CONF-9605146--2

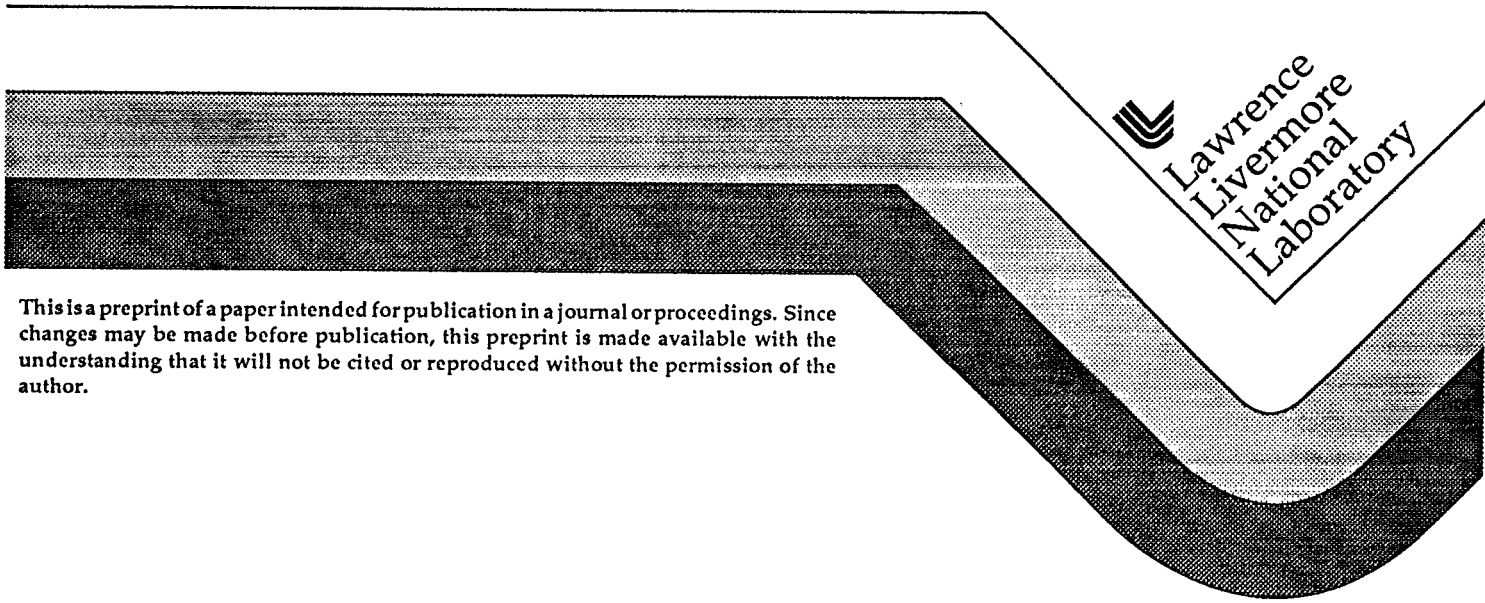
UCRL-JC-122845
PREPRINT

Open Architecture Controller Activities in TEAM

Howard K. McCue

This paper was prepared for submittal to the
World Automation Congress (WAC) '96
Montpellier, France
May 27-30, 1996

February 4, 1996



Open Architecture Controller Activities in TEAM

Howard K. McCue, Ph.D.
Lawrence Livermore National Laboratory

ABSTRACT

As part of its manufacturing initiative, TEAM is actively involved in open architecture controller activities. Within the TEAM community of members, TEAM is developing an open architecture controller requirements document and an open architecture controller application programming interface document. In addition, TEAM is also evaluating early open architecture controllers in a shop floor environment.

KEYWORDS: application programming interface, controller, open architecture controller, API, OAC, TEAM

INTRODUCTION

TEAM (Technology Enabling Agile Manufacturing) is an alliance of over 30 individual US industry members, industry consortia, US federal organizations (DOE, NIST, DoD, NSF), and US universities. Its activities are guided by an industry led steering board with technical progress provided through industry/ government co-led thrust area teams. TEAM's goal is to streamline product development, reduce cost, shorten time to market, and enhance quality by deploying integrated, validated design-to-manufacturing tools and processes. As part of this manufacturing initiative, TEAM is actively involved in open architecture controller (OAC) activities. To date, TEAM's OAC activities have centered around OAC reference specifications and evaluation of early OACs. Within the TEAM community of members, TEAM is developing an OAC requirements document and an OAC application programming interfaces (APIs) document. One goal of TEAM is to develop OAC APIs that will be supported by multiple US controls vendors on PC-based controllers. In addition to the OAC specifications work, TEAM is also evaluating early OACs in a shop floor environment. The purpose of the evaluation work is to validate controller performance in a shop floor environment and identify problem areas that need improvement.

