have the potential for filling in the remaining pieces of the NGC structure. These activities include controller development projects as well as more generalized real-time, distributed architecture projects. The controller projects provide much of the necessary detail needed to arrive at a complete and useful NGC structure. Specifically, they provide a set of primitive components, requirements definition, application architectures, but most importantly, specific messaging structures and an initial set of implementation components. It is also important to note that the identified controller projects are not limited to strictly machine tool control architectures. In addition to general machine tool applications, there are ongoing projects that also address inspection (coordinate measurement) and robotics (both autonomous and teleoperated.) Similarly, a number of ongoing real-time, distributed control projects are focusing on many of the "domain independent" communication issues that are important to NGC as well as the tool structure for both building and maintaining NGC-like systems.

The controller projects that could directly contribute to completing the overall structure of the NGC system include the NIST Enhanced Machine Controller, (EMC), the Unified Telerobotic Architecture Project, (UTAP), sponsored by the Aircraft Directorate at the San Antonio Air Logistics Center, (SA-ALC), with participation by NIST and JPL, the Air Force ManTech Title III activity, the Department of Energy Technologies Enabling Agile Manufacturing, (TEAM), as well as others. Even the European version of NGC, Open System Architecture for Controls within Automation Systems, (OSACA), is worthy of further study with respect to what it can offer a long term NGC activity. All of the programs described above offer elements that can be directly incorporated into the evolving NGC structure that was described in section 3.4. The NGC structure benefits from the standpoint of additional depth and completeness. Each of the individual programs also benefits from the ability to capture program legacy in the consistent framework of NGC.

Similarly, there are a number of projects that are addressing many of the domain independent and tools issues that are crucial to an effective and complete NGC system. The ARPA DSSA program is currently felt to be the most important among these programs. This program is working many of the issues essential to the advanced manufacturing domain. Issues that are currently at the forefront of this project include requirements, system structural development, testing, and validation of systems. Most importantly, DSSA is focusing on a consistent set of tools that will make opên, reusable systems a reality for all real-time control system applications. Even programs seemingly unrelated to NGC and advanced manufacturing such as the Reusable Software Architecture for Spacecraft, (RSAS), sponsored by the Air Force Phillips Lab, are in the process of working issues and developing tools that are directly relevant to NGC. Finally, the National Information Infrastructure Program (NIIP) is of direct interest to the NGC development process. This program has the goal of establishing and unifying many of the key information transfer standards necessary for NGC.

5.3 Recommendation

An effective overall structure has been developed for the NGC vision for advanced manufacturing systems. This structure facilitates the development of open, interoperable controllers for all advanced manufacturing operations. At the same time, the structure allows for an unprecedented growth in the vendor base that can field products relevant to this area. It is not an overstatement to say that this structure *could* serve as the foundation for a revolution in advanced manufacturing systems similar to that which has occurred in personal and workstation computing over the past ten years.

To make the NGC vision a reality, it will be necessary to continue the development of the overall system structure. This is not an overly daunting task because, as pointed out in the earlier parts of this section, the activity should focus on capture of technology from ongoing activities rather than additional development. It is safe to say that all the major elements required to complete the NGC system either currently exist , (e.g. NIST EMC), or are in the process of being developed (e.g. ARPA DSSA, Title III, TEAM, NIIP). The integration of these available elements is not an especially complex task, but it does require that some type of central repository site be identified and solid interfaces with these programs be developed.

6.0 GLOSSARY OF TERMS

ADL Rules: TBD

Aggregate Component: Two or more primitive components retaining notion of responsibilities and message requirements.

Application: The use of capabilities (services/ facilities) provided by an information system specific to the satisfaction of a set of user requirements.

Application Architecture: The result of applying specific application requirements to the selection of Reference Architecture components. Still an abstraction but messages become more specific. Many possible configurations of Reference Architecture components for any given applications.

Application Environment Profile (AEP): A profile, specifying a completed and coherent specification of the Open Systems Environment (OSE), in which the standards, options, and parameters chosen are necessary to support a class of applications.

Application Platform: A set of resources that support the services on which application software will run. The application platform provides services at its interfaces that, as much as possible, make the specific characteristics of the platform irrelevant to the application software.

Application Program Interface (API): The interface between the application software and the application platform across which all services are provided. The API is primarily in support of application portability, but system and application interoperability are also supported by a communication API.

Application Software: Software that is specific to an application and is composed of programs, data and documentation.

Application System: One of many possible implementations of a specific application architecture. The computing platform is fully specified by a profile published by the control builder. Software module granularity conforms to chosen boundaries of the application architecture.

Architecture Description Language (ADL): TBD

Base Standard: A standard or specification that is recognized as appropriate for normative reference in a profile by the body adopting that profile, but is not a profile itself.

Component Profile: A profile that is made up of a defined subset of a single standard.

Conformance: Action or behavior in correspondence with current customs, rules, or styles. In particular, behavior in correspondence with SOSAS rules, requirements, and styles and documented by a SOSAS-coconsistent standardized profile.

External Environment Interface (EEI): The interface between the application platform and the external environment across which information is exchanged. The EEI is defined primarily in support of system and application interoperability. The primary services at the EEI comprise of human/ computer interaction services, information services, and communication services.

Hardware: Physical equipment used in data processing as opposed to programs, procedures, rules, and associated documentation.

Implementation Component: A hardware, software, or human entity that fulfills a specific set of responsibilities with a specific interface. The form of implementation is not specified.

Interface: The shared boundary between two functional units defined by functional characteristics and other characteristics, as appropriate.

Interchangeability: A characteristic of system components that makes it possible to replace one component with another component of equivalent functionality made by a different vendor.

Interoperability: The ability of two or more systems to exchange information and to mutually use the information that has been exchanged.

Open System Application Program Interface: A combination of standards based interfaces specifying a complete interface between an application program and the underlying application platform.

Open System: A system that implements sufficient open specifications for interfaces, services, and supporting formats to enable properly engineered applications software: to be ported with minimal changes across a wide range of systems, to interoperate with other applications on local and remote systems, and to interact with users in a style that facilitates user portability

Open Systems Environment (OSE): The comprehensive set of interfaces, services, and supporting formats, plus user aspects for interoperability or for portability of applications, data, or people as specified by information technology standards and profiles.

Performance: A measure of a computer system or subsystem to perform its functions; for example, response time, throughput, number of transactions per second.

Portability: The ease with which software can be transferred from one information system to another.

Primitive Component: A component with a single responsibility defined as an abstraction (no code) with an abstract message interface and architecture constraints defined.

Process: A address space and one or more threads of control that execute within that address space and their required system resources.

Profile: A set of one or more base standards and, where applicable, the identification of chosen classes, subsets, options and parameters of those base standards necessary for accomplishing a particular function. A profile lists the choices of platforms and defines the interfaces for an implementation of SOSAS consistent NGC.

Protocol: A set of semantic and syntactic rules that determine the behavior of entities in performing communication functions.

Reference Architecture: The collection of all known components including both primitive and aggregate components.

Scaleability: The ease with which software can be transferred from one graduated series of application platforms to another.

Software: The programs, procedures, rules and any associated documentation pertaining to the operation of a data processing system.

Specification: A document that prescribes in a complete, precise, verifiable manner, the requirements, design, behavior or characteristics of a system or system component.

Standardized profile: A balloted, formal, harmonized document that specifies a profile.

Standards: Documents, established by consensus and approved by a recognized body, that provide for common and repeated use, rules, guidelines, or characteristics for activities or their results aimed at the achievement of the optimum degree of order in a given context.

Thread: A single flow of control within a process.

APPENDIX A — REFERENCE REQUIREMENTS

Reference requirements guide the grouping and specialization of responsibilities from the Reference Architecture when synthesizing an application architecture for a specific controller application. For any specific application, only a subset of the Reference Requirements are relevant, and when the relevant requirements are selected out of the full set of Reference Requirements, they must be made specific to the application. For example, if Reference Requirement A.1.2.2, "Availability shall be at ? percent." is selected for a given application, then it is made specific by changing it to "Availability shall be at 99 percent." The NGC Reference Requirements are listed below. The reference numbers provide a convenient grouping of the requirements.

A.1 General Requirements

A.1.1 Environmental

A.1.1.1 The controller shall meet ? environmental standards.

A.1.1.2 The controller shall be able to operate in ? temperature range.

A.1.1.3 The controller shall/shall not be in an enclosure that keeps out dust-laden air.

A.1.1.4 The controller shall/shall not be of modular construction.

A.1.1.5 The controller shall/shall not automatically halt operation when the temperature within the control unit reaches ? degrees.

A.1.1.6 The controller shall/shall not be equipped with an air conditioner that will assure operation in a temperature of ? and humidity of ?.

A.1.2 Reliability

A.1.2.1 Availability shall/shall not be a function of mean time between failures and mean time to repair.

A.1.2.2 Availability shall be at ? percent.

A.1.2.3 The controller shall/shall not have a test program that will execute all control responsibilities implemented.

A.1.3 Factory Supervision

A.1.3.1 The controller shall/shall not interface with a supervisory computer system.

A.1.3.2 The supervisory computer system shall/shall not exert complete control over the controller.

A.1.4 Probing

A.1.4.1 The controller shall/shall not incorporate measuring probes.

A.1.4.2 The controller shall/shall not incorporate touch probes.

A.1.4.3 The controller shall/shall not incorporate optical probes.

A.1.4.4 The controller shall/shall not provide standard inspection (probing/gauging/measuring) cycles.

A.1.4.5 The controller shall/shall not provide for user-defined measuring cycles.

A.1.4.6 The controller shall/shall not measure the amount of excess or lack of workpiece material before operations.

A.1.4.6 The controller shall/shall not adjust process parameters according to pre-operation measurements.

A.1.4.7 The controller shall/shall not gauge the workpieces throughout operations to determine the necessity of tool changes.

A.1.4.8 The controller shall/shall not gauge the workpieces throughout operations to determine the necessity of repeating operations.

A.1.4.9 The controller shall/shall not gauge the workpieces throughout operations to determine the necessity of tool changes.

A.1.4.10 The controller shall/shall not record the results of the inspection cycle.

A.1.4.11 The controller shall/shall not generate digital coordinate point data describing any complex-shaped three-dimensional object.

A.1.5 Multitasking

A.1.5.1 The controller shall/shall not support multiple concurrent tasks.

A.1.5.2 The controller shall/shall not support background tasks of part programming.

A.1.5.3 The controller shall/shall not support background tasks of program edit.

A.1.5.4 The controller shall/shall not support background tasks of program loading.

A.1.5.5 The controller shall/shall not support background tasks of program downloading.

A.1.5.6 The controller shall/shall not support background tasks of program uploading.

A.1.6 User Interface

A.1.6.1 The controller shall/shall not extract geometric information from program files.

A.1.6.2 The controller shall/shall not graphically display geometric information in orthographic views.

A.1.6.3 The controller shall/shall not graphically display geometric information in perspective views.

A.1.6.4 The controller shall/shall not display a ?D simulation of the tool path.

A.1.6.5 The controller shall/shall not display a ?D simulation of a cross sectional view of the part.

A.1.6.6 The user interface design shall/shall not use colors and sounds.

A.1.6.7 Colors and sounds shall/shall not be used sparingly and redundantly.

A.1.6.8 The user interface shall provide access to information with a maximum of ? steps.

A.1.6.9 The user interface shall have active ? windows at any one moment.

A.1.6.10 The user shall/shall not be able to close all open windows with a single command.

A.1.6.11 The user interface shall support ?-language presentation.

A.1.6.12 Information shall/shall not be updated dynamically as it changes.

A.1.6.13 The user interface shall provide timely feedback within ? ms for every input the user makes.

A.1.7 Block Processing Time

A.1.7.1 Block processing time shall/shall not be fast enough to prevent data starvation.

A.1.7.2 The controller shall/shall not automatically decelerate axes movement when the block execution time is less than the average block processing time.

A.1.7.3 The controller shall have a look-ahead capacity of ? blocks.

A.1.7.4 The average block processing time shall be ? milliseconds.

A.1.7.5 The servo update time shall be from ? to ? milliseconds.

A.1.8 Software

A.1.8.1 Software components shall be added or replaced incrementally with/without recompilation.

A.1.8.2 I/O counts and types shall be expandable for ? type of equipment.

A.1.8.3 The software shall/shall not preclude the use of a distributed system.

A.1.8.4 The software shall/shall not preclude the use of a centralized system.

A.1.8.5 The start-up sequence shall/shall not support appropriate start-up sequencing of all machine components.

A.1.8.6 The start-up sequence shall/shall not support both a default and a user-defined idle state after start-up.

A.1.8.7 The system shall/shall not come up in an operational mode following the start-up sequence.

A.1.8.8 The system shall/shall not be orderly and responsible to emergency and catastrophic shutdowns.

A.1.8.9 System shutdown shall/shall not result in graceful termination of process execution.

A.1.8.10 A shutdown shall/shall not leave the system in a safe state with regard to the positions of valves, motors, and actuators.

A.1.8.11 The system shall/shall not automatically save critical machine-state data.

A.2 Axis Motion Control Requirements

A.2.1 Continuous Path Control

A.2.1.1 The controller shall/shall not provide continuous path control of tool motion.

A.2.1.2 The controller shall/shall not provide point-to-point programming.

A.2.1.3 The controller shall/shall not provide NURBS programming.

A.2.1.4 The controller shall/shall not provide feature-based programming.

A.2.2 Interpolation

A.2.2.1 The controller shall/shall not perform linear interpolation in simultaneous motion of all axis drives.

A.2.2.2 The controller shall/shall not perform circular interpolation with 3D motion control.

A.2.2.3 The controller shall/shall not perform helical interpolation.

A.2.3 Interpolation Resolution

A.2.3.1 Interpolation resolution shall be a maximum of ? mm or ? inch in linear mode.

A.2.3.2 Interpolation resolution shall be a maximum of ? mm or ? inch in circular mode.

A.2.4 Methods of Specifying Interpolation

A.2.4.1 Circular interpolation shall/shall not be specified by radius and end point.

A.2.4.2 Circular interpolation shall/shall not be specified by I, J, K parameters.

A.2.4.3 Circular interpolation shall/shall not be performed in more than one quadrant in one command block.

A.2.5 Interpolation in Rapid Traverse

A.2.5.1 Rapid traverse motion shall/shall not be in linear interpolation mode in all axes.

A.2.6 System Resolution

A.2.6.1 Control system least input increment shall be ? mm / ? inch and ? degree.

A.2.6.2 Control system least command increment shall be ? mm / ? inch and ? degree.

A.2.7 Acceleration and Deceleration Control

A.2.7.1 The controller shall/shall not automatically control axis acceleration/deceleration.

A.2.7.2 The controller shall/shall not perform exponential acceleration for feed motions.

A.2.7.3 The controller shall/shall not perform linear acceleration for rapid traverse.

A.2.8 Mechanical Error Compensation

A.2.8.1 The controller shall/shall not provide automatic compensation of lead screw errors (pitch error compensation).

A.2.8.2 The controller shall/shall not provide automatic compensation of backlash.

A.2.8.3 Lead screw error compensation shall/shall not be reprogrammable for wear compensation.

A.2.8.4 The controller shall/shall not perform cross compensation for axis alignment, beam sag and axis motion geometry errors.

A.2.9 Thermal Compensation

A.2.9.1 The controller shall/shall not perform temperature growth compensation.

A.2.9.2 The controller shall use ?# of thermal sensors.

A.2.10 Software Limitation of Axis Motion

A.2.10.1 The controller shall/shall not establish spatial zones (safe or forbidden zones) in addition to the fixed control limits provided by the machine builder.

A.2.11 Position Sensing

A.2.11.1 The controller shall/shall not measure position with linear digit scales.

A.2.11.2 The controller shall/shall not measure position with inductosyn devices.

A.2.12 Axis Velocity

A.2.12.1 The controller shall/shall not support programmable feedrates.

A.2.12.2 Programmable feedrates shall have a minimum of ? mm/min and a maximum of ? mm/min for the ? axis.

A.2.12.3 Rapid traverse feedrates shall have a minimum of ? mm/min and a maximum of ? mm/min for the ? axis.

A.3 Auxiliary Machine Control Functions

A.3.1 Tool Management

A.3.1.1 Tool management shall/shall not have storage for the maximum number of tool data sets.

A.3.1.2 The number of data sets may/may not be equal to or greater than the number of tool changer pockets.

A.3.1.3 The tool identification code shall have a minimum number of ? digits.

A.3.1.4 The tool shall/shall not be referenced by the tool identification code.

A.3.1.5 The tool shall/shall not be referenced by the tool location identification code.

A.3.1.6 The tool magazine shall/shall not provide random storage of tools with/without preassignment.

A.3.1.7 Tool slots, compartments, or other storage elements shall/shall not have sensors for identification and location of each tool.

A.3.1.8 The tool magazine shall/shall not hold the complement of tools and spares for specific jobs.

A.3.1.9 The tool magazine shall/shall not support removable tool cartridges.

A.3.1.10 The tool magazine shall/shall not support sensors for identification of tool cartridges and their job associations.

A.3.1.11 The tool length shall have a measurement of ? digits.

A.3.1.12 Cutter diameter offset compensation shall have a measurement of ? digits.

A.3.1.13 Cutter nose radius compensation shall/shall not be used with bull and ball nose cutters.

A.3.1.14 Cutter taper compensation shall/shall not be supported.

A.3.1.15 Tool change sequence shall be initiated by a ? code.

A.3.1.16 Tool change sequence shall/shall not be initiated manually.

A.3.1.17 Tool change operations shall/shall not automatically adjust to variations in spindle attachment geometries.

A.3.1.18 Tool length shall/shall not be verified as part of the tool change procedure.

A.3.1.19 Tool diameter shall/shall not be verified as part of the tool change procedure.

A.3.1.20 Tool form shall/shall not be verified as part of the tool change procedure.

A.3.2 Pallet Changer

A.3.2.1 The controller shall/shall not identify pallets by non-contact sensors.

A.3.2.2 Pallet data shall consist of at least ? bytes.

A.3.2.3 The controller shall track data for at least ? individual pallets.

A.3.2.4 Pallet identification shall/shall not be used to automatically select part programs.

A.3.2.5 Pallet identification shall/shall not be used to automatically select fixture offsets.

A.3.2.6 Pallet identification shall/shall not be used to automatically select pallet offsets.

A.3.2.7 Multiple part programs shall/shall not be selected per pallet.

A.3.3 Cutting Process Control

A.3.3.1 Cutting tool life shall/shall not be monitored based on user programmed time intervals/tool life time.

A.3.3.2 A tool life data shall/shall not include: tool number, programmed tool life, accumulated cutting time, remaining tool life, and tools which have exceeded the programmed tool life.

A.3.3.3 The controller shall/shall not automatically select an alternative tool should a tool's life elapse.

A.3.3.4 Tool life data shall/shall not be passed to external devices or communication systems.

A.3.3.5 The controller shall/shall not sense a broken tool.

A.3.3.6 The controller shall/shall not sense significant changes in tool performance.

A.3.3.7 The controller shall/shall not support a worn tap monitoring system for tapping speeds below ? RPM.

A.3.3.8 The controller shall/shall not automatically change tools upon detection of a broken or worn tool.

A.3.3.9 The controller shall/shall not measure engineering force levels on the spindle bearings.

A.3.3.10 The controller shall/shall not measure temperature of the spindle bearings.

A.3.3.11 The controller shall/shall not scan excessive loads for collision protection.

A.3.3.12 The controller shall/shall not monitor for unbalanced spindle loads.

A.3.3.13 The controller shall/shall not take appropriate actions when monitored spindle conditions exceed programmed tolerances.

A.3.3.14 The controller shall/shall not adjust feedrate to maintain constant cutting force.

A.3.3.15 The controller shall/shall not access work material parameter tables for automatic selection of cutting speeds, feeds, and depth of cut.

A.4 Programming Requirements

A.4.1 Automatic Programming

A.4.1.1 Programming shall/shall not include high-level language part programming.

A.4.1.2 Programming shall/shall not include inch or metric units.

A.4.1.3 Programming shall/shall not include absolute or incremental programming.

A.4.1.4 Programming shall/shall not include programmed dwell, stop, and optional stop.

A.4.1.5 Programming shall/shall not include feedrate programming: direct feedrate, inverse time feedrate, per revolution unit feed for turning controls.

A.4.1.6 Programming shall/shall not include direct spindle speed programming in RPM.

A.4.1.7 Programming shall/shall not include block delete.

A.4.1.8 Programming shall/shall not include program number search and sequence number search.

A.4.1.9 Programming shall/shall not include conditional and unconditional jumps in the part program.

A.4.1.10 Programming shall/shall not include mirror image.

A.4.1.11 Programming shall/shall not include fixture offsets for multiple pallet and table machines.

A.4.1.12 Programming shall/shall not include multiple program storage and management.

A.4.1.13 Programming shall/shall not include safe zone definitions.

A.4.1.14 Programming shall/shall not include parametric subroutines.

A.4.1.15 Programming shall/shall not include custom macro routines.

A.4.1.16 Programming shall/shall not include programmed tool change.

A.4.1.17 Programming shall/shall not include programming for cutting process control.

A.4.1.18 Programming shall/shall not include programmed tool life with designated replacement.

A.4.1.19 Programming shall/shall not include programmable adaptive control parameters.

A.4.1.20 Programming shall/shall not include programmable selection in tool failure detection modes.

A.4.1.21 Programming shall/shall not include circular interpolation designated by radius.

A.4.1.22 Programming shall/shall not include helical interpolation programming.

A.4.1.23 Programming shall/shall not include preprogrammed (canned) cycles for drilling, boring, and tapping.

A.4.1.24 Programming shall/shall not include preprogrammed cycles for area milling, rectangular pocket milling, circular pocket milling, and bolt hole circles.

A.4.1.25 Programming shall/shall not include automatic chamfering and corner radiusing.

A.4.1.26 Programming shall/shall not include look-ahead cutter compensation.

A.4.1.27 Programming shall/shall not include coordinate system rotation.

A.4.1.28 Programming shall/shall not include work coordinate system setting.

A.4.1.29 Programming shall/shall not include inspection probe programming.

A.4.1.30 Programming shall/shall not include constant surface speed programming.

A.4.1.31 Programming shall/shall not include tool center point programming for 5-axis machining.

A.4.1.32 Programming shall/shall not include programming scaling.

A.4.2 Manual Programming Requirements

A.4.2.1 Manual programming shall/shall not include provisions for conversational input.

A.4.2.2 Manual programming shall/shall not include automatic selection of cutting speed and feed.

A.4.2.3 Manual programming shall/shall not include cutter offset.

A.4.2.4 Manual programming shall/shall not include programmable spindle orientation.

A.4.2.5 Manual programming shall/shall not include pallet or table change cycles.

A.4.2.6 Manual programming shall/shall not include program editing.

A.4.2.7 Manual programming shall/shall not include the following setup capabilities: test run of part program, dry run of part program, graphic display of part position, graphic tool path display, execution of automatic probe routine to align program, axis inversion, and zero shift.

A.4.2.8 Manual control shall/shall not include: emergency stop, program stop, manual axis select, manual positioning control, sequence number search and display, spindle power and RPM readouts, X, Y, Z, and other axis position readouts.

A.4.3 Status Record Requirements

A.4.3.1 Status records shall/shall not include work order number: job id and quantity.

A.4.3.2 Status records shall/shall not include job id number: part number and resource number or class number.

A.4.3.3 Status records shall/shall not include job completion status: good, bad, rework, trail, active, unscheduled stop, program stop, and emergency stop.

A.4.3.4 Status records shall/shall not include time stamp of start of job.

A.4.3.5 Status records shall/shall not include time stamp of end of job.

A.4.3.6 Status records shall/shall not include time stamp of last status record update for job cycle.

A.4.3.7 Status records shall/shall not include elapsed time active for job cycle.

A.4.3.8 Status records shall/shall not include data transfer enabled.

A.4.3.9 Status records shall/shall not include edit mode enabled.

A.4.3.10 Status records shall/shall not include block by block mode enabled.

A.4.3.11 Status records shall/shall not include feedrate (percent override).

A.4.3.12 Status records shall/shall not include spindle speed (percent override).

A.4.3.13 Status records shall/shall not include programmable logic controller input, output, and status bits.

A.4.3.14 Status records shall/shall not include machine activities: power on, start and stop, idle/running, remote or local, ready, hold, cycle start, controller stop, resume, emergency stop, test run, and dry run.

A.4.3.15 Status records shall/shall not include tool status: list of tools, tool matrix data table (type of tool, number of pockets, pockets currently occupied, description), and tool object table (job number, T code, cutter compensation, tool offsets, gauge lengths, programmed tool life, pocket number).

A.4.3.16 Status records shall/shall not include pallet status: number of pallets, pallet identification, list of pallet offsets, pallet object table (pallet type, pallet size, pallet priority levels, number of parts present, list of part coordinate reference) parts mounted, and parts machined.

A.4.3.17 Status records shall/shall not include fixture status: type, id, and offsets.

A.4.3.18 Status records shall/shall not include part program status: id, size, available memory size, create/modify data, restrictions on use, restrictions on modifications, running program is ?% over, program complete, and elapsed time.

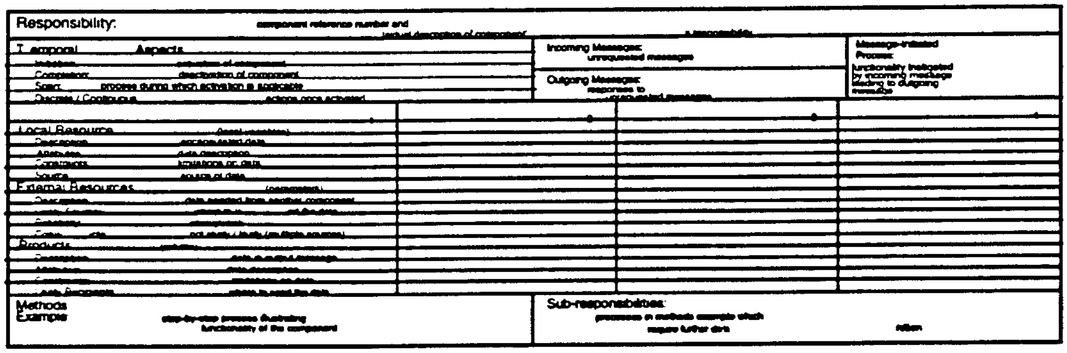
A.4.3.19 Status records shall/shall not include logical control parameters: feedrate override, spindle override, block delete, single block, and optional stop.

A.4.3.20 Status records shall/shall not include diagnostic status: system hardware faults, tool magazine faults, tool changer faults, pallet magazine faults, spindle drive faults, and axes drive faults.

A.4.3.21 Status records shall/shall not include event log: setting or resetting of conditions.

APPENDIX B — PRIMITIVE COMPONENTS

Primitive components represent indivisible responsibilities in the Reference Architecture. For each primitive component, a descriptive template represents temporal, resource, and product information. Temporal information explains the intended timing of the primitive component, such as start-up responsibilities. resource information explains what the primitive component will need in order to carry out its responsibility, such as coordinate system data. Product information explains what the primitive component can provide, such as translated coordinates. Shaded areas of each template represent information that is not appropriate at the reference architecture level. Once an application architecture is built, the determination of the contents of these areas brings the development process into the design phase. Below is the template including explanations for each entry. The remainder of this appendix contains the starter set of primitive components for NGC.



Responsibility: component reference number and textual description of component's responsibility

Temporal Aspects
Initiation: activation of component
Completion: deactivation of component
Span: process during which activation is applicable
Discrete / Continuous: actions once activated

Incoming Messages: unrequested messages

Outgoing Messages: responses to unrequested messages

Message-initiated Process: functionality instigated by incoming message leading to outgoing message

Local Resource (local variables)
Description: encapsulated data
Attributes: data description
Constraints: limitations on data
Source: source of data

External Resources (parameters)
Description: data needed from another component
[illegible] [illegible] the data
[illegible] [illegible]
[illegible] not shared / shared / multiple sources

Products [illegible]
Description: data in output message
Attributes: data description
Constraints: limitations on data
Data Recipients: where to send the data

Methods Example: step-by-step process illustrating functionality of the component

Sub-responsibilities: processes in methods example which require further definition

Responsibility: NGC-1 Handle operator modifications to part program statements			
TemporalAspects		Incoming Messages:	Message-Initiated Process:
Initiation:	modification to part program necessary		
Completion:	program completed	Outgoing Messages:	
Span:	prove-out		
Discrete / Continuous			

	1	2	3	4
Local Resources				
Description:				
Attributes:				
Constraints:				
Source:				
External Resources				
Description:	part program	machine model	modifications	
Likely Source:	workstation mgt	knowledge base	operator interface	
Reliability:	weak	strong	weak	
Conflicts:	not likely	not likely	not likely	
Products				
Description:	part program	error message		
Attributes:	code: parameters	code: parameter		
Constraints:	dependent on machine	modification invalid		
Likely Recipients:	wks mgt/plan interp	operator interface		

Procedures/Methods Example: • accept program • halt program when modifications needed • modify blocks if valid • restart segment of prove-out • send modified blocks	Sub-responsibilities: • halt program • modify blocks if valid • restart segment of prove-out

Responsibility: NGC-2 Part program storage activities			
TemporalAspects		Incoming Messages:	Message-Initiated Process:
Initiation:	system startup		
Completion:	system shutdown	Outgoing Messages	
Span:	execution and prove-out		
Discrete / Continuous			

	1	2	3	4
Local Resources				
Description				
Attributes				
Constraints:				
Source				
External Resources				
Description:	activity request	part program		
Likely Source:	workstation mgt	storage medium		
Reliability:	strong	strong		
Conflicts:	not likely	not likely		
Products				
Description:	part program	program status		
Attributes:	code: parameters	code: parameters		
Constraints:				
Likely Recipients:	workstation mgt	workstation mgt		

Procedures/Methods Example: • accept program activity request • access part program from storage • send part program or program status	Sub-responsibilities:

Responsibility: NGC-3 Determine each part's program and coordinate offsets

Temporal Aspects		Incoming Messages:	Message-Initiated Process:
Initiation:	system startup		
Completion:	system shutdown	Outgoing Messages:	
Span:	execution and prove-out		
Discrete / Continuous			

	1	2	3	4
Local Resources				
Description:				
Attributes:				
Constraints:				
Source:				
External Resources				
Description:	job list			
Likely Source:	factory scheduler			
Reliability:	strong			
Con½cts:	not likely			
Products				
Description:	part program id	part coordinate offsets		
Attributes:		zero positions		
Constraints:				
Likely Recipients	workstation mgt	workstation mgt		

Procedures/Methods Example: • accept job list • parse part program and offsets • send part program and offsets	Sub-responsibilities:

Responsibility: NGC-4 Maintain information on available tools

Temporal Aspects		Incoming Messages:	Message-Initiated Process:
Initiation:	system startup		
Completion:	system shutdown	Outgoing Messages:	
Span:	execution and prove-out		
Discrete / Continuous			

	1	2	3	4
Local Resources				
Description:				
Attributes:				
Constraints:				
Source:				
External Resources				
Description:	tool list	tool data updates	tool model	
Likely Source:	operator interface	operator interface	knowledge base	
Reliability:	strong	strong	strong	
Con½cts:	not likely	not likely	not likely	
Products				
Description:	updated tool model			
Attributes:				
Constraints:				
Likely Recipients:	knowledge base			

Procedures/Methods Example: • accept tool list, tool data updates • access tool model • update model • send updated tool model	Sub-responsibilities:

Responsibility: NGC-5 Ensure that resources required by the jobs are available

TemporalAspects		Incoming Messages:	Message-Initiated Process:
Initiation:	system startup		
Completion:	system shutdown	Outgoing Messages:	
Span:	execution and prove-out		
Discrete / Continuous			

	1	2	3	4
~~Local Resources~~				
Description:				
Attributes:				
Constraints:				
Source:				
External Resources				
Description:	job description	tool, machine model	part program	
Likely Source:	workstation mgt	knowledge base	workstation mgt	
Reliability:	strong	strong	strong	
Conflicts:	not likely	not likely	not likely	
Products				
Description:	verification results			
Attributes:	code: parameters			
Constraints:				
Likely Recipients	workstation mgt			

Procedures/Methods Example:

- accept job description
- access tool and machine models
- verify availability of resources
- send verification results

Sub-responsibilities:

- verify availability of resources

Responsibility: NGC-6 Determine if next part's program matches current part's program

TemporalAspects		Incoming Messages:	Message-Initiated Process:
Initiation:	start of part program		
Completion:	program completed	Outgoing Messages:	
Span:	execution and prove-out		
Discrete / Continuous			

	1	2	3	4
Local Resources				
Description:				
Attributes:				
Constraints:				
Source:				
External Resources				
Description:	next part's program id	current part's program id		
Likely Source:	workstation management	plan interpretation		
Reliability:	strong	strong		
Conflicts:	not likely	not likely		
Products				
Description:	comparison result			
Attributes:	value			
Constraints:	match / no match			
Likely Recipients:	workstation management			

Procedures/Methods Example:

- accept next part's program id
- accept current part's program id
- compare ids
- send comparison result

Sub-responsibilities:

Responsibility: NGC-7 Determine status of active program

Temporal Aspects		Incoming Messages:	Message-Initiated Process:
Initiation:	program startup		
Completion:	program completed	Outgoing Messages:	
Span:	execution and prove-out		
Discrete / Continuous			

	1	2	3	4
Local Resources				
Description:				
Attributes:				
Constraints:				
Source:				
Extemal Resources				
Description:	software status			
Likely Source:	components			
Reliability:	strong			
Conflicts:	not likely			
Products				
Description:	program status			
Attributes:	code: parameters			
Constraints:				
Likely Recipients	workstation mgt			

Procedures/Methods Example: • accept software status • correlate for program status • send program status	Sub-responsibilities:

Responsibility: NGC-8 Request additional segments of part program

Temporal Aspects		Incoming Messages:	Message-Initiated Process:
Initiation:	program startup		
Completion:	program shutdown	Outgoing Messages:	
Span:	execution and prove-out		
Discrete / Continuous			

	1	2	3	4
Local Resources				
Description:				
Attributes:				
Constraints:				
Source:				
Extemal Resources				
Description:	part program segments	additional segments request		
Likely Source:	workstation mgt	plan interpretation		
Reliability:	strong	strong		
Conflicts:	not likely	not likely		
Products				
Description:	part program request			
Attributes:	program id			
Constraints:				
Likely Recipients	workstation mgt			

Procedures/Methods Example: • accept additional segments request • request part program • send next appropriate segment	Sub-responsibilities:

Responsibility: NGC-9 Initialize start of part program activities

TemporalAspects	Incoming Messages:	Message-Initiated Process:
Initiation: part program loaded		
Completion: part program started	Outgoing Messages:	
Span: execution and prove-out		
Discrete / Continuous		

	1	2	3	4
Local Resources				
Description:				
Attributes:				
Constraints:				
Source:				
External Resources				
Description:	cycle start command			
Likely Source:	operator interface			
Reliability:	strong			
Conflicts:	not likely			
Products				
Description:	commands for starting			
Attributes:	code: parameters			
Constraints:				
Likely Recipients	plan interpretation			

Procedures/Methods Example: • accept cycle start command • send start commands for various activities	**Sub-responsibilities:**

Responsibility: NGC-10 Initiate end of part program activities

TemporalAspects	Incoming Messages:	Message-Initiated Process:
Initiation: program startup		
Completion: program shutdown	Outgoing Messages:	
Span: execution and prove-out		
Discrete / Continuous		

	1	2	3	4
Local Resources				
Description:				
[illegible]				
[illegible]				
[illegible]				
External Resources				
Description:	part program	next command step	machine model	
Likely Source:	workstation mgt	plan interpretation	knowledge base	
Reliability:	strong	strong	strong	
Conflicts:	not likely	not likely	not likely	
Products				
Description:	ending commands			
Attributes:	code: parameters			
Constraints:				
Likely Recipients	plan interpretation			

Procedures/Methods Example: • accept part program & next step • access machine model • when next step is nil, determine end of program activities • send ending commands	**Sub-responsibilities:** • determine end of program activities

Responsibility: NGC-11 Determine where to send part program codes

Temporal Aspects		Incoming Messages:	Message-Initiated Process:
Initiation:	program startup		
Completion:	program shutdown	Outgoing Messages:	
Span:	execution and prove-out		
Discrete / Continuous			

	1	2	3	4
Local Resources				
Description:				
Attributes:				
Constraints:				
Source:				
External Resources				
Description:	part program	machine model		
Likely Source:	workstation mgt	knowledge base		
Reliability:	weak	strong		
Conflicts:	not likely	not likely		
Products				
Description:	modified part program			
Attributes:	code: parameters			
Constraints:				
Likely Recipients	plan interpretation			

Procedures/Methods Example:
- accept program
- access machine model
- determine recipient of each component
- augment code with destinations
- send modified program

Sub-responsibilities:
- determine recipient of each component
- augment code with destinations

Responsibility: NGC-12 Interpret part program

Temporal Aspects		Incoming Messages:	Message-Initiated Process:
Initiation:	beginning of program stream		
Completion:	end of program stream	Outgoing Messages:	
Span:	execution and prove-out		
Discrete / Continuous			

	1	2	3	4
Local Resources				
Description:				
Attributes:				
Constraints:				
Source:				
External Resources				
Description:	part program file	machine model		
Likely Source:	workstation management	knowledge base		
Reliability:	weak	strong		
Conflicts:	not likely	not likely		
Products				
Description:	parsed part program	error message	mechanism commands	
Attributes:	code: parameters	code: parameters	code: parameters	
Constraints:				
Likely Recipients:	plan interpretation	operator interface	mechanisms	

Procedures/Methods Example:
- accept program file
- access machine model
- validate code for machine
- parse codes for motion / mechanism

Sub-responsibilities:
- validate code for machine
- parse codes for motion / mechanism

Responsibility: NGC-13 Augment code with component destinations

Temporal Aspects		Incoming Messages:	Message-Initiated Process:
Initiation:	program startup		
Completion:	program shutdown	Outgoing Messages:	
Span:	execution and prove-out		
Discrete / Continuous			

	1	2	3	4
Local Resources				
Description:				
Attributes:				
Constraints:				
Source:				
External Resources				
Description:	part program codes	recipient		
Likely Source:	plan interpretation	plan interpretation		
Reliability:	strong	strong		
Conflicts:	not likely	not likely		
Products				
Description:	modified part program			
Attributes:	code; parameters			
Constraints:				
Likely Recipients	plan interpretation			

Procedures/Methods
Example:

- accept part program codes and recipients
- update codes with recipients
- send modified code

Sub-responsibilities:

Responsibility: NGC-14 Inform operator of part program comments

Temporal Aspects		Incoming Messages	Message-Initiated Process:
Initiation:	program startup		
Completion:	program shutdown	Outgoing Messages:	
Span:	execution and prove-out		
Discrete / Continuous			

	1	2	3	4
Local Resources				
Description:				
Attributes:				
Constraints:				
Source:				
External Resources				
Description:	part program	part requirements	physical laws	
Likely Source:	workstation mgt	engineering	knowledge base	
Reliability:	strong	weak	strong	
Conflicts:	not likely	not likely	not likely	
Products				
Description:	comments to operator			
Attributes:	code; parameters			
Constraints:	priority levels			
Likely Recipients	operator interface			

Procedures/Methods
Example:

- accept part program
- identify comments to send to operator
- send comments

Sub-responsibilities:

Responsibility: NGC-15 Augment code for coordination between motion and mechanism

TemporalAspects
Initiation: particular tasks necessary
Completion: tasks completed
Span: execution and prove-out
Discrete / Continuous

Incoming Messages:

Outgoing Messages:

Message-Initiated Process:

	1	2	3	4
Local Resources				
Description:				
Attributes:				
Constraints:				
Source:				
External Resources				
Description:	part program			
Likely Source:	workstation management			
Reliability:	strong			
Con⅓cts:	not likely			
Products				
Description:	augmented part program			
Attributes:	additional temporal sequencing			
Constraints:	machine dependent			
Likely Recipients:	motion / mechanism			

Procedures/Methods
Example:
- accept part program
- determine tasks requiring tight coordination
- look ahead to determine degree of coord.
- determine new sequencing
- send newly ordered commands

Sub-responsibilities:
- determine tasks requiring tight coordination
- look ahead to determine degree of coord.
- determine new sequencing

Responsibility: NGC-16 Translate coordinate systems

TemporalAspects
Initiation: start of part program
Completion: program completed
Span: execution and prove-out
Discrete / Continuous

Incoming Messages:

Outgoing Messages:

Message-Initiated Process:

	1	2	3	4
[illegible]				
[illegible]				
[illegible]				
[illegible]				
[illegible]				
External Resources				
Description:	part program	machine model	coordinate frame models	
Likely Source:	plan interpretation	knowledge base	knowledge base	
Reliability:	strong	strong	strong	
Con⅓cts:	not likely	not likely	not likely	
Products				
Description:	coordinate transformations			
Attributes:	code; parameters			
Constraints:	dependent on machine			
Likely Recipients	plan interpretation			

Procedures/Methods
Example:
accept program
- analysis for transformation required
- access machine models
- access coordinate frame models
- transform coordinates
- send part program with new coordinates

Sub-responsibilities:
- analysis for transformation required
- transform coordinates

Responsibility: NGC-17 Interpret augmented code

Temporal Aspects	
Initiation:	program startup
Completion:	program shutdown
Span:	execution and prove-out
Discrete / Continuous	

Incoming Messages:

Outgoing Messages:

