Validation of Standard Interfaces for Machine Control*

F.M. Proctor, J. Michaloski,
W. Shackleford, and S. Szabo
National Institute of Standards and Technology
Gaithersburg, MD 20899

ABSTRACT

Open architecture controllers offer a multitude of benefits to users of machine tools, robots, and coordinate measuring machines, ultimately reducing the life-cycle costs of installing, operating, and maintaining manufacturing equipment. Aside from those benefits resulting from basing a controller on common operating systems and computing platforms, the main feature of an open architecture is the public availability of interfaces to controller functionality. These interfaces allow third parties who are not associated with the original equipment manufacturers to provide enhancements to the functionality of the machine. Efforts to standardize the interfaces to open architecture machine tool controllers are underway both in the United States. and abroad. In the United States, the Department of Energy and the National Institute of Standards and Technology have cooperatively undertaken this standards effort. One of the most important aspects of this program is the validation of interfaces on actual machinery in production applications. The goal of this validation process is to ensure that the interface specification is broad enough to encompass a significant portion of manufacturing applications, while still being practical to implement. This paper explores the problems resulting from defining interfaces to general controller functionality, when faced with the realities of validating the interfaces on controllers with specific operating systems, computing platforms, and control components.

KEYWORDS: *open architecture controller, motion control, standards*

1. INTRODUCTION

In the early 1990s, the Manufacturing Engineering Laboratory of the National Institute of Standards and Technology (NIST) began the Enhanced Machine Controller (EMC) program to develop a modular definition of components for machine control [1].

The intent was to document the interfaces to these modules to the degree that would allow independent third parties to provide interoperable products. The development of this modular architecture grew out of NIST's experience developing controllers based on the Real-time Control System (RCS) architecture which has evolved within NIST over many years [2]. NIST and the Department of Energy national laboratories have combined their efforts in this area under the auspices of the Technologies Enabling Agile Manufacturing (TEAM) program, and are undertaking a formal review of the interface specification that has resulted from recent implementations of interface-based controllers [3]. This review also includes researchers from General Motors and the University of Michigan. A broader review is anticipated within the EMC Consortium, established by NIST in January 1996 to formalize a three-year review of controller implementations based on these interfaces.

2. INTERFACE DESCRIPTION

The development of the EMC architecture, shown in Figure 1, was the first step toward defining an interface specification. In this figure, boxes indicate the individual modules for which interfaces have been defined and validated. These include Task Sequencing, Trajectory Generation, Servo Control, and Discrete Input/Output. The Operator Interface, shown at the side, does not require any specific interfaces itself, but can be developed using only the interfaces provided by the other modules. Implementations of the operator interface need only avail themselves of messages to the controller and data provided by the controller: no additional interfaces are required to be defined in order to incorporate an operator interface into an EMC controller.

---

* No approval or endorsement of any commercial product by the National Institute of Standards and Technology is intended or implied. Certain commercial equipment, instruments, or materials are identified in this report in order to facilitate understanding. Such identification does not imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the materials or equipment identified are necessarily the best available for the purpose.

This publication was prepared by United States Government employees as part of their official duties and is, therefore, a work of the U.S. Government and not subject to copyright.

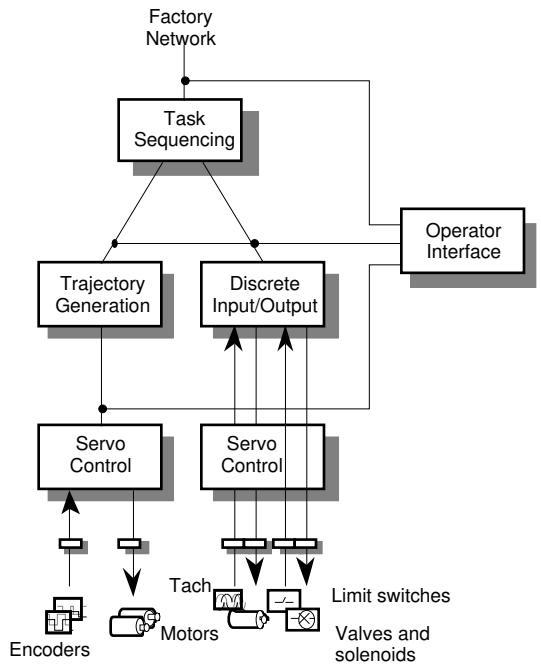

Figure 1. The EMC Architecture. Interfaces for Task Sequencing, Trajectory Generation, and Servo Control were targeted during the validation process.