CONTROLLER SITUATION ANALYSIS

In today's worldwide manufacturing environment, industries must constantly improve their manufacturing processes just to remain competitive. The dominant parameter(s) of improvement (cost, time to market, quality, and functionality) will vary depending on the particular situation, but constant improvements in manufacturing processes are required. Future improvements in manufacturing processes depends strongly on two factors: a detailed understanding of the process as it is and as it could be in addition to advanced process control to exploit this understanding. If processes were currently fully understood and if processes could currently be economically exploited via advanced controls, then worldwide competition would demand that this be done. Today, we encounter barriers and bottlenecks when trying to apply needed control upgrades using traditional discrete manufacturing control systems. Traditional manufacturing controls cost too much, have limited functionality, and take too long to implement the needed advanced control features. The traditional controller costs are kept high due to proprietary vendor hardware and

software. Proprietary solutions are typically not in the mainstream, high volume computer arena. Proprietary hardware and software means uncommon or unpublished key interfaces which limit or eliminates third party suppliers. Proprietary hardware and software also means training costs are higher to cover the specialized, niche products. Finally, costs can be higher because once a customer is locked into a proprietary solution, then the controls vendor has less incentive to be innovative and cost sensitive. Continuous process improvements will require more functionality than traditional controllers can provide. Future process improvements will use and exploit a detailed understanding of the process. Future discrete manufacturing controllers must accommodate: detailed process models, multiple sensed and actuated process variables, advanced control algorithms and strategies, intelligent decision making, and process communication with the enterprise. Implementing non-standard controls on the traditional controller takes too long. Non-standard controls features are typically programmed by the controls vendor. Turn around times are often too long for agile manufacturing.

TEAM views open architecture controllers as a way around the barriers and bottlenecks inherent in traditional manufacturing controllers. With open architecture controllers, costs will be reduced by providing solutions that are in the mainstream of computer hardware, software, and standards. In addition, TEAM is pursuing the TEAM APIs with the goal of opening up these controller interfaces. An industry-accepted, published APIs would enable third party vendors to write controller modules for a wide array of controllers. The added competition would keep costs down while increasing the array of available functionality in the modules. If an end user needs some non-standard functionality for his control process, then the APIs allow the end user to get this functionality through several means: controls vendor, third party vendors, or end user written. The bottom line is: OACs are time and cost efficient mechanisms for implementing the present and future discrete manufacturing controls needed by industry.

TEAM OAC API

Figure 1 illustrates the basic elements of a TEAM API compliant open architecture controller. First, the computation platform supporting the controller is built from mainstream computer hardware and software that utilizes industry-accepted computer hardware and software standards. This keeps the basic computational platform costs low and platform component availability high. The controls vendor supplies the infrastructure software that enables the operating system to interface to the TEAM APIs, and thus the TEAM API modules. In addition to the infrastructure software, the controls vendor also provides support tools that will allow the user to connect modules as needed, to develop module functionality as needed, and to support other basic functions (i.e. initialization of hardware and software from an application parameter file, etc.). The TEAM API modules operate on top of services and functionality provided by the mainstream computational platform and controls vendor's infrastructure software. The TEAM API modules are interconnected to provide the level of control functionality needed by the discrete manufacturing process. TEAM has several goals it wishes to reach in applying the above technique. First, the APIs must be open. That is, the module interface definitions and supporting documents must be made public for a nominal fee (i.e. essentially free). The modules must support growing, shrinking, and modifying controller functionality to meet a wide range of price/functionality points. For example, the modules should support controllers ranging from simple PLCs to machine tool controllers to multiaxis robotic controllers. The modules must support scaling of controller performance by easily moving to faster or slower platforms. This allows the end user to select the price/performance point needed for an application.