Message-Initiated Process:

	1	2	3	4
Local Resources				
Description:				
Attributes:				
Constraints:				
Source:				
External Resources				
Description:	augmented program	status reports	machine model	
Likely Source:	plan interpretation	motion/mechanism	knowledge base	
Reliability:	strong	strong	strong	
Con⅓cts:	not likely	not likely	not likely	
Products				
Description:	coordination commands			
Attributes:	code: parameters			
Constraints:				
Likely Recipients	motion/mechanism			

Procedures/Methods
Example:
- accept part program and status reports
- access machine model
- track program execution
- determine wait/continue commands for coordination
- send coordination commands

Sub-responsibilities:
- determine wait/continue commands for coordination

Responsibility: NGC-18 Determine wait/continue commands for coordination

Temporal Aspects	
Initiation:	program startup
Completion:	program shutdown
Span:	execution and prove-out
Discrete / Continuous	

Incoming Messages:

Outgoing Messages:

Message-Initiated Process:

	1	2	3	4
Local Resources				
Description:				
Attributes:				
Constraints:				
Source:				
External Resources				
Description:	program status	part program	machine model	
Likely Source:	plan interpretation	plan interpretation	knowledge base	
Reliability:	strong	strong	strong	
Con⅓cts:	not likely	not likely	not likely	
Products				
Description:	coordination command			
Attributes:	code: parameters			
Constraints:				
Likely Recipients	motion/mechanism			

Procedures/Methods
Example:
- accept part program & program status
- access machine model
- determine next coordination required
- send coordination commands

Sub-responsibilities:

Responsibility:	NGC-19 Swap current tool with required tool		
Temporal Aspects		Incoming Messages:	Message-Initiated Process:
Initiation:	tool change required		
Completion:	tool change accomplished	Outgoing Messages:	
Span:	execution and prove-out		
Discrete / Continuous			

	1	2	3	4
Local Resources				
Description:				
Attributes:				
Constraints:				
Source:				
External Resources				
Description:	current tool id	next tool id	swap command	
Likely Source:	plan interpretation	plan interpretation	plan interpretation	
Reliability:	strong	strong	strong	
Con¼cts:	not likely	not likely	not likely	
Products				
Description:	swap commands			
Attributes:	code: parameters			
Constraints:				
Likely Recipients	tool mechanism			

Procedures/Methods Example: • access current, next tool ids • determine location of next tool • track location of current tool • send swap commands to mechanism	Sub-responsibilities:

Responsibility:	NGC-20 Map tool id to actual tool location		
Temporal Aspects		Incoming Messages:	Message-Initiated Process:
Initiation:	system startup		
Completion:	system shutdown	Outgoing Messages:	
Span:	execution and prove-out		
Discrete / Continuous			

	1	2	3	4
Local Resources				
Description:				
Attributes:				
Constraints:				
Source:				
External Resources				
Description:	required tool id	tool model		
Likely Source:	plan interpretation	knowledge base		
Reliability:	strong	strong		
Con¼cts:	not likely	not likely		
Products				
Description:	tool location			
Attributes:				
Constraints:				
Likely Recipients:	tool changer			

Procedures/Methods Example: • accept required tool id • access tool model • determine tool location • send tool location	Sub-responsibilities:

Responsibility: NGC-21 Load and unload tools

TemporalAspects		Incoming Messages:	Message-Initiated Process:
Initiation:	system startup		
Completion:	system shutdown	Outgoing Messages:	
Span:	setup		
Discrete / Continuous			

	1	2	3	4
Local Resources				
Description:				
Attributes:				
Constraints:				
Source:				
External Resources				
Description:	tools required	tools supplied	tool model	
Likely Source:	factory scheduler	operator interface	knowledge base	
Reliability:	strong	weak	strong	
Conflicts:	not likely	not likely	not likely	
Products				
Description:	modified tool model			
Attributes:				
Constraints:				
Likely Recipients:	knowledge base			

Procedures/Methods
Example:
- accept tools required and supplied
- access tool model
- update model with new tool data
- send modified tool model

Sub-responsibilities:

Responsibility: NGC-22 Track tool locations

TemporalAspects		Incoming Messages:	Message-Initiated Process:
Initiation:	system startup		
Completion:	system shutdown	Outgoing Messages:	
Span:	execution and prove-out		
Discrete / Continuous			

	1	2	3	4
Local Resources				
Description:				
Attributes:				
Constraints:				
Source:				
External Resources				
Description:	last tool id	next tool id	next tool's location	
Likely Source:	plan interpretation	plan interpretation	knowledge base	
Reliability:	strong	strong	strong	
Conflicts:	not likely	not likely	not likely	
Products				
Description:	last tool's new location			
Attributes:	code: parameters			
Constraints:	tool size			
Likely Recipients:	knowledge base			

Procedures/Methods
Example:
- access last, next tool ids
- access next tool's location
- assign last tool to next tool's location
- send last tool's new location

Sub-responsibilities:
- design part

Responsibility: NGC-23 Swap current part with next part

TemporalAspects		Incoming Messages:	Message-Initiated Process:
Initiation:	part fixturing mechanism started		
Completion:	part fixturing mechanism shut down	Outgoing Messages:	
Span:	program execution		
Discrete / Continuous			

	1	2	3	4
Local Resources				
Description:				
Attributes:				
Constraints:				
Source:				
External Resources				
Description:	part fixturing commands	machine model		
Likely Source:	mechanism control	knowledge base		
Reliability:	strong	strong		
Conflicts:	not likely	not likely		
Products				
Description:	command ack	actuator signals		
Attributes:	code: parameter	voltage		
Constraints:	success of execution	device types		
Likely Recipients	mechanism control	actuators		

Procedures/Methods Example: • accept part fixturing commands • convert to actuator signals • send signals • send command acknowledgment	Sub-responsibilities:

Responsibility: NGC-24 Identify current pallet

TemporalAspects		Incoming Messages:	Message-Initiated Process:
Initiation:	pallet fixturing mechanism started		
Completion:	pallet fixturing mechanism shut down	Outgoing Messages:	
Span:	execution		
Discrete / Continuous			

	1	2	3	4
Local Resources				
Description:				
Attributes:				
Constraints:				
Source:				
External Resources				
Description:	sensor data			
Likely Source:	sensor interface			
Reliability:	weak			
Conflicts:	not likely			
Products				
Description:	pallet id			
Attributes:	code: parameters			
Constraints:				
Likely Recipients	workstation mgt			

Procedures/Methods Example: • accept sensor data • identify pallet • send pallet id	Sub-responsibilities:

Responsibility: NGC-25 Maintain control of feedrate-dependent operations

Temporal Aspects		Incoming Messages:	Message-Initiated Process:
Initiation:	start of part program		
Completion:	program completed	Outgoing Messages:	
Span:	execution and prove-out		
Discrete / Continuous			

	1	2	3	4
Local Resources				
Description:				
Attributes:				
Constraints:				
Source:				
External Resources				
Description:	command rate	mechanism commands		
Likely Source:	motion	plan interpretation		
Reliability:	strong	strong		
Conflicts:	not likely	not likely		
Products				
Description:	mechanism commands			
Attributes:	code: parameters			
Constraints:	altered for smooth flow			
Likely Recipients:	mechanism			

Procedures/Methods Example:
- accept command rate, commands
- analysis for continuous flow information
- modify flow according to rate
- send modification commands

Sub-responsibilities:
- analysis for continuous flow information

Responsibility: NGC-26 Control acceleration and deceleration

Temporal Aspects		Incoming Messages:	Message-Initiated Process:
Initiation:	motion started		
Completion:	motion stopped	Outgoing Messages:	
Span:	execution and prove-out		
Discrete / Continuous			

	1	2	3	4
Local Resources				
Description:				
Attributes:				
Constraints:				
Source:				
External Resources				
Description:	goal rate	axis step limits		
Likely Source:	trajectory generator	knowledge base		
Reliability:	strong	strong		
Conflicts:	not likely	not likely		
Products				
Description:	command rate			
Attributes:				
Constraints:				
Likely Recipients:	trajectory generator			

Procedures/Methods Example:
- accept goal rate
- access axis step limits
- modify goal rate to be within limits
- send command rate

Sub-responsibilities:

Responsibility: NGC-27 Define normals to path

Temporal Aspects		Incoming Messages:	Message-Initiated Process:
Initiation:	motion started		
Completion:	motion stopped	Outgoing Messages:	
Span:	execution and prove-out		
Discrete / Continuous			

	1	2	3	4
Local Resources				
Description:				
Attributes:				
Constraints:				
Source:				
External Resources				
Description:	trajectory			
Likely Source:	trajectory generator			
Reliability:	strong			
Conflicts:	not likely			
Products				
Description:	normals			
Attributes:				
Constraints:				
Likely Recipients	trajectory generator			

Procedures/Methods
Example:
- accept trajectory
- calculate normals
- send normals

Sub-responsibilities:

Responsibility: NGC-28 Normalize trajectory with respect to time

Temporal Aspects		Incoming Messages:	Message-Initiated Process:
Initiation:	motion started		
Completion:	motion stopped	Outgoing Messages:	
Span:	execution and prove-out		
Discrete / Continuous			

	1	2	3	4
Local Resources				
Description:				
Attributes:				
Constraints:				
Source:				
External Resources				
Description:	trajectory	machine model		
Likely Source:	trajectory generator	knowledge base		
Reliability:	strong	strong		
Conflicts:	not likely	not likely		
Products				
Description:	normalized trajectory			
Attributes:	setpoints: parameters			
Constraints:				
Likely Recipients:	trajectory generator			

Procedures/Methods
Example:
- accept trajectory
- access machine model
- segment trajectory into setpoints for timesteps
- send normalized trajectory

Sub-responsibilities:
- segment trajectory into setpoints for timesteps

Responsibility: NGC-29 Translate motion commands to trajectory

Temporal Aspects	
Initiation:	start of part program
Completion:	program completed
Span:	execution and prove-out
Discrete / Continuous	

Incoming Messages:

Outgoing Messages:

Message-Initiated Process:

	1	2	3	4
Local Resources				
Description:				
Attributes:				
Constraints:				
Source:				
External Resources				
Description:	motion command	cutter offsets	position corrections	rate corrections
Likely Source:	plan interpretation	motion	motion	motion
Reliability:	strong	weak	weak	weak
Conflicts:	not likely	not likely	not likely	not likely
Products				
Description:	trajectory information			
Attributes:	setpoints: parameters			
Constraints:	per coordinate frame			
Likely Recipients	actuator control			

Procedures/Methods Example:

- accept program
- analysis for destination information
- access corrections
- determine setpoints and parameters
- send trajectory

Sub-responsibilities:

- analysis for destination information
- determine setpoints and parameters

Responsibility: NGC-30 Coordinate with mechanisms

Temporal Aspects	
Initiation:	program started
Completion:	program shutdown
Span:	execution and prove-out
Discrete / Continuous	

Incoming Messages:

Outgoing Messages:

Message-Initiated Process:

	1	2	3	4
Local Resources				
Description:				
Attributes:				
Constraints:				
Source:				
External Resources				
Description:	augmented code	continue command		
Likely Source:	plan interpretation	plan interpretation		
Reliability:	strong	strong		
Conflicts:	not likely	not likely		
Products				
Description:	wait indicator			
Attributes:	code: parameter			
Constraints:				
Likely Recipients	plan interpretation			

Procedures/Methods Example:

- accept augmented code
- execute codes
- if code is to wait, send wait indicator
- if waiting, accept continue command

Sub-responsibilities:

Responsibility: NGC-31 Avoid data starvation

Temporal Aspects		Incoming Messages:	Message-Initiated Process:
Initiation:	start of part program		
Completion:	program completed	Outgoing Messages:	
Span:	execution and prove-out		
Discrete / Continuous			

		2	3	4
	1			
Local Resources				
Description:				
Attributes:				
Constraints:				
Source:				
External Resources				
Description:	motion commands	machine model		
Likely Source	plan interpretation	knowledge base		
Reliability:	strong	strong		
Conflicts:	not likely	not likely		
Products				
Description:	look-ahead analysis results			
Attributes:	code: parameters			
Constraints:	time / memory limitations			
Likely Recipients:	plan interpretation			

Procedures/Methods
Example:
- accept motion commands
- analysis for look-ahead information
- send look-ahead response to control rate of command transfer

Sub-responsibilities:
- analysis for look-ahead information

Responsibility: NGC-32 Determine rate of movement

Temporal Aspects		Incoming Messages:	Message-Initiated Process:
Initiation:	motion started		
Completion:	motion stopped	Outgoing Messages:	
Span:	execution and prove-out		
Discrete / Continuous			

		2	3	4
	1			
Local Resources				
Description:				
Attributes:				
Constraints:				
Source:				
External Resources				
Description:	rate commands	spindle speed	tool model	
Likely Source:	trajectory generator	sensor interface	knowledge base	
Reliability:	strong	weak	strong	
Conflicts:	not likely	not likely	not likely	
Products				
Description:	goal rate			
Attributes:				
Constraints:				
Likely Recipients:	trajectory generator			

Procedures/Methods
Example:
- accept rate commands
- access spindle speed, tool model
- determine appropnate goal rate
- send goal rate

Sub-responsibilities:

Responsibility: NGC-33 Adjust trajectory for tool deviations

Temporal Aspects		Incoming Messages:	Message-Initiated Process:
Initiation:	start of part program		
Completion:	program completed	Outgoing Messages:	
Span:	execution and prove-out		
Discrete / Continuous			

		1	2	3	4
Local Resources					
Description:					
Attributes:					
Constraints:					
Source:					
External Resources					
Description:	normals to trajectory		tool offset amounts	plane selection	
Likely Source:	trajectory generation		knowledge base	plan interpretation	
Reliability:	strong		strong	strong	
Con¼cts:	not likely		not likely	not likely	
Products					
Description:	cutter offsets				
Attributes:	values				
Constraints:	tool offset accuracy				
Likely Recipients	motion				

Procedures/Methods Example: • accept trajectory • access models • determine cutter compensation • send new trajectory	Sub-responsibilities: • determine cutter compensation

Responsibility: NGC-34 Determine trajectory corrections from predictable variations

Temporal Aspects		Incoming Messages:	Message-Initiated Process:
Initiation:	machine startup		
Completion:	machine shutdown	Outgoing Messages:	
Span:	execution and prove-out		
Discrete / Continuous			

		1	2	3	4
[illegible]					
[illegible]					
[illegible]					
[illegible]					
[illegible]					
External Resources					
Description:	thermal data		temporal data	tables, polynomials	position
Likely Source:	sensor interface		operating system	knowledge base	sensor interface
Reliability:	weak		strong	strong	weak
Con¼cts:	likely		not likely	not likely	not likely
Products					
Description:	trajectory corrections				
Attributes:	code, parameters				
Constraints:					
Likely Recipients	trajectory generation				

Procedures/Methods Example: • access machine model • access temporal data • access thermal data • determine necessary corrections • send corrections	Sub-responsibilities:

Responsibility: NGC-35 Determine trajectory corrections from sensed variations

Temporal Aspects		Incoming Messages:	Message-Initiated Process:
Initiation:	sensor data beyond tolerance		
Completion:	sensor data with tolerance	Outgoing Messages:	
Span:	execution and prove-out		
Discrete / Continuous			

	1	2	3	4
Local Resources				
Description:				
Attributes:				
Constraints:				
Source:				
External Resources				
Description:	sensor data (force, torque)	normals to trajectory	sensor tolerances	sensor ranges
Likely Source:	sensor interface	trajectory generation	plan interpretation	knowledge base
Reliability:	weak	strong	strong	strong
Conflicts:	likely	not likely	not likely	not likely
Products				
Description:	error message	position correction	rate correction	spindle correction
Attributes:	code; parameters	value	value	value
Constraints:	bad path / bad sensors	machine dependent	machine dependent	machine dependent
Likely Recipients:	operator interface	trajectory generation	trajectory generation	trajectory generation

Procedures/Methods Example:

- accept normals
- continually monitor sensors
- compare sensor data against tolerances
- calculate correction
- send correction

Sub-responsibilities:

- continually monitor sensors
- calculate correction

Responsibility: NGC-36 Adjust trajectory with corrections generated by sensor data

Temporal Aspects		Incoming Messages:	Message-Initiated Process:
Initiation:	start of part program		
Completion:	program completed	Outgoing Messages:	
Span:	execution and prove-out		
Discrete / Continuous			

	1	2	3	4
Local Resources				
Description:				
Attributes:				
Constraints:				
Source:				
External Resources				
Description:	trajectory	sensed corrections		
Likely Source:	trajectory generation	motion		
Reliability:	strong	strong		
Conflicts:	not likely	not likely		
Products				
Description:	modified trajectories			
Attributes:	setpoints; parameters			
Constraints:	non-permanent program change			
Likely Recipients:	actuator control			

Procedures/Methods Example:

- accept trajectory
- accept sensed corrections
- modify trajectory as per tolerances
- send new trajectory

Sub-responsibilities:

Responsibility: NGC-37 Translate servo commands into drive readable movement commar

Temporal Aspects	
Initiation:	start of part program
Completion:	program completed
Span:	execution and prove-out
Discrete / Continuous	

Incoming Messages:

Outgoing Messages:

Message-Initiated Process:

	1	2	3	4
Local Resources				
Description:				
Attributes:				
Constraints:				
Source:				
External Resources				
Description:	servo command positions	kinematic equations	feedback	
Likely Source:	trajectory generation	knowledge base	servo interface	
Reliability:	strong	strong	weak	
Con¼cts:	not likely	not likely	not likely	
Products				
Description:	movement commands	following error		
Attributes:	voltage	value		
Constraints:	accuracy of models	accuracy of encoders		
Likely Recipients:	axis drives	machine model		

Procedures/Methods Example:
- accept setpoints
- interpolate between current position and setpoint
- convert to command voltage
- send voltage

Sub-responsibilities:
- interpolate between current position and setpoint
- convert to command voltage

Responsibility: NGC-38 Control motors to produce movement

Temporal Aspects	
Initiation:	receipt of voltage commands
Completion:	feedback provided
Span:	execution and prove-out
Discrete / Continuous	

Incoming Messages:

Outgoing Messages:

Message-Initiated Process:

	1	2	3	4
[illegible]				
[illegible]				
[illegible]				
[illegible]				
[illegible]				
External Resources				
Description:	movement commands	feedback		
Likely Source:	actuator control	motors		
Reliability:	strong	strong		
Con¼cts:	not likely	not likely		
Products				
Description:	motor commands	feedback		
Attributes:	voltage	values		
Constraints:	motor type	accuracy of devices		
Likely Recipients:	motors	actuator control		

Procedures/Methods Example:
- accept movement commands
- convert for motors
- send voltage
- accept leadscrew feedback
- convert for actuator control
- send feedback

Sub-responsibilities:
- convert for motors
- convert for actuator control

Responsibility: NGC-39 Initiate startup procedures

Temporal Aspects		Incoming Messages:	Message-Initiated Process:
Initiation:	system power on		
Completion:	startup completed	Outgoing Messages:	
Span:	prior to all operations		
Discrete / Continuous			

	1	2	3	4
Local Resources				
Description:				
Attributes:				
Constraints:				
Source:				
External Resources				
Description:	reference setpoints	machine model		
Likely Source:	operator interface	knowledge base		
Reliability:	strong	strong		
Con½cts:	not likely	not likely		
Products				
Description:	diagnostic results	axes coordinate zero positions		
Attributes:	codes; parameters	codes: parameters		
Constraints:	device types	axes		
Likely Recipients	op if / maintenance	trajectory generation		

Procedures/Methods Example:
- initialization of power
- perform diagnostics
- send diagnostics results
- determine axes zero positions
- send axes zero positions

Sub-responsibilities:
- perform diagnostics
- determine axes coordinate zero positions

Responsibility: NGC-40 Coordinate with factory scheduler/control system

Temporal Aspects		Incoming Messages:	Message-Initiated Process:
Initiation:	system start up		
Completion:	system shut down	Outgoing Messages:	
Span:	no program executing		
Discrete / Continuous			

	1	2	3	4
Local Resources				
Description:				
Attributes:				
Constraints:				
Source:				
External Resources				
Description:	job instructions	machine model	heartbeats	
Likely Source:	factory system	knowledge base	all components	
Reliability:	strong	strong	strong	
Con½cts:	not likely	not likely	not likely	
Products				
Description:	health information	job list	tool list	job status
Attributes:	code: parameters	id. pallet/part id. offsets	tool id. location	code: parameters
Constraints:				
Likely Recipients	factory system	workstation mgt	workstation mgt	factory system

Procedures/Methods Example:
- accept instructions
- access machine model
- validate, send instructions
- accept machine heartbeats
- determine machine health
- send health information

Sub-responsibilities:
- validate instructions
- determine machine health

Responsibility: NGC-41 Path planning based on part features, surface model, etc.

Temporal Aspects		Incoming Messages:	Message-Initiated Process:
Initiation:	beginning of model-based program		
Completion:	program completed	Outgoing Messages:	
Span:	execution and prove-out		
Discrete / Continuous			

	1	2	3	4
Local Resources				
Description:				
Attributes:				
Constraints:				
Source:				
External Resources				
Description:	part description	machine model	stock description	
Likely Source:	workstation mgt	knowledge base	knowledge base	
Reliability:	weak	strong	strong	
Conflicts:	not likely	not likely	not likely	
Products				
Description:	path plan			
Attributes:	trajectories			
Constraints:	machine dependent			
Likely Recipients	motion control			

Procedures/Methods Example:

- accept part description
- access machine model, stock description
- determine path for motion
- send path

Sub-responsibilities:

- determine path for motion given models

Responsibility: NGC-42 Modify part program based on sensor data

Temporal Aspects		Incoming Messages:	Message-Initiated Process:
Initiation:	sensor data beyond tolerance		
Completion:	sensor data within tolerance	Outgoing Messages:	
Span:	execution and prove-out		
Discrete / Continuous			

	1	2	3	4
Local Resources				
Description:				
Attributes:				
Constraints:				
Source:				
External Resources				
Description:	sensor data	part program	sensor tolerances	sensor ranges
Likely Source:	sensor interface	workstation mgt	plan interpretation	knowledge base
Reliability:	weak	strong	strong	strong
Conflicts:	likely	not likely	not likely	not likely
Products				
Description:	part program	error message		
Attributes:	blocks of codes	code: parameters		
Constraints:	machine dependent	bad program / bad sensors		
Likely Recipients	workstation planning	operator interface		

Procedures/Methods Example:

- accept program
- continually monitor sensors
- compare sensor data against tolerances
- adjust part program
- send new part program

Sub-responsibilities:

- continually monitor sensors
- adjust part program

Responsibility: NGC-43 Switch between position control to force control

Temporal Aspects		Incoming Messages:	Message-Initiated Process:
Initiation:	start of part program		
Completion:	program completed	Outgoing Messages:	
Span:	execution and prove-out		
Discrete / Continuous			

	1	2	3	4
Local Resources				
Description:				
Attributes:				
Constraints:				
Source:				
External Resources				
Description:	switch command	machine model	part model	current position & force
Likely Source:	trajectory generation	knowledge base	knowledge base	sensor interface
Reliability:	strong	strong	strong	weak
Conflicts:	not likely	not likely	not likely	not likely
Products				
Description:	activate command	deactivate command		
Attributes:	code: parameters	code: parameters		
Constraints:				
Likely Recipients	motion	motion		

Procedures/Methods Example:
- accept switch command
- intermediate conversion between control
- send deactivate to position/force component
- send activate to force/position component

Sub-responsibilities:
- intermediate conversion between control

Responsibility: NGC-44 Plan for multi-axis interactions

Temporal Aspects		Incoming Messages:	Message-Initiated Process:
Initiation:	start of part program		
Completion:	program completed	Outgoing Messages:	
Span:	execution and prove-out		
Discrete / Continuous			

	1	2	3	4
Local Resources				
Description:				
Attributes:				
Constraints:				
Source:				
External Resources				
Description:	trajectory	machine model		
Likely Source:	motion	knowledge base		
Reliability:	strong	strong		
Conflicts:	not likely	not likely		
Products				
Description:	movement commands	error message		
Attributes:	code: parameters	code: parameter		
Constraints:	time limitations	interaction invalid		
Likely Recipients	motion	operator interface		

Procedures/Methods Example:
- accept trajectory
- perform look-ahead to determine possible multi-axis coordination optimization
- send modified part program

Sub-responsibilities:
- perform look-ahead to determine possible multi-axis coordination optimization

Responsibility:	NGC-45 Touch off for automatic setup
Temporal Aspects	
Initiation:	setup of part
Completion:	tool in position
Span:	prove-out
Discrete / Continuous	

Incoming Messages:

Outgoing Messages:

Message-Initiated Process:

	1	2	3	4
~~Local Resources~~				
Description:				
Attributes:				
Constraints:				
Source:				
External Resources				
Description:	trajectory rough termination point		sensor data	servo feedback
Likely Source:	trajectory generation		sensor interface	hardware interface
Reliability:	strong		weak	strong
Conflicts:	not likely		likely	not likely
Products				
Description:	servo commands			
Attributes:	voltage			
Constraints:	sensor accuracy			
Likely Recipients	axis motors			

Procedures/Methods Example:
- accept trajectory rough termination point
- move to termination point
- continue move until sensors or servos pass tolerance

Sub-responsibilities:
- continue move until sensors or servos pass tolerance

Responsibility:	NGC-46 Perform sensor fusion
Temporal Aspects	
Initiation:	start of part program
Completion:	program completed
Span:	prove-out and execution
Discrete / Continuous	

Incoming Messages:

Outgoing Messages:

Message-Initiated Process:

	1	2	3	4
Local Resources				
Description				
Attributes				
Constraints				
Source				
External Resources				
Description:	sensor data	machine model		
Likely Source:	sensor interface	knowledge base		
Reliability:	weak	strong		
Conflicts:	likely	not likely		
Products				
Description:	sensor information			
Attributes:	values			
Constraints:	sensor accuracy			
Likely Recipients	trajectory gen. / wks planning			

Procedures/Methods Example:
- accept sensor data
- fuse data from multiple sensors
- convert data into information
- send information

Sub-responsibilities:

Responsibility: NGC-47 Initiate shutdown procedures

TemporalAspects		Incoming Messages:	Message-Initiated Process:
Initiation:	system start up		
Completion:	system shut down	Outgoing Messages:	
Span:	always running		
Discrete / Continuous			

		1	2	3	4
Local Resources					
Description:					
Attributes:					
Constraints:					
Source:					
External Resources					
Description:	watchdog warning		machine model		
Likely Source:	watchdog system		knowledge base		
Reliability:	strong		strong		
Con¼cts:	not likely		not likely		
Products					
Description:	shutdown command				
Attributes:	code: parameters				
Constraints:	time limitations				
Likely Recipients:	all components				

Procedures/Methods Example:
- accept watchdog warning
- determine safe shutdown procedures
- send shutdown command

Sub-responsibilities:

Responsibility: NGC-48 Cue the operator for manual tasks

TemporalAspects		Incoming Messages:	Message-Initiated Process:
Initiation:	receipt of speci½ commands		
Completion:	valid operator response	Outgoing Messages:	
Span:	execution and prove-out		
Discrete / Continuous			

		1	2	3	4
Local Resources					
Description:					
Attributes:					
Constraints:					
Source:					
External Resources					
Description:	part program		machine model	operator response	
Likely Source:	workstation management		knowledge base	code: parameters	
Reliability:	weak		strong	strong	
Con½cts:	not likely		not likely	not likely	
Products					
Description:	halt/continue cues			error message	
Attributes:	code: parameters			code: parameters	
Constraints:					
Likely Recipients:	plan interpretation/		operator interface	operator interface	

Procedures/Methods Example:
- accept program blocks
- access machine model
- verify manual tasks
- halt processing
- cue operator for task
- resume processing with operator response

Sub-responsibilities:
- verify manual tasks
- halt block processing
- resume processing

Responsibility: NGC-49 Handle operator command interactions

Temporal Aspects		Incoming Messages:	Message-Initiated Process:
Initiation:	system start up		
Completion:	system shut down	Outgoing Messages:	
Span:	always running		
Discrete / Continuous			

	1	2	3	4
Local Resources				
Description:				
Attributes:				
Constraints:				
Source:				
External Resources				
Description:	part program	manual interaction		
Likely Source:	workstation management	operator interface		
Reliability:	weak	weak		
Con⅛cts:	not likely	not likely		
Products				
Description:	execution commands	error message		
Attributes:	code: parameters	code: parameter		
Constraints:	non-permanent change	interaction invalid		
Likely Recipients	plan interpretation	operator interface		

Procedures/Methods Example:

- accept program
- halt program when interactions needed
- insert execution commands if valid
- send execution commands

Sub-responsibilities:

- halt program
- insert execution commands if valid

Responsibility: NGC-50 Handle modi⅛ations of offset values by operator

Temporal Aspects		Incoming Messages:	Message-Initiated Process:
Initiation:	operator signal		
Completion:	part program start or shutdown	Outgoing Messages.	
Span:	always running		
Discrete / Continuous			

	1	2	3	4
[illegible]				
[illegible]				
[illegible]				
[illegible]				
[illegible]				
External Resources				
Description:	offset values	machine models		
Likely Source:	operator interface	knowledge base		
Reliability:	strong	strong		
Con⅛cts:	not likely	not likely		
Products				
Description:	modi⅛ed machine models	operator log		
Attributes:	codes: parameters	code: parameter		
Constraints:	operator capabilities	memory size		
Likely Recipients	knowledge base	operating system		

Procedures/Methods Example:

- accept operator offsets
- log commands
- modify machine models
- send new models

Sub-responsibilities:

Responsibility: NGC-51 Handle operator interface for manual interpolation commands

Temporal Aspects		Incoming Messages:	Message-Initiated Process:
Initiation:	system start up		
Completion:	system shut down	Outgoing Messages:	
Span:	always running		
Discrete / Continuous			

	1	2	3	4
Local Resources				
Description:				
Attributes:				
Constraints:				
Source:				
External Resources				
Description:	trajectory	manual interpolation command		
Likely Source:	trajectory generation	operator interface		
Reliability:	strong	strong		
Con¼cts:	not likely	not likely		
Products				
Description:	servo commands	halt commands		
Attributes:	voltage	code: parameter		
Constraints:	accuracy of operator			
Likely Recipients	axis motor	trajectory generation		

Procedures/Methods
Example:
- accept trajectory
- accept manual interpolation command
- send halt to trajectory generation
- interpolate for next setpoint
- send servo commands

Sub-responsibilities:
- interpolate for next setpoint

Responsibility: NGC-52 Ensure safety

Temporal Aspects		Incoming Messages:	Message-Initiated Process:
Initiation:	system start up		
Completion:	system shut down	Outgoing Messages:	
Span:	always running		
Discrete / Continuous			

	1	2	3	4
Local Resources				
Description:				
Attributes:				
Constraints:				
Source:				
External Resources				
Description:	voltages	heartbeats	tolerances	
Likely Source:	hardware interface	all components	knowledge base	
Reliability:	strong	strong	strong	
Con¼cts:	not likely	not likely	not likely	
Products				
Description:	watchdog warning			
Attributes:	codes: parameters			
Constraints:	out of tolerance			
Likely Recipients	shutdown component			

Procedures/Methods
Example:
- accept voltages
- access heartbeats
- compare against tolerances
- send warning if tolerances exceeded

Sub-responsibilities:

Responsibility: NGC-53 Generate a part program

Temporal Aspects
Initiation: part design completed
Completion: program developed
Span: product development
Discrete / Continuous

Incoming Messages:

Outgoing Messages:

Message-Initiated Process:

	1	2	3	4
Local Resources				
Description:				
Attributes:				
Constraints:				
Source:				
External Resources				
Description:	part model	stock model	operations available	machine model
Likely Source:	knowledge base	knowledge base	knowledge base	knowledge base
Reliability:	strong	strong	strong	strong
Conflicts:	not likely	not likely	not likely	not likely
Products				
Description:	part program			
Attributes:	code; parameters			
Constraints:				
Likely Recipients	workstation mgt			

Procedures/Methods
Example:
- accept part, stock, machine models
- access operations available
- determine sequence of operations
- send part program

Sub-responsibilities:
- determine sequence of operations that will change stock (as is) to part (to be)

Responsibility: NGC-54 Customize part program for particular machine

Temporal Aspects
Initiation: new part program
Completion: part program customized
Span: product development
Discrete / Continuous

Incoming Messages:

Outgoing Messages:

Message-Initiated Process:

	1	2	3	4
Local Resources				
Description:				
Attributes:				
Constraints:				
Source:				
External Resources				
Description:	part program	machine model		
Likely Source:	workstation mgt	knowledge base		
Reliability:	strong	strong		
Conflicts:	not likely	not likely		
Products				
Description:	customized part program			
Attributes:	code; parameters			
Constraints:				
Likely Recipients	workstation mgt			

Procedures/Methods
Example:
- accept part program
- access machine model
- customize program for machine
- send customized part program

Sub-responsibilities:
- customize program for machine

Responsibility: NGC-55 Schedule processes in part program for optimal coordination

Temporal Aspects		Incoming Messages:	Message-Initiated Process:
Initiation:	part program completed		
Completion:	part program re-sequenced	Outgoing Messages:	
Span:	product development		
Discrete / Continuous			

	1	2	3	4
Local Resources				
Description:				
Attributes:				
Constraints:				
Source:				
External Resources				
Description:	part program	part model	stock model	
Likely Source:	workstation mgt	knowledge base	knowledge base	
Reliability:	strong	strong	strong	
Conflicts:	not likely	not likely	not likely	
Products				
Description:	resequenced part program			
Attributes:	code; parameters			
Constraints:				
Likely Recipients	workstation mgt			

Procedures/Methods Example:
- accept part program
- access part, stock models
- analyze program for optimality
- send resequenced part program

Sub-responsibilities:
- analyze program for optimality

Responsibility: NGC-56 Simulate part program execution

Temporal Aspects		Incoming Messages:	Message-Initiated Process:
Initiation:	simulation initiated		
Completion:	simulation complete	Outgoing Messages:	
Span:	maintenance		
Discrete / Continuous			

	1	2	3	4
Local Resources				
Description:				
Attributes:				
Constraints:				
Source:				
External Resources				
Description:	part program	start command	hardware models	software models
Likely Source:	workstation mgt	operator interface	knowledge base	knowledge base
Reliability:	strong	strong	strong	strong
Conflicts:	not likely	not likely	not likely	not likely
Products				
Description:	simulation results			
Attributes:	code; parameters			
Constraints:				
Likely Recipients	operator interface			

Procedures/Methods Example:
- accept part program and start command
- retard commands to actuators, axes
- respond to commands as devices would
- send results

Sub-responsibilities:

Responsibility: NGC-57 Simulate software faults

Temporal Aspects	
Initiation:	simulation initiated
Completion:	simulation complete
Span:	maintenance
Discrete / Continuous	

Incoming Messages:

Outgoing Messages:

Message-Initiated Process:

	1	2	3	4
~~Local Resources~~				
Description:				
Attributes:				
Constraints:				
Source:				
External Resources				
Description:	part program	software errors model		
Likely Source:	workstation mgt	knowledge base		
Reliability:	strong	strong		
Conflicts:	not likely	not likely		
Products				
Description:	modified software models			
Attributes:	code: parameters			
Constraints:	errors inserted			
Likely Recipients	knowledge base			

Procedures/Methods

Example:
- accept part program and errors model
- modify software model to contain selected errors
- send modified software model

Sub-responsibilities:

Responsibility: NGC-58 Simulate hardware faults

Temporal Aspects	
Initiation:	simulation initiated
Completion:	simulation complete
Span:	maintenance
Discrete / Continuous	

Incoming Messages:

Outgoing Messages:

Message-Initiated Process:

	1	2	3	4
[illegible]				
[illegible]				
[illegible]				
[illegible]				
[illegible]				
External Resources				
Description:	part program	hardware errors mode		
Likely Source:	workstation mgt	knowledge base		
Reliability:	strong	strong		
Conflicts:	not likely	not likely		
Products				
Description:	modified hardware model			
Attributes:	code: parameters			
Constraints:	errors inserted			
Likely Recipients	knowledge base			

Procedures/Methods

Example:
- accept part program and errors model
- modify hardware models to contain selected errors
- send modified hardware model

Sub-responsibilities:

Responsibility: NGC-59 Create a part design

Temporal Aspects	
Initiation:	part identified
Completion:	design completed
Span:	product development
Discrete / Continuous	

Incoming Messages:

Outgoing Messages:

Message-Initiated Process:

	1	2	3	4
Local Resources				
Description:				
Attributes:				
Constraints:				
Source:				
External Resources				
Description:	part description	part requirements	physical laws	
Likely Source:	engineering	engineering	knowledge base	
Reliability:	weak	weak	strong	
Conflicts:	not likely	not likely	not likely	
Products				
Description:	part design model			
Attributes:	features			
Constraints:				
Likely Recipients	model management			

Procedures/Methods
Example:
- accept part description & requirements
- access physical laws
- design part
- send part design model

Sub-responsibilities:
- design part

Responsibility: NGC-60 Notify maintenance when parameters exceed tolerances

Temporal Aspects	
Initiation:	system startup
Completion:	system shutdown
Span:	execution and prove-out
Discrete / Continuous	

Incoming Messages:

Outgoing Messages:

Message-Initiated Process:

	1	2	3	4
Local Resources				
Description:				
Attributes:				
Constraints:				
Source:				
External Resources				
Description:	machine model	machine status	sensed status	
Likely Source:	knowledge base	components	sensor interface	
Reliability:	strong	weak	weak	
Conflicts:	not likely	not likely	not likely	
Products				
Description:	exception notification			
Attributes:	code; parameters			
Constraints:				
Likely Recipients:	factory control			

Procedures/Methods
Example:
- accept machine model
- monitor machine status and sensed status
- compare status against tolerances
- send exception if out of tolerance

Sub-responsibilities:

Responsibility: NGC-61 Maintain models

TemporalAspects		Incoming Messages:	Message-Initiated Process:
Initiation:	system startup		
Completion:	system shutdown	Outgoing Messages:	
Span:	execution and prove-out		
Discrete / Continuous			

	1	2	3	4
Local Resources				
Description:				
Attributes:				
Constraints:				
Source:				
External Resources				
Description:	updates to models	models		
Likely Source:	op if/mechanism/motion	knowledge base		
Reliability:	strong	strong		
Con¼cts:	not likely	not likely		
Products				
Description:	updated models			
Attributes:				
Constraints:				
Likely Recipients:	knowledge base			

Procedures/Methods
Example:
- accept updates to models
- access models
- validate updates
- send updated models

Sub-responsibilities:

Responsibility: NGC-62 Initiate job sequence

TemporalAspects		Incoming Messages:	Message-Initiated Process:
Initiation:	system startup		
Completion:	system shutdown	Outgoing Messages:	
Span:	execution and prove-out		
Discrete / Continuous			

	1	2	3	4
Local Resources				
Description:				
Attributes:				
Constraints:				
Source:				
External Resources				
Description:	current program status	job list		
Likely Source:	plan interpretation	workstation mgt		
Reliability:	strong	strong		
Con¼cts:	not likely	not likely		
Products				
Description:	request to start next cycle			
Attributes:				
Constraints:				
Likely Recipients:	operator interface			

Procedures/Methods
Example:
- accept current program status
- access job list
- determine if current program is completed
- send request to start next cycle

Sub-responsibilities:

Responsibility: NGC-63 Verify proper part program and revision number for each part

Temporal Aspects	
Initiation:	system startup
Completion:	system shutdown
Span:	execute and prove-out
Discrete / Continuous	

Incoming Messages:

Outgoing Messages:

Message-Initiated Process:

	1	2	3	4
Local Resources				
Description:				
Attributes:				
Constraints:				
Source:				
External Resources				
Description:	part id	part program	job list	
Likely Source:	workstation mgt	workstation mgt	factory scheduler	
Reliability:	strong	strong	strong	
Conflicts:	not likely	not likely	not likely	
Products				
Description:	verification notification			
Attributes:				
Constraints:				
Likely Recipients:	workstation mgt			

Procedures/Methods Example:

- accept part id and part program
- access job list
- verify that all match
- send verification notification

Sub-responsibilities:

Responsibility: NGC-64 Identify current part

Temporal Aspects	
Initiation:	system startup
Completion:	system shutdown
Span:	execution and prove-out
Discrete / Continuous	

Incoming Messages:

Outgoing Messages:

Message-Initiated Process:

	1	2	3	4
Local Resources				
Description:				
Attributes:				
Constraints:				
Source:				
External Resources				
Description:	request for part id			
Likely Source:	workstation mgt			
Reliability:	strong			
Conflicts:	not likely			
Products				
Description:	part id			
Attributes:	code; parameters			
Constraints:				
Likely Recipients:	workstation mgt			

Procedures/Methods Example:

- accept request for part id
- access sensor for id
- send part id

Sub-responsibilities:

Responsibility: NGC-65 Ensure sanctity of safe zones

Temporal Aspects		Incoming Messages:	Message-Initiated Process:
Initiation:	system startup		
Completion:	system shutdown	Outgoing Messages:	
Span:	execution and prove-out		
Discrete / Continuous			

	1	2	3	4
Local Resources				
Description:				
Attributes:				
Constraints:				
Source:				
External Resources				
Description:	trajectory	part model	machine model	
Likely Source:	trajectory generator	knowledge base	knowledge base	
Reliability:	strong	strong	strong	
Conflicts:	not likely	not likely	not likely	
Products				
Description:	modified trajectory			
Attributes:	setpoints; parameters			
Constraints:				
Likely Recipients:	trajectory generator			