The interface specifications are formalized in the C++ programming language, using header files. The specification consists of messages into each module, and world model data provided by each module. Both the messages and world model data are implemented using C++ classes.

Class definitions alone are not sufficient to describe the interfaces. The specification needed to include the expected behavior of the control modules in response to each control message, and their effect on the world model of each control module. This information is provided in manual-style pages accompanying the C++ class definitions, using Hypertext Markup Language (HTML) format.

Supplementing the message specification is a model of data transfer, the Neutral Manufacturing Language (NML) [4]. This model provides for "mailboxes" of data, with one or more readers and writers. Each module is modeled as a cyclic process, which reads its input command from its supervisor, reads the status of its subordinates (or sensors), and computes and sends outputs to its subordinates (or actuators).

The interface specification is divided into two parts: commands that each module will perform, and status that each module will maintain. Both commands and status are derived from the NML message base class, and require a unique identifier and zero or more data fields representing the parameters to the command or fields in the status. During the development of the specification, the intent was to analyze the general requirements of each module in terms of which commands it should be responsible for carrying out, and what world model status it should be responsible for maintaining. Ideally, the specification would be complete enough to never require modifications. In the case of the Servo Control module, a literature survey conducted over a period of years resulted in a cataloguing of 62 algorithms, and a generalization of an interface to this module that would support implementation of the servo control function with any of these algorithms. However, no matter how comprehensive such a survey, new algorithms will inevitably arise which require input data or provide output data that are not available in the interface. Because of this, extensions to the interfaces are anticipated. Indeed, the need for extensions was evidenced during several of the validation tests discussed in subsequent sections of this paper.

3. SUPPORT INFRASTRUCTURE

A central problem faced when developing the interfaces was eliminating suppositions for computing platforms, operating systems, and programming languages. Already one can see that the C++ form of the interfaces presumes a programming language. Vendors of modules (toward whom these interfaces are aimed) need to provide the command initiation functions and world model access functions for the platforms the vendors have selected. This does not prevent developers from using another language for the implementation (e.g., the graphical user interface was coded in Microsoft Visual Basic in this case), but it does require that the external interfaces be C++-linkable.

The picture is complicated when considering how one assembles a system with components that provide these interfaces. For example, as seen in Figure 1, the Trajectory Generator module supervises the Servo Control module, typically generating points in a world coordinate system and sending them to the Servo Control subordinate. However, the Trajectory Generator cannot, in general, call functions provided by the Servo Controller, since the Trajectory Generator may be running on a different computer than the Servo Controller.

This problem is one aspect of the more general configuration problem, which leads to the question, "What services can be expected in an open architecture controller that allow components to interoperate?" Aside from a vendor's need to access memory, disk files, timers, and other operating system resources, there is more required from the support infrastructure as the example above illustrates.

Because we needed to implement controllers for various machines to embark on the validation effort, we required some particular support infrastructure. We used the NIST RCS Library, which has been ported to a variety of computing platforms, and directly supports the RCS methodology for implementing real-time control systems by providing a uniform programming interface to communication, timing, shared memory, and mutual exclusion primitives. However, by selecting this library, we sidestepped an important problem: which support

infrastructure should be presumed (and indeed accompany) the interface specification? Architectures exist which can serve the purpose, such as the Common Object Request Broker Architecture (CORBA) from the Object Management Group, but requiring a particular infrastructure greatly constrains the potential spectrum of applications. Resolving this problem was outside the scope of this effort.

3.1. *Validation Tests*

The interface validation tests were conducted in testbeds at NIST in Gaithersburg, Maryland and at the General Motors Powertrain facility in Pontiac, Michigan. At NIST, the testbed consisted of a UNIX workstation simulation for initial development, and a two-computer controller for run-time tests on a desktop milling machine (minimill). The graphical user interface ran on Microsoft Windows on the first computer, and the real-time controller ran on a UNIX operating system on the second computer. Communication between the two took place via ethernet.

This is not a full paper. This is only an example of how to produce your camera-ready paper for the ISIC/CIRA/ISAS '98 conference.

4. REFERENCES

[1] Proctor, F. M., and Michaloski, J., "Enhanced Machine Controller Architecture Overview," NIST Internal Report 5331, December 1993.

[2] Albus, J. S., "Outline for a Theory of Intelligence," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 21, No. 3, May/June 1991.

[3] Shackleford, W., and Proctor, F. M., "The Real-time Control System Library, Internet Location: http://isd.cme.nist.gov/ ~shackle/rcslib/

[4] Object Management Group, "What is CORBA?," Internet Location: http://ruby. omg.org/corba.htm