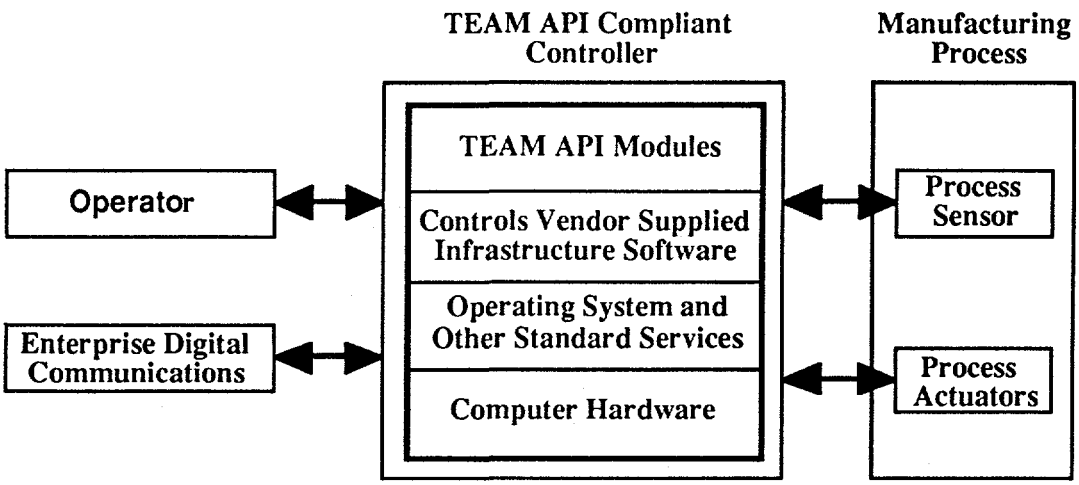


Figure 1: Basic Elements of a TEAM API compliant Open Architecture Controller

APIs play a central role in open architecture controllers. One goal of TEAM is to define a set of open APIs for discrete manufacturing controllers that will be adopted by US controls vendors and used as a basis for building OACs. The development of the TEAM APIs and the TEAM OAC Requirements started in September 1994 at a TEAM meeting at General Motors. At this meeting, many large industrial companies (Cincinnati Milacron, Chrysler, Ford, General Motors, Pratt & Whitney, Wizdom) and Federal agencies (NIST, DOE) were represented. One starting point for the TEAM APIs was the General Motors Open Modular Architecture Controller document [1] that had been written by GM and reviewed by Chrysler, Ford, and General Motors. The participants of the meeting put together a first-cut at the modules and established a small working group to define the TEAM APIs. Some of the initial results from the TEAM API working group are now available and are discussed below. This meeting also used the NIST EMC Functionality document [2] and participants input as starting points for the TEAM OAC Requirements document [3].

The TEAM API model defines ten modules; see Table I and [4] for details. In specifying the modules, the controller was decomposed based on logical functionality and anticipation of which interfaces the controls builders (controls vendors, third party developer, and end users) will need access to. Each module has the following properties: It has a publicly defined interface. It has defined functionality and services that it performs within the TEAM API context. Any external functionality and services that it uses are defined in terms of other modules. A module only interact with other modules through its defined interfaces. If a developer adheres to the TEAM API model, then any of the modules can be replaced by other similar modules having different price/performance and price/ functionality points.

Not all TEAM APIs need be implemented in an open architecture controller; a controls vendor may want to aggregate functionality. For example, if a motion control card vendor incorporates the functionality of many TEAM API modules in hardware, then only the TEAM APIs that interface the motion control card to the remaining portion of the controller need be implemented. If at a later date, one wishes to replace the motion control card, then all the functionality represented by the hardware card must be replaced as a unit. Going the opposite direction, the TEAM APIs may not represent all the interfaces a controls vendor needs to implement an OAC. For example, a vendor supplying a software solution for the motion control portion may wish to define additional, internally used, interfaces to help develop the vendor's software product; these internal interfaces may or may not be made public. The controls vendor can define additional, internal interfaces and still matchup to the TEAM APIs at a higher level.

Table I: TEAM API Modules and Functionality Description

Module	Functionality
Human Interface	responsible for data access between controller internal modules and operator control panel.
Communications	responsible for digital communications over local area links and sensor/actuator bus communications.
Task Coordination	responsible for sequencing operations plus coordinating motion, sensing, and event-driven control processes
Part Program Interpreter	responsible for translating the part program into control sequences
Trajectory Generator	responsible for coordinating the motions of individual axes.
Single Axis Control	responsible for motion control of a single axis
Axis Control Law	responsible for control calculations (typically loop closure and control law).
User Defined Process	responsible for detailed process models that include specialized sensor and actuator calculations.
Discrete Control	responsible for discrete logic calculations
I/O Points	responsible for reading input devices and writing to output devices using generic read/write interfaces. Specialized sensor/actuator buses are handled in the communications module.