Procedures/Methods	Sub-responsibilities:
Example: • accept trajectory • access part model, machine model • modify trajectory if safe zones violated • send modified trajectory	

Responsibility: NGC-66 Reassignment of tasks between machine and operator

Temporal Aspects		Incoming Messages:	Message-Initiated Process:
Initiation:	system startup		
Completion:	system shutdown	Outgoing Messages:	
Span:	execution and prove-out		
Discrete / Continuous			

	1	2	3	4
Local Resources				
Description:				
Attributes:				
Constraints:				
Source:				
External Resources				
Description:	part program	operator reassignments		
Likely Source:	plan interpretation	operator interface		
Reliability:	strong	strong		
Conflicts:	not likely	not likely		
Products				
Description:	modified part program			
Attributes:	code; parameters			
Constraints:				
Likely Recipients:	plan interpretation			

Procedures/Methods	Sub-responsibilities:
Example: • accept operator reassignments • access part program • modify program per operator reassignments • send modified part program	

Responsibility: NGC-67 Coordinate spindle gear changes with servo control

Temporal Aspects	Incoming Messages:	Message-Initiated Process:
Initiation: motion started		
Completion: motion stopped	Outgoing Messages:	
Span: execution and prove-out		
Discrete / Continuous		

	1	2	3	4
Local Resources				
Description:				
Attributes:				
Constraints:				
Source:				
External Resources				
Description:	spindle gear state	movement commands	machine model	
Likely Source:	mechanism	motion	knowledge base	
Reliability:	strong	strong	strong	
Conflicts:	not likely	not likely	not likely	
Products				
Description:	movement commands			
Attributes:				
Constraints:				
Likely Recipients:	motion			

Procedures/Methods
Example:
- accept movement commands
- access spindle gear state, machine model
- modify movement commands according to spindle gear
- send modified movement commands

Sub-responsibilities:

Responsibility: NGC-68 Re-evaluate coordinate zeros for alternate tool choices

Temporal Aspects	Incoming Messages:	Message-Initiated Process:
Initiation: system startup		
Completion: system shutdown	Outgoing Messages:	
Span: execution and prove-out		
Discrete / Continuous		

	1	2	3	4
Local Resources				
Description:				
Attributes:				
Constraints:				
Source:				
External Resources				
Description:	tool substitution	coordinate system model	tool model	
Likely Source:	mechanism	knowledge base	knowledge base	
Reliability:	strong	strong	strong	
Conflicts:	not likely	not likely	not likely	
Products				
Description:	new coordinate zeros			
Attributes:				
Constraints:				
Likely Recipients	plan interpretation			

Procedures/Methods
Example:
- accept tool substitution
- access tool and coordinate system model
- determine new coordinate zeros
- send new coordinate zeros

Sub-responsibilities:

Responsibility: NGC-69 Translate NURBS trajectories into motion commands

Temporal Aspects
Initiation: start of part program
Completion: program completed
Span: execution and prove-out
Discrete / Continuous

Incoming Messages:

Outgoing Messages:

Message-Initiated Process:

	1	2	3	4
~~Local Resources~~				
Description:				
Attributes:				
Constraints:				
Source:				
External Resources				
Description:	NURBS trajectory	NURBS model		
Likely Source:	plan generation	knowledge base		
Reliability:	strong	strong		
Con¼cts:	not likely	not likely		
Products				
Description:	motion commands			
Attributes:	setpoints; parameters			
Constraints:				
Likely Recipients	actuator control			

Procedures/Methods
Example:
- accept NURBS trajectory
- accept NURBS model
- determine setpoints and parameters
- send new trajectory

Sub-responsibilities:
- determine setpoints and parameters

Responsibility: NGC-70 Manage canned cycles

Temporal Aspects
Initiation: start of part program
Completion: program completed
Span: execution and prove-out
Discrete / Continuous

Incoming Messages:

Outgoing Messages:

Message-Initiated Process:

	1	2	3	4
Local Resources				
Description:				
Attributes:				
Constraints:				
Source:				
External Resources				
Description:	part program	machine model	canned cycle lexicon	
Likely Source:	workstation management	knowledge base	knowledge base	
Reliability:	weak	strong	strong	
Con¼cts:	not likely	not likely	not likely	
Products				
Description:	detailed program commands			
Attributes:	codes; parameters			
Constraints:	limited cycles			
Likely Recipients	trajectory generation			

Procedures/Methods
Example:
- accept part program
- access machine model
- access canned cycle lexicon
- parse cycle into individual commands
- send detailed individual commands

Sub-responsibilities:
- parse cycle into individual commands

Responsibility: NGC-71 Noti¼cation of errors such as singularityashup, unreachable point

Temporal Aspects		Incoming Messages:	Message-Initiated Process:
Initiation:	start of part program		
Completion:	program completed	Outgoing Messages:	
Span:	execution and prove-out		
Discrete / Continuous			

	1	2	3	4
Local Resources				
Description:				
Attributes:				
Constraints:				
Source:				
External Resources				
Description:	trajectory setpoints	machine model	sensor data	error descriptions
Likely Source:	trajectory generation	knowledge base	sensor interface	knowledge base
Reliability:	strong	strong	weak	strong
Con¼cts:	not likely	not likely	likely	not likely
Products				
Description:	error noti¼cation	halt command		
Attributes:	code: parameters	code: parameters		
Constraints:	accuracy of sensors			
Likely Recipients	operator interface	all components		

Procedures/Methods
Example:
- model machine as program executes
- compare model against error descriptions
- send halt command if match
- send noti¼cation if match

Sub-responsibilities:
- model machine as program executes
- compare model against error descriptions

Responsibility: NGC-72 Automatically set feedrate / feedforce at point of contact

Temporal Aspects		Incoming Messages:	Message-Initiated Process:
Initiation:	automatic feedrate/force activated		
Completion:	automatic feedrate/force deactivated	Outgoing Messages:	
Span:	prove-out and execution		
Discrete / Continuous			

	1	2	3	4
[illegible]				
[illegible]				
[illegible]				
[illegible]				
[illegible]				
External Resources				
Description:	feedrate / feedforce value	sensor data	machine model	
Likely Source:	motion	sensor interface	knowledge base	
Reliability:	strong	weak	strong	
Con¼cts:	not likely	likely	not likely	
Products				
Description:	servo commands			
Attributes:	voltage			
Constraints:	accuracy of machine model			
Likely Recipients	axis motor			

Procedures/Methods
Example:
- accept feedrate/force desired
- monitor sensors
- compare against machine model tolerances
- adjust voltage to maintain tolerance
- send servo commands

Sub-responsibilities:
- monitor sensors
- compare against machine model tolerances

Responsibility: NGC-73 Execute plans targeted at discrete actuators			
Temporal Aspects		Incoming Messages:	Message-Initiated Process:
Initiation:	start of part program		
Completion:	program completed	Outgoing Messages:	
Span:	execution and prove-out		
Discrete / Continuous			

	1	2	3	4
Local Resources				
Description:				
Attributes:				
Constraints:				
Source:				
External Resources				
Description:	discrete commands	machine model		
Likely Source:	plan interpretation	knowledge base		
Reliability:	strong	strong		
Conflicts:	not likely	not likely		
Products				
Description:	actuator commands			
Attributes:	voltage			
Constraints:	device types			
Likely Recipients:	hardware interface			

Procedures/Methods Example: • accept discrete commands • convert for appropriate voltage per device • send voltage	Sub-responsibilities:

Responsibility: NGC-74 Evaluate part with probe			
Temporal Aspects		Incoming Messages:	Message-Initiated Process:
Initiation:	program startup		
Completion:	program shutdown	Outgoing Messages:	
Span:	execution and prove-out		
Discrete / Continuous			

	1	2	3	4
[illegible]				
[illegible]				
[illegible]				
[illegible]				
[illegible]				
External Resources				
Description:	part model	probe readings	machine model	
Likely Source:	knowledge base	sensor interface	knowledge base	
Reliability:	strong	weak	strong	
Conflicts:	not likely	not likely	not likely	
Products				
Description:	probe selection	probe commands	movement commands	probe results
Attributes:	code: parameters	code: parameters	code: parameters	code: parameters
Constraints:				
Likely Recipients	tool change mechanism	sensor interface	motion	workstation mgt

Procedures/Methods Example: • access part, machine models • send command to change tool to probe • measure part • send probe results	Sub-responsibilities: • measure part

Responsibility: NGC-75 Evaluate tool with probe

Temporal Aspects		Incoming Messages:	Message-Initiated Process:
Initiation:	program startup		
Completion:	program shutdown	Outgoing Messages:	
Span:	execution and prove-out		
Discrete / Continuous			

	1	2	3	4
Local Resources				
Description:				
Attributes:				
Constraints:				
Source:				
External Resources				
Description:	tool model	probe readings	machine model	
Likely Source:	knowledge base	sensor interface	knowledge base	
Reliability:	strong	strong	strong	
Con½cts:	not likely	not likely	not likely	
Products				
Description:	probe selection	probe commands	movement commands	probe results
Attributes:	code: parameters	code: parameters	code: parameters	code: parameters
Constraints:				
Likely Recipients:	tool change mechanism	sensor interface	motion	workstation mgt

Procedures/Methods Example: • access tool, machine models • send command to change tool to probe • measure tool • send probe results	**Sub-responsibilities:** • measure part

Responsibility: NGC-76 Automatically learn machine'kinematics

Temporal Aspects		Incoming Messages:	Message-Initiated Process:
Initiation:	system startup		
Completion:	system shutdown	Outgoing Messages:	
Span:	execution and prove-out		
Discrete / Continuous			

	1	2	3	4
Local Resources				
Description:				
Attributes:				
Constraints:				
Source:				
External Resources				
Description:	machine status	sensed status	kinematic equations	kin eq. history
Likely Source:	components	sensor interface	knowledge base	knowledge base
Reliability:	strong	weak	strong	strong
Con½cts:	not likely	likely	not likely	not likely
Products				
Description:	modi½ed kinematics			
Attributes:	matrices			
Constraints:				
Likely Recipients	knowledge base			

Procedures/Methods Example: • access kinematic equations and history • monitor machine status • analyze history and status • modify and send equations	**Sub-responsibilities:** • analyze history and status

APPENDIX C — ARCHITECTURE DESCRIPTION RULES

The following rules guide the construction of the components of an application architecture. Some of them are based on requirements, others are based on primitive components. The rules are expressed in simple English, although the architecture description language (ADL) explained in Appendix F could be used for ease in computation. Sequence numbers were added for ease of reference.

Most Important Structural Rule:

If a primitive component (PC NGC-#) needs a resource,
you need some other PC that has that resource as a product.

C.1 ADL Rules Accessing Multiple Primitive Components

C.1.1 If you want to control an axis,
you need PC NGC-38, Control motors to produce movement,
you need PC NGC-49, Handle operator command interactions.

C.1.2 If you have PC NGC-29,
you need PC NGC-28, Normalize trajectory with respect to time,
you need PC NGC-37, Translate servo commands into drive readable movement commands.

C.1.3 If you want to maximize speed,
you need PC NGC-26, Control acceleration and deceleration,
you need PC NGC-32, Determine rate of movement.

C.1.4 If you want to optimize cutting,
you need PC NGC-4, Maintain information on available tools,
you need PC NGC-26, Control acceleration and deceleration,
you need PC NGC-32, Determine rate of movement,
you need PC NGC-35, Determine trajectory corrections from predictable variations,
you need force sensors,
you need torque sensors.

C.1.5 If you want to control tool changes,
you need PC NGC-21, Load and unload tool,
you need PC NGC-20, Map tool id to actual tool location,
you need PC NGC-19, Swap current tool with required tool,
you need PC NGC-22, Track tool locations.

C.1.6 If you want to control pallet changes,
you need PC NGC-23, Swap current part with next part,
you need PC NGC-24, Identify current part.

C.1.7 If you have PC NGC-15,
you need PC NGC-17, Interpret augmented code,
you need PC NGC-18, Determine wait/continue commands for coordination.

C.1.8 If you are coordinating mechanisms and motion through a high-level language,
you need PC NGC-11, Determine where to send part program codes,
you need PC NGC-12, Interpretation of part program,
you need PC NGC-13, Augment code with component destinations.

C.1.9 If your high-level language commands are embodied in a part program,
you need PC NGC-14, Inform operator of part program comments,
you need PC NGC-9, Initialize start of part program activities,
you need PC NGC-10, Initialize end of part program activities,
you need PC NGC-48, Cue the operator for manual tasks,
you need PC NGC-49, Handle operator command interactions.

C.1.10 If you want to work multiple parts with multiple programs as a job,
you need PC NGC-3, Determine each part's program and coordinate offsets,
you need PC NGC-7, Determine status of active program,
you need PC NGC-5, Ensure that resources required by the jobs are available,
you need PC NGC-6, Determine if next part's program matches current part's program,
you need PC NGC-62, Initiate job sequence.

C.1.11 If you want to create new parts,
you need PC NGC-59, Create a part design,
you need PC NGC-53, Generate a part program,
you need PC NGC-1, Handle operator modifications to part program statements,
you need PC NGC-54, Customize part program for particular machine,
you need PC NGC-55, Schedule processes in part program for optimal coordination.

C.1.12 If you want to simulate operations,
you need PC NGC-56, Simulate part program execution,
you need PC NGC-57, Simulate software faults,
you need PC NGC-58, Simulate hardware faults.

C.2 ADL Rules Linking Primitive Components

C.2.1 If you have PC NGC-34,
you need PC NGC-36, Adjust trajectory with corrections generated by sensor data.

C.2.2 If you have PC NGC-33,
you need PC NGC-4, Maintain information on available tools.

C.3 ADL Rules Accessing Single Primitive Components

C.3.1 If you want the operator to customize the part program,
you need PC NGC-1, Handle operator modifications to part program statements.

C.3.2 If you are going to store part programs for automatic access,
you need PC NGC-2, Part program storage activities.

C.3.3 If you want to execute a job list of multiple arts and programs,
you need PC NGC-3, Determine each part's program and coordinate offsets.

C.3.4 If you have dynamic tool data,
you need PC NGC-4, Maintain information on available tools.

C.3.5 If you want to execute programs with various resources,
you need PC NGC-5, Ensure that resources required by the jobs are available.

C.3.6 If you want to execute various programs,
you need PC NGC-6, Determine if next part's program matches current part's program.

C.3.7 If you need to keep track of the executing program,
you need PC NGC-7, Determine status of active program.

C.3.8 If you cannot load an entire program at once,
you need PC NGC-8, Request additional segments of part program.

C.3.9 If you need to perform activities at the start of every program,
you need PC NGC-9, Initialize start of part program activities.

C.3.10 If you need to perform activities at the end of every program,
you need PC NGC-10, Initiate end of part program activities.

C.3.11 If you need to distribute code to various components,
you need PC NGC-11, Determine where to send part program codes.

C.3.12 If you need to parse code for motion and mechanisms,
you need PC NGC-12, Interpretation of part program.

C.3.13 If you need to track recipients of code,
you need PC NGC-13, Augment code with component destinations.

C.3.14 If you want the operator to follow the program execution,
you need PC NGC-14, Inform operator of part program comments.

C.3.15 If you want to coordinate mechanism control with motion control,
you need PC NGC-15, Augment code for coordination between motion and mechanisms.

C.3.16 If you have multiple coordinate systems,
you need PC NGC-16, Translate coordinate systems.

C.3.17 If you need to track program execution,
you need PC NGC-17, Interpret augmented code.

C.3.18 If you need to coordinate notion and mechanisms,
you need PC NGC-18, Determine wait/continue commands for coordination.

C.3.19 If you have multiple tools,
you need PC NGC-19, Swap current tool with required tool.

C.3.20 If you have dynamic tool location assignments,
you need PC NGC-20, Map tool id to actual tool location.

C.3.21 If your operator places tools in the tool changer,
you need PC NGC-21, Load and unload tools.

C.3.22 If you the tool changer swaps tool locations,
you need PC NGC-22, Track tool locations.

C.3.23 If the part handler can automatically change parts,
you need PC NGC-23, Swap current part with next part.

C.3.24 If you need to verify the loaded pallet,
you need PC NGC-24, Identify current part.

C.3.25 If you want to control a continuous flow device,
you need PC NGC-25, Maintain control of feedrate-dependent operations.

C.3.26 If you want to ensure the trajectory follows the machine's physical limitations,
you need PC NGC-26, Control acceleration and deceleration.

C.3.27 If you need to modify the trajectory,
you need PC NGC-27, Define normals to path.

C.3.28 If you need to follow a trajectory based on setpoints,
you need PC NGC-28, Normalize trajectory with respect to time.

C.3.29 If you want to enter high-level commands instead of axis commands,
you need PC NGC-29, Translate motion commands to trajectory.

C.3.30 If you need motion and mechanisms to coordinate,
you need PC NGC-30, Coordinate with mechanisms.

C.3.31 If you want to avoid witness marks,
you need PC NGC-31, Avoid data starvation.

C.3.32 If you want the goal rate to adjust for a chip per tooth rate,
you need PC NGC-32, Determine rate of movement.

C.3.33 If you want to adjust for tool variations,
you need PC NGC-33, Adjust trajectory for tool deviations.

C.3.34.1 If you want to adjust for predictable variations in axis linearity,
you need PC NGC-34, Determine trajectory corrections from predictable variations.

C.3.34.2 If you want to adjust for predictable variations with respect to temperature,
you need temperature sensors,
you need to track time of machine powered up,
you need PC NGC-34, Determine trajectory correction from predictable variations.

C.3.35 If you want to adjust for unpredictable variations in trajectory following,
you need PC NGC-35, Determine trajectory corrections from sensed variations,
you need force sensors,
you need torque sensors.

C.3.36 If you want to quickly modify the trajectory with pre-analyzed corrections,
you need PC NGC-36, Adjust trajectory with corrections generated by sensor data..

C.3.37 If you need to interpolate between current position and setpoints,
you need PC NGC-37, Translate servo commands into drive readable movement commands.

C.3.38 If you need a closed loop motion control,
you need PC NGC-38, Control motors to produce movement.

C.3.39 If you need diagnostics and zero positions determined,
you need PC NGC-39, Initiate startup procedures.

C.3.40 If you want automatic job control by the factory scheduling system,
you need PC NGC-40, Coordinate with factory scheduler/control system.

C.3.41 If you want to generate paths with as-is / to-be algorithms,
you need PC NGC-41, Path planning based on part features, surface model, etc.

C.3.42 If you want sensor data to be used for program modifications,
you need PC NGC-42, Modify part program based on sensor data.

C.3.43 If you can control axes by both force and position,
you need PC NGC-43, switch between position control and force control.

C.3.44 If you have more than one axis,
you need PC NGC-44, Plan for multi-axis interactions.

C.3.45 If you want automatic setup of parts,
you need PC NGC-45, Touch off for automatic setup,
you need touch sensors.

C.3.46 If you have multiple sensors providing one piece of information,
you need PC NGC-46, Perform sensor fusion.

C.3.47 If you want to gracefully shutdown form exception conditions, you need PC NGC-47, Initiate shutdown procedures.

C.3.48 If you expect the part program to require interaction with the operator, you need PC NGC-48, Cue the operator for manual tasks.

C.3.49 If you expect the operator to control the program execution, you need PC NGC-49, Handle operator command interactions.

C.3.50 If you want the operator to enter offsets, you need PC NGC-50, Handle modifications of offset values by operator.

C.3.51 If you expect the operator to perform manual interpolation, you need PC NGC-51, Handle operator interface for manual interpolation commands.

C.3.52 If you need a system watchdog, you need PC NGC-52, Ensure safety.

C.3.53 If you want to generate programs with as-is / to-be algorithms, you need PC NGC-53, Generate a part program.

C.3.54 If you need to execute the same program on various machines, you need PC NGC-54, Customize part program for particular machine.

C.3.55 If you want to improve the efficiency of the part program, you need PC NGC-55, Schedule processes in part program for optimal coordination.

C.3.56 If you want to test the system without cutting a part, you need PC NGC-56, Simulate part program execution.

C.3.57 If you want to introduce software errors into the system for testing, you need PC NGC-57, Simulate software faults.

C.3.58 If you want to introduce hardware errors into the system for testing, you need PC NGC-58, Simulate hardware faults.

C.3.59 If you want to generate a manufacturable part design from a concept, you need PC NGC-59, Create a part design.

C.3.60 If you want maintenance to be informed automatically of problems, you need PC NGC-60, Notify maintenance when parameters exceed tolerances.

C.3.61 If you want to keep machine, part, tool (etc) models in a central knowledge-base, you need PC NGC-61, Maintain models.

C.3.62 If you want to setup multiple executions in one job list, you need PC NGC-62, Initiate job sequence.

C.3.63 If you want to ensure the right program is loaded on the right machine for the right part,
you need PC NGC-63, Verify proper part program and revision number for each part.

C.3.64 If you need to verify the part in the fixture,
you need PC NGC-64, Identify current part.

C.3.65 If you want to modify the trajectory to stay within safe zones,
you need PC NGC-65, Ensure sanctity of safe zones.

C.3.66 If you want to allow the operator to perform some machine functions,
you need PC NGC-66, Reassignment of tasks between machine and operator.

C.3.67 If you want to coordinate motion with the spindle gear setting,
you need PC NGC-67, Coordinate spindle gear changes with servo control.

C.3.68 If you allow substitute tools with different offsets,
you need PC NGC-68, Re-evaluate coordinate zeros for alternate tool choices.

C.3.69 If you want to follow complex curves,
you need PC NGC-69, Translate NURBS trajectories into motion commands.

C.3.70 If you want to program using canned cycles,
you need PC NGC-70, Manage canned cycles.

C.3.71 If you want to warn of impending movement errors,
you need PC NGC-71, Notification of errors such as singularity, lashup, unreachable points.

C.3.72 If you want the system to maintain a constant rate or force,
you need PC NGC-72, Automatic feedrate / feedforce at point of contact.

C.3.73 If you want to control discrete actuators,
you need PC NGC-73, Executing plans targeted at discrete actuators.

C.3.74 If you want to measure for part accuracy,
you need PC NGC-74, Evaluate part with probe.

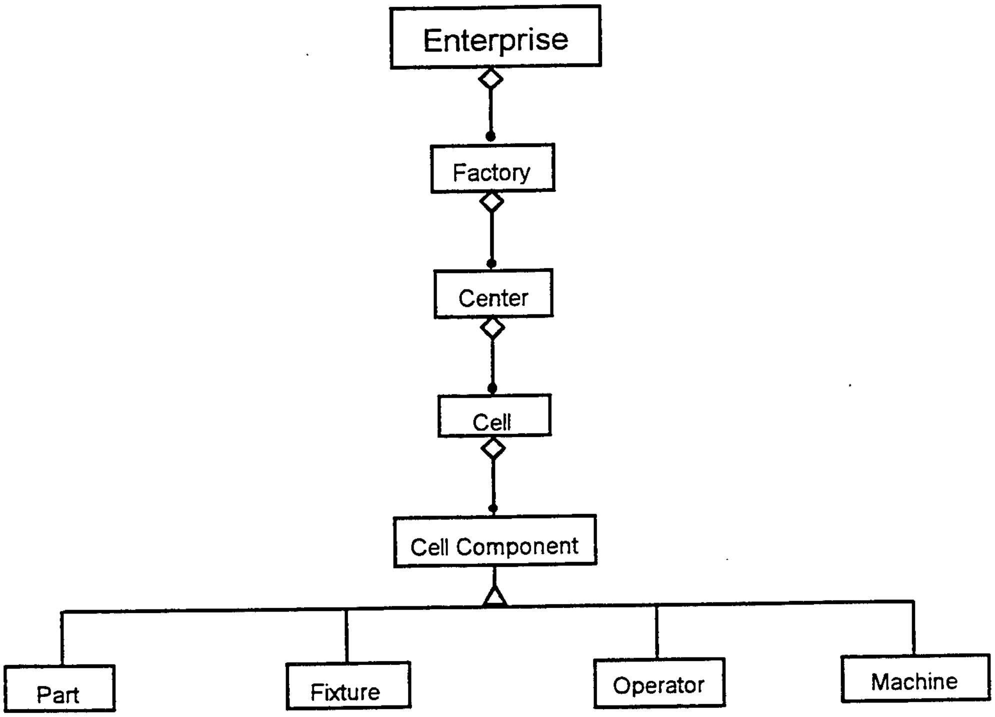
C.3.75 If you want to measure for tool accuracy,
you need PC NGC-75, Evaluate tool with probe.

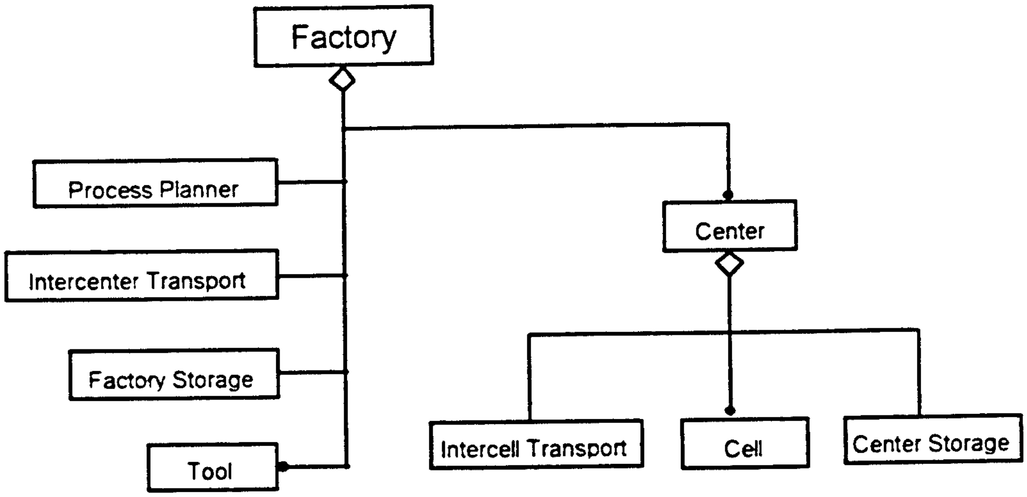
C.3.76 If you want the machine to learn its own physical nature,
you need PC NGC-76, Automatically learn machine's kinematics.

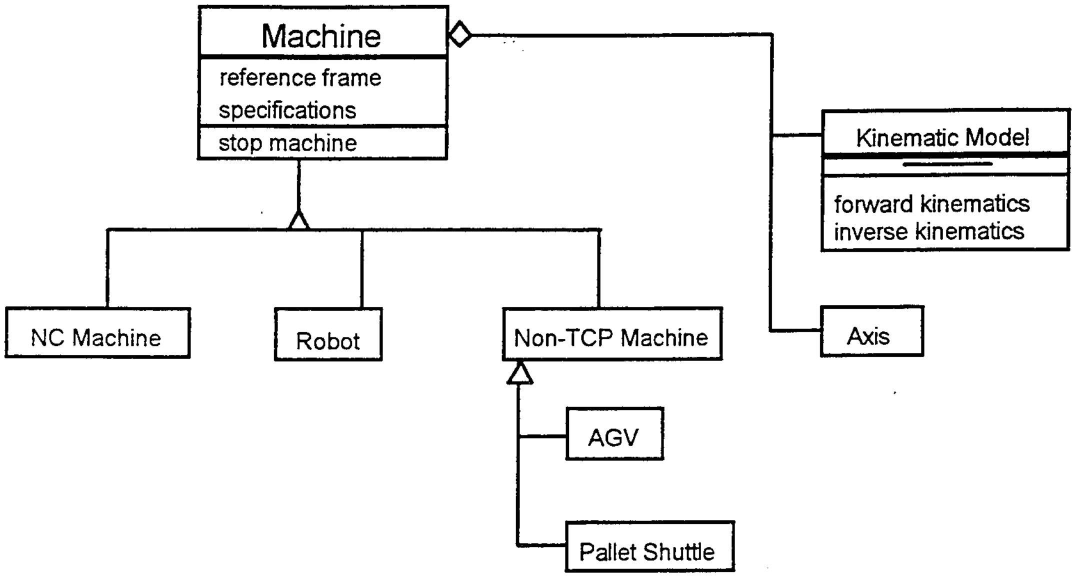
APPENDIX D — DOMAIN MODELS

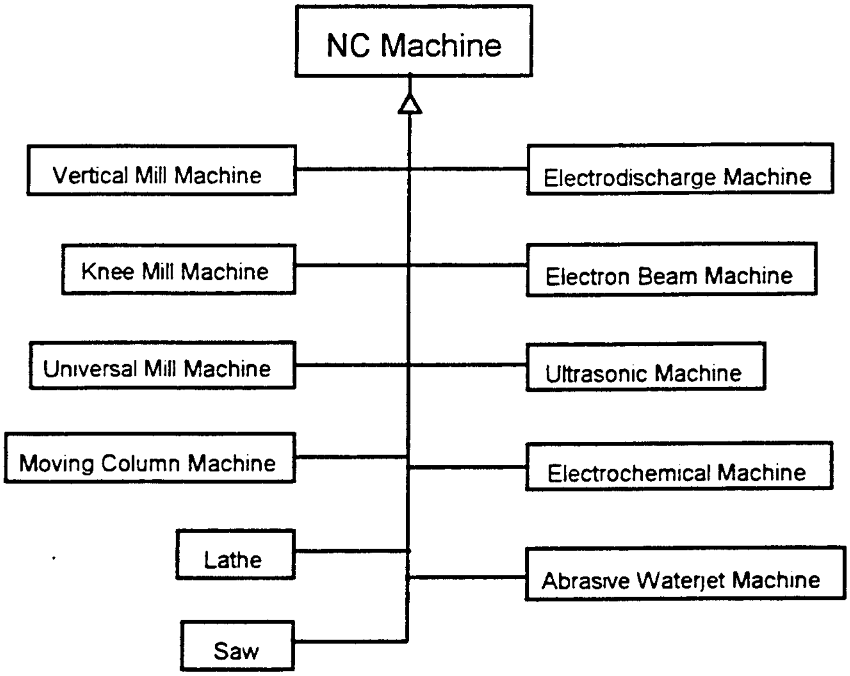
Domain models describe the domain. They include a description of, actions of, relationships to, communications among, and constraints of entities. The implementation may follow an object-oriented paradigm or a functional or other paradigm. Some representations of domain models include object diagrams, task diagrams, topology diagrams, and interaction diagrams. Some domains may be better expressed in specialized models, such as algorithm models or hybrid control models.

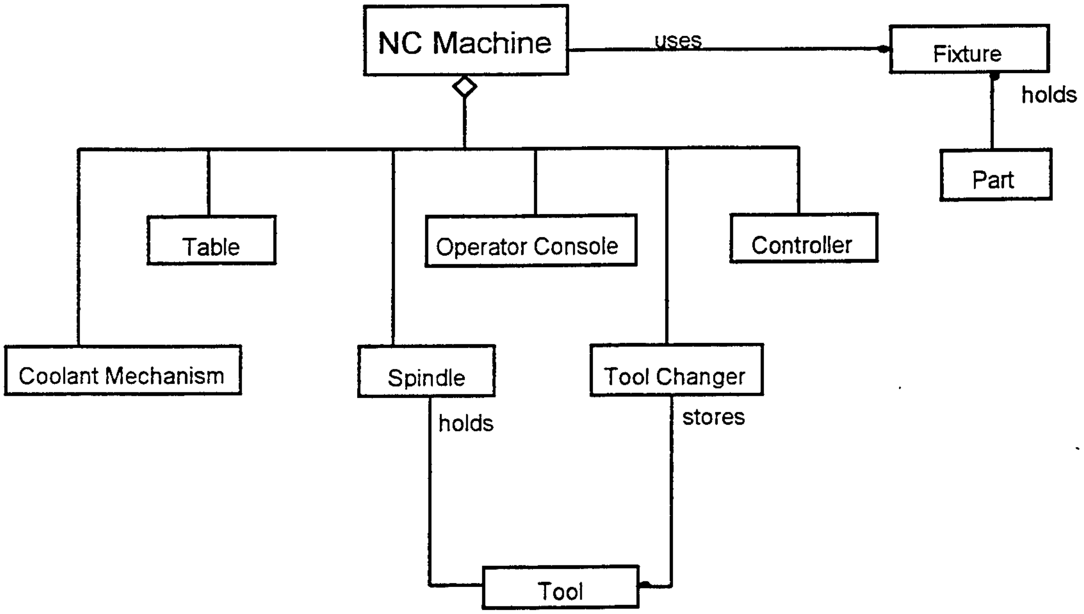
While domain models contain a full view of the domain, the models' appearance changes to support multiple viewpoints. Particular objects, subsets of attributes and services, and certain relationships are only relevant to certain views. Viewpoint also drives the collection of responsibilities into components and the relevance of reference requirements. The domain models for NGC are listed below.

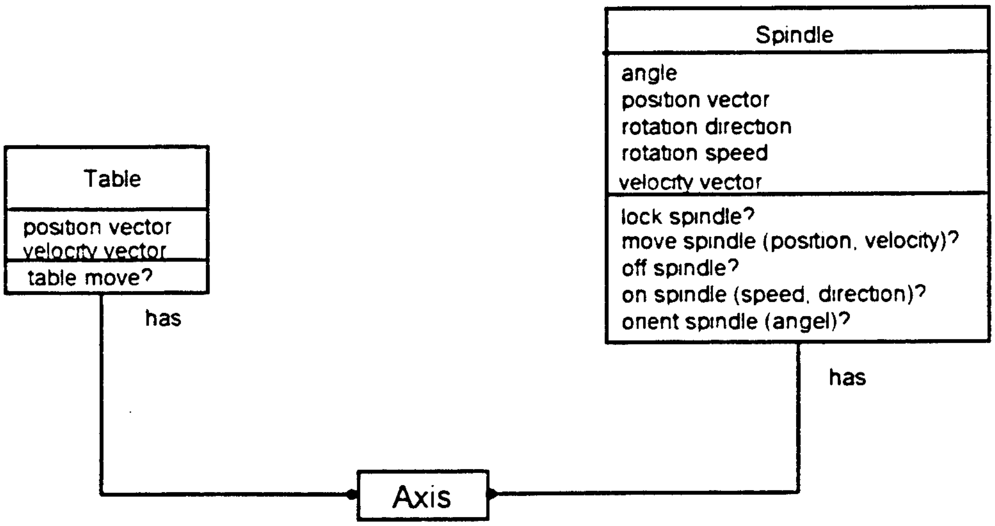

Enterprise
Factory
Center
Cell
Cell Component
Part
Fixture
Operator
Machine

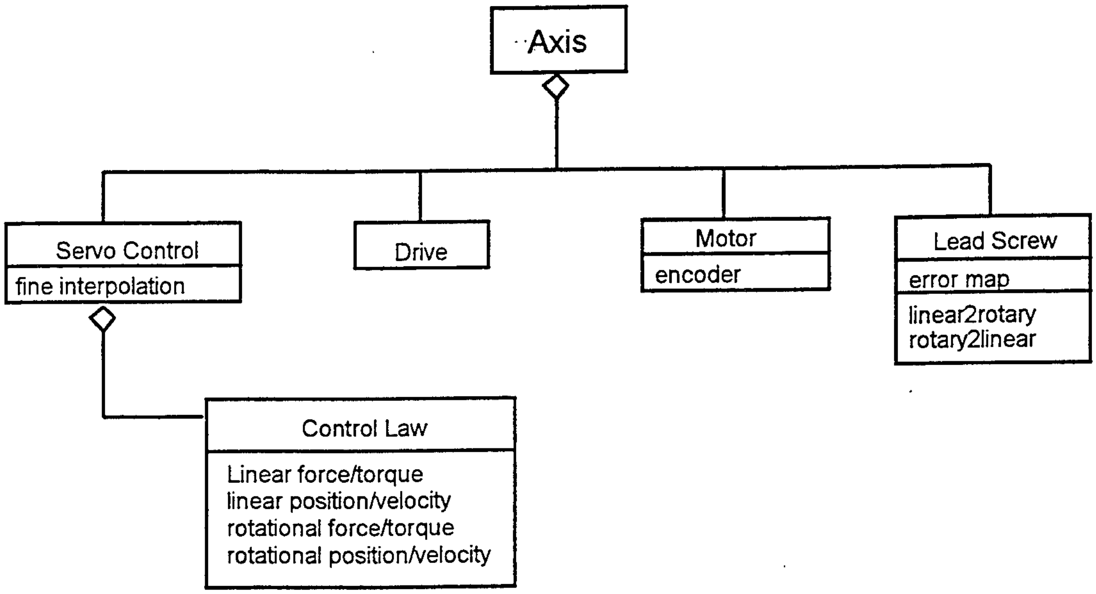

Factory
Process Planner
Intercenter Transport
Factory Storage
Tool
Center
Intercell Transport
Cell
Center Storage

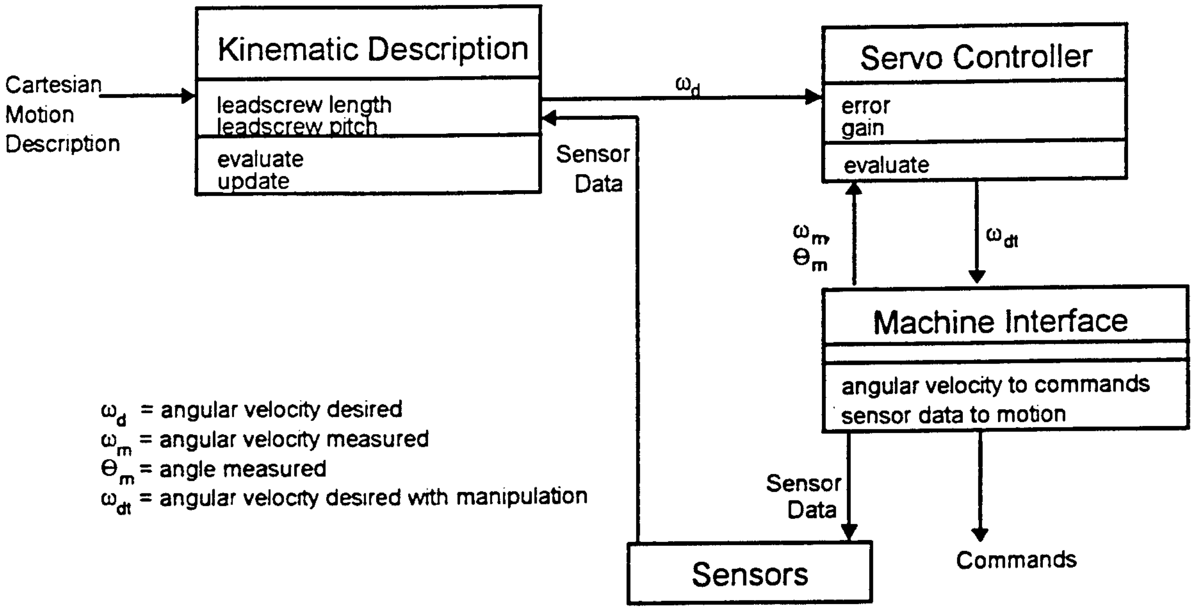

Machine
reference frame
specifications
stop machine
Kinematic Model
forward kinematics
inverse kinematics
NC Machine
Robot
Non-TCP Machine
Axis
AGV
Pallet Shuttle


NC Machine
Vertical Mill Machine
Electrodischarge Machine
Knee Mill Machine
Electron Beam Machine
Universal Mill Machine
Ultrasonic Machine
Moving Column Machine
Electrochemical Machine
Lathe
Abrasive Waterjet Machine
Saw

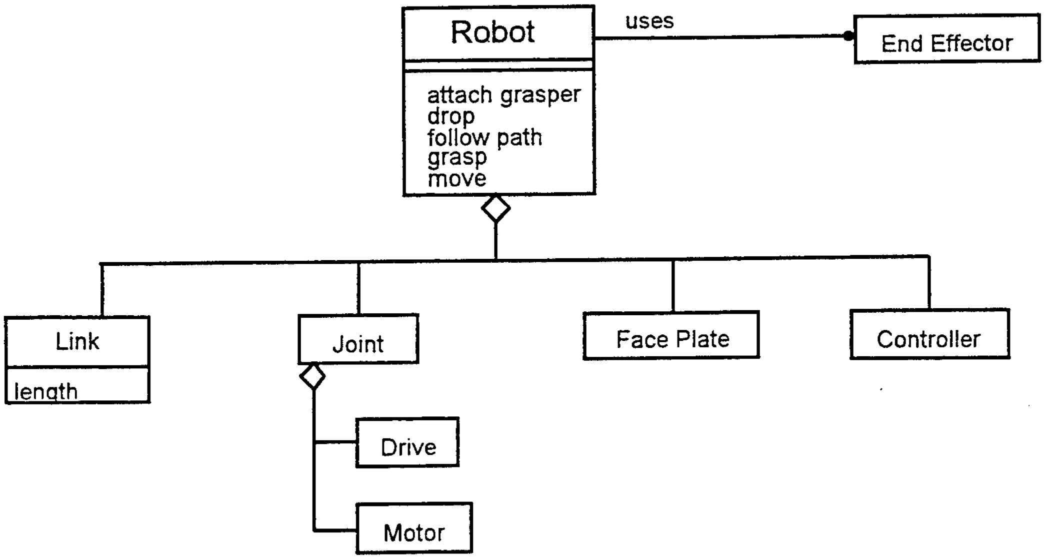

NC Machine
uses
Fixture
holds
Part
Table
Operator Console
Controller
Coolant Mechanism
Spindle
Tool Changer
holds
stores
Tool

