AIM

Association for Information and Image Management

1100 Wayne Avenue, Suite 1100
Silver Spring, Maryland 20910
301/587-8202

Centimeter

Inches

MANUFACTURED TO AIM STANDARDS
BY APPLIED IMAGE, INC.

1 of 1

SAND 24-10-12

Sandia National Laboratories

Albuquerque, New Mexico 87185

date: February 14, 1994

to: Major Michael B. Leahy, Jr.
SA-ALC/TIEST, Bldg. 183
450 Quentin Roosevelt Rd.
Kelly AFB, TX 78241-6416

from: William E. Ford
Sandia National Laboratories
M/S 0952
Albuquerque, NM 87185
Phone: (505) 844-7893
Fax: (505) 844-6161
Email: weford@isrc.sandia.gov

subject: Submission of paper to 9th IEEE International Symposium on
Intelligent Control, August 16-18, 1994.

Enclosed you will find five copies of a paper titled "What is an Open
Architecture Robot Controller?". I would like to publish this paper in
the symposium proceedings. The following is a repeat of the abstract.

Abstract

This paper addresses the issue of what is an open architecture robot
controller. Three different classifications are defined along with the
various advantages and shortcomings of each approach. Knowledge
from past research and new technology development has been included
in this analysis. The conclusions recommend a communication-based
hybrid approach with well defined interfaces between modules.

2161:WEF:wef

Copy to:
MS 0952 R. Palmquist, wo/enc.,

What is an Open Architecture Robot Controller?

William E. Ford

Sandia National Laboratories,
Intelligent Systems and Robotics Center,
Albuquerque, NM 87185¹

Abstract

This paper addresses the issue of what is an open architecture robot controller. Three different classifications are defined along with the various advantages and shortcomings of each approach. Knowledge from past research and new technology development has been included in this analysis. The conclusions recommend a communication-based hybrid approach with well defined interfaces between modules.

1: Introduction

Taking a broad definition, robot controllers can be classified into three main categories: proprietary (also called closed), open, and hybrid systems. A proprietary system is one in which it is very difficult if not impossible to integrate external hardware including sensors and software. A completely open architecture is one in which all aspects of the design can be changed or modified by someone other than the original manufacturer. In other words, the hardware and software structure is such that integrating sensors, operator interfaces (e.g. force reflection), new control laws for servos, etc... can be done without any difficulties. In a hybrid system, certain aspects such as the control laws are closed whereas most elements are accessible. The vast majority of the robotic systems sold throughout the world today have proprietary controllers. This is desirable when the application is well defined and is not expected to change. This type of "turn-key" system requires little in-house expertise and development. In recent years, the demands for increased capability and flexibility has led to a dramatic increase in the use of controllers based on open architectures. The following describes some of the advantages and shortcomings of each approach.

Why select a proprietary robot controller? The two major reasons are a large installed user base and low initial cost. The benefit of a large installed user base is that the system has a proven track record. Low cost is the result of several market factors. The most dominant is the amortization of the robot controller development over a large number of units. If the customer has high confidence in knowing the current and future tasks of the robotic system, then selecting a proprietary system which meets the needs is most likely the preferred choice. The downfall, however, comes when such a proprietary system that meets all these needs does not exist and customization is required. Modifying a closed system can be very expensive and often

times is not technically possible without extensive redesign.

It is possible to gain some of the benefits of a proprietary controller and still have the ability to customize and upgrade the system as needs dictate. In the hybrid approach, the low level control functions are provided by the vendor along with the means to externally access these functions. For example, the vendor could be responsible for servoing joint angles which are generated by external software at a given update rate (typically 10-50 Hz). Thus, this proprietary part of the system could interact with the trajectory generation, sensor integration, and operator interface subsystems which could all be based on a completely open architecture.

This hybrid approach has been used by the U.S. Department of Energy's Robotics Technology Development Program (RTDP) to develop faster, safer, and cheaper robotics technologies for hazardous waste site cleanup. Because of the unstructured environments and evolving needs presented by these waste sites, a key approach has been the development of this hybrid approach which employs modular-based technologies that allow the creation of robotic systems from well tested subsystem modules, including commercially supplied items such as robot controllers [1]. Faster technology development results from adapting previously developed generic technology to new applications. Safer robotic systems result from the reuse of modular generic technologies in diverse applications. Robotic systems that reuse well understood generic technologies can be much better characterized than robotic systems using only technology developed for that particular application. When the development activities satisfy the needs of many applications, the total development cost is spread out over many projects and time for technology development are reduced because there is less duplication of effort. It is important to realize that generic technology will most likely have applications outside of hazardous environments. This will contribute to reducing the costs of commercial technology.

In the past, much time was spent developing and redeveloping the communication and subsystem interface infrastructure. Thus, the ability to add high level intelligence is often delayed or hindered by the initial system infrastructure development efforts [2]. Ideally, a system software architecture should bridge the gap between high level sub-task descriptions and low level robot commands. Current commercial robot control systems fail to do this. Therefore, it is necessary to develop customized

¹ This work was performed at Sandia National Laboratories and supported by the Department of Energy under contract DE-AC04-94AL85000.