Figure 2 illustrates how a simple, conventional two axis motion controller could be implemented using the TEAM API modules. Under normal operating conditions, the controller interacts with the world through the operator's control panel, an Ethernet connection, and various I/O connections to drives, encoders, switches, et cetera. The operator's control panel interfaces with the human interface module. This module has access to those module variables and parameters that are in the TEAM APIs. These variables and parameters are shown as an ellipse in the figure. The part program (e.g. RS274) is downloaded from the Ethernet connection using the communications module. The task coordination module sequences various internal operations and coordinates the motion control. The part program interpreter takes the part program and translates it into control sequences used by the trajectory generation module. The trajectory generation module takes the control sequences and coordinates the motions of the two single axis control modules. In Figure 2, any "shadow box" with a two on it denotes a duplicate module for the second axis; arrows are not shown on the second axis. The single axis control module has the responsibility for the motion of a single axis. The control law module supports the single axis control module by providing loop closure and control law calculations. If one wishes to change the control law from PID to fuzzy logic, this is the module that needs to be modified or replaced. The discrete logic module supports all the binary logic operations. If discrete logic, tool changer control is needed for a machine tool, it would be located in this module. Various I/O points are used to read and write to the process sensors and actuators.

Figure 2 shows the high level interconnects for a simple, two axis motion controller. The benefits of the open API approach are amplified when more complex situations are considered. Figure 3 shows how the controller can easily accommodate the uncoupling calculations needed in precision machine tools. When one axis moves, it can cause changes in position in other axes (i.e. straightness). Figure 3a shows the uncompensated version of axis position input. Figure 3b shows how layered I/O can be used to compensate for repeatable crosscoupling motions. The calculations could include lookup tables or pure calculations based on geometry. Note that the APIs defined for the sensor

input do not change and are re-used for the specialized calculation module; the remaining controller modules do not need to know this change has occurred. This is symbolically shown as interlocking puzzle pieces in Figure 3. Module functionality can be dramatically changed while preserving APIs. For example, in Figure 2, one could replace the part program interpreter with a CAD file interpreter while preserving module APIs.