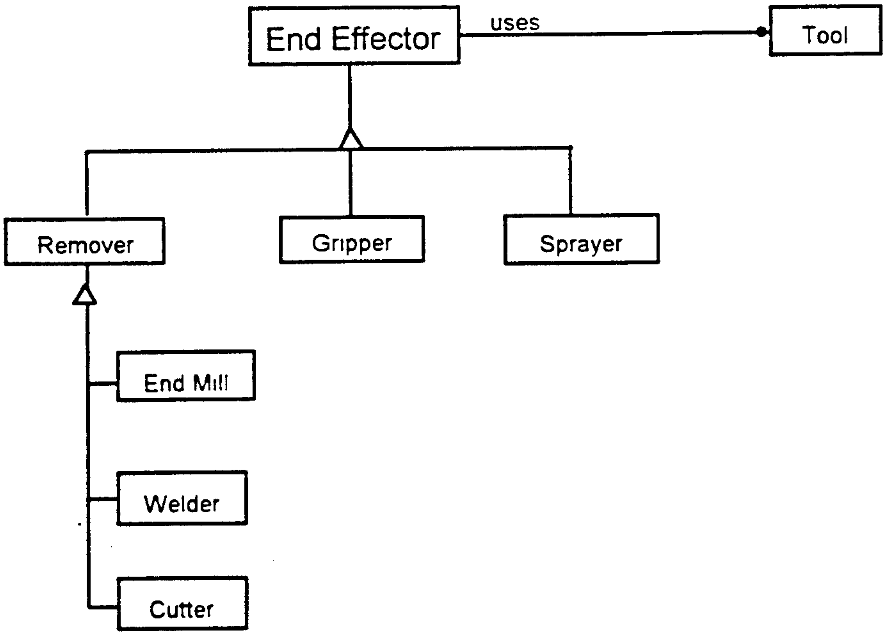

Spindle
angle
position vector
rotation direction
rotation speed
velocity vector
lock spindle?
move spindle (position, velocity)?
off spindle?
on spindle (speed, direction)?
orient spindle (angel)?
Table
position vector
velocity vector
table move?
has
has
Axis

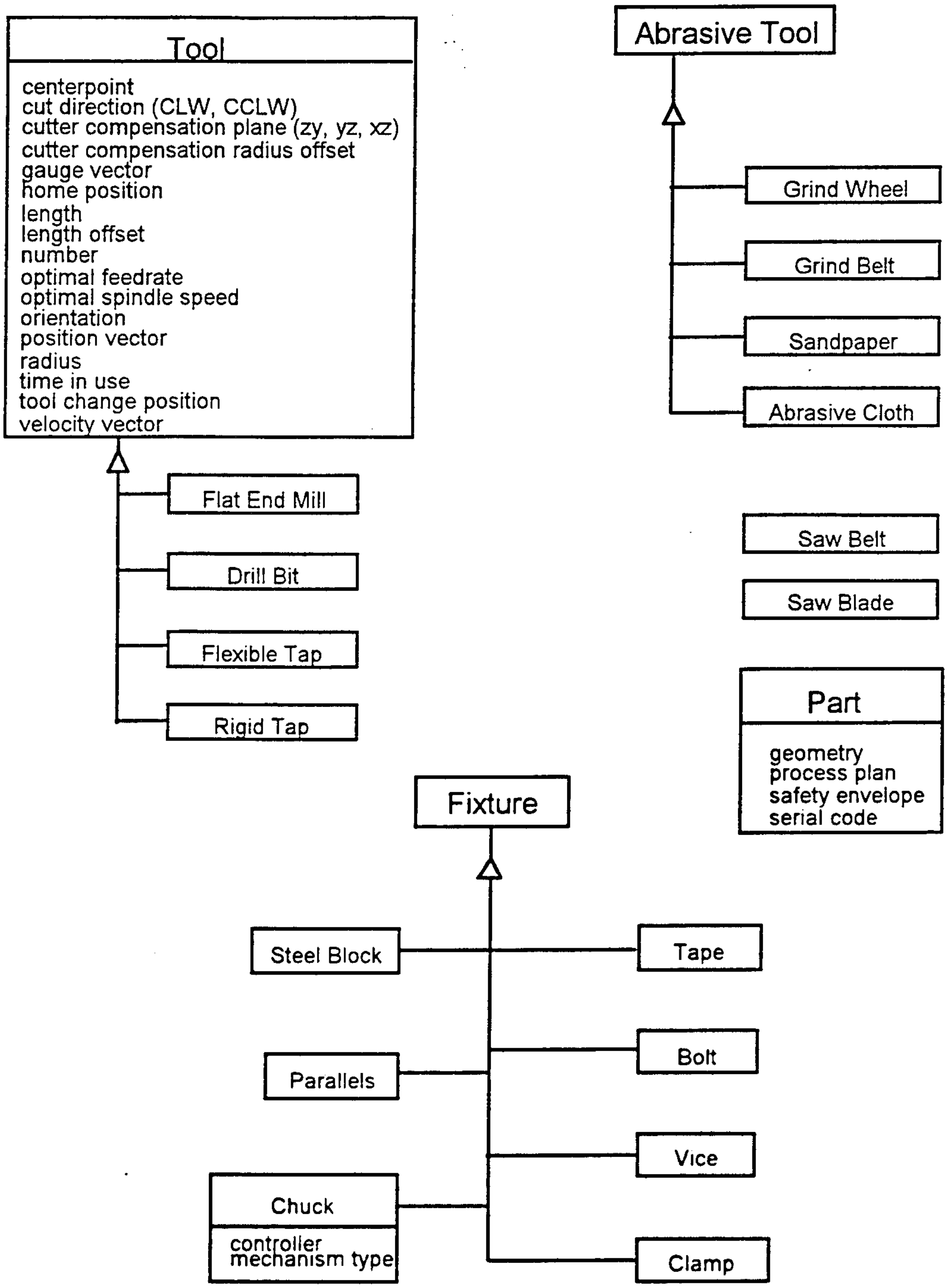


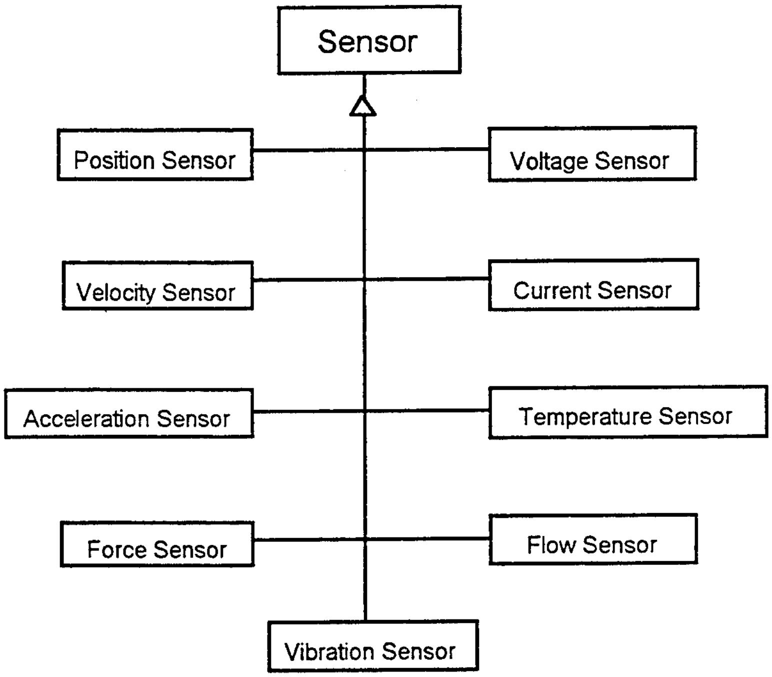


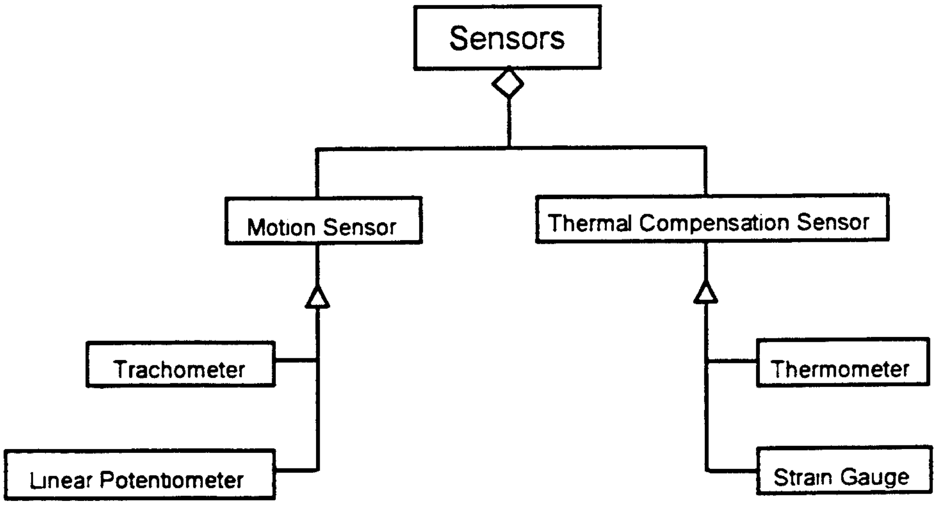
ω_d = angular velocity desired
ω_m = angular velocity measured
Θ_m = angle measured
ω_{dt} = angular velocity desired with manipulation

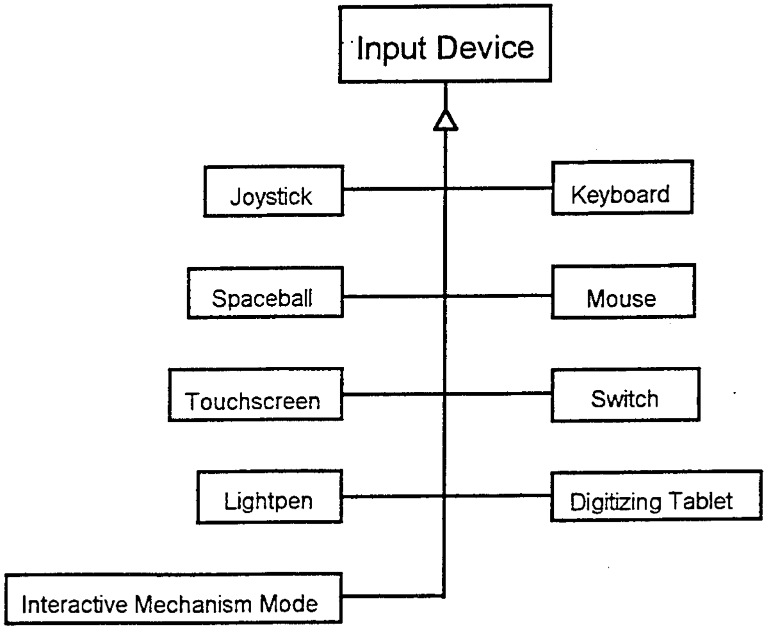

Robot
attach grasper
drop
follow path
grasp
move
uses
End Effector
Link
length
Joint
Face Plate
Controller
Drive
Motor

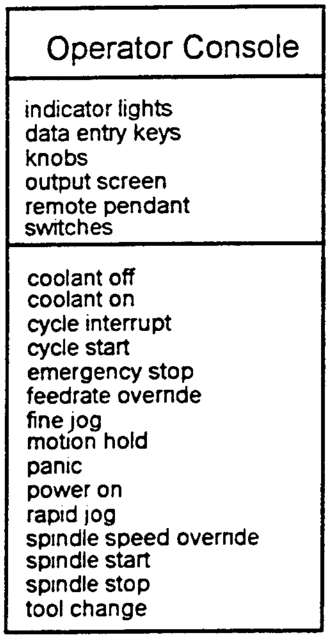

End Effector
uses
Tool
Remover
Gripper
Sprayer
End Mill
Welder
Cutter

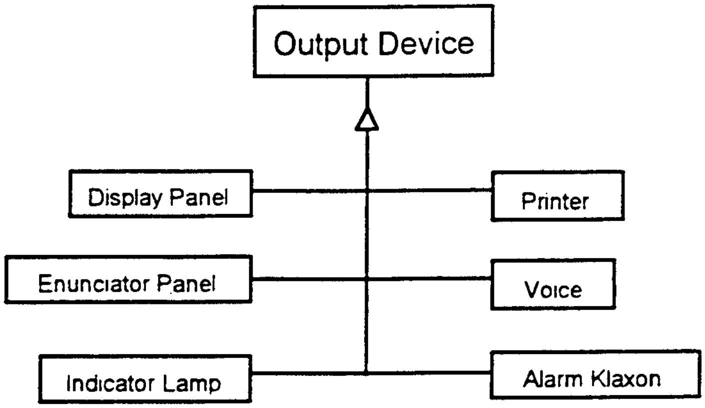

Tool
centerpoint
cut direction (CLW, CCLW)
cutter compensation plane (zy, yz, xz)
cutter compensation radius offset
gauge vector
home position
length
length offset
number
optimal feedrate
optimal spindle speed
orientation
position vector
radius
time in use
tool change position
velocity vector
Flat End Mill
Drill Bit
Flexible Tap
Rigid Tap
Abrasive Tool
Grind Wheel
Grind Belt
Sandpaper
Abrasive Cloth
Saw Belt
Saw Blade
Part
geometry
process plan
safety envelope
serial code
Fixture
Steel Block
Tape
Parallels
Bolt
Vice
Chuck
controller
mechanism type
Clamp

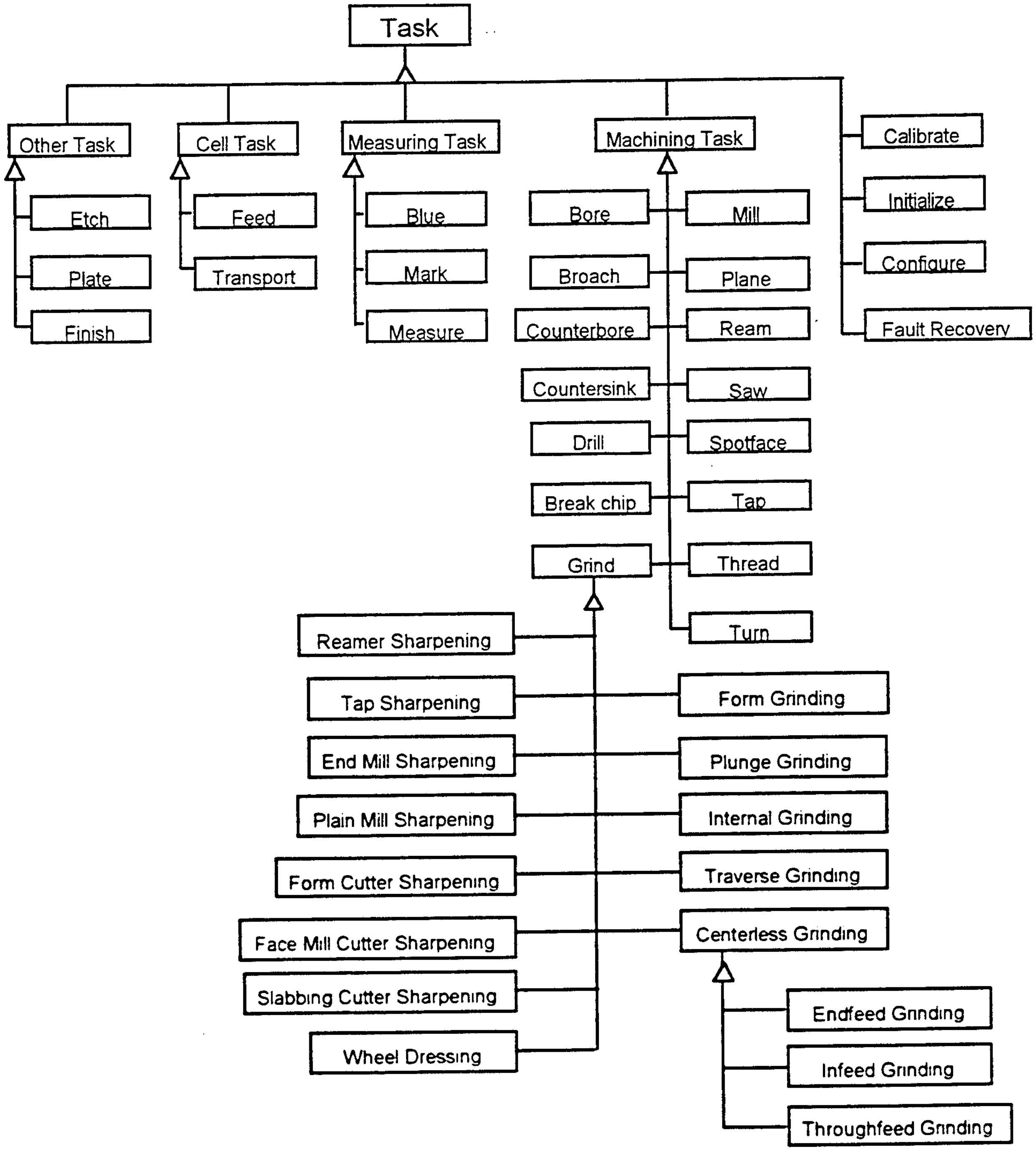

Sensor
Position Sensor
Voltage Sensor
Velocity Sensor
Current Sensor
Acceleration Sensor
Temperature Sensor
Force Sensor
Flow Sensor
Vibration Sensor


Sensors
Motion Sensor
Thermal Compensation Sensor
Trachometer
Linear Potentiometer
Thermometer
Strain Gauge

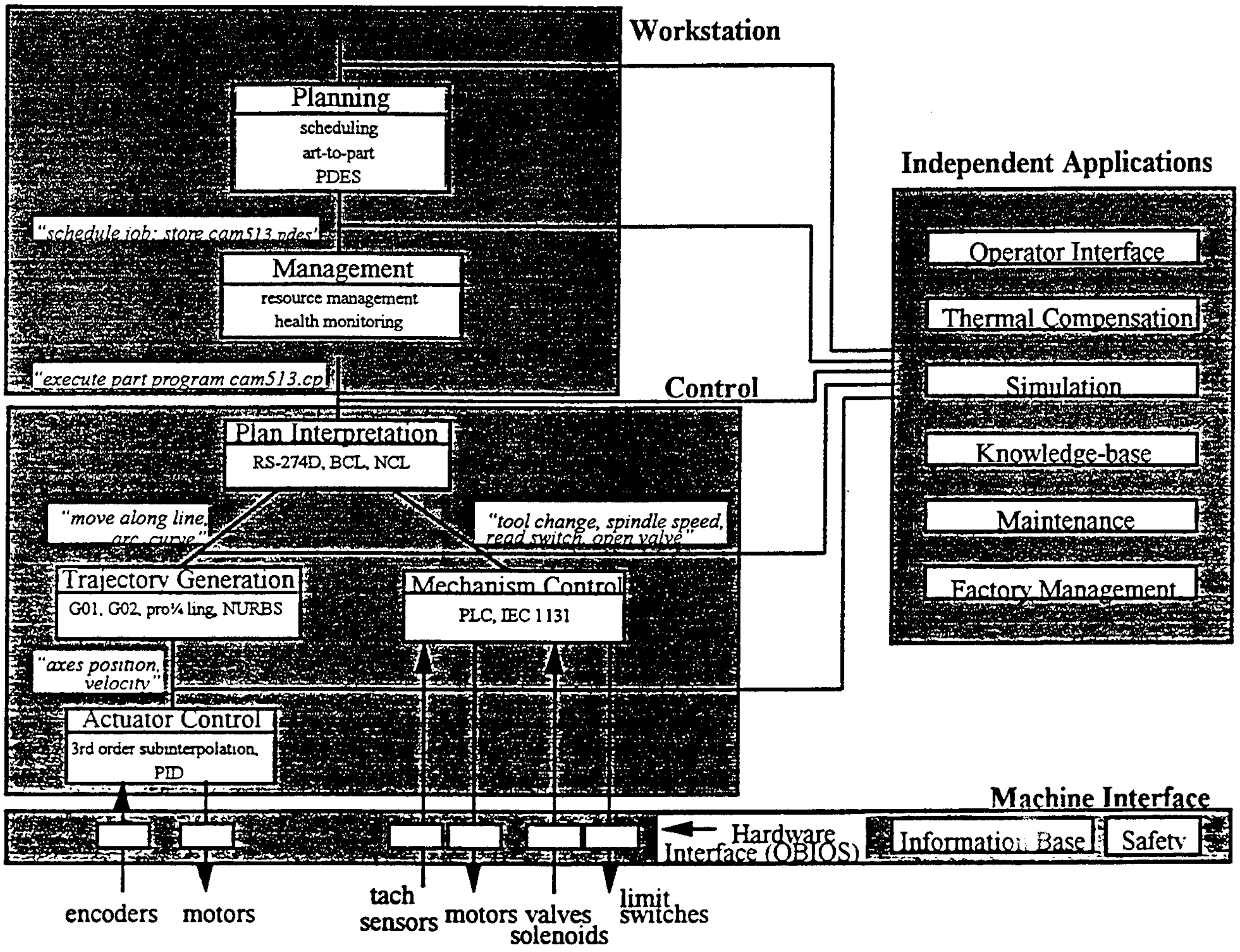

Input Device
Joystick
Keyboard
Spaceball
Mouse
Touchscreen
Switch
Lightpen
Digitizing Tablet
Interactive Mechanism Mode


Operator Console
indicator lights
data entry keys
knobs
output screen
remote pendant
switches
coolant off
coolant on
cycle interrupt
cycle start
emergency stop
feedrate override
fine jog
motion hold
panic
power on
rapid jog
spindle speed override
spindle start
spindle stop
tool change

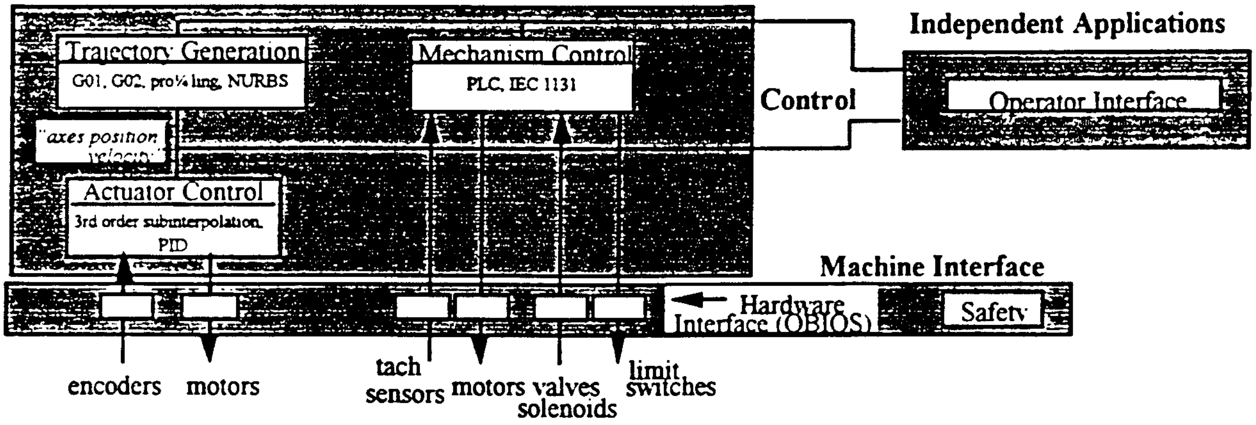

Output Device
Display Panel
Printer
Enunciator Panel
Voice
Indicator Lamp
Alarm Klaxon


Task
Other Task
Etch
Plate
Finish
Cell Task
Feed
Transport
Measuring Task
Blue
Mark
Measure
Machining Task
Bore
Mill
Broach
Plane
Counterbore
Ream
Countersink
Saw
Drill
Spotface
Break chip
Tap
Grind
Thread
Turn
Calibrate
Initialize
Configure
Fault Recovery
Reamer Sharpening
Tap Sharpening
Form Grinding
End Mill Sharpening
Plunge Grinding
Plain Mill Sharpening
Internal Grinding
Form Cutter Sharpening
Traverse Grinding
Face Mill Cutter Sharpening
Centerless Grinding
Slabbing Cutter Sharpening
Endfeed Grinding
Wheel Dressing
Infeed Grinding
Throughfeed Grinding

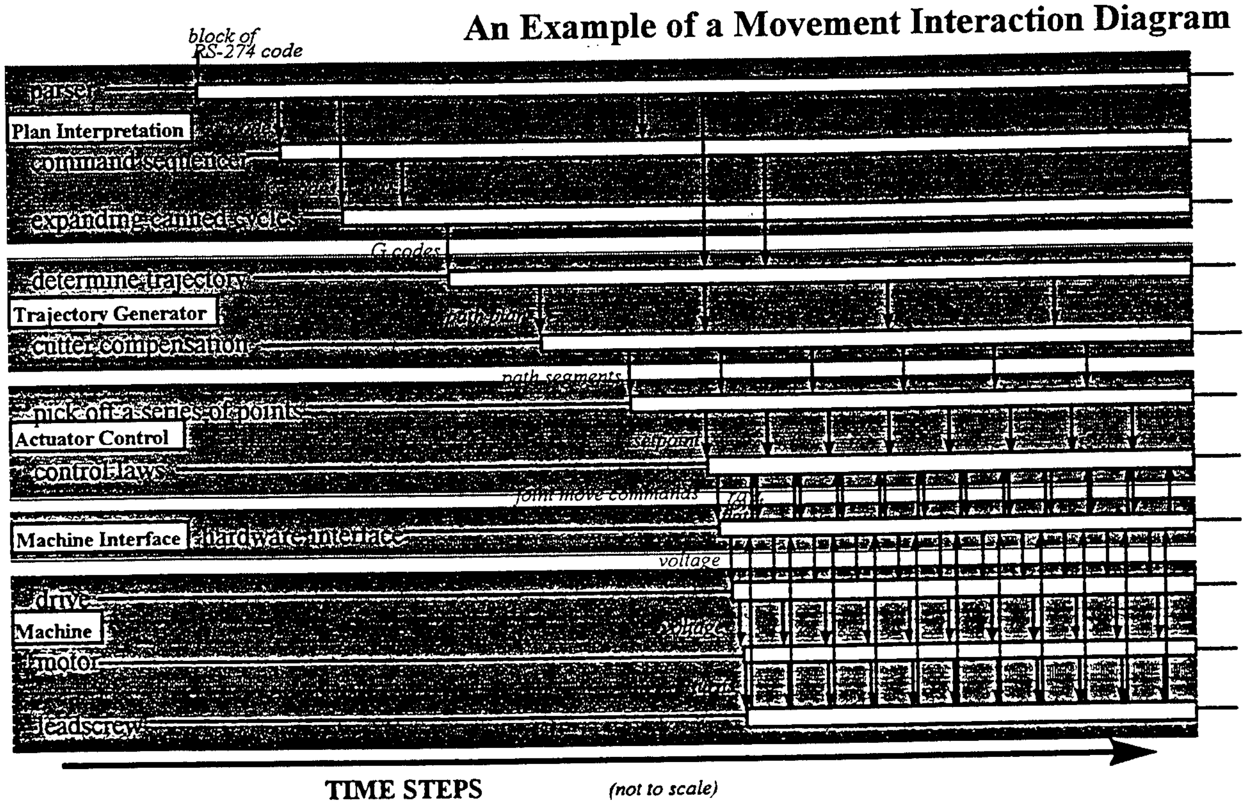
An Example of a Controller Topology



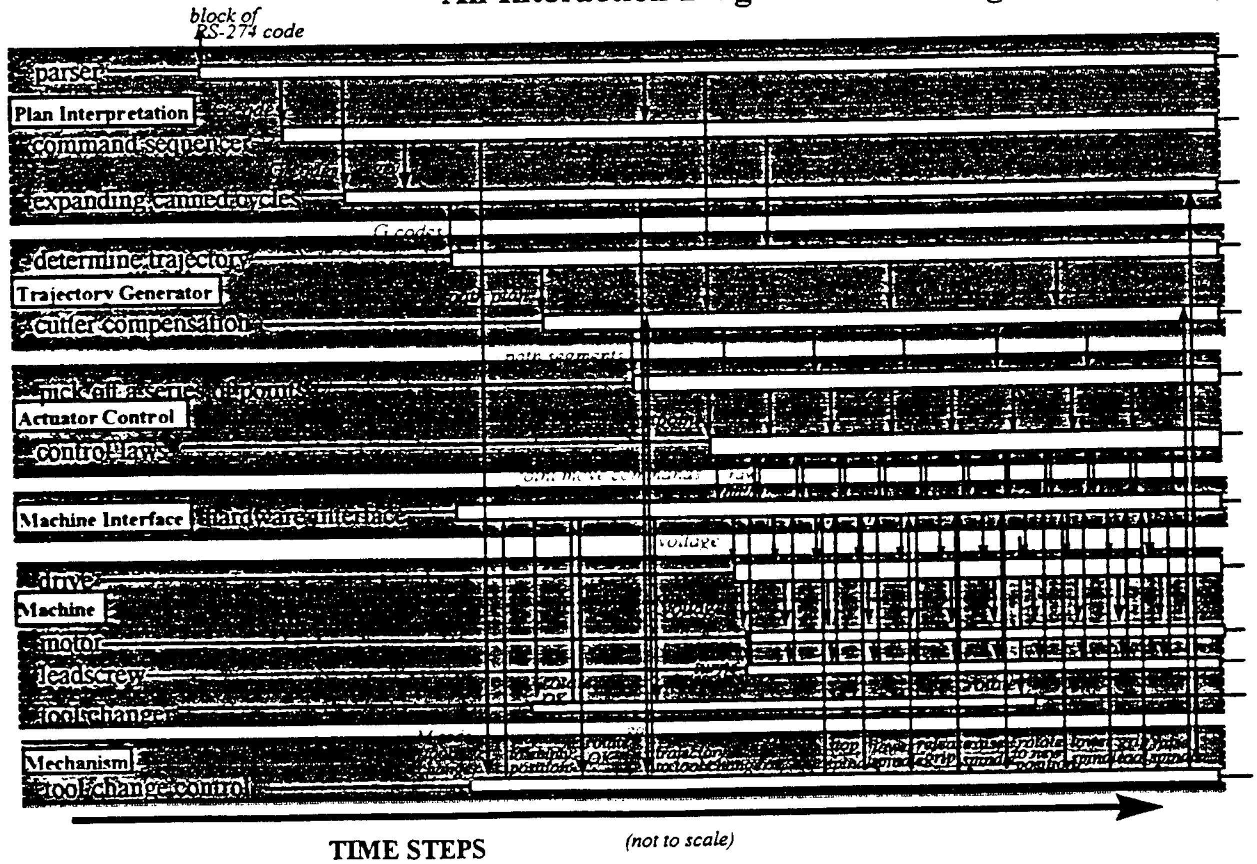
Topology for a Less-Sophisticated Controller



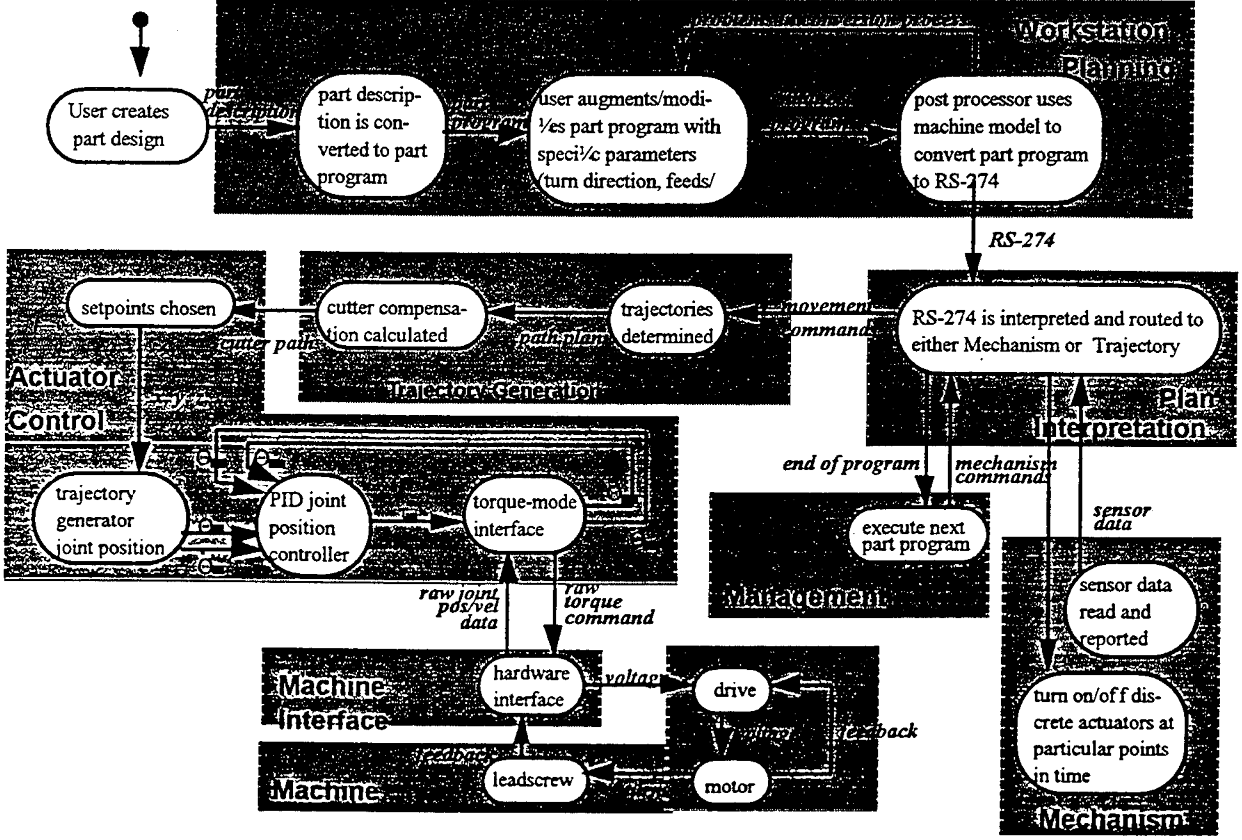
An Example of a Movement Interaction Diagram



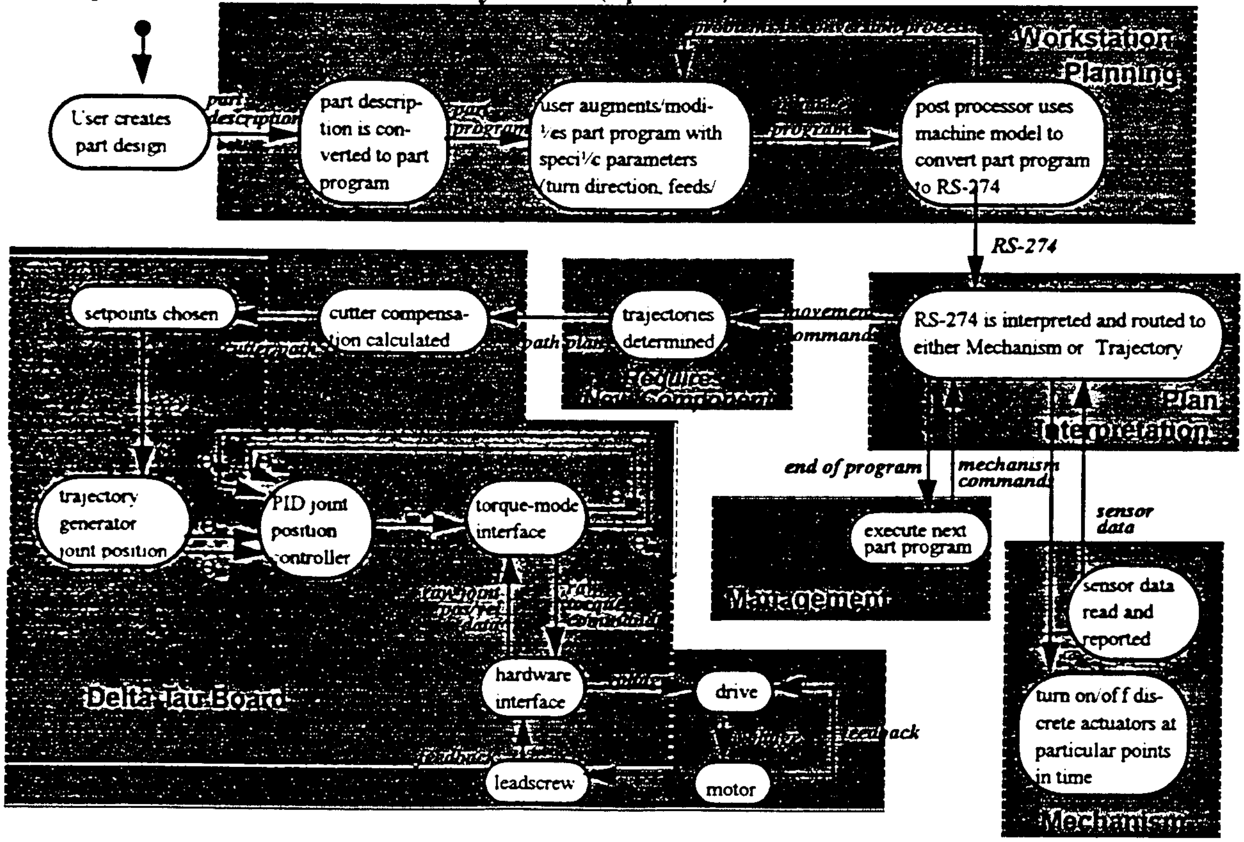
An Interaction Diagram Illustrating a Tool Change



Dynamic Model with Subsystems (option 1)


Workstation Planning
User creates part design
part description
part description is converted to part program
part program
user augments/modi½es part program with speci½c parameters (turn direction, feeds/
program
post processor uses machine model to convert part program to RS-274
RS-274
RS-274 is interpreted and routed to either Mechanism or Trajectory
Plan Interpretation
movement command
trajectories determined
path plan
cutter compensation calculated
cutter path
setpoints chosen
Trajectory Generation
Actuator Control
trajectory generator joint position
PID joint position controller
torque-mode interface
raw joint pos/vel data
raw torque command
end of program
mechanism command
execute next part program
Management
sensor data
sensor data read and reported
turn on/off discrete actuators at particular points in time
Mechanism
Machine Interface
hardware interface
voltage
drive
feedback
motor
leadscrew
Machine

Dynamic Model with Subsystems (option 2)


Workstation Planning
User creates part design
part description
part description is converted to part program
part program
user augments/modi½es part program with speci½c parameters (turn direction, feeds/
program
post processor uses machine model to convert part program to RS-274
RS-274
RS-274 is interpreted and routed to either Mechanism or Trajectory
Plan Interpretation
movement command
trajectories determined
path plan
Requires New Component
cutter compensation calculated
cutter path
setpoints chosen
trajectory generator joint position
PID joint position controller
torque-mode interface
raw joint pos/vel data
raw torque command
hardware interface
Delta Tau Board
drive
feedback
motor
leadscrew
end of program
mechanism command
execute next part program
Management
sensor data
sensor data read and reported
turn on/off discrete actuators at particular points in time
Mechanism

APPENDIX E — DOMAIN DICTIONARY

This dictionary contains the terminology and definitions from the domain of manufacturing in general and in the domain of the Next Generation Controller in particular. The dictionary is divided into several sections: a bibliography, a glossary of terms, notation, and a category index. The bibliography lists the information sources used. Sources include books, documents generated by NCMS members, commercial brochures, and notes from knowledge acquisition sessions. The glossary lists all terms (which are boldfaced) and their definitions alphabetically. Each definition includes a bibliographical annotation of the form [#] to indicate its source. Where a term has definitions from more than one source, each definition is annotated separately. The notation section includes tables of symbols, nomenclature, and units. The category index lists all terms divided into meaningful categories. Although a term may apply to more than one category, Most are listed only once in one category.

E.1 Bibliography

[1] Allen Bradley, PAL Programmer's Guide

[2] Asimov, Isaac & Karen A. Frenkel, Robots: Machines in Man's Image, Nightfall Inc. (1985). [TJ 211.A83]

[3] Automation Intelligence, IntelliPost Command Reference Manual for Turning Equipment (Mar 93).

[4] Hurco, "NGC Architecture and Modeling Project: Scenarios" (Jun 93).

[5] IBM, "Dictionary for the ADAGE of the DSSA Project" (31 Jul 92).

[6] IBM & Northrop, "Enterprise Integration Framework Next Generation Workstation/ Machine Controller Glossary" (24 Jan 91). This document contains many other software engineering and manufacturing terms that were not included here.

[7] Martin Marietta, "NGC SOSAS Volume I" (Mar 92 draft).

[8] New England Affiliated Technologies, product catalog.

[9] Walker, John R., Machining Fundamentals (1977). [TJ 1160.W25]

[10] Wizdom, sales brochures.

[11] Dictionary of the English Language: 2nd Edition unabridged, Random House (1987). [PE 1625.R3]

[12] Webster's New Universal Unabridged Dictionary: 2nd Edition, Simon & Schuster (1983).

[13] Webster's 3rd New International Dictionary: unabridged (1981). [PE 1625.W36]

[14] The Dictionary of Computing, Oxford University Press (1983). This reference was given in [6].

[15] Knowledge Acquisition Session with Mike Wright, Hurco (20-21 May 93).

[16] Knowledge Acquisition Session with Lester Godwin (6 Jun 93).

[17] Knowledge Acquisition Session with Whitey Simon & Dan Ma, GM, and Tom Carbone, Hughes (10-11 Jun 93).

[18] Knowledge Acquisition Session with Mike Nelson, Mayday (8 Jul 93).

[19] Knowledge Acquisition Session with Automation Intelligence (25-26 Aug 93).

[20] Knowledge Acquisition Session with Ken Stoddard, Trellis.

[21] Knowledge Acquisition Session with Wizdom.

[22] Computer Graphics: Principles and Practice: 2nd Edition.

E.2 Glossary of Terms

2.5 D
a 2D representation (topographical map) of a 3D object. 2.5D cannot be rotated, because the necessary 3D information is only from one perspective, so there is insufficient information to produce a 3D model. [16]

abbe error
a linear positioning error caused by a combination of an angular error in the ways, and an offset between the precision determining element (leadscrew, feedback device, etc.) and the actual point of interest. [8]

abort
to cease normal operations [5]

abrasive waterjet machine (AWJ)
a machine which performs material removal via a stream of water containing an abrasive solute at high pressure.

absolute
applies to measurements, in a standard, fixed reference, as opposed to moving reference; compare with relative [5]

absolute value
common math function; synonym for magnitude; standard notation |x|; for vectors, absolute value means length (magnitude) of the vector [5]

abstract data type

1 a non standard, user defined data type. [6]

2 a class of objects together with a set of operations which can be applied to them. The objects are entirely characterized through the external behavior of the operations. Neither the storage representation nor the implementation of the operations should be known to the user. [6]

AC
abbreviation for alternating current.

acceleration
rate of change of velocity; either scalar or vector, often with subscripts to denote the coordinate frame; time derivative of velocity; time integral of jerk; standard symbol a, A; standard units ft/s^2, g; primary units L/θ' [5]

accelerometer
an inertial device for measuring acceleration, usually in three orthogonal axes (lateral X, longitudinal Y, and vertical Z); accelerometers usually consists of a mass, spring, and damper, accelerometers are usually included in inertial sensors [5]

AC induction motor
a device that uses alternating current to produce mechanical energy. These motors do not have brushes and are simple to operate as well as inexpensive to produce. However, they are very difficult to control. DC brushless motors are used as AC servo motors. [4]

accuracy
1 measure to exactness, possibly expressed in percent, for example, position measured +10% ; compare with precision. [5]

2 extent of agreement of measured/reported value with true magnitude. [7]

3 extent to which a given value for a measurement agrees with the standard for that measurement. The degree of correctness of a quality or expression. [11]

activity
a repetitive, well-defined set of tasks that have common functional characteristics. [7]

actuator
a motor or transducer that converts electrical, hydraulic, or pneumatic energy to motion. [7]

adaptive control
a response to time-varying characteristics of the machinery, material, and process involving an adjustment of elements in the feedback control loop. [7]

ADL
abbr for Application Development Language.

AGV
abbr for Automated Ground Vehicle.

AI
abbr for Artificial Intelligence.

aiding
a process by which one or more sensors provide data to another sensor to produce results better than any single sensor. Aiding occurs at the physical device level, depending upon specific implementation of the device. Aiding is automatically controlled by software without input from an operator [5]

algorithm
1 a sequence of mathematical operations that precisely performs a specific task. [4]

2 a finite set of well defined rules for the solution of a problem in a finite number of steps. [6]

American National Standards Institute (ANSI)
an organization consisting of producers, consumers, and general interest groups, that establish the procedures by which accredited organizations create and maintain voluntary industry standards in the United States. [6]

AMICE
abbr for European Computer Integrated Manufacturing Architecture (in reverse)

analog input/output
continuous data input or output presented as a measurable quantity such as voltage and current. [7]

angle
synonym for angular position; standard units rad, deg; primary units 1 [5]

angular acceleration

rate of change of angular velocity, either scalar or vector, often with subscripts to denote the coordinate frame; time derivative of angular velocity; time integral of angular acceleration; standard symbol α; standard units rad/s'; primary units 1/θ' [5]

angular position
amount of rotation about an axis, either scalar or vector, often with subscripts to denote the coordinate frame; time integral of angular velocity; synonym for angle; standard symbol θ; standard units rad, deg; primary units 1 [5]

angular velocity
rate of change of rotation about an axis, either scalar or vector, often with subscripts to denote the coordinate frame; time derivative of angular position; time integral of angular acceleration; see tachometer; standard symbol ω; standard units rad/s, rpm; primary units 1/θ [5]

ANSI
abbr for American National Standards Institute.

aperiodic
a process that executes based on events rather than a fixed rate, it is not synchronized to other processes of interest; compare with periodic [5]

API
abbr for Application Programming Interface.

application program
a program that, when executed on a NGC computing platform, performs designated functions. [7]

application programming interface (API)
an interface that provides modules access to vendor components, shared components, and computing platform services. [7]

application-specific integrated circuit (ASIC)
a custom hardwired chip that performs a desired function.

application viewpoint
a DSSA view of a system.

approach vector
machine's normal approach of moving to the piece. Can be dynamically changed. [20]

architecture description language (ADL)
an ADL describes components' structures and specifies their protocols.

architecture viewpoint
a DSSA view of a system.

Arcnet
Arcnet is classified as a deterministic, high speed 2.5 Mb/sec, peer-to-peer token passing LAN. Arcnet is deterministic in that is allows the user to calculate the worst case latency of the network based on the number of nodes in use. Arcnet provides the software with a high-level interface that minimizes software overhead. Packets are written to the Arcnet controller along with the destination address and a single command is given to begin message transmission. The completed transmission and reception of a packet can be signaled with an interrupt.

Arcnet can be used by a number of network configurations. It can be used in a star configuration which uses active and passive hubs to grow the network in any shape required. Arcnet also supports a bus configuration which uses a high impedance version of the network transceiver on each node, and enables several nodes to be connected to a bus section of cabling by short cable spurs. Star and bus configurations can be mixed to provide the most effective layout for any installation.

Arcnet supports multiple media. Physically, the connection can be made with coaxial, fiber optic, and twisted pair cabling including 485. typically, the network is tied together with inexpensive RG62 coaxial cable, making it highly immune to electrical noise. A network can be up to 4 miles long. the maximum node-to-node or node-to-hub distance is 2000 feet.

Arcnet was designed to allow nodes to be added or removed during live everyday use while the network is running. There is no need to shut down. The protocol will automatically reconfigure itself within 60 milliseconds of a node being removed or failing. [10]

arctangent
common math function; standard notation arctan x, atan x, $\tan^{-1}x$, $\tan^{-1}(y,x)$; $\tan^{-1}(y,x)$ means two-argument arctangent of y/x [5]

area clear milling
method of surface finishing in which the tool follows a simple back-and-forth pattern across the part surface. [15]

arm
an interconnected set of links and articulated joints between an end effector and its support structure. [7]

artificial intelligence (AI)
the ability of a process to perform functions normally associated with human intelligence, such as reasoning, learning, and self-improvement. A technology or field of study that encompasses all attempts to emulate or reproduce human neurological functions, such as cognition, perception and action, through symbolic means. [14]

ASCII
abbr for American Standard Code for Information Interchange.

ASIC
abbr for Application-Specific Integrated Circuit

assembly
a group of parts and/or subassemblies that are put together. An assembly may be an end item or a component of a higher level assembly. [6]

asset
any expendable resource within the manufacturing workstation. Asset does not include the machines under control. [7]

async
abbr for Asynchronous.

asynchronous

tasks or events that occur at non-fixed time intervals and are large independent of one another; compare with synchronous [7]

attribute

1 data element that represents a characteristic used to describe an object. [7]

2 a quality or characteristic element of an entity, having a name and a value; an item of information about an entity; properties which describe data objects. [6]

autonomous control

independent control over the sequence, options, and schedule of tasks. [7]

availability

the measure of time that a manufacturing workstation is free to carry out its designated functions. [7]

averaging filter

a filter for combining multiple data sources, usually of the same type, by adding with weighted averages; a simple average of the data sources; compare with complementary filter, Kalman filter.

Averaging filters are often designed as:

$$y = \sum_{1}^{n} \alpha i x i$$

where α_i are usually computed at run time such that [5]

$$\sum_{1}^{n} \alpha_i = 1$$

AWJ

abbr for Abrasive WaterJet machine.

axis

1 one direction in an orthogonal reference frame [5]

2 Used to locate or move the part. An axis is mathematically described as one of the reference lines of a coordinate system, such as Cartesian coordinates (x, y, and z). There can be x, y, z, a, b, c, u, v, and w axes. [4]

3 Any line used as a fixed reference in conjunction with one or more other references for determining the position of a point or of a series of points. [11]

4 the center line, real or imaginary, passing through an object about which it could rotate. a point of reference. [9]

backlash

the amount of free play between a leadscrew and nut, or worm and worm gear. It is determined by measuring the range of angular movement of the driven shaft which results in no motion. [8]

band-pass filter (BPF)

a filter that allows frequencies between two cutoff frequencies to pass while attenuating frequencies outside the cutoff frequencies; a band-pass filter can be constructed as the composition of a low-pass filter and a high-pass filter [5]

band saw

a type of saw machine in the form of a continuous steel belt running over pulleys. [13]

batch execution
a condition that occurs when the controller is in the normal production mode. Interaction may be required initially, but file-based operations are generally thought of as being "automatic". This operation is characterized by the execution of internally represented plans, and facilitates part/process production while performing planning operations. A file typically consists of pre-generated batched control plan or process plan information that is executed automatically from start to finish. [7]

BCD
abbr for Binary Coded Decimal.

BCL
abbr for Binary Cutter Location.

binary cutter location (BCL)
a numeric control programming and interpretation scheme that is based on a generic machine reference. This eliminates the need for post processing the CNC program for a specific machine or type of machine. CNCs designed for BCL interpretation can execute the BCL instructions directly. [6]

bias
an offset applied to a measurement for error correction; standard engineering term synonym for offset [5]

bed
machine's cutting surface, also referred to as the "table"; the part is attached to this surface with a clamping device. [4]

bed mill
a.k.a. mill or machining center. A type of machine that is more powerful than a knee mill and can be used to cut large parts in large quantities. The bed can travel horizontally or vertically on a bed mill, and the cutting area is usually enclosed. [4]

block
a group of instructions from a part program to control machine actions. A block usually describes a discrete movement or action such as cutting a bolt circle. [4]

block cycle time
time it takes to execute one line of executable RS-274 code [15]

blue
a finishing task.

BOC
abbr for Buick, Oldsmobile, Cadillac; a now defunct GM organization.

bolt
a type of fixture; any of several types of strong fastening rods, pins, or screws, usually threaded to receive a nut. [11]

bore
a machining task; to make a hole in a material by turning a tool.

BPF

abbr for Band-Pass Filter [5]

break chip
temporarily stop z-motion of the tool to break off the chip. [15]

broach
1 v. a machining task; to cut with a broach tool.

2 n. a cutting tool for removing material from metal or plastic to shape an outside surface or hole that has been previously formed, consisting of a bar of suitable length provided on its surface with a series of cutting edges or teeth that increase in size from the entering or starting end. [13]

brushless motor
an "inside-out" DC motor, with a permanent magnet rotor, and electrical coils in the stator. Commutation of current in the windings is typically achieved via external switching transistors, and Hall effect detectors. This avoids the limited life of brushes and their radiated EMI. [8]

CAD
abbr for Computer-Aided Design.

calibration
1 an interactive condition that occurs when the controller is in the maintenance mode; starts processes unique to a specific machine or process that enables the operator to fine tune the controller. This mode shall be used to adjust control law gains, modify axis soft limits, input positional offsets, verify setpoint accuracy, adjust overshoot and balance process parameters, or adjust other known model inaccuracies though the use of error correction tables and/or other methods. It is assumed that this mode will rely heavily on information provided by the operator interactively. It is also assumed that manual operation of the machine may be required in this mode. [7]

2 calibrate sequence is used to determine a reference point or zero location on a part. [4]

CAM
abbr for Computer-Aided Manufacturing.

CANBus
1 an I/O bus used by numeric controllers. [4]

2 CAN could be classified as a Small Area Network (SAN). It runs on an inexpensive twisted pair bus and provides an interface to I/O and other peripherals, peer-to-peer networking of controllers, deterministic prioritized message passing, small efficient packet sizes, robust error detection, and a reduced wiring solution. CAN is an industry standard protocol supported by multiple vendors. Packets are written to the bus controller along with priority and the destination address. A single command is given to begin message transmission. The completed transmission and reception of a packet can be signaled with an interrupt. The data transmission rate can be up to 1mb/sec for networks less than 130 feet in length. Lower data rates can support longer networks. Up to 32 nodes can be attached to a given network.

CAN is a deterministic network with prioritized message passing. Worst case message transmission latency in a CAN network can be calculated based on the number of nodes in use and the priority of the message. The use of prioritized messages guarantees that critical messages are transmitted ahead of lower priority ones. [10]

canned cycle
common machining operations, such as drilling, tapping, pecking, and boring and reaming, that are pre-programming into a CNC. These cycles allow the operator to specify a type of operation and define only the necessary variable information. Then the system makes additional calculations to define the operation fully. [16]

cache
1 an architected area of computer main storage for system and/or application use. [6]

2 the virtual storage within which objects are presented to methods. [6]

3 an architected area made up of control structures and pointers that point to a variable number of Logical Views. The Logical Views may contain internal and/or external references to the same/other Logical views which are resolved as relocatable addresses when loaded into computer storage. The entire cache structure may be treated as a unique entity and may be modified, saved and restored as a means of maintaining the context of a user or unit of work between user logons or automatic activation of enterprise activities. [6]

CALS
abbr. for Computer-aided Acquisition and Logistics Support. [6]

canonical
conforming to a standard form of semantics and syntax. [6]

CCLW
abbr for Counterclockwise.

cell
1 a manufacturing unit that has the capacity to manufacture a family of products that are a subset of a product type within a fixed domain of operations. It may contain several manufacturing workstations. [7]

2 a manufacturing unit which has the capacity to manufacture a family of products which are a subset of a product type (e.g. sheet metal, composites, circuit card, etc.) and which share the same manufacturing operations with minor changes. Therefore, a cell is a manufacturing unit which has the ability to produce a family of parts within a fixed domain of operations. [6]

center
a manufacturing unit consisting of two or more cells and the materials transport and storage buffers that interconnect them. [7]

centerless grinding
a type of grinding.

CEP
abbr for Circular Error Probability [5]

chatter
the quick repeated sounds resulting from a machine tool vibrating against the workpiece while the machine is in operation. [7]

checkpointing
a recovery mechanism that take a snapshot of everything pertinent to system state, producing a virtual memory page, and forces this page periodically to disk. [7]

chip
a small residue piece of material remaining from cutting, shaping or finishing a part with a machine tool. [7]

chuck
a type of fixture; a device for centering and clamping work in a lathe or other machine tool. [11]

CIM
abbr for Computer Integrated Manufacturing

circular error probability (CEP)
a probability that a percentage of two-dimension measurements will lie within a circle of given radius, with the circle centered at truth or mean of the measurements; compare with radial error probability, spherical error probability

CEP specifies test cases for measurement errors of sensors of two dimensions, such as X and Y. For example, a velocity error of 1 ft/s (50% CEP) means that for any given measurement of velocity

$$\vec{V}_i = \{V_E, V_N\}$$

then

$$p(|\vec{V}_i - \vec{V}_\mu| \leq 1) = 0.5$$

where $\vec{V}_\mu$ is the average of all $\vec{V}_i$ or $\vec{V}$ truth, depending upon context. [5]

clamp
a type of fixture; an appliance with opposite sides or parts which may be adjusted or brought closer together to hold or compress something. [11]

class
1 the object-oriented paradigm's abstract data typing mechanism that groups objects with commonalty of attributes and services. [7]

2 a type of data object. There are primitive classes or types such as fixed, character, float, and bit. There are also more complex types that are constructed from the primitive types. [6]

3 a group of objects which carry the same instance variables, the same methods, and respond to the same messages. [6]

class hierarchy
defines superclass and subclass relationships and supports the is-a relationship. [6]

climb milling
milling in which the tool motion is against the tool rotation, such that feedrate + rotation = actual speed of cutting tool; compare with conventional milling [15]

clockwise (CLW)
positive rotational direction.

close dancing
two robots working on the same thing independently without interfering with each other. Matching parts takes 12 degrees of freedom (position, velocity, etc.), 6 degrees

of freedom on each of 2 parts (force, torque, etc.). [19]

closed loop
control achieved by measuring the degree to which actual system response (feedback) conforms to desired system response and using the difference to drive the system in conformance. Also called feedback control. [4]

CLW
abbr for Clockwise.

CMM
abbr for Coordinate Measuring Machine.

CNC
abbr for Computer Numerical Control.

coefficient of friction
the ratio of the force required to move a given load to the magnitude of that load. [8]

command
a signal that actuates a device. [7]

commercial off-the-shelf package (COTS)
a software system that is commercially-available and not specifically targeted toward NGC system needs. [7]

commit
the act of writing buffered data to permanent storage upon completion of a transaction. Commits are usually carried out periodically for groups of data. [7]

communications viewpoint
a DSSA view of a system.

complementary filter
a filter in which the complement of the filter is desired, for example {EMBED Equation |}, giving the effect of a high-pass filter implementing a low-pass filter; a filter for combining multiple data sources, usually of different types, by adding filtered values, where the sum of the filters in the frequency domain is unity; a Kalman filter with fixed gains; compare with averaging filter, Kalman filter

Complementary filters are often designed in the frequency domain as

$$Y(s) = \sum_{1}^{n} F_i(s) X_i(s)$$

where $F_i(s)$ are filters determined at build time such that

$$\sum_{1}^{n} F_i(s) = 1$$

For example, with velocity sources V_x and V_y, one might choose

$$V_{sys}(t) = f_{LPF}(V_x(t) + f_{HPF}(V_y(t)))$$

where the cutoff frequency of the LFP is equal to that of the HPF. [5]

component viewpoint
a DSSA view of a system.

computer-aided design (CAD)
the use of computers in interactive engineering drawing and storage of design. Programs complete the layout, geometric transformations, projections, rotations,

magnifications, and interval (cross-section) view of part and its relation with other parts. [6]

computer-aided manufacturing (CAM)

1 the use of computers to program, direct, and control production equipment in the fabrication, assembly, and distribution of manufactured items. [6]

2 the application of a computer in a manufacturing environment to bridge various systems and connect them into a coherent, integrated whole. For example, budgets, CAD/CAM, process controls, group technology systems, MRPII, financial reporting systems, etc., would all share data in an integrated environment. [6]

3 the application of information systems technology to increase the productivity and responsiveness of the organization. [6]

4 a business philosophy aimed at reducing the development time of industrial products, increase its quality and create an environment where the production of goods will be based on market requirements. Use of (C)omputer technology to enable us to work faster and more precisely in dealing with complex problems. (I)ntegration means that we make our enterprise work as an undivided unit. Transferring information about our products, plans, methods and decisions effortlessly and without risk of introducing errors. (M)anufacturing refers to all industrial activities necessary for design, manufacture, installation and maintenance of products. [6]

5 a computer-based technology that helps define and create a program to drive a CNC machine. CAM generates tool path. [4]

computer cycle

in a periodic, cyclical computer system, the most basic, fastest timing loop [5]

computer numerical control (CNC)

1 controlling machine tools using microcomputers attached to the machines in some manner. The computer controls the machining sequence of a machine to provide faster, more accurate production and the capability to make parts that are essentially impossible to create by hand control. Within the machine industry, the ìcontrolî is usually the enclosure and the computer boards within it that contain the machine controlling software and handle communications between the machine and the operator console. [4]

2 a type of machine controller designation, in which the computer stores and recalls programs. [16]

computing platform

the processing infrastructure of the NGC manufacturing workstation consisting of processor(s), related hardware, system, and system utility software, and communication mechanisms. [7]

concurrency

proper control of concurrent updates made on shared data so that inconsistent data does not result. The results produced by multiple concurrent transactions are equivalent to results that would be produced by serial transactions. [7]

configuration

the arrangement of a manufacturing workstation as defined by the nature, number, interaction, and main characteristics of its functions. The features, customer options, software modules and engineering specifications that collectively define the functionality of the manufacturing workstation. [7]

configuration management
establishing a process for managing and controlling changes to functional and physical characteristics during the life cycle; this provides tracability and status of change activities to previous configurations. [7]

conformance
action or behavior in correspondence with current customs, rules, or styles. In particular, behavior in correspondence with SOSAS rules requirements, and styles. [7]

conformance class
a set containing members having common behavior in correspondence with a defined set of (SOSAS) rules, requirements, and styles. [7]

consumable
material that is physically used or transformed by a production process in the completion of that process. It includes items such as coolants and lubricants. [7]

context
the conditions that bound and relate discrete events temporarily, spatially, or semiannually. [7]

continuous path control
motion control of a device that is not planned to stop at intermediate points along a path. [7]

continuous time
time which can have any point expressed as a real quantity, without regard for any specific interval or processing rate; compare with discrete time [5]

continuous-time equation
a mathematical relationship to describe a function of time, expressed in terms of continuous time; compare with difference equation, differential equation, discrete-time equation, Laplace transform, Z transform; see first order filter, second-order filter, unit functions for examples.

Example standard notation for continuous-time equations is: [5]

$$y(t) = \frac{-5}{16}e^{-t} + \frac{5}{16}e^{3t} - \frac{1}{4}te^{-t}$$

control law
the mathematical definition of a system used to control or to change the dynamic response of a system. [5]

control plan (CP)
a plan that is executable by a specific machine within the workstation. [7]

controls standardized application (CSA)
provides both continuous and discrete control of an individual machine. The SA receives machine specific task commands from the task execution SA, and produces effector commands to support all modes of operation. Process control commands are also decomposed to a sequential set of discrete effector commands and sent to SESA. Coordination between machine

specific motion and process commands is provided by this SA. [7]

conventional machine tool
a machine tool that is not controlled by computers. [7]

conventional milling
milling in which the tool motion is with the tool rotation; compare with climb milling [15]

conversational part programming
method implemented by software within the control that allows part programming in a question/answer format on the operator console. This method of part programming uses multiple choice and fill-in-the-blank questions, as well as clearly worded operator prompts. It is the method not the software. [4]

coolant
a cutting fluid that keeps the tool and part cool to prevent reduction in hardness and resistance to abrasion. It therefore helps prevent distortion of the work. [4]

coolant mechanism
delivers coolant in either a flood, tap, or mist fashion to prevent the tool or part from overheating.

coordinate
any of the magnitudes which serve to define the position of a point in terms of a fixed reference frame such as coordinate axis. [17]

coordinated joint
a joint that is coordinated to begin and end at prescribed, controlled, programmed points with special tolerances on the (interpolated) path in between them. [7]

coordinate measuring machine (CMM)
a machine that uses a probing tool to determine coordinate points on a part.

correlation
the degree to which two or more attributes or measurements on the same group of elements show a tendency to vary together. [11]

cosine
common math function; standard notation cos x, cx [5]

COTS
abbr for Commercial Off-The-Shelf package

counterbore
a machining task; to form a flat-bottomed enlargement in the mouth of a cylindrical bore [13]

counterclockwise (CCLW)
negative rotational direction.

countersink
a machining task; to cut a funnel shaped enlargement at the outer end of a drill hole, usually for the reception of a screw, bolt, or rivet head. [13]

CP
abbr for Control Plan

CPC
abbr for Chevrolet, Pontiac, CM of Canada.

CPU
abbr for Central Processing Unit

CSA
abbr for Controls Standardized Application.

cutoff frequency
the frequency at which the gain of a filter is at an edge of a band, usually taken to be when gain is 0.5, or Å-3.01 dB; the frequency at which the output of a filter is half the power of the input; see band-pass filter, high-pass filter, low-pass filter, standard symbol ω_c; standard units rad/s, Hz; primary units 1/θ} [5]

cutter compensation
tool path is programmed for a tool of zero diameter. Cutter compensation uses algorithms to determine the offset of the tool path to compensate for the actual diameter of the tool. [4]

cutter compensation plane
plane in which cutter compensation is measured. Either the xy-, yz-, or xz-plane.

cutter compensation radius offset
attribute of a cutting tool.

damped frequency
the frequency of oscillation of an underdamped second-order filter

$$\omega = \omega_n \sqrt{1-\zeta^2}$$

where ω_n is the natural frequency and ζ is the damping ratio; see second-order filter; standard symbol ω; standard units rad/s, Hz; primary units 1/θ [5]

damping ratio
see second-order filter; standard symbol ζ; standard units 1; primary units 1 [5]

data
a representation of facts, concepts, or instructions in a formalized manner suitable for communications, interpretation, or processing. [6]

database
a structured repository for information, with well defined methods for storing and accessing data. Instances of objects and relationships as defined by a schema. [7]

database management system (DBMS)
a software system (often COTS) for managing one or more, centralized or distributed data bases. Data management facilities typically provide mechanisms for data entry, update, retrieval, and storage. [7]

data bus
communication path between one or more CPUs and related hardware. [7]

data dictionary
a centralized repository of information about data such as its meaning, relationships to

other data, origin, usage, format, and business rules governing the data. It assists management, database administrators, system analysts, and application programmers in planning, controlling, and evaluating the collection, storage and use of data. [6]

data starvation
finish executing a block of code before next block has been analyzed. Eliminated with sufficient look-ahead. [15]

data structure
a formal representation of information for communication and processing purposes without regard to actual storage configuration. [7]

DBMS
abbr for Database Management System

DC
abbr for Direct Current.

DC motor
such a motor uses direct current. There are two types of DC motors: brushed and brushless. The brushed DC motors use brushes for commutation. Brushless motors are externally commutated. [4]

deadband
the deadband range is adjustable and lets the programmer select the appropriate error range both above and below setpoints outside of which the output will not change. With the deadband, the programmer can closely match the process variable to the setpoint without changing the output. (see PID control loop) [10]

dedicated channel
point-to-point wiring. [7]

delta
the difference between the desired value and the measured value

derivative
common math function; rate of change, usually with respect to time; standard notation $\dot{x}$, x', $x^{(1)}$, dx/dt, Dx, $sx+x(0^{+})$ [5]

design viewpoint
a DSSA view of a system.

desired
what must be achieved in order to match a plan; synonym for reference [5]

determinant
common math function; standard notation det[A], |A| [5]

deviation
difference from desired [5]

device driver
a software component that provides software platform access to a hardware platform. [7]

diagnostic
an interactive condition that occurs when the controller is in the maintenance mode. Off-line diagnostics are algorithms that extensively check the entire NGC system.

They are intended to extensively verify all system hardware. A complete system diagnostics verification requires some or all of the NGC operations to be suspended. This will normally require a mode change at the standardized application level and perhaps at the system level. Due to the various possible configuration of hardware and software on a NGC, the system integrator must tailor this package for the delivered environment. Health and status monitoring includes data consistency checking, and may be enabled any time during system operation for any combination of standardized applications in the system. [7]

difference equation
a mathematical relationship to model a discrete function, expressed in terms of other values in the sequence; compare with continuous-time equation, differential equation, discrete-time equation, Laplace transform, Z transform; A difference equation usually models periodic process in terms of past values; see first-order filter, integrator, second-order filter for examples

Example standard notations for difference equation are

$$y[n] = y[n-1] + y[n-2]$$
$$y_n = y_{n-1} + y_{n-2}$$
$$y_{n+2} = y_{n+1} + y_n$$

where y is a function of time, n denotes the current cycle, n-1 denotes the last cycle, and in general n-1 denotes the ith prior cycle. Initial conditions, like y[0]=0, are usually given or implied. Difference equations are usually derived from differential equations. [5]

differential equation
a mathematical relationship to model a continuous function, expressed in terms of derivatives; compare with continuous-time equation, difference equation, discrete-time equation, Laplace transform, Z transform. A differential equation usually models continuous-time phenomenon in terms of time derivatives; see first-order filter, integrator, second-order filter for examples.

Example standard notation for differential equations are

$$y'' - 2y' - 3y = e^{-t}$$
$$\ddot{y} - 2y - 3y = e^{-t}$$
$$y^{(2)} - 2y^{(1)} - 3y = e^{-t}$$
$$(d^2y/dt^2) - 2(dy/dt) - 3y = e^{-t}$$
$$(D^2 - 2D - 3)y = e^{-t}$$
$$y - 2\_ydt - 3\_\_ydt^2 = \_\_e^{-t}\,dt^2$$
$$s^2y - sy(0^+) - y(0^+) - 2sy + 2y(0^+) - 3y = e^{-t}$$

where y is a function of time t. Initial conditions, like y(0)=0 or y(0) =1 ft/s are usually given or implied.

Differential equations are commonly used by systems engineers to model systems. The systems engineer usually converts differential equations to difference equations for specification and implementation in software. [5]

digital numerical control (DNC)
a type of machine controller designation, in which programs downloaded from elsewhere. [16]

dimension
1 a standard quantity, such as ft or mi; synonym for units [5]

2 a degree of freedom within a vector space. [17]

dimensional measurement interface specification (DMIS)
standard language to program coordinate measuring machines. [7]

dimensionless
no units, such as ratios; synonym for unitless [5]

direct numerical control (DNC)
a host computer controls a group of machine tools by downloading part programs to them. Operators do not program parts at the machine. They start machines and fixture parts. This control structure is common in large shops that mass produce parts. [4]

discrete I/O
input or output data represented by on or off states. [7]

discrete time
time divided into quantized intervals; time is usually divided into equal intervals to create a periodic process; compare with continuous time [5]

discrete-time equation
a mathematical relationship to describe a function of time, expressed in terms of discrete time; compare with continuous-time equation, difference equation, differential equation, Laplace transform, Z transform; see first-order filter, unit functions for examples.

Example standard notation for discrete-time equations is
[5]

$$y[n] = \frac{-5}{16}e^{-n} + \frac{5}{16}e^{3n} - \frac{1}{4}ne^{-n}$$

distance
synonym for range; method of measurement dependent on use [5]

distributed control
distributed control means that a number of PLCs on a network can control I/O in local or remote places. They communicate over a control network like Arcnet or CANbus. In distributed control, the task of implementing the control system is divided into a number of small modular and easily managed parts. Each part of the control is performed by individual stations networked together to communicate needed information. The idea is to minimize the communication between stations to allow each of them to run autonomously. This approach works well in large complex systems, applications with many similar repeated elements, and when machine upgrade and configuration flexibility is essential. [10]

distributed database
a fully distributed database is one that has multiple data bases stored on multiple servers at physically separate locations. The computers that store, manage, and interface with users across the various locations are linked by network. A centralized distributed database utilizes a central server to access dispersed database sources. [7]

DLL
abbreviation for Dynamically Linked Library.

DML
abbr for data manipulation language. [6]

- **DMS**
abbr contained in definition for transaction

DMIS
abbreviation for Dimensional Measurement Interface Specification

DNC
1abbr for Direct Numerical Control.

2abbr for Digital Numerical Control.

domain
the set of values assigned to the independent variable of a function [11]

domain independent application
an applications that supplies general functionality and was not necessarily developed for NGC. [7]]

drawing interchange file (DXF)
the AutoCAD systemís ASCII drawing interchange files describe a CAD drawing created by that system software. These files can be easily translated into formats for other CAD systems. The files and their format are of interest to companies manufacturing CNCs because customers want to use AutoCAD to design parts and then dump the DXF files into the CNCs on their machine tools and quickly cut the parts. [4]

drag
to move the tool with a dead spindle. [3]

drift
slow monotonic change in measured data [5]

drill
a machining task; to make a rounded hole or cavity in a solid by removing bits with a rotating cutting tool. [13]

drill bit
a cutting tool for boring holes in material.

drive
a mechanism for controlling a motor.

DSP
abbr contained in definition for servo motor control

dwell
to stop tool motion while the spindle is turning; to break chip, for example. [3]

DXF
abbr for Drawing Interchange File.

dynamic binding
the code associated with a given procedure call is not necessarily linked until the moment of the call at run-time. [7]

dynamic error
the difference between the true and the reported values of a machine motion variable due to error sources. [7]

dynamic following error
the distance between the commanded position and the actual position of the device under motion control. [7]

EBM
abbr for Electron Beam Machine.

ECM
abbr for ElectroChemical Machine.

EDM
abbreviation for electrodischarge machine.

EIA
abbr for Electrical/Electronic Industry Association.

effector
a device that physically performs an action or transformation function. [7]

electrical cabinet
large metal box, also called the magnetic cabinet, containing the CNC computer boards, servo amplifiers, communications boards and other wiring necessary to control the machine electronically. Some OEMs attach the operator console to the outside of this cabinet. [4]

electrochemical machine (ECM)
a machine which performs material removal via corrosive chemicals.

electrodischarge machine (EDM)
a machine which performs material removal via electric potential.

element
a constituent of the NGC system at the level directly below system level. [7]

emergency stop
a systemís emergency stopping circuits and devices were designed to avoid injury to personnel and damage to the machine. Usually large, red E-stop buttons are the emergency stopping devices on the operator console, the remote jog unit, and the machine itself. [4]

enclosure
screening around the cutting surfaces of a machine to protect the operator from material and coolant that may be propelled from the cutting area during operation. Enclosures are usually metal doors with windows in them to allow the operator to see inside. Many countries have regulations defining these enclosures and the mechanisms for opening and closing them. [4]

encoder
a mechanic device for translating motion into a unique electronic signal or combination of signals. Modern machine tools use optical encoders. See also quadrature. [4]

encoder feedback

an interface that allows a controller to interface to incremental or Quadrature encoders or other high speed pulse sources. The interface is capable of accepting Phase 0, Phase 90, and index pulse inputs. These signals can be TTL, single-ended or differential (strappable) and are optically isolated. The interface provides logic to detect overflow/underflow conditions and desired position comparison triggers. The index pulse can be used to latch counter data. These conditions can be made to generate interrupts to the host processor. Inputs are conditioned by a four stage digital filter. Five jumper selectable sampling frequencies are available for filter use. Selecting the lowest frequency compatible with the highest expected input rate will maximize noise immunity. The interface accepts quadrature input and input pulse rates. The counter register operates in quadrature decoding, pulse and direction input counting, or in a pulse input up/down mode. Counter output is available to the host bus as a binary or coded decimal form. [10]

end effector
an effector, gripper, or mechanical device, located at the end of last joint, by which objects can be grasped or acted upon. [7]

endfeed grinding
a type of centerless grinding.

end mill
a type of task.

end mill sharpening
a type of grinding.

endpoint control
any control scheme in which only the motion of the tip of the end effector may be commanded. [7]

end user
anyone who uses a manufacturing workstation for the execution of tasks. [7]

enterprise
the top level of a manufacturing hierarchy under which exists the factory, center, and cell [7]

enterprise information
the enterprise information is any data from the enterprise level that is communicated to lower levels. Enterprise information may include procedures, policies and schedules. [7]

environment
the aggregate of physical factors that act on the manufacturing workstations, manufacturing processes, and final products. [7]

error
difference between desired and measured data; synonym for delta; standard engineering term [5]

ESPRIT
abbr for the European Strategic Programme for Research and Development of Information Technology.

E-stop
abbr for Emergency Stop.

etch
a task; to produce a design, usually in a metal or glass surface by covering it with an acid resistant ground through which a design is scratched with a pointed instrument, then submitting the surface to an acid bath. [13]

euler parameters
four parameters for specifying quaternions; standard symbols e_{1234},a,b,c,s; standard units 1; primary units 1 [5]

European Strategic Programme for Research and Development of Information Technology (ESPRIT)
a consortium of 21 European companies now working on the specification and definition of CIM-OSA. The consolidated efforts which focus on CIM has resulted in what is today the AMICE consortium. [6]

exception handling
managing deviations from the planned/nominal production or process. [7]

executive viewpoint
a DSSA view of a system.

expert system
a system that captures a human expert's knowledge in a computer application for general application to provide solutions for similar problems. An expert system attempts to capture some of the intuitive reasoning that experts use to make decisions. [6]

exponential
common math function; standard notation e^x, exp x [5]

external communication
communication of a manufacturing workstation with the outside world through its computing platform. [7]

external interface
interfaces of a manufacturing workstation with the outside world, to cell level above, and to sensors/effectors below at the machine level. [7]

extrapolate
function to determine values from two or values in a table; usually linear but can be higher order [5]

fabrication
a term used to distinguish manufacturing operations for components, as opposed to assembly operations; to make or produce a part from raw stock. [6]

face mill cutter sharpening
a type of grinding.

face plate
the part of a robot at the end of the last joint, to which an end effector can be attached.

factory
a manufacturing unit consisting of two or more centers and the materials transport, storage buffers and communications that interconnect them. [7]

failure recovery
a condition that begins with an assessment of the damage. Repair of equipment and/or replacement of components by the operator may be required. In some instances the process may be allowed to resume after a failure recovery operation has been successfully performed, but at a lower level of performance. Failure recovery shall involve the performance of specific automatically generated and sequentially executed steps, and may require some interactive step execution as well. In most instances, a combination of automatic and interactive steps are required to facilitate a successful failure recovery. [7]

fault detection/isolation
an automatic condition that occurs when the controller is in the normal production mode. It automatically detect failures or interruptions in service, indicates location, and diagnoses cause when possible, Fault recovery actions are also initiated. Detection takes place when a flag is set or when some metric takes on a significantly different value. Possible metrics include the following: a process parameter that has exceeded its control limits, a statistical process control parameter, the output of a sensor fusion/integration module, or derived data such as runtime statistics that monitor performance. [7]

fault management
prevention, correction, or recovery from major deviations from current planned operations; predicts impending failures in time for corrective action; detects failures or interruptions in service; indicates location and diagnoses the cause when possible, determines and initiates recovery action. [7]

Fanuc
a controller manufacturer.

FBD
abbr for Function Block Diagram

feature
a physical attribute such as a surface, hole, or slot, used to describe a part. [7]

feature-based definition
a method of describing a part in terms of its features. [7]

feed
to provide a machine with material.

feedback
1 Input describing what is being controlled. This data is used by the CNC to adjust positioning and velocity. [4]

2 the furnishing of data concerning the output of a machine to an automatic control device or to the machine itself, so that subsequent or ongoing operations of the machine can be altered or corrected [11]

3 correcting or controlling a system by using part of the output as input; the flow of information back into the control system so that actual performance can be compared with planned performance. [6]

feedrate
feeding material into a tool. [4]

feedrate override
a way, via the operator console, for the operator to manually change the feedrate.

- **feeds and speeds**
machine tool movements determining the rate at which a material is machined or transported. This can be in the form of stock feedrate into the tool, tool rotational speed, or tool translational rate. [7]

field replaceable unit
a hardware unit or software module that can be physically replaced on site, thus reducing downtime. [7]

filter
a device to alter a signal; software to alter a data steam; see averaging filter, band-pass filter, complementary filter, first-order filter, high-pass filter, hysteresis, Kalman filter, limiter, low-pass filter, rate limiter, second-order filter, smoothing filter, wash-out filter; standard engineering term [5]

finish
a task; to polish a raw edge to form a smooth surface; to finish a raw edge by hemming, pinking, overcasting, or facing. [13]

first-order filter
a filter in which the output follows the input, only more slowly; It is usually implemented in software as a difference equation of period T as

$$y_n = e^{-\frac{T}{\tau}} y_{n-1} + (1 - e^{-\frac{T}{\tau}}) x_n = x_n + e^{-\frac{T}{\tau}} (y_{n-1} - x_n)$$

where τ is a filter constant. If the input is a unit step, then

$$y(t) = 1 - e^{-\frac{t}{\tau}}$$
$$y(\tau) \approx 0.6321$$
$$y[n] = 1 - e^{-\frac{T}{\tau}(n+1)}$$

The Laplace transform is

$$\frac{Y(s)}{X(s)} = \frac{1}{\tau s + 1}$$

The Z transform is

$$\frac{Y(s)}{X(s)} = \frac{(1 - e^{-\frac{T}{\tau}})z}{z - e^{-\frac{T}{\tau}}}$$

The differential equation is

$$y = x - \tau \dot{y}$$

The first-order filter is commonly used in avionics to smooth data, and to wash out transients at mode change, where typical values for τ are 1-5s. It is also used as a low-pass filter, with cutoff frequency $\omega_c = 1/\tau$.

When implementing a second-order filter on normalized variables, such as angles, the discontinuities require special treatment. [5]

fixture
any device that holds workpieces during the machining operation; a fixture by itself does not provide location information. [7]

flat end mill
a type of cutting tool.

flexible tap
a type of tapping tool in which the rotation and the z-motion of the tool are not tightly coupled. [15]

FLEXIS
performs "offline programming" of robot/manufacturing processes. Uses Grafcet language.

flute
a groove machined in a cutting tool to facilitate easy chip removal and to permit cutting fluid to reach the cutting point. [9]

following error
the difference between where a cutting tool actually is and where the control commands it to be. [4]

form cutter sharpening
a type of grinding.

form grinding
a type of grinding

function
a mathematical relation between two sets, called the domain and range, which assigns a unique value in the domain to each value in the range. i.e. f(domain coordinate) = range coordinate [17]

function block diagram (FBD)
a graphical language that allows program elements that appear as blocks to be wired together in a form analogous to a circuit diagram. FBD is well suited for applications that involve continuous flow of information or data between control. [10]

gauge vector
attribute of a tool.

geometric error
the difference between the true location of a particular end effector and the location as reported by the machinery scales. The machinery is assumed to be in a static, unloaded posture at a reference temperature. [7]

geometric modeling applications program (GMAP)
built on the results of PDDI, its goal is to identify and organize geometric and nongeometric product definition data required for the engineering, manufacturing, and logistics support of complex structured components throughout the product life cycle. [6]

geometry
the definition (measurements and relationships) of the dimensional properties of points, lines, curves, surfaces, and solids, as required to convey the shape of configuration aspects of a design in human processing terms. [6]

GM
abbr for General Motors

GMAP
abbr for Geometric Modeling Applications Program.