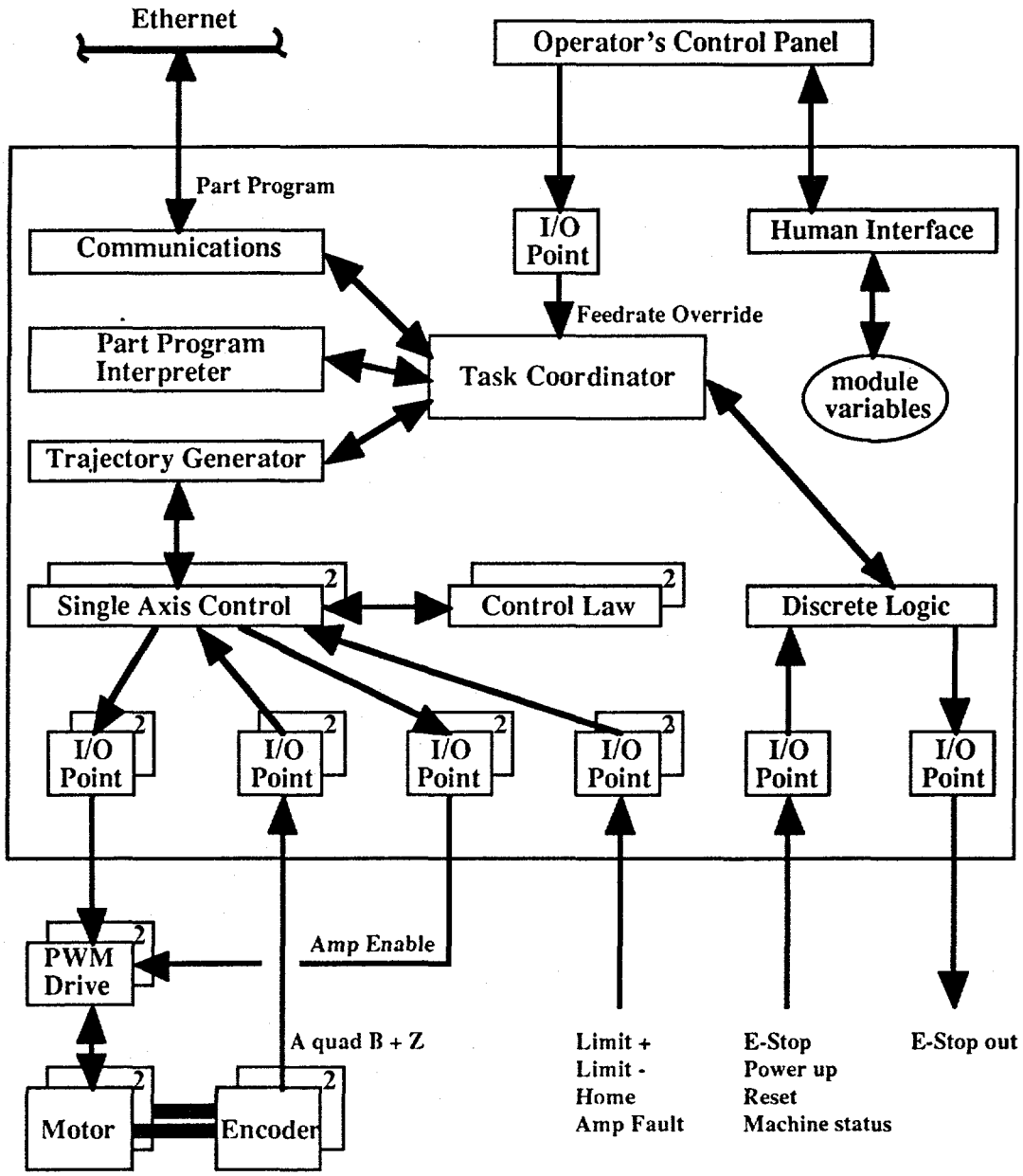


Figure 2: Simple Two Axis Motion Controller based on TEAM APIs

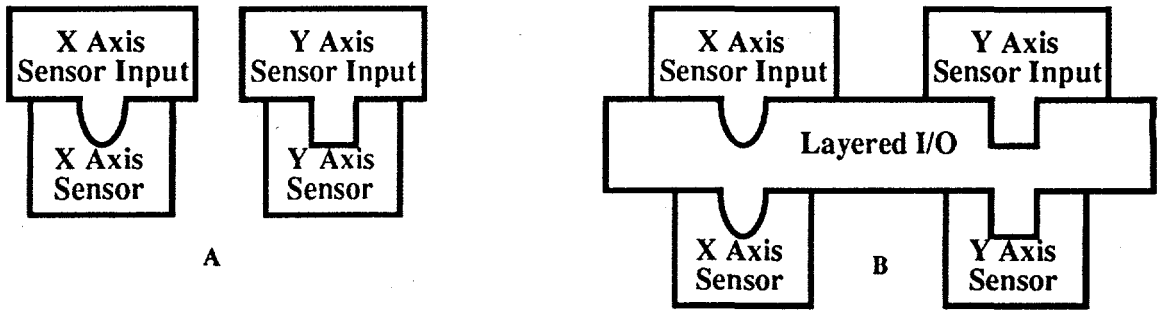


Figure 3: Extending Controller Functionality by Using Layered I/O

INITIAL OAC EVALUATIONS

As part of its OAC activities, TEAM is evaluating initial OAC controllers in the shop floor environment. At present, three TEAM OAC activities are in progress and more are planned. The three TEAM initial OAC evaluations are: NIST EMC on a four-axis K&T 800 mill with tool changer at General Motors Power Train, specialized Trellis drilling controller on a Sharnoa mill at Ford Scientific Research Laboratories, and ICON on a Bostomatic mill at LLNL. The NIST EMC on the K&T 800 mill is being used daily to machine parts in the GMPT prototype shop. Future extensions include spindle thermal growth compensation and part/fixturing variation compensation using part probing. The other two controllers are being developed and installed and are not yet operational. The purpose of the evaluations is to verify the controller performance in a shop floor environment plus identify where future improvements need to be made.

ACKNOWLEDGMENTS

I wish to thank all the TEAM members who participated in the TEAM OAC activities that I describe here. In particular, I would like to thank Jerry Yen of General Motors and John Michaloski of NIST for leading the TEAM API effort, Doug Sweeney of LLNL for being the editor for the TEAM OAC Requirements document, and Fred Proctor of NIST for his EMC evaluation at GMPT. This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.

REFERENCES

1. Jerry Yen and Clark Bailo, "Requirements of Open, Modular Architecture Controllers for Applications in the Automotive Industry" version 1.0, August 15, 1994
2. Charles Yang, "Enhanced Machine Controller Functionality Requirements", NIST, Manufacturing Engineering Laboratory, Automated Production Technology Division
3. Douglas Sweeney, "TEAM Open Architecture Controller Requirements", Lawrence Livermore National Laboratory, UCRL-ID-122503 Dr
4. TEAM API Working Group, "Preliminary TEAM API Specification Document", available on the world wide web at: http://isd.cme.nist.gov/info/teamapi For more information, contact John Michaloski, NIST 301-975-3458