GMF
abbr for General Motors/Fanuc.

GMTS

abbr for General Motors Technical Staff. Authors of a Standard for Machinery and Equipment.

goal
the end result of the manufacturing process. [7]

granularity
degree of refinement. [7]

graphical user interface (GUI)
common user interfaces applied to the controller domain include: Citect, Fix, Genesis, Labview, Lotus@Factory, Paragon, PCIM, Wonderware [10]

gravitational acceleration
acceleration caused by the force of gravity; standard symbol g', standard units lbf, kip; primary units, ML/θ; gravity sometimes includes effects of the earth's rotation; gravity is often treated as a constant, but for greater accuracy gravity is a function of latitude, altitude, and the phase of the moon [5]

grind
a machining task; to wear down, polish, or sharpen by friction. [13]

grind belt
a type of grinding tool.

grind wheel
a type of grinding tool.

gripper
a type of end effector, a device by which physical objects are grasped and held. [7]

GUI
abbr for Graphical User Interface

Hall effect sensor
highly accurate, non-contact limit switch which detects the proximity of a magnet and provides a digital output to assure an accurate position reference. [8]

hand crank
usually a wheel with a handle attached used to control the motion of an axis on a manual machine. [4]

hierarchical control
control in which tasks are decomposed into sequences of subtasks that are passed to the next lower level in the hierarchy until the lowest level is reached. [7]

high-pass filter (HPF)
a filter that allows frequencies above a cutoff to pass while attenuating frequencies below the cutoff frequency [5]

holding torque
stepper motors, when energized, hold position via a magnetic field. The holding torque is the maximum torque which can be generated before the rotor slips to the next pole location. [8]

home position
a tool may have a home position to which it traverses to get out of the way of the machine.

home switch
any of a variety of sensors which can be used to establish an accurate initial position. This may consist of a standard end-of-travel Hall sensor; a center position optointerupter with half-travel blocking vane; an index signal on a linear encoder; a shaft coupling mounted magnet with Hall sensor; or a once-per-revolution encoder index signal. Once-per-revolution sensors will usually require a logical ORing with a linear signal if a unique home position is required. [8]

HPF
abbr for High-Pass Filter [5]

HUI
abbr for Human User Interface

human-user interface (HUI)
an information channel that conveys information between a human user and a system. It is composed of the surface layout, domain-independent applications (such as window and graphic packages), open HUI applications, and the PMSs. [7]

hysteresis
1 the error which can occur when a motion system is commanded to return to a starting position after several interim moves are made. It is attributable to lead screw reversing errors, backlash, etc. [6]

2 a function in which the algorithm for computing output changes at defined events or thresholds, such that output follows as input increases and another path as input decreases.

Hysteresis can be formalized:

```
    at initialization, select algorithm-0
    if event-1 occurs, switch to algorithm-1
    if event-2 occurs, switch to algorithm-2
    :
    .
    if event-n occurs, switch to algorithm-n
```

Hysteresis can prevent a test (x = x_c) from oscillating near the transition point (x Å x_c) due to noise. Implementation is usually:

```
    at initialization, set y = 0
    if $x_c$ + (h/2) ² x, then set y = 1
    if $x_c$ - (h/2) < x < $x_c$ + (h/2), then let y
    retain its value
    if x ² xc - (h/2), then set y = 0  [5]
```

IB
abbr for Information Base.

ICAM
abbr for Integrated Computed Aided Manufacturing.

identity
identity matrix; standard symbol I; standard units 1; primary units 1 [5]

IEC-1131
1 A standard PLC programming and execution environment that would facilitate the proliferation of many third party vendor, reusable software modules, and new software development tools. [21]

2 A controller language standard which is a collection of four different languages: Pascal-like, Grafcet-like, Signals on IC chip like, assembly language like. Statements in each language can be included in each other.

IGES

1 abbr for Interactive Graphics Exchange Specification
2 abbr for International Graphics Exchange Standard.

- IL
abbr for Instruction List.

impulse
synonym for unit impulse [5]

IMW
abbr for Intelligent Machining Workstation.

Industrial Real-Time Operating System Nucleus (ITRON)
part of Japan's TRON program to develop a new operating system for use across the spectrum of computer systems and industrial machinery. [6]

infeed grinding
a type of centerless grinding.

information base (IB)
an abstract storage mechanism that may consist of multiple, coordinated information sources (e.g., data bases, knowledge bases, memory maps) distributed across heterogeneous or homogeneous platforms. The IB is the conceptual structure to which all application-and service-generated requests for shared data manipulation are made. [7]

information base application
code that allows a user to interact with or manipulate specific information in a manner that particularly suits NGC IB needs, e.g., a graphical schema browser capable of taking STEP input. [7]

information base support package
a software component (often COTS) that provides specific IB functionality to be accessed through the data management services abstraction layer, e.g., a DBMS, a KBMS. [7]

initialization
1 a procedure to reset physical devices to a known state. During initialization, the device is usually not available. [5]

2 an interactive condition that occurs when the controller is in the start-up mode. Initialization establishes all late bindings and communications links between SAs, and establishes the initial state for all SAs. Initialization also retrieves the appropriate models for the selected modes determined by the configuration. [7]

in-process gauging
measure the part with a probe while the part is still on the machine. Problems are encountered due to additional cycle time required for sensing with devices such as lasers. [18]

input/output artifacts
objects, either physical or virtual, such as buttons, dials, readouts, sliders, etc., that provide specific user input or output capabilities and that can be combined to create user interface devices. [7]

instance

a distinguished occurrence of a particular object. [7]

installation
the creation of an instance. [7]

instruction list (IL)
a low level language, similar to assembler language. It is useful for smaller applications or for optimizing parts of an application. [10]

instrumentation
hardware to measure and to monitor a system [5]

integrate/configure
an interactive condition that occurs when the controller is in the start-up mode. Hardware integration, physical integration and configuration shall include installing mechanisms and computing hardware, and/or connecting wires to the hardware ports; software integration and configuration implies loading the code and data required for the controller onto the computing platform. [7]

integrated computed aided manufacturing (ICAM)
1 an activity modeling methodology that represents the functional decomposition of any activity. An activity in the model converts inputs to outputs using the defined mechanisms and under the constraints imposed by the controls, and linked by those inputs, outputs, and controls, to other activities in the model. [6]

2 a representation of structurally integrated information. A methodology for developing a data model which expresses the structural characteristics of information. It is used as a foundation for inter-division data standards and data base design, showing data and entity relationships, and attribute identification. [6]

integration
the formation of one complete and harmonious entity or coordinated entities of different applications that will accomplish a complete process solution. [7]

integration architecture
the components that provide the interconnection structure and interface definition between interoperating applications. It is composed of the platform services, communication services, data management services, presentation management services, task management services, geometric modeler services, basic I/O services, and the data standards. [7]

integrator
a function or filter that mathematically integrates [5]

intelligent machining workstation (IMW)
a USAF Program conducted by Carnegie Mellon University, Cincinnati Milacron and Pratt & Whitney. [6]

interactive graphics exchange specification, or International Graphics Exchange Standard (IGES)
1 files conforming to this specification are generated by CAD systems to define part geometry. However, interpretation of these files by different CAD systems may vary causing variations in the parts on different systems. [4]

2 a format for the exchange of graphical information (such as lines, ellipses, and text) between dissimilar drawing systems. This data enables the electronic transmission of prints, but does not include information about the items represented by those prints. [6]

3 a CAD/CAM data exchange specification adopted by ANSI. IGES attempts to standardize communication of two-dimensional drawing and geometric product information between computer systems. [6]

interchangability
a characteristic of system components that makes it possible to replace one component with another component of equivalent functionality made by a different vendor. [7]

interface
the definition of common boundary or a point or means of interaction between two or more distinct entities. It may be a shared boundary between two subsystems, which are defined by functional characteristics, common physical interconnections, signal specifications, and other characteristics. [7]

internal grinding
a type of grinding.

International Organization for Standardization (ISO)
an organization of national standards bodies from various countries established to promote development of standards to facilitate international exchange of goods and services, and develop cooperation in intellectual, scientific, technological, and economic activity. [6]

interoperability
1 a characteristic of independent system components that makes it possible for a set of components that comprise a system to intercommunicate and work properly together. [7]

2 that characteristic of a SOSAS compliant module which makes it possible for the module to operate on different computing platforms, or be integrated with differently configured NGCs. [6]

interpolate
function to determine intermediate values from two or values in a table; usually linear but can be higher order; endpoints are either extrapolated or limited [5]

interpolation
1 the process of approximating a given function by using its values at a discrete set of points [11]

2 an algorithmic process to generate intermediate points along a prescribed path or specific geometric shape between two points. For example, interpolation might be used to generate an arc with a 3.5-in. radius (circular interpolation) between points A and B. In addition to linear and circular interpolation, there are advanced algorithms to produce helices, splines, etc. [6]

interpolation block
offline system generates more than machine can handle, so machine ignore blocks coming in when its not ready for them. [20]

interrupt

interrupts are controlled with interrupt registers consisting of an end interrupt, poll, poll status, interrupt mask, priority mask, in-service, interrupt request, and interrupt status. [10]

inventory
the aggregate of tangible company property items which are held for sale in the ordinary course of business, or are in process of production for such sale, or are to be currently consumed in the production of products or services to be available for sale. [6]

inverse
standard math function; standard matrix operator; standard notation x^{-1}, X^{-1} [5]

I/O interface
different I/O types are either accessed directly from the base hardware platform via serial or parallel interfaces or may otherwise be accessed via add-on expansion modules. I/O types include: CANbus, Opto-22 PAMUX and OPTOMUX, APC Seriplex, Keithley Metrabyte and Workhorse, TURCK Sensoplex, AB1771 Digital, GE Series One, TI/Series, TI/Siemens, Transition Technologies, Grayhill, Gordos, Burr Brown Sensorbus, Du Tec [10]

iron
the frame of a machine without any electrical equipment, such as the CNC or servo drives, attached. The iron is produced at a foundry and then the bed, tool changer, and other electrical equipment is added. [4]

I/O
abbr for Input/Output

IPM
abbr for Inches Per Minute.

IPR
abbr for Inches Per Revolution.

ISO
abbr for International Organization for Standardization.

ITRON
abbr for Industrial Real-Time Operating System Nucleus.

jerk
rate of change of acceleration, either scalar or vector, often with subscripts to denote the coordinate frame; time derivative of acceleration; standard symbol j, J; standard units ft/s^3; primary units L/θ [5]

jig
1 a mechanical construction that determines location dimensions that are going to be machined into a workpiece. A jig can be incorporated into a fixture so that both accurate location and clamping are implemented (see fixture). [7]

2 a plate, box, or open frame for guiding work and for guiding a machine tool to the work, used especially for location and spacing drilled holes. [11]

jig saw

a type of saw machine with a narrow, vertically reciprocating blade for cutting curved and irregular lines or ornamental patterns in open work. [13]

JIT
abbr for Just-In-Time.

job shop
businesses the supply small quantities of parts (20 to 30) on a contract basis to larger companies. The large companies send specifications for the parts to the job shops and award the contracts to the lowest job shop bidders. These shops usually specialize in particular types of parts (e.g., springs for the automotive industry or hand cranks for knee mills). [4]

jog
1 manual control of an axis. [4]

2 an interactive condition that occurs when the controller is in the normal production mode; allows a machine axis to moved in any direction within the machine envelope or limited parameters via specified increments, or continuously until the human interface button is released. [7]

joint
a point of articulation between moveable parts. [7]

just-in-time (JIT)
a logistic approach to the movement of material to the necessary place at the necessary time. It allows for no buffer stock and is designed to minimize inventory during the manufacturing process. In short, synchronized production scheduling. [6]

kalman filter
a filter for combining multiple data sources, usually of different types, to produce an estimate better than any single source; compare with averaging filter, complementary filter [5]

Karel
an robot programming language, a successor to VAL. Named for the Czech playwright Karel Capek who wrote RUR.

KBMS
abbr for Knowledge Base Management System

kinematic error
the difference between the true and the reported locations of an end effector due to mechanical distortion of the machinery caused by motion. [7]

kinematic model
the controller's representation of the machine's motion.

kinematics
the branch of physics which deals with pure motion, without reference to the masses or forces involved in it [11]

knee mill machine
an inexpensive type of machine used to cut small parts in limited quantities. These machines usually do not have enclosures around the cutting area, and they have lower horsepower motors than bed mills. Knee mills may have CNCs attached, but they all

use at least one hand crank to move the bed with the part attached up to or away from the cutting tool. The "Z" axis on these mills has limited travel up and down, so the operator always needs to adjust the position of the bed. In fact, this machine received its nickname because operators use their knees to bump the cranks and adjust the beds when their hands are busy. [4]

knowledge base

1 semantically rich set of symbols, relations, procedures, and constraints. [7]

2 the data, data rules of use, and procedures stored to support expert systems and artificial intelligence applications. It implies that some semantic (knowledge about the meaning and possible uses of the data) content is included. [6]

knowledge base management system (KBMS)

a software system (often COTS) for managing and providing reasoning mechanisms for one or more centralized or distributed knowledge bases, e.g., rule sets. [7]

knowledge base system

a knowledge base system consists of several components: a knowledge base and a mechanism for reasoning about stored information. The knowledge base contains domain knowledge, rules and heuristics. The reasoning mechanism includes the control for rule selection and execution. [6]

ladder diagram (LD)

a powerful collection of standard graphical symbols and function blocks that can be combined into complex control programs that portray the actual execution component taking place in the SFC step or action. [10]

ladder logic

1 written representation of the electrical relays required to operate a machine. There are several different forms of ladder logic (see PAL). Ladder logic in the United States looks like rungs of a ladder with contacts and relays on the vertical rungs; however, European ladder logic is written horizontally. (Also referred to as relay ladder logic.) [4]

2 a programming language utilized for PLCs; the language for discrete control originally based on relays, rails, and rungs. [7]

ladder logic execution

ladder logic is driven (started) by a timer. It executes for each millisecond chunk and always responds to time interrupts. After ladder is evaluated, a delay time is hardcoded to give background tasks a chance to run. Background tasks include operator interface messages and other user-defined code sequences. Ladder logic has a higher priority, so it always takes place. It must be interrupted so that other processing can be executed.

Software interrupts are used to tie user-defined code to particular points of the ladder logic. Software interrupts occur at the end of every OPI state, timer interrupt, input phase, output phase, and during some particular instruction in the ladder logic. This allows a particular ladder logic command to initiate a customized program. [10]

ladder logic programming

a simple high-level graphical language for drawing control logic diagrams. Input contacts can be represented as normally open or closed. Output coils can be energized, latched, or unlatched. Boolean logic is represented by horizontal rungs (AND function) and vertical branches (OR function) read left rail to right rail with logic conditions on the left and output instructions on the right. Current logic states are graphically displayed as connected/disconnected sections of the rung(S). If a complete unbroken path exists from the left rail of the rung to the right trail, the rung is TRUE and the output instruction is acted upon. This discussion has only illustrated a small portion of the entire ladder logic instruction set. [10]

Laplace transform
a mathematical relationship to model a continuous function in the complex frequency domain (S-plane); compare with continuous-time equation, discrete-time equation, Z transform; Laplace transforms are commonly used by systems engineers to describe systems; see first-order filter, integrator, second-order filter, unit functions for examples [5]

latency
time between the request for an action and the initiation/completion of that action. [7]

lathe
a machine used to shape a part by gripping it in a holding device and rotating it under power against a cutting tool. Lathes are commonly used for turning, boring, facing, and threading. [4]

LD
abbr for Ladder Diagram

lead
synonym for pitch.

lead error
the deviation of a leadscrew from its nominal pitch. The error is often monotonic (linear), although periodic error and thermal expansion set limits to its predictability. [8]

leadscrew
1 a mechanical device, also called a ball screw, for translating rotary motion into linear motion, consisting of an externally threaded screw and an internally threaded carriage (nut). These typically move the bed, the spindle, and the head surrounding the spindle. [4]

2 a long precision screw on the front of a lathe bed that is geared to the spindle to transmit motion of the carriage for thread cutting [9]

leadscrew mapping
leadscrew motion can be mapped by indicating the number of revolutions per inch. This technique allows for compensation in leadscrew pitch because the pitch per inch is not constant (see lead error). [4]

lead through teach
record the trajectories. Use teach points to avoid obstacles [20]

length offset
attribute of a tool

life-cycle costs
the cost of a product and its related activities that occur over the entire life of the product. [7]

limiter
a filter that passes the input to the output, except that the output is limited to a minimum value and a maximum value; compare with rate limiter.

Limiters are usually implemented as

$$y_n = \max(x_{min},\min(x_{max},x_n))$$

where x_{min} is the lower limits and x_{max} is the upper limit. [5]

limit switch
a sensor, typically Hall effect, optical, eddy current, or mechanical, which is used to sense the end of travel of a linear motion assembly. In addition to preventing overtravel, it is frequently used to establish a precision reference. [8]

linear force/torque
a type of control law.

linear position/velocity
a type of control law.

link
part of a robot.

locking
preventing access by one transaction to an object by another transaction. Locking may prevent data modification only or all read/write access. [7]

logarithm
common math function: standard notation $\log_b x$; $\log x \equiv \log_{10} x$ [5]

logging
shadowing system activities over time by creating a time-stamped log file of transactions and state information. [7]

look ahead
algorithm that takes into account future commands to make decisions about current motions. For example, the control will note that the cutting is going to be commanded to stop so it begins to decelerate. [4]

look and feel
surface-level representation attributes and behaviors independent of specific presentation layout, i.e., the style of presentation. [7]

low-pass filter (LPF)
a filter that allows frequencies below a cutoff frequency to pass while attenuating frequencies above the cutoff frequency; see first-order filter [5]

low rate initial production (LRIP)
normally, the phase of production used to assess readiness for full rate production. [6]

LPF
abbr for Low-Pass Filter [5]

LRIP
abbr for Low Rate Initial Production.

lube
synonym for lubricant.

lubricant
a substance used to reduce friction between the cutting tool and the part. [4]

LWP
abbr for Light-Weight Process.

M & G codes
a slang phrase describing combinations of the letters (common letters being ìMî and ìGî) with multiple numbers to indicate where a machine cutter should move and what it should do. These codes are used in NC programs. The codes were originally fed into a machine as a punched tape of instructions. Many other letters are used, such as ìTî codes to describe tool changes. ìGî codes describe positions, and ìMî codes identify miscellaneous functions. [4]

machine model
a structure data representation that provides a description of a machine and simulates its properties and characteristics of interest. [7]

machine tool
a stationary power-driven device used to shape, cut, turn, bore, drill, grind, or polish solid parts, especially metal. [4]

machine under control
any machine or process being controlled by NGC. [7]

machinist
a highly skilled machine operator who makes decisions about how a part will be programmed, fixtured, and cut. [4]

MADE
abbreviation for Manufacturing Automation Design & Engineering.

magnitude
common math function; synonym for absolute value [5]

maintainability
the measure of the ability of a system or product to be retained in or restored to specified conditions when used by personnel having the specified skills, using prescribed procedures and resources. [6]

maintenance
any activity to eliminate faults or to keep hardware or programs in satisfactory condition, including tests, measurements, replacements, adjustments, and repairs. [6]

manipulator
a mechanism usually consisting of a series of articulated links, for the purpose of grasping and moving physical objects. [7]

man-machine language (MML)
a user interface.

manual machine

a machine that does not use computer components to control operation of the cutting tools. Usually the operator interacts with the machine by turning hand cranks to adjust the spindle speed and to move the cutting tools and/or bed. [4]

manufacturing

a series of interrelated actions involving the process design, material selection, planning, production, quality assurance, management and marketing of discrete consumer and desirable goods. [6]

Manufacturing Automation Protocol (MAP)

IEEE 802.4 token-bus specification. [6]

manufacturing workstation

1 a material transformer device that includes a controller, computing platform, human interface, and peripheral devices. [7]

2 a manufacturing workstation is defined as a single material transformer device and related support equipment (e.g., material handling or inspection) together with the processing and control capability for autonomous response to commands and inquiries. [6]

MAP

abbr for Manufacturing Automation Protocol.

mark

a measuring task.

materials resource planning (MRP)

scheduling for the manufacturing floor.

matrix

standard notation $[a_{ij}]$, A [5]

maximum

common math function; standard notation $\max(x_1,...,x_n)$ [5]

metrology

science/system of weights and measures or of measurement. [13]

microstepping

a technique which, instead of switching phase currents in a stepper motor on and off, slightly decreases the current in one winding, while slightly increasing it in another. This increases the resolution. [8]

mill

1 A machining process which removes material, usually metal, from a part using one or more rotating cutting tools. [4]

2 to shape or dress by means of a rotary cutter. [13]

milling machine

a machine capable of performing the operation of cutting, shaping, or finishing a workpiece. [7]

minimum
common math function; standard notation $\min(x_1,...,x_n)$[5]

MML
abbr for Man-Machine Language.

MMPM
abbr for Millimeters Per Minute.

MMPR
abbreviation for Millimeters Per Revolution.

MMST
it features interoperability of a growing number of machines used in semiconductor manufacturing.

modal
a state during which an established operation remains unchanged; compare with one-shot [3]

mode
an enumerated mechanism for coordinating behavior. [7]

model
1 a structured data representation of a process or system that simulates properties and characteristics of interest. [7]

2 a mathematical representation [5]

modular
a standardized and flexible construction consisting of logically self-contained and discrete parts; it allows for scalability and interchangability within a NGC. [7]

mold
a form made on a milling or punch machine into which a liquid is poured that hardens into the shape of the mold. The resulting molded product may also be milled as a finishing step. [4]

motion control
control of continuous joint motion of machines. [7]

motor
device that converts electrical energy into mechanical energy by using forces produced by magnetic fields on current-carrying conductors. There are three basic types of motors available on machine tools: DC brushed, DC brushless, and AC induction. [4]

moving column machine
in such a machine, the part remains in a fixed position while the spindle performs all motion.

MRP
abbr for Materials Resource Planning.

multiaxis mode
keeping the cutting tool perpendicular to the surface [19]

multiprocessor control

a control computing platform that contains more than one processing unit. [7]

natural frequency
see second-order filter; standard symbol ω_n; standard units rad/s, Hz; primary units 1/θ [5]

NC
abbr for Numerical Control.

NCMS
abbr for National Center for Manufacturing Sciences.

network interface
Arcnet and CANbus networks allow controllers to be networked with each other and with PCs. Arcnet is a Local Area Network (LAN) industry standard protocol that has been well accepted and proven in factory control applications with over 1.67 million installed nodes in the United States. CAN could be classified as a Small Area Network (SAN). It runs on an inexpensive twisted pair bus and provides an interface to other peripherals, peer-to-peer networking of controllers, deterministic prioritized message passing, small efficient packet sizes, robust error detection, and a reduced wiring solution. CAN is an industry standard protocol supported by multiple vendors.

Peer-to-peer networking means that each node in the network is of equal status and is able to transmit at any time within the guidelines of the network access method. This is in contrast to master-slave or server-consumer based networks in which a particular node(s) on the network has special network functions and privileges. In master-slave based networks, a slave node must wait for the master to poll it and give it permission to transmit before being able to send messages. This method can cause large latencies to exist when high priority messages such as alarms and fault conditions need to be reported. [10]

neutral manufacturing language (NML)
1 a proposed language for communications between NGC applications and between a NGC and the cell. [7]

2 used by humans to communicate with hardware/software modules and for communication between such modules. NML is the language used in communicating within an NGC, between NGCs and between an NGC and other systems. Manufacturing process plans, product descriptions, processes (at several levels of detail), machine management, motion control, and other aspects of manufacturing are expressed in NML. [6]

NGC
abbr for Next Generation Controller.

NGIS
abbr for Next Generation Inspection System.

NGIS - NC Inspection Project
a CMM with an open architecture. This project will eventually add additional sensor types besides just touch probes...vision, laser, ultrasound. It will marry wrist and machine controllers. A wrist controller has a standard I/O interface and holds a variety of different sensors. This will allow the machine controller to select a sensor for a particular process, much as it currently selects a tool for a particular process. Some

sensors include SAMI (a CCD based camera with software for vision, providing information on tool wear, surface conditions, etc.), laser for interferometries for surface conditions, Strudhome capacitive touch sensor (probe nears surface but does not touch). [21]

NML
abbr for Neutral Manufacturing Language.

NIST
abbr for National Institute of Standards and Technology.

noise
part of received data that is undesired, consisting of random sinusoidal terms added to a signal; compare with signal [5]

non-uniform rational b-spline
a spline is a function that has specified values at a finite number of points and consists of segments of polynomial functions joined smoothly at those points, enabling it to be used for approximation and interpolation of functions. For example, a cubic b-spline approximates a series of m+1 control points with a curve consisting of m-2 cubic polynomial curve segments. Non-uniform indicates that the control points do not have to be evenly spaced along the curve. Rational indicates that each parameter is defined by the quotient of two b-spline polynomials. [22]

normalizer
function to restrict input to a specific range, such as restricting an angle α in radians so that $-^{1}\;^{2}\;\alpha\;^{2}\;^{1}$; angles usually require normalizing following any computation; normalized variables often present problems for filters and other functions as their discontinuities [5]

numerical control (NC)
controlling the motion of machine components using numbers (M & G codes) often fed into the controller on punched tape. The controller mechanism has no intelligence and cannot be modified in any way without changing the wiring of the machine. (see also RS-274-D, CNC.) [4]

NURBS
abbr for Non-Uniform Rational B-Spline.

nut compliance
the reciprocal of the stiffness of a leadscrew/nut assembly, measured in length per axial force. [8]

object
an abstraction of an entity representing an encapsulation of its attributes and services on those attributes. [7]

object module
software that has been compiled, but not yet linked. [7]

object-oriented
a viewpoint that models data and behavior as objects. [7]

object-oriented programming
programming in a language that embodies the concepts of objects, classes, inheritance, polymorphism, and dynamic binding. [7]

object-oriented structures
two methods used to manage complexity; classification structure, which captures class/member organization; and assembly structure, which portrays whole/part organization. [7]

odd pockets with islands milling
method of surface milling in which the tool follows an inside-out or outside-in pattern across the part surface. [15]

OEM
abbr for Original Equipment Manufacturer.

offset
the amount of compensation required to account for the difference between the actual dimensions of a tool, probe, or end effector and the dimensions that were used to program the motion. [7]

offset vector
attribute of a fixture, which describes by how much the tool must be moved to avoid the stationary fixture

one-shot
a state after which an established operation is canceled upon execution; compare with modal [3]

open applications
applications developed for NGC whose functionality is not defined by the SOSAS but whose external interfaces are specified within the SOSAS. [7]

open loop
a control system in which data flows unidirectionally, that is, only from the control to the mechanism but not from the mechanism back to the control. [2]

open system
a system that provides capabilities that enable properly implemented applications to run on a wide variety of platforms from multiple vendors, interoperate with other system applications, and present a consistent style of interaction with the user (IEEE P1003.0). [7]

open system architecture
a specification of the capabilities or services that provides the interconnection structure and defines the interface between interoperating components, thus allowing applications to be integrated into a system with a consistent style of interaction. [7]

open system interconnection
a seven-layer model of intersystem communications specified by the International Standard Organization. [7]

open technology
a technology for which there are publicly available specification allowing the reinvention, from scratch, without royalties or license fees. [7]

operating system
software that controls the execution of application programs and manages computing platform services. [7]

operational
functioning correctly [5]

operator
human who controls movements of a machine and communicates instructions to a machine. Usually an operator is a lower skilled worker than a machinist. [4]

operator console
a device attached to a machine that allows the operator to communicate to the machine and receive message from the machine indicating its operating state. The console communicates to the control. [4]

operator interface
for the machine shop, it is best to train operators on one controller for all the machines in the shop. Plug in particular machine model for specifics on that machineís configurations, i.e. lathe model, mill model, grinder model. [18]

orient
to move the spindle to a desired rotational position

orientation
direction in reference to a coordinate frame [5]

original equipment manufacturer (OEM)
they build machine tools and fit controls, electrical devices, and operator consoles on the machines. They may completely build their own machines, or they may receive CNC-prepared iron from a manufacturer and integrate their own control or another manufacturerís control onto the iron. [4]

Open-System Interconnection (OSI)
the ISO/OSI model is a seven-layer model of intersystem communications. The model has been adopted for use in MAP. [6]

optical encoder
a linear or angular position feedback device, typically providing incremental two channel information in quadrature format (sine or square waves with a 90 phase shift between each channel). Such two channel information allows simple counter circuits to function as absolute position indicators. [8]

optimal feedrate
attribute of a tool

optimal spindle speed
attribute of a tool

orthogonality
the degree of perpendicularity, or squareness, between two axes (X, Y, Z). usually measured in arc seconds. [8]

OSI
abbr for Open-System Interconnection.

override
to alter selection made automatically by software [5]

overshoot
the amount of distance a cutting tool goes past the point where it was told to stop by the control [4]

PAL
abbr for Programmable Application Logic.

pallet
a portable platform on which materials (parts) are stacked for transportation [12]

pallet shuttle
a mechanism for transporting pallets of parts to and from the machining area.

PAMUX bus
a high-speed parallel bus. This bus can be up to 500 feet in length. It is a very reliable high-current bus that is extremely immune to noise. An individual digital I/O byte on the bus can be accessed in less than 3 microseconds, even at distances of 500 feet. Analog channels are accessed in less than 150 microseconds. A completely populated PAMUX digital bus could be read in approximately 250 microseconds. It uses a standard 40 conductor ribbon cable that is daisy-chained between I/O interface brain boards and terminated actively at each end. [10]

paperless factory
the automated and on-line control of the factory without the aid of printed reports or forms. [6]

parallels
a type of fixture; a block or strip of metal made with two parallel sides and used especially in machine shop work, as for a gauge block or for setting up work. [13]

part

1 end product of a milling operation on stock/workpiece. For example, a part could be a small piece used inside a motor or a large metal casting. Also called a workpiece itself. [4]

2 normally refers to a material item which is used as a component and is not an assembly. [6]

part geometry
the dimensional features of a part or an assembly; see geometry. [6]

part number
1 a number which serves to uniquely identify a component, product or raw material. [6]

2 the number that uniquely identifies each purchased, manufactured, or assembled part. It is also referred to as an item number. [6]

part program
1 definition of the machining features of a part such as process order number, part orientation and position, cutter tool types and size, depth of cuts, spindle speed, and feed rates. Then the programming system calculates the information needed to position the cutter and table. Many systems use computer graphics to show the tool path on a computer monitor. [4]

2 a manufacturing process description for a workpiece. [7]

part programming
used to create tool path. It describes an object to be made by a machine. Part programming can be accomplished on a

CAD/CAM system and downloaded to the machine or it can be programmed on the machine itself using through the operator console. [4]

part transformation
the removal or addition of material, assembly, or inspection of a part. Part transformation adds value to a part. [7]

part zero
the "zero"location on a part; where the X and Y axis meet on the part. [4]

path planner
software that produces paths for an end effector and computes collision avoidance algorithms, and satisfies other constraints. [7]

PCTE
abbr for Portable Common Tool Environment.

PDD
abbr for Product Definition Data

PDDI
abbr for Product Definition Data Interchange

PDES
abbr for Product Data Exchange using STEP.

period
time of a periodic process; $1/f_s$ where f_s is the sampling frequency; standard symbol T; standard units s; primary units θ [5]

periodic
a process that executes at a fixed rate; compare with aperiodic [5]

phase current
the rated current which a stepper motor requires to generate its rated holding torque. This value is usually based on unipolar (half-coil) operation. This choice of how the motor is wired has significant impact on performance. [8]

phase sequence
the specific sequence of coil current changes used to advance a stepper motor clockwise and counterclockwise, in either full or half step modes. [8]

PHIGS
abbr for Programmer's Hierarchical Interactive Graphics System. [6]

physical resource viewpoint
a DSSA view of a system.

PID control loop
a PID control loop is used to maintain a selected process variable at a desired set point. Many control applications require closed loop real-time control to set and maintain critical process characteristics which might be expected to vary or to drift over time. The PID function constantly monitors these parameters and calculates the error from the desired set-point and outputs a control parameter to return the process to the desired set point.

the update time of the PID loops determines the amount of total PID loops a single program/processor can handle. Simple loops and long update times allow for more PID loops in a program.

- PID can be used in both machine and process control applications. In a process control with temperature settings, PID can monitor the thermocouples and upon a programmed action, turn on a fan, open a valve, energize a pump and more. Other functions of PID include scaling, deadband, and zero-crossing. [10]

pitch
1 for leadscrews specified in British units, the number of full rotations required to advance the nut 1". For example, a 5 pitch leadscrew has a lead of .200". Metric screws are specified by lead only, usually in millimeters.

2 an angular deviation possible in positioning system, in which the tables' leading edge rises or falls as the table translates along its direction of travel. This represents rotation around a horizontal axis, perpendicular to the direction of travel. [8]

plain mill sharpening
a type of grinding.

plan
an ordered set of actions for accomplishing a goal. [7]

plane
a machining task.

planner based
an automatic condition that occurs when the controller is in the normal production mode; all resources are focused on part transformation or process control, and use of the production resources are in direct control of the WPSA. The WPSA provides the dialog or communications coordination that allows multiple SAs within the workstation to interact and perform simultaneous operations. [7]

planning
a procedure for determining the operations or actions necessary to transform material from one state to another. It includes the preparation of detailed instructions to produce a part transformation. [7]

plate
to cover with an adherent layer, as of metal, by mechanical, chemical, or electrical means. [13]

platform
a basic structure consisting of virtual machines, operating systems, communication mechanisms and hardware. [6]

PLC
abbr for Programmable Logic Controller.

plunge grinding
a type of grinding.

PMS
abbr included in definition of human-user interface

point-to-point control
a scheme whereby the inputs commands specify only points along a desired path or motion. [7]

polymorphism
in an object language, the characteristic of an operator that allows it to work on objects of different classes. [7]

portability
the ability to operate the same component on different computing platforms. [7]

position
location, either scalar or vector, often with subscripts to denote coordinated frame; time integral of velocity; standard symbol p, P, x, y, z; standard units ft; primary units L [5]

power-up
an interactive condition that occurs when the controller is in the startup mode; involves bringing the computing platform on-line via the normal startup routines. If auto-start files are integral to the controller, the NGC automatically start; otherwise, the appropriate start command must be issued to initialize the NGC environment. In either instance, power-up starting and self-test routines, as determined by the configuration, until they are successfully completed; [7]

precision
1 measure of exactness, possibly expressed in number of digits, for example, computed to the nearest millimeter; compare with accuracy [5]

2 the degree to which a given set of measurements of the same sample agree with their mean [11]

predicate
an expression that can be evaluated as either TRUE or FALSE. [6]

primary units
a standard set of four units to which all units can be resolved. The primary units are (1) mass-M, (2) length-L, (3) time, and (4) temperature-T. For example, standard units for velocity might be ft/s or m/s, but primary is always L/θ. [5]

primitive function
the lowest level of functionality of the NGC manufacturing workstation. [7]

prismatic
with regard to axes.

probing
synchronize with measure and motion. Probe must be quickly brought near to part, then slowing moved in close enough to part to just sense it for measurement. Only barely touching or not touching part at all. [17]

procedure
a systematic, controlled sequence of actions or operations performed in order to achieve a specific result. [6]

process

a set of procedures required to transform resources (inputs) into specific objectives (outputs) on a returning basis. [7]

process plan
the sequence and description of fabrication or assembly operations. [6]

Product Data Exchange using STEP (PDES)
1 a proposed data format and data exchange standard which enables the sharing of physical and functional product information among various computer systems and applications. [6]

2 an international standard for data developed due to the problems discovered with the IGES format. PDES, represented in the EXPRESS specification language, uses specific CAD translators and application specific protocols to reduce part geometry translation differences between CAD systems. [4]

product definition data (PDD)
machine interpretable product, design, and manufacturing information that includes geometric models, features, surface finish, heat treatment, materials specifications, and tolerances. [7]

product definition data interchange (PDDI)
a format that allows for interchanging of PDD between dissimilar CIM systems. [7]

product life cycle
all the stages of a product from initial concept to finish abandonment. [7]

programmable application logic
an application specific "ladder logic" that is continuously executed by the controlís executive firmware. This "ladder logic" is composed of PAL ladder language elements. These elements are combined to form ladder rungs that make up PAL ladder modules. These ladder modules are linked together to form a PAL program.

The PAL program coordinates and controls the I/O interface between the 9/240 control and the machine tool. Use PAL to perform the following functions: 1) sequence machine functions; 2) operate machine functions; 3) monitor machine functions. PAL communicates with the controlís executive firmware and discrete I/O to coordinate these machine functions.

The controlís executive firmware sends information to the PAL program through specific PAL flags and variables. The PAL program sends information back to the control through other PAL flags and variables. The PAL program can also turn on PAL messages and custom display pages that appear on the operator panel CRT. These flags are divided into the following areas:

primary system flags
manual motion flags
axis mover flags
offset modification flags
part program selection flags
part program execution flags
feedrate flags
G code flags
M code flags
tool change flags
random tool flags
spindle and gear change flags
spindle orient flags
mode select flags

status flags from the control
PAL message flags
display page flags
digitizing flags [1]

- **programmable logic controller (PLC)**
1 a device for discrete control characterized by repetitive cycling typically through the ladder logic program. [7]

2 a control device designed to replace banks of relays for the repetitive cycling through a series of mechanical device control commands. Current designs are computers with sophisticated logic and communications capabilities. [6]

Programmable Universal Machine for Assembly (PUMA)
"At a luxurious press conference in 1978, General Motors unveiled the PUMA system that included conveyors, parts feeders, and robots small enough to work alongside humans. But it was the small robots in that system, supplied by Unimation, that became known as PUMAs". [2]

protocol
a set of semantic and syntactic rules that determine the behavior of functional units in achieving communication. [6]

PUMA
abbr for Programmable Universal Machine for Assembly.

punch press
a machine with a table controlled in the x and y plane used to cut or press holes usually into sheet metal. Punches are often used to press vents into the sheet metal for the sides of electrical cabinets. [4]

quadrature
state of being separated in phase by 90° or one quarter cycle. A quadrature encoder provides both position and direction information to the control. [4]

quality
conformance to customer expectations. [6]

query
to request information from a database. [6]

query language
a language close to natural language allowing flexibility of expressing syntactical forms. It serves as the interface between the user and the mechanics of the system and enables him to query data. [6]

radial error probability (REP)
a probability that a percentage of one-dimension measurements will lie on a radial (line) of given length, with the origin centered at truth or mean of the measurements; compare with circular error probability, spherical error probability; used to specify test cases for measurement errors of sensors of one dimension, such as velocity.

For example, a position error of 1 ft/s (50% REP) means that for any given measurement of position, p(distance from truth to measurement is less than 1) = 0.5.

In another example, a velocity error of 1 ft/s (50% REP) means that for any given measurement of velocity

$$V_i = V_v$$

then

$$p(|\vec{V}_i - \vec{V}_\mu| \leq 1) = 0.5$$

where $\vec{V}_\mu$ is the average of all $\vec{V}_i$ or $\vec{V}$ truth, depending upon context. [5]

radius
a straight line extending from the center of a circle or sphere to the circumference or surface [11]

RAMP
abbr for Rapid Acquisition of Manufacturing Parts. [6]

ramp
1 n. synonym for unit ramp [5]

2 v. to gradually accelerate a motor, essential if performance beyond the start/stop range is required. The slope of the ramp is a function of screw pitch, load, motor, and drive voltage and design. [8]

rapid traverse
cutting feedrate to quickly position the cutting tool for the next cut. [4]

rate
rate of change or data; derivative with respect to real time [5]

rate limiter
a filter that passes the input as the output, except that rate of change of the output is limited to a maximum absolute value; compare with limiter.
Rate limiters are usually implemented as

$$y_n = y_{n-1} + sgn(x_n - y_{n-1}) \min(\dot{Y}_{\max}, |x_n - y_{n-1}|)$$

where $\dot{Y}_{\max}$ is the maximum absolute change per period. [5]

real time
1 time in a computational process which runs at the same rate as a physical process; for example, algorithms designed to run a fixed period T (filter time constants at set for T) and actually execute with frequency 1/T in real time [5]

2 within a strict, predictable time interval. Real-time processing often involves event waiting, timed reminders, validity intervals, and some form of approximate or progressive reasoning that allows creation of an acceptable (in some sense "best") response within the time allowed. [7]

real-time function
a function that must be carried out in a relatively brief time interval (typically a minute or less) and that is characterized by a commitment to execution. Here, any attempt to make a significant alteration in the planned course of action will result in some negative effect on the workpiece. [6]

real-time operation
an activity within a strict, predictable bounded-time interval. [7]

ream
a machining task; to finish a drilled hole to an exact size using a reamer [9]

reamer
a cutting tool used to produce a smooth accurate hole by removing a small amount of metal from a drilling hole [9]

reamer sharpening
a type of grinding.
reciprocating saw
a type of saw machine.

recovery
restoration of IB consistency in the event of hardware or software error. [7]

reference
what must be achieved in order to match a plan; synonym for desired [5]

referential integrity
the capability of ensuring that repository data does not contain references to non existent data or definitions. It involves automatically deleting a relationship instance whenever the source or target of that instance is deleted. [6]

relative
applies to measurements, in a non-standard, moving reference, as opposed to fixed reference; compare with absolute [5]

reliability
the measure of a system's ability to perform in a stated manner for a given period of time under specified operating conditions and environments. [7]

remote jog pendant
a hand-held device with wheels and buttons used to manually move axes. Also called a teach pendant. [4]

REP
abbr for Radial Error Probability [5]

repeatability
capability of CNC software to exactly imitate accurate motions. [4]

replication
the existence of multiple instantiations of a module's functionality within an implementation. [7]

requirement
a declarative statement describing customer needs. [7]

resolution
the smallest unit of measurement that a measuring system is capable of distinguishing. [7]

resolver
analog position feedback device. [4]

resonance
1 midrange resonance: a parasitic oscillation which is endemic to stepper motors, although frictional loads may mask its effect. It typically sets in from 5 to 15 revolutions per second, and can easily cause a loss of synchronization (stalling). [8]

2 primary resonance: The rotor inertia of a stepper motor, coupled to its spring-like magnetic field characteristics, constitutes a basic spring-mass oscillator. In the absence of sufficient damping, stepper at certain frequencies may excite resonance in this system, or resonate with the load, resulting in loss of synchrony. The addition of system damping, operation in half-step mode, or ramping through problem speeds will usually eliminate resonance. [8]

resource
any item within the manufacturing workstation used to transform a part, assist in the transformation of a part, or support other resources. These include machines-under-control, such as robots, machine tools, and material handling devices, as well as fixtures, tools, lubricants, coolants, and workpieces. [7]

retrofitter
a person or company who installs (integrates) new CNCs and other electrical equipment on old machine tools. Some large companies have retrofitting departments that are responsible for constantly upgrading the machine tools. Small job shops usually wait until a machine is too expensive or impossible to fix or it cannot be used to fulfill a new contract for parts before trying to retrofit a new control on the machine. [4]

reusable software
software that can be used for more than one implementation. [7]

reverse engineering
examining an existing part to determine its structure and make a copy or use that structure to create another part based on the features of the first. [4]

revolutions per minute (RPM)
this unit of measurement indicates the velocity at which a object must travel at a uniform angle to complete a circle in one minute. [4]

RMS
abbr for Root Mean Square

rigid tap
a type of tapping tool; compare flexible tap

robot
a programmable manipulator and its associated system. [7]

roll
an angular deviation from ideal straight line motion, in which the positioning table rotates around its axis of travel as it translates along that axis. [8]

rollback
return a database to its state before an uncommitted transaction was updated. In order to do this, a "before image" that reflects the state of the data items prior to update must be available for restoration. [7]

rollforward

restore updated information following a rollback. In order to do this, an "after image" that reflects the state of the data items following update must be available for restoration. [7]

root mean square (RMS)
a statistical measure of data; the square root of the mean of the square; compare with root sum square; for variables with mean of zero, the standard deviation is equal to the RMS. [5]

In discrete time, RMS is computed

$$x_{RMS} = \sqrt{\frac{\sum_{i=0}^{n} x_i^2}{n}}$$

In continuous time, RMS is computed:

$$x_{RMS} = \sqrt{\frac{1}{t}\int_o^t x_i^2(\tau)d\tau}$$

root sum square (RSS)
a statistical measure of data; the square root of the sum of the square; compare with root mean square; for a vector, its length is equal to the RSS of its scalar elements. [5]

In discrete time, RSS is computed:

$$x_{RSS} = \sqrt{\sum_{i=0}^{n} x_i^2}$$

In continuous time, RSS is computed:

$$x_{RSS} = \sqrt{\int_o^t x_i^2(\tau)d\tau}$$

rotary saw
a type of saw machine.

rotational force/torque
a type of control law.

rotational position/velocity
a type of control law.

roughing pass
a cycle of milling in which a significant margin is added to the tool motion, so that large amounts of material can be removed quickly. [15]

RPM
abbr for Revolutions Per Minute.

RS-274-D
an industry standard, part programming language for generating numerical control (NC) programs (using M & G codes). The programs describe tool path. This standard specifies the use of the EIA Standard RS-358 character code set and the basic characteristics of a numerical controller. [4]

RSS
abbr for Root Sum Square [5]

RT
abbr for Real-Time.

SA
abbr for Standardized Application

saber saw
a type of saw machine which is a portable electric jigsaw. [13]

safety
safety relates to procedures to keep humans free from injury or exposed to hazards. [7]

safety envelope
attribute of a part which describes the volume about the part in which it is safe to move the tool.

safety interlock
a safety interlock is dependent upon the design and purpose of the equipment. It provides a workstation warning and control in the event of human errors and misjudgments. It also detects component failures such as electrical overloads and shorts, valve sticking, or lack of flows and pressure. [7]

sampling frequency
rate of a periodic process; 1/T where T is the period; standard symbol fs; standard unit Hz; primary units 1/θ [5]

saw
a type of machine; a machine tool used to cut hard material, usually consisting of a thin, flat blade or plate or tempered steel with a continuous series of teeth on the edge and mounted in a handle or frame. [13]

scalability
the capability to increase or decrease the functionality of a manufacturing workstation or its components. [7]

scalar
a quantity possessing only magnitude; compare with vector; standard notation x [11]

scaling
the process in the PID instruction which allows the setpoint and zero crossing values to be displayed in engineering units. Setpoint, deadband, process variable, and error may be scaled. (see PID control loop) [10]

schema
a data model that defines all abstract object data types including relationships, attributes and constraints. [7]

sculpted surface
a machined or molded object whose surface is contoured and non-linear, e.g. the shape of a light bulb. [6]

SDT
abbr for System Development Tool

SEB
abbr for Sensor/Effector Bus

second-order filter
a smoothing filter in which the output follows the input, only more slowly. It is usually implemented in software as a difference equation of period T as:

$$\begin{aligned} y_n &= Ay_{n-1} - By_{n-2} + (1 - A + B)\frac{x_n + 2x_{n-1} + x_{n-2}}{4} \\ &\approx Ay_{n-1} - By_{n-2} + (1 - A + B)x_n \end{aligned}$$

A and B vary depending upon case:

$$A = 2e^{-\zeta\omega_n T}\cos\left(\omega_n T\sqrt{1-\zeta^2}\right) = e^{-aT} + e^{-bT}$$

$$B = e^{-2\zeta\omega_n T} = e^{-aT} + e^{-bT}$$

where $\omega_n>0$ (natural frequency) and $0<\zeta<1$ (damping ratio) are filter constants for underdamping (case 1), $\omega_n>0$ and $\zeta=0$ for no damping (case 2), a and b are filter constants for overdamping (case 3), and a=b or $\omega_n>0$ and $\zeta=1$ for critical damping (case 4).

If the input is a unit step then, the time responses are:

underdamping, no damping

$$y(t) = 1 - \frac{e^{-\zeta\omega_{nt}}}{\sqrt{1-\zeta_2}}\sin\left(\omega_n\sqrt{1-\zeta^{2}t} + \cos^{-1}\zeta\right)$$

overdamping, a_b

$$y(t) = 1 + \frac{b}{a-b}e^{-at} + \frac{a}{b-a}e^{-bt}$$

critical damping, a=b

$$y(t) = 1-(1+at)e^{-at}$$

The Z transform is:

$$\frac{Y(z)}{X(z)} = \frac{(1-A+B)z^2}{z^2 - Az + B}$$

The differential equations are:

$$\begin{aligned} y &= x - \frac{2\zeta}{\omega_n}\dot{y} - \frac{1}{\omega_n^2}\ddot{y} \\ &= x - \frac{a+b}{ab}\dot{y} - \frac{1}{ab}\ddot{y} \\ &= x - \frac{2}{a}\dot{y} - \frac{1}{a^2}\ddot{y} \end{aligned}$$

The Laplace transform is:

$$\frac{Y(s)}{X(s)} = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} = \frac{ab}{(s+a)(s+b)} = \frac{a^2}{(s+a)^2}$$

When the second-order filter is used, it is commonly to smooth data, and to wash out transients at mode change. Usually, a first-order filter suffices, and it being less expensive, is chosen over a second-order filter. Typical values for ω_n, a, and b are 0.1-2 rad/s, and ζ 0.1-0.9 (unitless). It should also be noted that two first-order filters can be chained together to form a second-order filter that is critically damped or overdamped.

When implementing a second-order filter on normalized variables, such as angles, the discontinuities require special treatment. [5]

semantics

the meaning or interpretation assigned to a group of characters in a language; compare syntax

semicircle

a measure of angle, 1 semicircle = ' rad = 180 deg; angles from physical devices may be reported in semicircles in order to compress data [5]

sensor

1 feedback device used on a CNC machine to monitor the operation and check output. Common types of sensors are force and power sensors, touch probes, and acoustic monitors. [4]

2 a device that measures, receives, or generates data [5]

3 a transducer whose input is a physical event and whose output is a quantitative measure. [7]

sensor control
control of a machine, based on sensor measurements. [7]

sensor/effector bus (SEB)
a shared medium for distributing electronic signals between the controllers, sensors, and effectors. [7]

sensor/effector standardized application (SESA)
connects the controller to the instrument(s) under control, i.e., it handles all input and output relating to controlled instruments on the manufacturing workstation. [7]

sensor feedback
the return of information obtained from sensors to maintain performance or to control a system or process. [7]

sensor hierarchy
a relationship of sensor information processing elements whereby the results of lower level processing are abstracted and used as inputs to higher level processing. [7]

SEP

1 abbr for Spherical Error Probability

2 abbr for Scenario-based Engineering Process

sequential function chart (SFC)
a graphical language that uses five simple graphical elements to provide a representative diagram of any sequential process. The basic elements are steps, transitions, and actions. A step consists of any number of actions or pieces of a program that are carried out until an action qualifier is met. The language supports alternative and parallel sequences. Because discrete manufacturing applications all run as a sequence of steps, SFC is an excellent way to logically decompose the process and automatically transfer the process into a structured program. SFC greatly enhances communication between the system architect, project manager, system integrator, and maintenance personnel. [10]

service
a domain-independent mechanism that provides fundamental capabilities enabling integration of independently developed, domain-specific functional entities. [7]

servo control

1 a feedback control system involving both hardware and software in which the output is some mechanical position, velocity, or acceleration. [4]

2 an interface that allows a controller to interface to industry standard servo drive amplifiers using +/- 10 volt motor commands and A quad B encoder feedback. The interface is capable of independently controlling the axis in closed-loop mode in either velocity or position modes using a DSP to perform the computations required for servo control. The DSP has a command set that eliminates most of the host CPU overhead associated with motion. Velocity profiling is supported. Loop compensation is accomplished by a digital Proportional

Integral Derivative (PID) filter updating the servo at 4khz. Feedback from an encoder is received at 1 million counts/second. The interface also handles motion control applications requiring opto-isolated inputs for end-of-travel limits, marker pulse, and open collector or differential quadrature encoders. [10]

servo-loop closure rate
in a closed-loop servo controller, the synchronous rate at which servo commands are generated, feedback sensors are sampled, control laws are computed and servo drive signals are generated. [7]

servo motor
a DC motor which produces a torque proportional to current. Precise positioning is achieved by linear or PWM (duty cycle) control of motor current or voltage, together with accurate monitoring of position via an external feedback device. [8]

SESA
abbr mentioned in the definition for controls standardized application.

SFC
abbr for Sequential Function Chart

SFM
abbr for Surface Feet per Minute.

shared axis
a programmable axis that is shared by two or more axes or groups of axes; e.g. two 5-axis milling heads sharing a common rail type axis that would position them. [6]

shop floor
the part of a manufacturing enterprise that has the responsibility to produce a product by performing manufacturing operations. The shop floor is the physical location through which the product passes on its way to completion. [6]

sign
common math function; returns whether the input value is positive or negative; standard notation sign x [5]

signal
part of received data that is desired; compare with noise [5]

signal-to-noise-ratio (SNR, S/N)
the ratio of magnitude of a desired signal to the magnitude of the noise received with it; standard units dB, 1; primary units 1 [5]

simultaneous axes
axes that move simultaneously at different or varying velocities as commanded by the control. These begin and end at programmed points but the simultaneous axis motion does not accommodate specific multi-axis paths in between these points. [6]

sine
common math function; standard notation
sin x, sx [5]

single instruction
an interactive condition that occurs when the controller is in the normal production mode; starts a process that enables the manual jog of machine axis for tool positioning

purposes during work piece set-up, or allows a process parameter to be temporarily varied; manual operation is also useful for activities where the workpiece remains fixture on the machine and a cursory inspection is performed via a digital probe or similar device; single instruction can also be an interactive condition that occurs when the controller is in the normal production mode. Examples of machine and process related macro execution include "Home Machine", "Auto Tool Change", "Set Point", "Null". Macro execution shall also involve the performance of a set of manual data input instructions intended to drive an axis or the machine to a predetermined position/location within the work envelope; or execute a predetermined collective set of process parameters. [7]

single-step normal production
an interactive condition that occurs when the controller is in the normal production mode; the machine/process pauses after each control plan segment and maintains that position until a human interface button is pressed. [7]

situation
current environment and surroundings [5]

slabbing cutter sharpening
a type of grinding.

slaved axis
an axis whose path follows that of another axis but in some translated or offset amount. [6]

SMM
abbr for Surface Meters per Minute.

smoothing filter
a filter to reduce quick changes of a signal by attenuating high frequencies; see first-order filter, second-order filter, wash-out filter [5]

S/N
abbr for Signal-to-Noise ratio [5]

SNR
abbr for Signal-to-Noise Ratio [5]

software library
a repository for software components. [7]

solution viewpoint
a DSSA view of a system.

SOSAS
abbr for Specification for an Open System Architecture Standard.

SOSAS-conformant
a characteristic of a NGC product indicating that it meets all applicable requirements in the NGC SOSAS. [7]

SPC
abbr for Statistical Process Control.

speed
the scalar component of velocity, measured in distance per unit time

spherical error probability (SEP)

a probability that a percentage of three-dimension measurements will lie within a sphere of given radius, with the sphere centered at truth or mean of the measurements; compare with circular error probability, radial error probability

SEP specifies test cases for measurement errors of sensors of three dimensions, such as velocity X, Y, and Z. For example, a total velocity error of 1 ft/s (50% SEP) means that for any given measurement of velocity

$$\vec{V}_i = \{V_E, V_N, V_V\}$$

then

$$p(|\vec{V}_i - \vec{V}_\mu| \leq 1) = 0.5$$

where $\vec{V}_\mu$ is the average of all $\vec{V}_i$ or $\vec{V}$ truth, depending upon context. [5]

spindle
a rotating mechanical assembly that serves as an axis for revolving tools, a workpiece, or an end effector for a machine tool. [7]

spindle lock
condition in which the spindle is stopped so that it cannot rotate

s-plane
continuous complex frequency plane; see Laplace transform; S-plane is used in control systems engineering in the design of control laws [5]

sprayer
a type of end effector.

spotface
a machining task; to machine a circular spot on the surface of a part to furnish a flat bearing surface for the head of a bolt or nut [9]

ST
abbr for Structured Text.

stall speed
the maximum speed which a stepper motor, properly ramped, can achieve without loss of synchrony. This speed is a function of motor inductance, ramp slope, applied load, drive voltage and design. [8]

stand alone control
one PLC controls all of the I/O. Stand alone is perfect for a smaller application, especially one that requires a continuous process. [10]

standardized application (SA)
an application that supports current controller practices and provides a baseline controller. [7]

standard notation
common form of a mathematical expression [5]

standard units
units commonly encountered for a particular quantity [5]

start/stop speed
the maximum step rate which can be applied to a stationary stepper motor and still retain error-free performance. Also, the rate from which a stepper motor may be instantaneously stopped without overshooting. This is a function of the

screw pitch, load, drive voltage and design, and motor. [8]

statistical process control (SPC)
the statistical techniques to analyze a process or its end product so appropriate actions can be taken to control and improve the process. [7]

steel block
a type of fixture.

step
synonym for unit step [5]

stepper motor
a type of motor featuring two or four stator coils and a toothed permanent magnet rotor, which moves through a small angle in response to a specific sequence of coil current changes. [8]

stepper motor control
an intelligent interface that allows a controller to interface to stepper motors. The interface provides precise motion control profiling requiring microstepping rates at millions of pulses per second. Each axis is controlled by an intelligent controller which provides five output signals and seven input signals, along with inputs for incremental encoders connected in either single-ended or differential form. It provides pulse, direction CW/CCW, hold, and user defined outputs. Inputs are provided for axis limits, ramp up/down, home, alarm and user defined input. All input and output is opto-isolated for noise-immunity and electrical isolation. Internal registers control the number of steps, acceleration/deceleration rates, starting rate, slew rate, and ramp-down point. [10]

step rate
the frequency of coil current changes, or input pulse train applied to a stepper motor, in pulses/second or hertz. For 200 step/revolution motors, the full step rate multiplied by .3 equals the rotation rate in RPM. [8]

stock
1 ahe raw material from which a part is cut. This material is often metal, but it can be any substance that can be cut such as plastic or even glass. [4]

2 any transformable material that is not currently involved in a transformation process. [7]

structured text (ST)
a high level block structured language that has a syntax that resembles PASCAL. ST can be used to express complex statements involving variables representing a wide range of data types. [10]

surface representation
the form presented to a user that represents surface function. [7]

synchronous
1 occurring at fixed time intervals (see asynchronous). [7]

2 refers to any tasks or electronic/computer events that occur at fixed intervals or periods and must be executed in a lock-step fashion (see asynchronous). [6]

syntax
set rules specified for a language. The relationships among characters or groups of characters independent of their meaning or the manner of their interpretation and use. [7]

system
an organized collection of people, machines, and procedures, required to accomplish a set of specific architectural or enterprise objectives. [6]

system development tool (SDT)
software that supports the analysis, development, integration, operation and maintenance of NGC-conformant elements. [7]

tachometer
a device for measuring angular velocity [5]

tactile sensor
a transducer that is sensitive to contact pressure. [7]

tangent
common math function; standard notation tan x [5]

tap
to cut an internal screw thread in an existing hole

tape
a type of fixture

tap sharpening
a type of grinding

target position
position command by the CNC. [4]

task
an activity within an operation to accomplish a goal. [7]

task execution standardized application (TESA)
responsible for routine coordination of multiple machines within a manufacturing workstation, as well as direct invocation of sensor/effector functionality as appropriate. [7]

task schedule
the highest or master schedule for the workstation. It is the combined schedules of all activities for the workstation. [7]

TCP
abbr for Tool CenterPoint.

temperature
the average kinetic energy of a body; standard symbol T; standard units °C; primary units T [11]

terminal emulation
display software that makes a computer display (CRT) and keyboard behave as a "dumb" terminal to be interfaced to a host or other computer. [6]

TESA
abbr for Task Execution Standardized Application

third party vendor
one who sells or offers for sale goods and/or services for use with equipment manufactured by other vendors. [6]

thread
a machining task; to form an external screw thread or threads on or in material. [13]

throughfeed grinding
a type of centerless grinding.

time
time relative to some defined time base; delta time as a fixed period (usually the average time for one computational cycle); standard symbol t, T; standard units s; primary units θ [5]

time constant
constant for a first-order filter determining time at which the output of the filter reaches Å0.6321% of a step input; standard symbol τ; standard units s; primary units θ [5]

tolerance
1 allowed error in measurements [5]

2 permissible variations in the dimensions of a machined part. A part that does not vary beyond these predefined limits is said to be "in tolerance." [4]

tool
1 a cutting tool used by a machine. [4]

2 an item that physically implements a predefined action. [7]

tool centerpoint (TCP)
a fixed point on the tool that serves as the reference point for the tool; all tool motion is expressed as motion of the TCP

tool change position
attribute of a tool, describing the point to which the tool should travel so that the tool changer can get to it.

tool changer
an automatic device to hold and dispense tools during part cutting. [4]

tool path
route a cutter takes when machining a part. A part program may display this path graphically on a computer monitor. [4]

tooth loading
material removal rate

top-down methodology
a process that begins with an analysis of key objectives, data or activities at a high level and works down to the lowest level of detail. [6]

torque
1 Force and the direction of the force combined to determine rotation movement. [4]

2 the moment of a force or system of forces tending to cause rotation; torque = (net force) (length of moment arm) [11]

torque compensation

used in digital control over analog movement, associated with tuning. Should the joystick move to the ìfull onî position, the system must back off the command until the engine can catch up with its RPM rate, creating a lag in response time. [21]

total quality management (TQM)
a leadership philosophy focused on customer satisfaction, achieved through empowered employees, teamwork, and continuous improvement. [6]

TQM
abbr for Total Quality Management.

transfer speed
in teaching a path through a teach pendant, the accuracy of the path generated from the path program decreases as the transfer speed is increased. This causes the teaching to be done at a slow rate, thereby hindering the testing of new algorithms. [17]

transaction
1 a mechanism for ensuring that all actions associated with one or more DMSs service call will be treated as a single unit of work. [7]

2 a single bi-directional exchange of messages from a requestor to its responding partner and from the responder to the requestor in that order. [6]

transducer
a device that converts a physical phenomenon into an electrical signal. [7]

transient-free switch
a switch with a wash-out filter so that the output contains no transients (steps) at switch time [5]

transpose
standard matrix operator; standard notation X^T [5]

travel
distance the bed moves along an axis. [4]

traverse grinding
a type of grinding.

trigger
a monitor placed on a data item that initiates some action based on access to or change in value of that item. [7]

tuning
digital control over analog movement. An example is joystick control of a process (like a hydraulic pump). As soon as the joystick crosses 10%, the valve should open (since it takes a specific amount just to start the movement). Every movement of the joystick causes the valve to open more. [21]

tuple
row of data in a (database) table. Relationally speaking, tuples are constituents of relations. [6]

turn
a machining task.

two-phase commit
process in which a commit manager sends out an intent-to-commit message to all subprocesses and those subprocesses must unanimously acknowledge consent before the commit can take place. [7]

ultrasonic machine (UM)
a machine which performs material removal via sound waves.

UM
abbr for Ultrasonic Machine.

unit
a standard quantity, such as ft or mi; synonym for dimension [5]

unit function
one of a collection of functions used as standard test cases in control systems engineering; standard notation $u_k(t)$; standard units 1; primary units 1. The primary unit functions of interest are the unit impulse, the unit step, and the unit ramp (see table in Notation section for properties) [5]

unit impulse
a function used as a standard test case in control systems engineering; a spike of "area" one at time t = 0; synonym for impulse; see unit functions; standard notation $u_0(t)$, δ (t); standard units 1; primary units 1; written $u_0(t - \tau)$ for a ramp starting at time $t = \tau$; standard engineering term [5]

unitless
no units, such as ratios; a quantity with standard units of 1; a quantity with primary units of 1; synonym for dimensionless [5]

unit ramp
a function used as a standard test case in control systems engineering; a line of slope 1 starting at zero at time t = 0; synonym for ramp; see unit functions; standard notation $u_{-2}(t)$; standard units 1; primary units 1; written $u_{-2}(t-\tau)$ for a ramp starting at time $t = \tau$ [5]

unit step
a function used as a standard test case in control systems engineering; a step from zero to one at time t = 0; synonym for step; see unit functions; standard notation $u_{-1}(t-\tau)$, u(t); standard units 1; primary units 1; written $u_{-1}(t-\tau)$ for a step at time $t = \tau$ [5]

upgradeability
the characteristic of enhancing speed, capacity or functionality of a software or hardware element within a manufacturing workstation (see scalability). [7]

user interface
the messages and informational and data entry screens displayed on a CRT to guide the system user in operation of the machine. Common elements of a CNC user interface are the graphic representations of parts, messages from the motion control subsystem, and part setup and programming option lists. [4]

user interface application

code that allows a user to interact with or manipulate specific information. [7]

user interface device
provides surface presentation functionality directly to a user. [7]

user interface support package
a COTS component that provides specific HUI functionality. [7]

VAL
an robot programming language, BASIC-like in syntax.

vector
a quantity possessing both magnitude and direction, represented by an arrow the direction of which indicates the direction of the quantity and the length of which is proportional to the magnitude; compare with scalar; standard notation $\bar{x}$ [11]

velocity
rate of change of location, either scalar or vector, often with subscripts to denote the coordinate frame; time derivative of position; time integral of acceleration; standard symbol v, V; standard units kt, ft/s; primary units L/θ [5]

verification
the methods or means used to ascertain that the SOSAS conforms to the requirements. [7]

version
a formal record used to help track an object's evolution over time; a version may be associated with a context and loaded according to that context. [7]

vice
a type of fixture; any of various tools having two jams for holding work that close, usually by a screw, lever, or cam. [13]

view
a presentation with limited perspective based on context. [7]

virtual machine
a specification of mechanisms providing a common software interface to computing platform services that isolates software modules from the computing platform. [7]

wash-out filter
a filter to smooth a transition due to change of input source, such as when changing modes; see transient-free switch

Wash-out filters are usually designed as $y = x_i - f_{HPF}(\text{Æ}x)$, where i selects a particular input x_i, $\text{Æ}x = x_p - x_r$, when i changes, and f_{HPF} is a high-pass filter, often first order. Time constants are usually about 5 s. [5]

waypoint
a discrete point along a path

ways

the rails on which a large portion of the machine, such as the bed or the spindle column, moves. [4]

welder
a type of end effector.

wheel dressing
a type of grinding.

WMSA
abbr for Workstation Management Standardized Application.

working envelope
a defined boundary representing the maximum extent or reach of a machine in all directions. [7]

working range
all reachable positions within the working envelope; the range of any variable within that the system normally operates. [7]

workpiece
the material that is being machined to produce a product or the product itself. [7]

workstation
see manufacturing workstation. [7]

workstation management standardized application (WMSA)
performs the control of the manufacturing workstation by determining and rescheduling its activities. These activities include coordination of startup, calibration, diagnosing and shutdown of the machines, and external communications. This SA provides for configuration management of the manufacturing workstation and controls its modes of operation. It monitors the controller and machines within the manufacturing workstation, predicts failures, and provides for failure mitigation. This WMSA tracks and maintains the status of all assets available within the manufacturing workstation. Furthermore, this SA provides the single point of communication with the cell. [7]

workstation model
the data necessary to control the manufacturing processes. It includes information such as parameters of the controlled equipment, quality control data, tool performance data, and part material characteristics. [7]

workstation planning standardized application (WPSA)
performs the preparatory planning required for the workstation. This planning includes determining and executing the manufacturing workstation operations plan, preparing a CCP to be executed by the TESA, and determining collision free path for a machine. This SA also incorporates modeling services for use in operations, task, and path planning and plan verification. [7]

world coordinate
a position relative to a frame of reference fixed on the manufacturing workstation. [7]

world model
the system's estimate and evaluation of the history, current state, and possible future

states of the world, including the states of the system being controlled. [7]

WPSA
abbr for Workstation Planning Standardized Application.

yaw
an angular deviation from ideal straight line motion, in which the positioning table rotates around the Z (vertical) axis as it translates along its travel axis. [8]

z axis
identifies the Cartesian axis that moves up and down (the "Z" plane). The movement of the "Z" axis controls the distance a cutting tool travels into a part. [4]

zero-crossing
a part of the deadband control. The PID instruction uses the deadband error range when a process variable moves into the deadband and crosses the setpoint. (see PID control loop) [10]

z-plane
discrete complex frequency plane; see z transform; z-plane is used in control systems engineering in the design of control laws [5]

z transform
a mathematical relationship to model a discrete function in the complex frequency domain (z-plane); compare with continuous-time equation, difference equation, differential equation, discrete-time equation, Laplace transform; Z transforms are commonly used by systems engineers to describe systems; see first-order filter, second-order filter, unit functions for examples [5]

E.3 Notation

The table below lists units used in this dictionary. Many entries list standard units for quantities. These are the units most likely to be used by domain experts. It is not intended to serve as the final word on units. Some engineers will prefer other units.

This dictionary lists primary units for many symbols. All units can be broken down into combinations of four primary units: (1) mass (M), length (2) length (L), (3) time (θ), and (4) temperature (T). Unitless quantities have primary units of 1. The primary units are useful for checking consistency of equations.

symbol	primary units	meaning	measures
g	L/θ	gravitational units	acceleration
deg	1	degrees	angle
DMS	1	degrees, minutes, seconds	angle
rad	1	radians	angle
semicircle	1	semicircles	angle
rpm	1θ	revolutions per minute	angle
lbf	ML/θ'	pound force	force
kip	ML/θ'	thousand pounds	force
Hz	1/θ	hertz	frequency
ft	L	feet	length
in	L	inch	length
lbm	M	pound mass	mass
slg	M	slug	mass
in Hg	ML/θ'L	inches mercury	pressure
mbar	ML/θ'L	millibar	pressure
psi	ML/θ'L	pounds per square inch	pressure
%	1	percent	ratio
dB	1	decibel	ratio of power
°C	T	degrees Celsius	temperature
hr	θ	hour	time
s	θ	second	time

The tables below list the symbols used in this dictionary.

nomenclature	standard units	primary units	meaning
A	ft/s',g	L/θ'	acceleration
a	ft/s',g	L/θ'	acceleration
a	1	1	Euler parameters
ac	ft/s',g	L/θ'	Coriolis acceleration

α	rad/s'	1/θ'	angular acceleration
b	1	1	Euler parameters
c	1	1	Euler parameters
e_{1234}	1	1	Euler parameters
fs	Hz	1/θ	sampling frequency
g	ft/s'	L/θ'	gravitational acceleration
g	lbf, kip	ML/θ'	gravity
l	1	1	identity
J	ft/s'	L/θ'	jerk
j	ft/s'	L/θ'	jerk
ω	rad/s, rpm	1/θ	angular velocity
P	ft, mi	L	position
p	rad/s, deg/s	1/θ	pitch rate
p	ft, mi	L	position
q	rad/s, deg/s	1/θ	roll rate
r	ft, mi	L	range (mg)
r	rad/s, deg/s	1/θ	yaw rate
s	1	1	Euler parameters
T	s	θ	period
T	°C	T	temperature
T	s	θ	time
t	s	θ	time
θ	rad, deg	1	angular position
ΔV	ft/s, kt	L/θ	speed error
V	kt, ft/s	L/θ	velocity
v	kt, ft/s	L/θ	velocity
x	ft, mi	L	position
y	ft, mi	L	position
z	ft, mi	L	position

Greek symbol	standard units	primary units	meaning
ω	rad/s, Hz	1/θ	damped frequency
ω_c	rad/s, Hz	1/θ	cutoff frequency
ω_n	rad/s, Hz	1/θ	natural frequency
Φ	rad, deg	1	roll
ϕ	rad, deg	1	roll
ϕ	rad/s, deg/s	1/θ	roll rate
Ψ	rad, deg	1	yaw
ψ	rad, deg	1	yaw
ψ	rad/s, deg/s	1	yaw rate
τ	s	θ	time constant
θ	rad, deg	1	pitch
θ	rad, deg	1	pitch

θ	rad/s, deg/s	$1/\theta$	pitch rate
ζ	1	1	damping ratio

notation	meaning
$\delta(t)$	unit impulse
_x dt	integral
A	matrix
$\lvert A \rvert$	determinant
arctan x	arctangent
atan x	arctangent
[aij]	matrix
cos x	cosine
cx	cosine
Dx	derivative
det [1]	determinant
dx/dt	derivative
ex	exponential
exp x	exponential
ln x	logarithm
log x	logarithm
$\log_b x$	logarithm
$\max(x_1,..., x_n)$	maximum
$\min(x_1,..., x_n)$	minimum
sgn x	sign
sin x	sine
sx	sine
$sx + x(0^+)$	derivative
tan x	tangent
$\tan^{-1}x$	arctangent
$\tan^{-1}(y,x)$	arctangent
u(t)	unit step
$u_{-1}(t)$	unit step
$u_{-2}(t)$	unit ramp
$u_k(t)$	unit functions
$u_0(t)$	unit impulse
X^{-1}	inverse
X^T	transpose
x	scalar
x	derivative
x(1)	derivative
x^{-1}	inverse
$\lvert x \rvert$	absolute value
x'	derivative

$\bar{x}$	vector

subscript	meaning	example
-2	ramp	unit ramp ($u_{-2}(t)$
-1	step	unit step ($u_{-1}(t)$, $u(t)$)
0	impulse	unit impulse ($u_0(t)$, $\delta(t)$)
b	base	logarithm (ln x, log x, $\log_b x$)
c	Coriolis	Coriolis acceleration (a_c)
c	cutoff	cutoff frequency (ω_c)
k	order	unit functions ($u_k(t)$)
max	maximum	limiter
min	minimum	limiter
n	cycle n	first-order, integrator, second-order filter
n	natural	natural frequency (ω_n)
s	sampling	sampling frequency (f_s)

E.4 Category Index

computer, communication

Arcnet
CANBus
dedicated channel
PAMUX bus
sensor/effector bus

computer science

algorithm
analog input/output
ASCII
autonomous control
batch execution
binary coded decimal
checkpointing
commit
computer cycle
computing platform
concurrency
database
database management system
data bus
data structure
device driver
discrete I/O
distributed database
dynamic binding
exception handling
graphical user interface
hierarchical control
human user interface
information base
information base application
interrupt
knowledge base
knowledge base management system
latency
locking
logging
operating system
protocol
schema
semantics
syntax
trigger
user interface

control law, categories of

linear force/torque
linear position/velocity
rotational force/torque
rotation position/velocity

controller, concepts regarding

block
block cycle time
canned cycle
controls standardized application
conversational part programming
data starvation
diagnostic
drawing interchange file
feedback
forward kinematics
interpolation
interpolation block
inverse kinematics
leadscrew mapping
multiaxis mode
offset
overshoot
part program
part programming
quadrature
roughing pass
servo-loop closure rate
stepper motor control
target position
torque compensation
transfer speed
waypoint

controller, parts of

control law
interpolater
kinematic model
machine model
path planner
servo control

control, types of

adaptive control
binary cutter location
closed loop
computer numerical control
continuous path control
direct numerical control
distributed control
endpoint control
motion control
numerical control
open loop
point-to-point control
sensor control
stand alone control

engineering & mathematics

absolute
acceleration
accelerometer
accuracy
angle
angular acceleration
angular position
angular velocity
aperiodic
asynchronous
axis
bias
coefficient of friction
coordinate
correlation
delta
desired

deviation
dimension
dimensionless
discrete time
distance
domain
drift
error
euler angles
euler parameters
function
granularity
gravitational acceleration
impulse
jerk
metrology
model
noise
non-uniform rational b-spline
orientation
period
periodic
position
precision
radius
ramp
rate
real time
reference
relative
resolution
root mean square
root sum square
sampling frequency
scalar
semicircle
signal
signal-to-noise ratio
speed
s-plane
standard notation
standard units
step
synchronous
tachometer
temperature
time
tolerance
torque
transducer
unit
unit impulse
unitless
unit ramp
unit step
vector
velocity
z-axis
z-plane

english
abort
activity
clockwise
counterclockwise
operational
situation

error
abbe error
dynamic error
dynamic following error
following error
geometric error
kinematic error

filter, concepts regarding
cutoff frequency
damped frequency
damping ratio
hysteresis
natural frequency
time constant
transient-free switch

filter, types of
averaging filter
band-pass filter
complementary filter
first-order filter
high-pass filter
integrator
kalman filter
limiter
low-pass filter
rate limiter
second-order filter
smoothing filter
wash-out filter

fixture, types of
bolt
chuck
clamp
jig
parallels
steel block
tape
vice

functions, types of
absolute value
arctangent
continuous-time equation
cosine
derivative
determinant
difference equation
differential equation
discrete-time equation
exponential
extrapolate
identity
interpolate
inverse
Laplace transform
logarithm
magnitude
matrix
maximum
minimum
normalizer

sign
sine
tangent
transpose
z transform

grinding, types of

centerless grinding
endfeed grinding
end mill sharpening
face mill cutter sharpening
form cutter sharpening
form grinding
infeed grinding
internal grinding
plain mill sharpening
plunge grinding
reamer sharpening
slabbing cutter sharpening
tap sharpening
throughfeed grinding
traverse grinding
wheel dressing

ladder logic

ladder diagram
ladder logic execution
ladder logic programming
programmable logic controller

machine, types of

abrasive waterjet machine
bed mill
saw
coordinate measuring machine
electrochemical machine
electrodischarge machine
electron beam machine
knee mill machine
lathe
manual machine
milling machine
moving column machine
Programmable Universal Machine for Assembly
punch
saw
ultrasonic machine
universal mill machine
vertical mill machine

manufacturing, concepts in

2.5 D
aiding
application-specific integrated circuit
approach vector
availability
backlash
chatter
circular error probability
close dancing
computer-aided manufacturing
configuration
consumable
control plan
emergency stop
encoder feedback
external communication
external interface
feature-based definition
feedrate
feedrate override
function block diagram
geometry
goal
I/O interfaces
lead
lead error
lead through teach
look ahead
materials resource planning
modal
network interfaces
NGIS-NC Inspection Project
nut compliance
one-shot
operator interface
orthogonality
part transformation
part zero
pitch
plan
planning
prismatic
process
radial error probability
rapid traverse
reference frame
repeatability
revolutions per minute
roll
safety
safety envelope
safety interlock
scalability
sensor feedback
spherical error probability
spindle lock
statistical process control
tool path
tooth loading
travel
tuning
upgradeability
virtual machine
working envelope
working range
world coordinate
world model
yaw

manufacturing, languages in

dimensional measurement interface specification
EXPRESS
FLEXIS
IEC1131-3
instruction list

interactive graphics
exchange specification
Karel
ladder logic
M & G codes
man-machine language
neutral manufacturing
language
Product Data Exchange
using STEP
programmable application
logic
RS-274-D
sequential function chart
structured text
VAL

manufacturing, levels of
cell
center
enterprise
factory
job shop

manufacturing, objects in
actuator
asset
automated ground vehicle
bed
chip
controller
conventional machine tool
coolant
coolant mechanism
coordinated joint
drive
effector
electrical cabinet
enclosure
encoder
end effector
end user
environment
face plate
feature
fixture
gripper
Hall effect sensor
hand crank
home switch
instrumentation
iron
joint
leadscrew
limit switch
link
lubricant
machinist
manipulator
manufacturing workstation
mold
motor
operator
operator console
optical encoder
original equipment
manufacturer
pallet
pallet shuttle
part
remote jog pendant
resolver
resource
retrofitter
robot
sensor
spindle
sprayer
stock
table
tactile sensor
tool changer
ways
welder
workpiece
workstation

motor, concepts about
holding torque
microstepping
phase current
phase sequence
ramp
resonance
stall speed
start/stop speed
step rate

motor, types of
AC induction motor
brushless motor
DC motor
servo motor
stepper motor

PID
deadband
PID control loop
scaling
zero-crossing

saw, types of
band saw
jig saw
reciprocating saw
rotary saw
saber saw

software engineering, concepts in
actor
agent
application program
architecture description
language
application programming
interface
attribute
class
commercial off-the-shelf
package

component
computer-aided design
configuration management
domain-independent
application
dynamically linked library
field replaceable unit
holon
input/output artifacts
instance
installation
integration architecture
interchangability
interface
interoperability
life cycle costs
modular
object
object module
object-oriented
object-oriented programming
object-oriented structures
open system
open system architecture
open system interconnection
open technology
polymorphism
portability
product life cycle
reusable software
reverse engineering
service
software library
verification
version
view
viewpoint

SOSAS, concepts in
conformance
conformance class
element
mode
machine under control
multiprocessor control

open applications
primitive function
product definition data
product definition data
interchange
real-time operation
recovery
replication
sensor/effector standardized
application
sensor hierarchy
single instruction
single-step normal production
standardized application
surface representation
system development tool
task execution standardized
application
task schedule
transaction
two-phase commit
user interface application
user interface device
user interface support package
workstation management
standardized application
workstation model
workstation planning
standardized application

task, types of
area clear milling
blue
bore
break chip
broach
calibration
climb milling
conventional milling
counterbore
countersink
drag
drill
dwell
etch

failure recovery
fault detection/isolation
fault management
feed
finish
grind
initialization
integrate/configure
jog
mark
mill
odd pockets with islands
milling
orient
plane
plate
probing
ream
saw
spotface
tap
thread
turn

tool, concepts regarding
cutter compensation
cutter compensation plane
cutter compensation radius
offset
flute
gauge vector
home position
in-process gauging
length offset
optimal feedrate
optimal spindle speed
tool centerpoint
tool change position

tool, types of
drill bit
end mill
flat end mill
flexible tap

grind belt
grind wheel
rigid tap
saw belt
saw blade

viewpoint, types of
application viewpoint
architecture viewpoint
communications viewpoint
component viewpoint
design viewpoint
executive services viewpoint
physical resource viewpoint
solution viewpoint

other
command
context
look and feel
override
planner based
power-up
reliability
requirement

recently added
abstract data type
American National
Standards Institute
artificial intelligence
assembly
cache
canonical
class hierarchy
data
data dictionary
European Strategic
Programme for
Research and
Development of
Information
Technology
expert system
fabrication
geometric modeling
applications
program
integrated computed aided
manufacturing
intelligent machining
workstation
inventory
International Organization
for
Standardization
Industrial Real-Time
Operating System
Nucleus
just-in-time
low rate initial production
maintainability
maintenance
manufacturing
Manufacturing Automation
Protocol
Open System
Interconnection
paperless factory
part geometry
part number
platform
predicate
process plan
quality
query
query language
real-time function
referential integrity
sculpted surface
shared axis
shop floor
simultaneous axes
slaved axis
system
terminal emulation
third party vendor
top-down methodology
total quality management
tuple

APPENDIX F — ARCHITECTURE DESCRIPTION LANGUAGE (ADL)

The architecture description language (ADL) describes rules for composing application architectures. The language constrains responsibilities, components, and interfaces. It is a constraint-based rather than a computational language. Categories of ADL rules include:

- applicable to architectural concepts, such as messages;
- applicable to domain concepts, such as mode;
- applicable to responsibilities, such as tool verification;
- applicable to components, such as motion control.

In a general sense, the ADL may contain a rule stating that the application architecture is not complete until all resources required by components are supplied by other components in the application. However, a great deal of domain knowledge is embodied in the ADL. More domain knowledge is necessary with rules that require the selection of particular primitive components, such as the technical selection of forward and inverse kinematics.

Future efforts in language research are focusing on communication concepts that will support the dynamic construction of component-based architectures. One such language is the Adaptive Semantic Language, described below.

THE ADAPTIVE SEMANTIC LANGUAGE

Introduction

As the complexity of software systems has increased, a number of formal models have been proposed to address the decomposition of large software systems into sets of reusable components. The Object Oriented (OO) paradigm is only the latest in a collection of models designed to divide the responsibilities of systems into separate objects which can provide a service to the rest of the system through a standard set of messages. The strength of the OO paradigm is that the methods and data needed to fulfill a message request can be completely held within an object, and therefore, hidden from the rest of the system. Thus, the high-level system designer needs only to understand the message-passing aspects of the object, not how the object internally deals with the requested service. At a higher level of abstraction, the complete set of objects making up a sub-component of a very large system can be abstracted to a single unit, an agent, which itself can provide a higher level of services to the overall system through a standard set of messages.

Both the object and the agent model have the potential of greatly reducing the amount of new software required to build a new system by allowing a standard method for the reuse of existing components. However, neither of these models addresses the complexity of design at the high level. Without clear documentation of a component's services and message interface, the component becomes unusable, but even with good documentation the selection of components requires a complex search over a database of standard messages and services. Further, new components cannot be considered for use by the system designer until their services and message requirements are added to this database. If this database exists within a standards document the process of adding new components can be very slow and not easily automated.

A better solution than a static database which must be modified for every change and addition to the set of available components would be a method by which the components themselves could contain semantic knowledge about their abilities and interface requirements. A system could then be designed by providing a set of system-construction agents with an ability to query the components, and then, based on the set of selected components tailor the system's internal interfaces to allow these components to carry out their required communication. However, such a method would require that both the abilities and interface information of a component be held in a common semantic representation which can be communicated between components using a semantic language.

Since the required semantic language would require an adaptiveness similar to human language while maintaining the non-ambiguity of a formal language, the logical source of such a language should be found by first formally defining the syntactic and semantic features of a human language required, and then, attempting to formally define the resulting features into a knowledge representation model which will allow the programmatic storage and transmission of the model's syntactic and semantic components. To accomplish this, both the requirements of the adaptive semantic language and a model for the semantic representation must be determined. Further, a testing platform must be constructed to ensure the resulting theory can be proven to be achievable.

This appendix will provide the theoretical support for an adaptive semantic language and the n-Towers model for the semantic representation which supports its use for communication from programmatic agent to programmatic agent and from programmatic agent to human. The method by which this model can interface to sub-language elements of the programmatic agents will also be addressed.

The n-Towers Model

The n-Towers model's purpose is to provide a method for computers to understand HL and to communicate with each other through a connection at the Adaptive Semantic Language (ASL) layer. The model has been based largely on the principles of Government and Binding (GB) linguistic theory. Using this foundation, the model attempts to incorporate the best features of three current semantic theories; 1) Situation Semantics, 2) Possible Worlds Semantics, and 3) Conceptual Semantics. Since no existing Knowledge Representation (KR) was found to be adequate, the model is based on the completely new KR defined as Situational Dependencies.

Figure 1 presents one tower of the of n-Towers Model. In this model there are five levels (or layers) between the Human Language (HL) Surface Structure and the Symbol Grounding Plane. The two lowest levels (Meta-Semantics and Situational Dependencies) are viewed as below linguistic knowledge. The model's breaking of HL syntactic processing into two layers (the surface and deep structure) should not be viewed as a total embracing of GB theory. The Deep Structure layer may best be viewed as a place holder for any HL information which cannot be captured at the surface layer. Thus, the contents of these two layers could depend on the requirements of any linguistic theory chosen to support the language processing above the ASL layer.

The interface between the HL layers and the lower three layers is the ASL layer. This layer provides a non-ambiguous language syntax which attempts to capture the ability to express the full semantic richness of the HL above it. Based on a GB model, an expression in ASL best maps to a logical form. The three lowest layers, as well as, the ASL and Symbol Grounding Plane will be discussed in more detail in the next four sections.

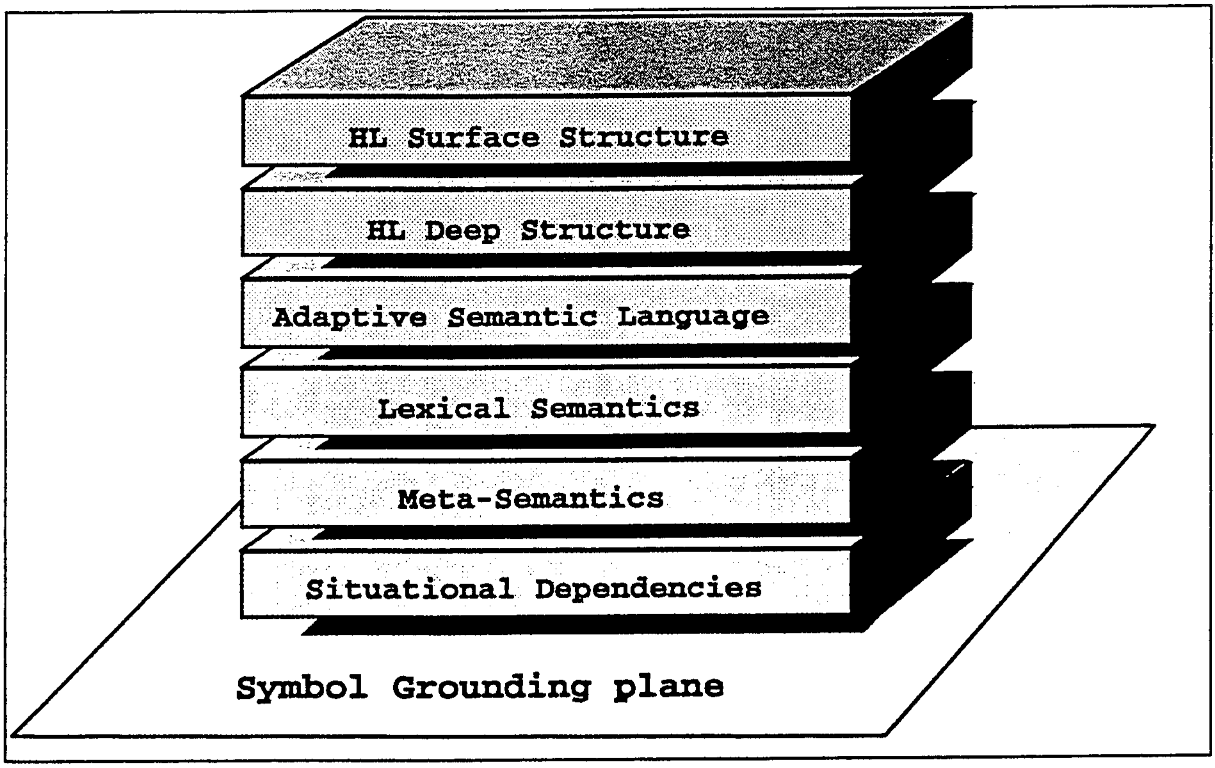


Figure F-1. A Tower of the n-Towers Model

The actual number of towers needed by a system will depend on the application being considered. For the communication of two agents within a generic system, each agent would normally need only one tower. If these agents were collocated within a discrete portion of the system, even some portions of these towers could be shared. An example of the way in which a group of agents controlling various portions of a factory floor would communicate is given in Figure 2. In this example, only the one agent whose duties include the system's man-machine interface would need the top two HL levels of the n-Towers model.

Theoretical Basis for an Adaptive Semantic Language

As stated above, the purpose of the ASL is to provide a non-ambiguous interface between the HL layers and the three lowest layers of the n-Towers model. As its name implies, ASL is viewed as semantic language. This means it theoretically falls between the syntactic D-structure and the conceptual structure represented by the three lower layers of the n-Towers model. This positioning in the HL model requires that the resolution of at least syntactic and semantic ambiguity be done in the top two layers of the model. This does not force a restriction in the expressive power of the n-Towers model as a whole since there is nothing to preclude the two top layers from generating any number of different ASL sentences for a given HL sentence.

Before the concepts of the ASL can be fully defined, a formal definition of the syntax of both a standard formal language and HL must discussed.

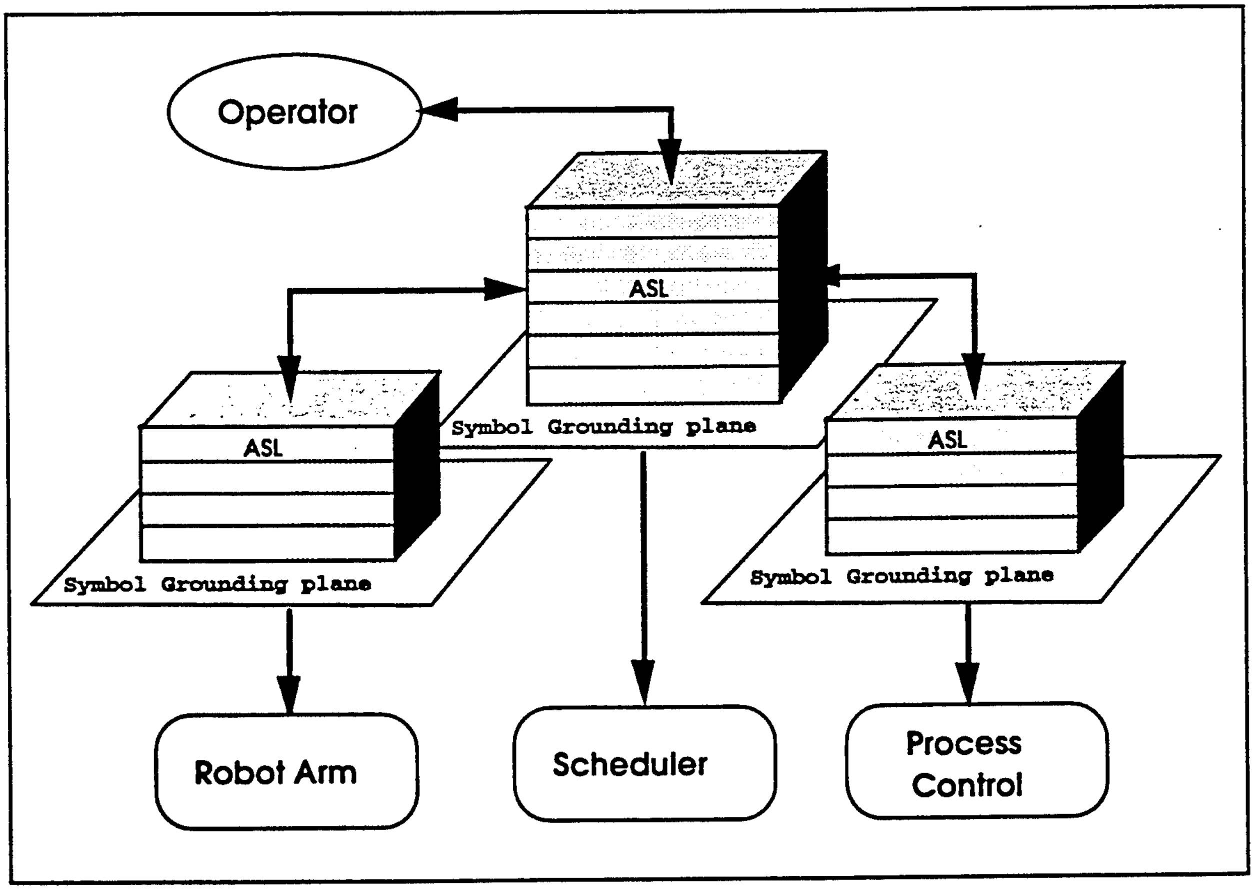


Figure F-2. A Factory Application of the n-Towers Model

The Standard Formal Language Model

Given a grammar G = (V, T, P, S), a standard formal language L(G) is the set:

$$\{w \mid w \in T^{*} \text{ and } S \overset{*}{\underset{G}{\Rightarrow}} w\}$$

This definition assumes that both the set of terminal symbols {t | t Œ T} and the set of variables {v | v Œ V} are finite sets of pre-defined symbols from which a potentially infinite set of sentences can be constructed. The actual number of well formed sentences which can be generated by a non-recursive set of production rules {p | p Œ P} depends on the total number of elements in the sets P, T and V. Any form of recursion over the set P will cause the number of well formed sentences to increase to infinity.

While the addition of recursion to the set P allows the standard formal language L(G) to produce an infinite number of sentences, the fact that such a L(G) depends on an increasing sentence length to increase its expressional power presents a problem in attempting to use any semantic model based on L(G). As the length of the sentence grows, the overall meaning of the sentence will begin to become computationally clouded due to the exploding size of the semantic representation. This problem becomes especially acute if the recursion is indirect, due to the lack of predictability about which portion of the representation will grow. Further, since only the syntactic meaning of sentences can be dynamically varied, the rate of increase in the expressional power of L(G) will naturally decay as the sentence length increases due to reduction in the contribution of the more powerful lexical elements of semantics.

The Human Language Model

In contrast to the grammar which generates a standard formal language, a human language HL(HG) can be defined using elements of GB theory as:

$$HG = (X_V, G_W, P_X, C, CP).$$

Here the set of variables {v | v Œ V} has been replaced with the set of word categories {x | x, x', x" Œ X_V} which can be plugged into the X-bar rules. The set of terminal symbols {t | t Œ T} has been replaced with a word grammar, G_W, which using its own grammar can generate the word level elements of the top level sentence grammar. In contrast to a formal language, a set of production rules {p | p Œ P} is not sufficient to capture the nature of the GB rules which control syntax, so a set P_X and C are provided as a replacement. The set P_X consists of the three X-bar rules from GB theory. The set C consists of all rules generated by the other five modules of the GB theory (i.e., Binding Theory, Bounding Theory, Case Theory, Control Theory and Theta Theory). The start symbol S has been replaced by the CP symbol which is the top phrase marker of a sentence in GB theory.

The word generation grammar can be defined as G_W = (M, LU, P, R) where M is a set of morphological variables, LU is a set of stored lexical units {u | u Œ LU}, MP is a set morphology rules, and R is the set of word roots {r | r Œ R} which can grow over time. Thus, W is the set of well formed words generated from G_W by:

$$\{w \mid w \in LU^* \text{ and } R \overset{*}{\underset{WG}{\Rightarrow}} w\}$$

As can be seen from these definitions of HL(HG) and G_W, human language allows for both the growth of the word set W (the resulting set of all possible word generations from all possible roots) and the recursive application of X-bar production rules within the set P_X under the control of C. In HL(HG), the extent of recursion of the X-bar rules within the set P is somewhat controlled by the set C which places some constraints on the application of the X-bar rules. Further, the power of the set W to grow and the resulting increase in the lexical semantics that results, reduces the need to draw additional semantic value from a more complex syntactic form. The result is a dynamic language which can adapt to changing semantic requirements without increasing the mean-length-of-utterance required for communication. In a massively parallel system such as our brains, the resulting optimization toward increased search complexity does not present a processing problem; however, in a modern computer such a complex grammar within a grammar construction for a language does present a substantial processing problem.

The Adaptive Semantic Language Model

Clearly the best language model for the syntax of the ASL, based on its role as a computer to computer communication language, would be one which is as expressive as the human language model but as algorithmically efficient as the standard formal language model. Of course, such a model cannot be constructed since these two requirements are nearly mutually exclusive, but there is a large middle ground between the two models in which to select a compromise solution. To determine exactly where the ASL model should fit between these two models, the syntactic and lexical requirements of the ASL are discussed. Based on these requirements an ASL model is proposed.

ASL Syntax Requirements

One of the difficulties in the processing of a HL is the complexity of the syntax necessary to provide the richness of expressions required for the socialization of humans. Since the surface structure form of a sentence needs to take in account the social environment of the utterance, a number of pragmatically different forms of a semantically identical D-structure form can be generated by a set of rules provided for this expressiveness. Further, since the human brain uses a massively parallel processor which can store a great deal of information in temporary registers and process the stored utterances almost simultaneously with the completion of input, the S-structure ordering of phrases presents little or no problem to the parsing of the utterances, as long as, this ordering falls within the range of acceptable possibilities.

Since the top two levels of the n-Towers model can handle the pragmatic aspects of a HL, the language selected for the ASL does not need a number of surface variations of the D-structure of an utterance. Thus, a model based on the logical forms of HL expressions is sufficient to express almost all of the semantic meaning needed. The only surface marker that needs to be preserved is one which differentiates between a declarative and interrogative utterance. However, a simple pre-punctuation of the utterance will convey the information without the need of the redundant word order changes used in such languages as English.

Based on ASL's proposed position below the D-structure, the need for the ASL to allow any recursion over its set of syntactic production rules can be eliminated if the lexical forms used have sufficient semantic power. In fact, based on the reduced scope of the semantic transmission inherent in a formal language, a major portion of the structure of the language can be contained within the lexicon. This assumes that the concept of external and internal q-roles is eliminated within the lexical entries, and that the potential difficulty in parsing a SVO or OVS word order is addressed.

Since the formal language of predicate calculus is very similar to a VSO or VOS language and presents little problem to a trained user, the mapping of the lexical forms to a predicate syntax would provide both a powerful level of expression and a very simple language to parse. Since the parser would be provided with the lexical unit's identity first (as the predicate label), it could easily look up the unit in the lexicon to determine its expected syntax and the number of thematic roles and start mapping the semantics of the contents as the terms of the predicate are read in. If the complete syntactic form of a predicate in predicate calculus is maintained, recursion within the s-grids could be easily recognized and interpreted.

The ASL's Lexical Requirements

During the natural evolution of a human language, the language's lexicon constantly changes. The phonology, morphology, syntactic relation and meaning of a lexical unit will naturally change over time and lexical units will be added or lost to the corpus of the language as the need to express new sets of ideas is created. Attempts to stop this evolution have proven fruitless since human language is closely tied to human society, and social change is a natural force not easily controlled. As a result of this natural change, human language contains methods of controlling the effect of a changing lexicon which can be exploited by an n-Towers model to allow the language to change to support the expression of new sets of ideas. Since the major forces for change in a human language reside outside the language, these control methods can be exploited without fear of encountering an internal force within the n-Towers model which would cause an uncontrollable change to the ASL.

Research in GB theory has provided evidence that all human languages use a concept of q-grids within the their lexicon to govern sentence construction and that the types of lexical categories containing the q-grids are remarkably uniform across all human languages. However, the number of lexical categories which have associated q-grids in HL is somewhat controversial. In addition, the amount of information stored in the q-grids varies greatly within different views of GB theory. Since there is some debate as to what amount, if any, of syntactic information is stored in the q-grid, the term s-grid (author's convention) will be used to refer to syntactic information stored in the lexicon. In addition a third grid, the a-grid (again author's convention) will be added to the lexicon to support the connection to the Symbol Grounding Plane.

In summary, the supporting lexicon will need to provide 1) the internal syntactic realization for all lexical units which can vary their internal syntactic realization, 2) a method of describing the semantic linking for lexical units of governing categories and 3) a method of firing some action as a result of the occurrence of a word in the ASL sentence. This will be done using s-grids, q-grids and a-grids, respectively. Since our goal is a formal semantic language which by nature will limit the number of ways the same semantic meaning can be expressed within the ASL syntax, the storage of a good portion of the language's syntax within the lexicon should be achievable, assuming that the number of lexical categories is reduced to a small set.

Lexical Categories

Human languages vary in the number of lexical categories used to logically divide the lexicon. For a majority of human languages, a set of 12 lexical categories will be sufficient to separate the language's lexicon. These are:

1) adjective
2) adverb
3) auxiliary verb
4) conjunctions
5) determiner
6) infinitive
7) main verb
8) noun
9) participle
10) particle
11) preposition
12) pronoun

Although this richness in the number of categories forces a complex syntax, the disadvantage of this complexity is far outweighed by the extreme expressive power it provides. Our task is reducing the number of categories which must exist at the semantic level while maintaining all of the expressive power of HL.

Due to their semantic importance, neither the noun nor preposition category can be eliminated or reduced in any real fashion. However, in almost all human languages, prepositions are notorious for having broad overlapping meanings. This fact can be remedied in the ASL since prepositions are a closed category, thus, allowing a more careful definition which is both narrow and non-overlapping.

The first reduction which can be made is in the number of categories required to express the overall concept of a verb. The major reason why this concept is spread over the auxiliary verb, main verb, infinitive and participle is to allow for the expression of mood, aspect, tense and voice. If given a method in which to encode a verb's usage within the lexical form, the ASL only needs to express externally the concept in the present active indicative without aspect and these categories can be reduced to a single 'verb' category.

Adverbs, in human language, normally qualify the meaning of the main verb. This qualification is normally in relative terms and often difficult to interpret in any concrete way. For these reasons, it seems best to eliminate adverbs from the list of the ASL's lexical categories. This may again force some sort of encoding of verb category or addition of verbs like 'run quickly' unless a method can be found to make adverbs act as more well behaved members of the adjective category.

Particles are sort of the waste bucket of human language lexical categories. Any concept which cannot be fully expressed by another category or any difficulty in syntactic construction may result in the use of a particle. In the ASL, the need for particle can be eliminated by a careful definition of the syntax and other lexical categories.

The discussion of pronouns is slightly more complex since their number and use varies greatly from human language to human language. For the sake of our discussion, we will take a 'standard' view based on a number of Indo-European languages. Based on this view, the personal, reflexive and relative pronouns will be treated as pronouns and other types will be treated as either adjectives or determiners.

In most human languages, a personal pronoun is normally coreferential with either a referent which has already been described in the discourse by a Noun Phrase (NP) or has been associated to a grounded symbol by some non-language method. Since we are dealing with a formal language, we can dismiss the non-language symbol grounding condition as beyond our realm of interest and assume that for our purposes all personal pronouns will be coreferential with a referent which has been described by a NP. The use of personal pronouns as external referents is driven mostly by the need to reduce the amount of redundant information being transmitted in HL. In a formal language, this need is less important and use of the personal pronoun can be eliminated.

The reflexive pronoun can be eliminated from a formal language for the same reason as the personal pronoun. The need for the ASL to support relative clauses, and thus, relative pronouns can be eliminated, but only if some type of conjunction with a temporal meaning is supplied at the syntactic level.

As with pronouns, adjectives serve to shorten the utterance by allowing a qualification of the NP to be directly stated within the NP. Therefore, adjectives can also be eliminated from the ASL without any loss of semantic expressive power as long as nouns with the same semantic value are provided.

Determiners on the other hand serve to quantify, not qualify, their associated NP. As such, they cannot be fully eliminated from the ASL without losing semantic value. Since their semantic value will need to be carefully defined for a proper semantic interpretation of the whole utterance, they will need to be a closed category in the ASL. This follows from human language where they are a closed category for the same reason.

The semantic value of conjunctions also cannot be eliminated from the ASL without losing semantic value. However, since the number of conjunctions needed at both the lexical and syntactic level is very small, they can be formalized in the ASL syntax.

Based on all the justifications given, the number of open categories needed in the ASL can be reduced to two; 1) nouns and 2) verbs. The number of closed categories required can be reduced to three; 1) determiners, 2) conjunctions, and 3) prepositions. Of these, verbs and prepositions will require q-grids. Only verbs and nouns will require s-grids.

Lexical Entries

Since the purpose of the lexicon is to support adaptive change within the ASL, the number of entries in the lexicon must be allowed to change over time. This will require some set of components in the ASL model to have meta-knowledge about how to interpret the syntactic and semantic information stored in the lexicon. The meta-knowledge about syntax can be relatively straightforward since a category will either have a built-in syntactic realization or a s-grid which can be directly read for this information. The meta-knowledge about the semantic model of the ASL will need to be a little more complex since the interpreting component will need to do more than simply read an entry's q-grid. This component will need to understand both, 1) the number of possible thematic roles and how these roles map into a semantic understanding of the language and 2) how to read a semantic description of a verb or preposition to complete the semantic understanding.

For a verb entry, the q-grid will need to consist of the required thematic elements and their thematic roles. To aid in parsing, no optional q-roles should be permitted. To allow some optional forms of a verb, the ASL should allow two or more entries for the same verb name in the lexicon. The s-grid for the verb will provide the phrase level constituents (either NP or PP) for each actor in the q-grid. For a preposition entry, the q-grid will need to consist of the thematic role of its actor. Since only NP's can fill this role, no s-grid will be required.

For a noun entry, the s-grid will define the other elements which can be used with the noun to form a NP. These elements will include determiners, conjunctions, prepositional phrases, and other NP's.

Determiners and conjunctions will be atomic elements within the ASL, and therefore, will have neither a s-grid or q-grid. Since determiners and conjunctions are both closed categories and are tightly defined, their storage in the lexicon is optional.

The ASL Model

Based on the requirements provided above, the grammar rules and lexical syntax can be now be defined for the ASL model. This is done in the next two sections. A set of examples is then given to show how the ASL could be used.

Grammar Level Syntax

Using a predicate syntax where a verb name is the predicate label and the verb's governed q-roles are the terms, the syntactic production rules of the ASL grammar can be reduced to the following non-recursive set of rules:

```
S Æ SI | ?(SI)
SI Æ VP | NOT VP
VP Æ V | V / V | V Æ V | V > V
```

where:
V is a verb predicate,
VP is a verb phrase,
SI is an declarative sentence, and
?(SI) is interrogative sentence.

This language allows a syntactic level negation (NOT) of the sentence. In addition, the following syntactic level conjunctions are allowed across two V's:

1) a logical disjunction (/)
2) an if-then relationship (Æ)
3) a temporal ordering (>)

A logical conjunction (AND) is not provided at the syntactic level since there is no semantic value added to two simple sentences being ANDed together to make a complex sentence over the same two simple sentences being stated as totally separate statements.

A more powerful recursive language could also be defined as:

S Æ SI | ?(SI)
SI Æ VP
VP Æ V | V / VP | V Æ VP | V > VP |NOT (VP)

However, such a language would present a more difficult semantic evaluation and may be overly expressive for the average domain.

Lexical Unit Level Syntax

As stated above, only verb phrases will appear at the sentence level. The syntax of the verb phase will be controlled by the grammar and the lexical syntax of the verb. The syntax of a verb will be:

verb-name ($s_0, s_1 \ldots s_N$)

where the number of s-roles (s_x) is determined by the verb's s-grid. These s-roles can either be filled by a NP or a PP. The syntax of these will be:

noun-name ($s_0, s_1 \ldots s_N)_0$ [j *noun-name* ($s_0, s_1 \ldots s_N)_1$
. . . j *noun-name* ($s_0, s_1 \ldots s_N)_N$]
preposition-name (NP)

where the symbol, j, stands for either a logical AND (Ÿ) or OR (/) operator and the number of s-roles (s_x) for the noun element within the NP is determined by the noun's s-grid. The s_0 position of this s-grid can only be filled by a determiner. The s_1 position will be reserved for NP's acting as adjectives. The remaining s-roles are optional and can be either filled by a NP or a PP.

Example Expressions

Based on the above design, the following sentence examples for a NGC type domain are given:

English: The lathe rounds the part with a cutting tool.
ASL: rounds(lathe(the, ø), part(the, ø), with(tool(a, cutting)))

English: Pick up the blue part on the green table with the stationary robot arm.
ASL: pick-up (part(the, blue, ø, on(table(the, green, ø, ø))),
with(robot-arm(the, stationary(ø))))

English: Get the cutting tool from the tool caddie, and then, cut a 2mm deep pocket in part.
ASL: get(tool(the, cutting(ø), ø, ø),
from(caddie(the, tool(a, ø, ø, ø), ø, ø))) >
cut(pocket(a, 2mm), part(a, ø, ø, ø))

English: Move the tool alone the x-axis plus 5mm.
ASL: move-linear (tool(the, ø, ø, ø),
position(relative, x-axis, +5mm))

In the examples, the ø symbol indicates a required s-role within a lexical unit being used for which no (or NULL) information is needed to complete the English sentence being translated.

Situational Dependencies

As stated above, the Situational Dependencies layer provides the basic semantic network for the storage of the Lexical Semantics. This semantic network allows for the existence of three types of nodes: 1) verb, 2) noun and 3) preposition. As shown in Figure 3, each of these node types provide the required storage of the tag and the needed grids.

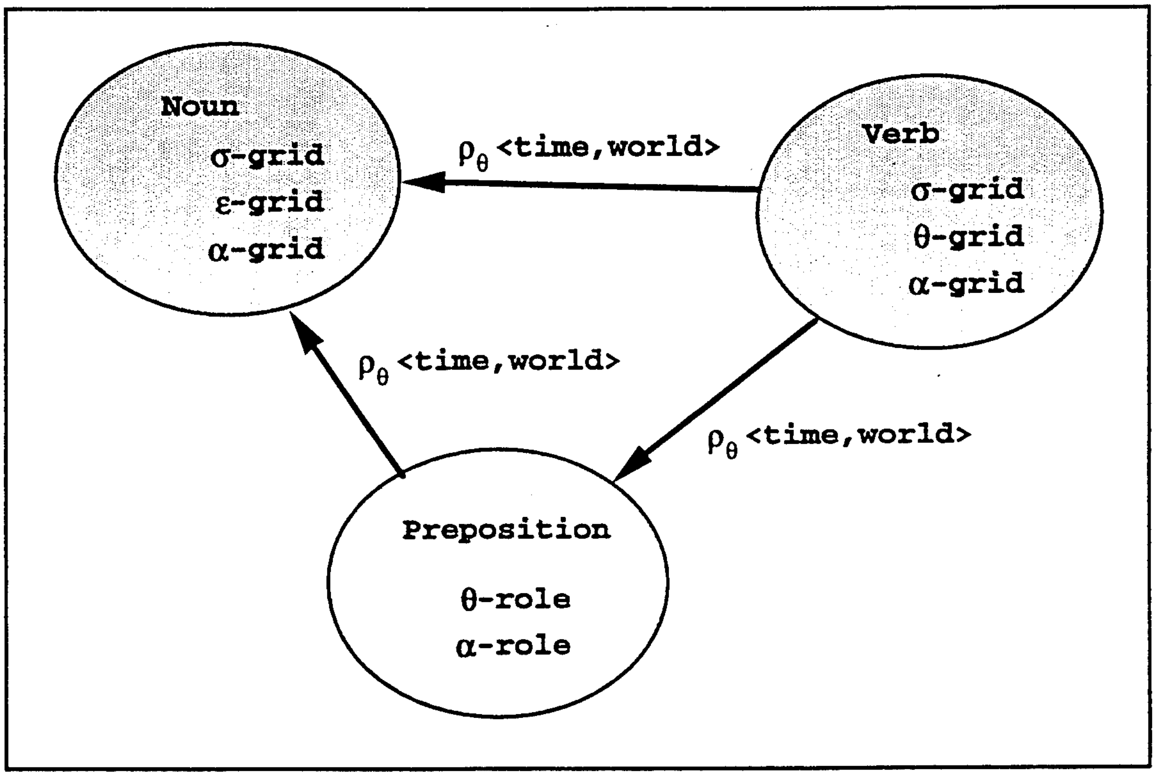


Figure F-3. Situational Dependency Nodes with Theta Relations

The expressive power of this network comes from the ability of the directed arcs (relations) to contain both type and situation information about the relation (i.e., the time and world in which the relation applies). Thus, procedural events, scenarios, and even beliefs can be directly superimposed on the lexical semantics without any risk of the information being misused by the system. As a result of this ability, the concept of long term and short term memory can be used with the n-Towers model without any of the complexity normally associated with systems that store these two types of knowledge in two different places.

As shown in Figures 3, 4, and 5, there are five different classes of situational relations define in the Situational Dependencies KR. These are:

r_q relations - defines the noun or preposition node allowed to fill a verb's theta role.

r_e relations - defines the noun or preposition node allowed to fill a noun's epsilon role.

r_i relations - defines a pure reciprocal relation between nodes of the same type.

r_n relations - defines non-reciprocal relation between nodes of the same type.

r_w relations - defines two relations which when taken to together produce a reciprocal relation between nodes of the same type.

- The actual number and names of the relations allowed in a relation class is defined in the Meta-Semantic level of the n-Towers model.

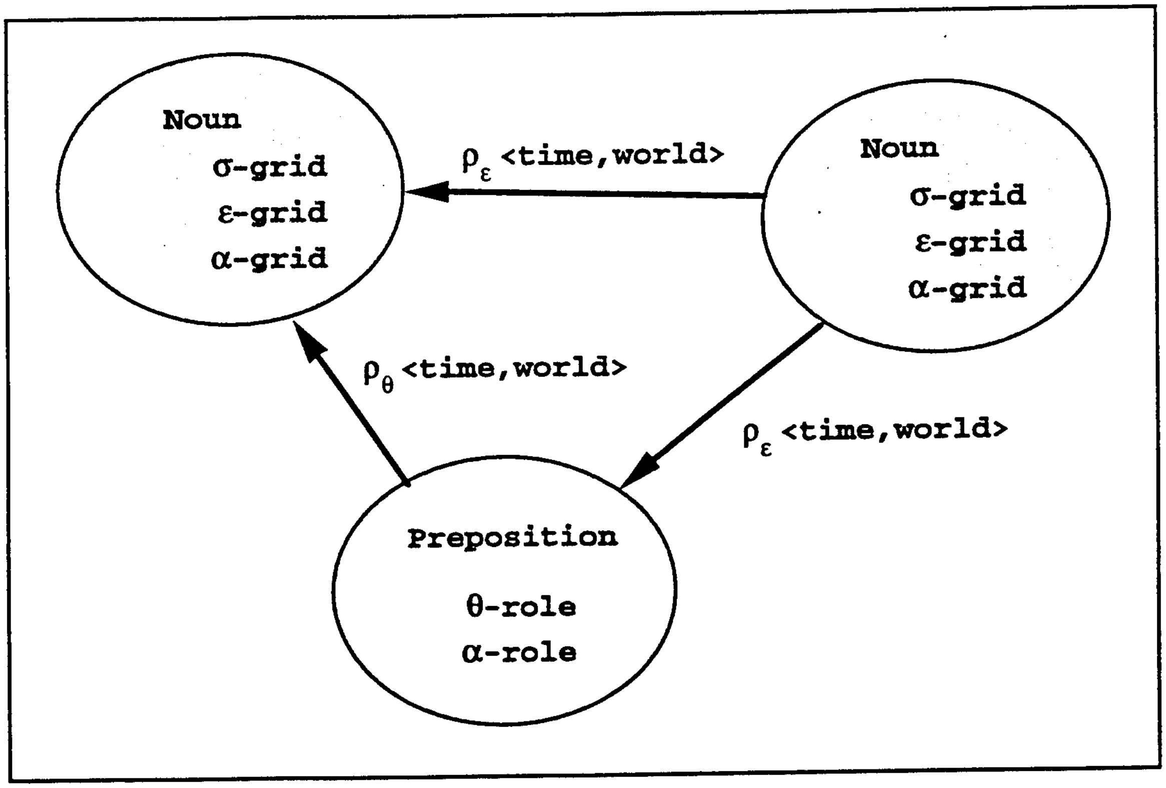


Figure F-4. Epsilon Relations of Situational Dependency Nodes

Meta-Semantics

The Meta-Semantics layer allows the definition of the legal relations allowed in a tower of the n-Tower model and the inferencing methods associated with those relations. Since relations must be defined as one of the five classes stated above, inferencing method can be associated with either a relation or a relation class.

The Meta-Semantics layer also handles the housekeeping for the reciprocal relations, ensuring that when a reciprocal relations is defined between Node A and Node B, the correct relation is defined between Node B and Node A. As an example of this, if the relation *A NC-machine* *always is-a machine in the real-world* Ò is added to the semantic knowledge, then the relation ·*A machine always could-be-a NC-machine in the real-world* Ò would be added by the meta-semantics if a reciprocal relation between *is-a* and *could-be* has been defined.

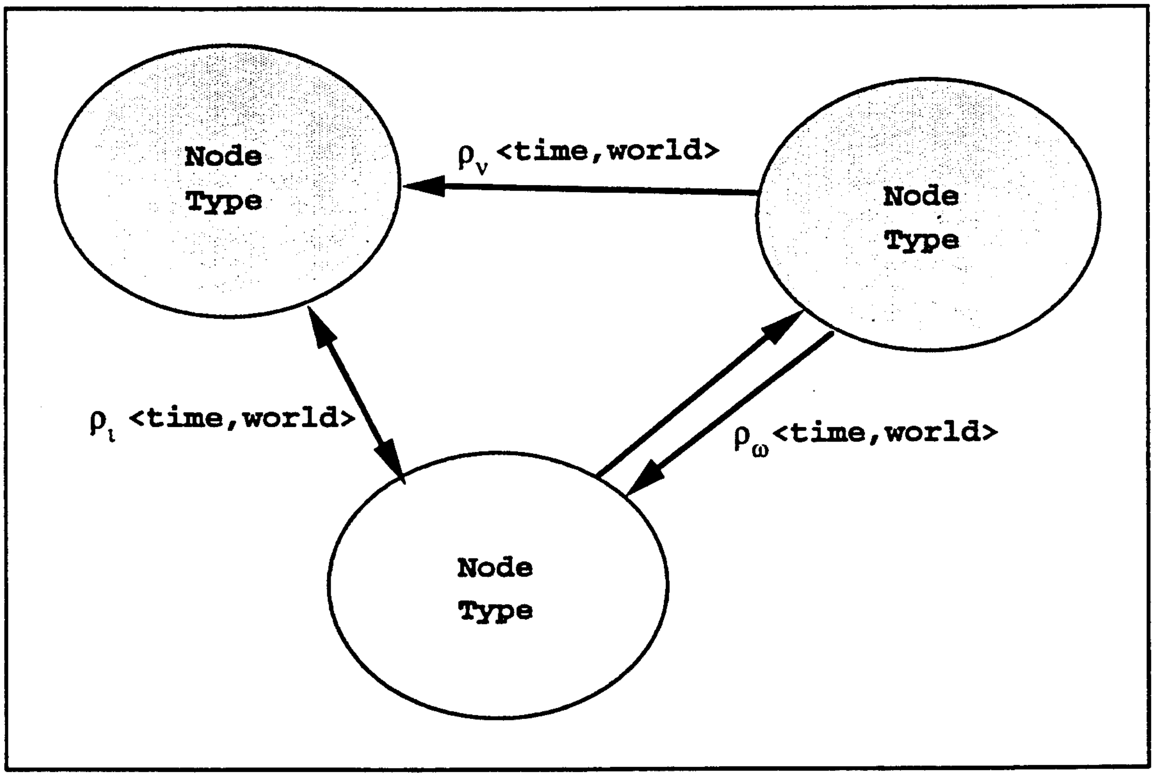


Figure F-5. Other Relations of Situational Dependency Nodes

Lexical Semantics

As stated above, the verbs, nouns and prepositions which can be used by the ASL will be stored in a lexicon which is free to grow as new concepts are needed to be expressed. In the n-Towers model, this lexicon will exist in the Lexical Semantic layer. The amount of information which must be stored in the Lexical Semantic layer for a lexical unit will depend on the category of the lexical unit. For all entries, a tag and the set of current situational links to the semantic representation of the domain knowledge (i.e., the other lexical units) will be required.

If the lexical unit is a verb, a q-grid containing the number and type of q-roles associated with the unit will be maintained to allow a road map to which noun and preposition nodes in the domain can be legally linked to that verb. If the lexical unit is a preposition, a single q-role will be maintained for this purpose. If the lexical unit is a noun, a e-grid containing the number and type of e-roles associated with the unit will be maintained to allow a road map to which noun and preposition nodes in the domain can be legally linked to that noun. The q-grid, q-role and the e-grid are represented as a simple ordered list of roles that corresponds to the unit's s-grid and a-grid.

Since a verb is one of the most complex types of entry, a verb will be used as an example. Given the verb, 'rounds', as used in the sentence:

The lathe rounds the part with a cutting tool.

The following lexical entry for 'rounds' would appear in the lexicon:

Tag:	hit				
q-grid:	1) Agent	s-grid:	1) NP	a-grid:	1) action X
	2) Patient		2) NP		2) action Y
	3) Instrument		3) PP		3) action Z

Both the domain knowledge and general knowledge of a tower will be contained in a set of situational dependent links between the lexical unit elements in the network (the nodes). The Meta-Semantic layer will determine the set of inferencing methods needed for this knowledge. As a result of the language information stored in the Lexical Semantic layer, the ASL layer is able to prohibit meaningless sentences. For example, given the sentence:

English: The part rounds the lathe with a cutting tool.
ASL: rounds(part(the, ø), lathe(the, ø), with(tool(a, cutting)))

using the information stored in the Lexical Semantic layer, the ASL parser would be able to determine that the q-roles for 'rounds' have been violated by the use of the NP:'the part' as a agent and the NP:'the lathe' as a patient.