MASTER

CONF-940869-1

20

architectures in order to provide the flexible and powerful environment that allows higher intelligence to be added. This effort can entail significant time and personnel resources. In addition, customized systems are rarely adaptable to new applications. Open architectures which are easily adaptable to new applications are needed. The main idea of an open or hybrid architecture is the following:

- Use a development system based on a non-proprietary computer platform. (e.g. Sun, SGI, PC's)
- Use a standard operating system (e.g. UNIX, VxWorks) and a standard control language (e.g. C or C++).
- Base the hardware on a standard bus architecture that can interface with a variety of peripherals and sensor devices.
- Utilize networking strategies that allow workcell controllers to share databases, and to be operated from remote locations.

Also, this type of interface-based system structure promotes flexibility, which in the past has been difficult to achieve because of the immense integration difficulties associated with multi-vendor equipment and hardware/software interfaces [3].

One of the greatest benefits from the existence, acceptance, and use of an open architecture approach for any key technology is the ability to foster competition. A third party will be able to add value to the hardware or software, whereas a proprietary system can only be modified by the original developers. It is this long term gain which will allow any system to be continually improved to meet unanticipated or changing needs and that will overcome the immediate gains offered by selecting a proprietary system. Another great benefit of open or hybrid controllers is the reuse of elements of an existing system [2]. Reuse of existing capabilities provides confidence that they will meet requirements and not fail catastrophically in unexpected ways. It also is easier to take advantage of rapid improvements in cost and performance driven by high volume markets. Open or hybrid architecture robot controllers based on modular, distributed, interconnected generic components allow leveraging of future advances in hardware and software for improved throughput and greater intelligence. Open controller designs allow a customer to standardize on platforms, operating systems, and user interfaces. Common open controller technologies promise reduced training requirements, reduced system support requirements, and reduced maintenance costs. They also promote increased quality and safety by the reuse of tested, well characterized components. Programming is simplified by prior experience with the robotic system, thereby reducing overall development time and cost. Open architecture controllers also provide greatly improved flexibility, reconfigurability, extendibility, interoperability, interchangeability, portability, scalability, reliability, and reusability. By enabling robot controller retrofits to be provided on a quality and needs basis, an open approach to hardware and software implementation can also stimulate competition among vendors of subsystem technology. A common approach will stimulate the robot industry to grow, with third party companies developing hardware and software packages that can lower implementation costs through competitive, multiple source procurements rather

than relying only on the original system developers. It also fosters rapid conversion from research systems to operational systems, thus shortening the research to commercial product cycle.

2: Current Approaches in Robotic Controller Architectures

This section summarizes some of the recent approaches to developing intelligent robotic systems. It is not meant to be all-encompassing but to be a survey of several current approaches.

2.1: NASREM

The National Aeronautics and Space Administration / National Bureau of Standards (NASA/NBS) Standard Reference Model for Telerobot Control System Architecture (NASREM) is based on research from several NASA, NIST, DOE, and university research centers [3]. It is one of the first attempts to fully specify all the components of an intelligent system [2].

The NASREM Standard Reference Model telerobot system architecture is a conceptual model which decomposes into a three legged hierarchy of levels [4]. Each of these levels performs a different fundamental mathematical transform. The architecture also splits each level into three sections, task decomposition, world modeling, and sensory processing. Each module is a finite state machine that accepts inputs, computes some function based on state and inputs, and produces outputs. The inputs consist of commands from higher levels, sensory data from the same level, and status/data from lower levels. The outputs are commands for the lower levels, processed sensory data for the same level, and data/status feedback for upper levels. This architecture is a standard methodology for implementing large telerobotic systems and not a physical working system.

2.2: NGC

The Next Generation Controller (NGC) is an attempt to develop a specification for an open system architecture standard (SOSAS) which is planned to become an ANSI standard for machine tools [5]. Sponsored by the U.S. Air Force, this project is important because it will provide well defined standards for the loosely specified NASREM concept. Martin Marietta Corp. was chosen as the prime contractor for the NGC program [3]. The objectives of the specification are to define an open architecture that provides increased capability, flexibility, and reduced cost/performance ratios for workstation/machine controllers. The approach requires all modules to be "plug" compatible between different manufacturers. The NGC is based on a virtual machine model in which each module has published specifications for its functionality and interfaces. This enables others to independently develop and test components that will extend or modify the capabilities of a controller without resorting to controller redesign. There are three concepts contained within the framework. The first is an integration architecture that provides a set of standard services for application programs. The second is a set of four applications that supply a basic set of machine controller functionality with messaging passing facilities

called Neutral Manufacturing Language (NML). Finally, an integration/configuration environment to provide a standard set of tools for augmenting, modifying, and integrating modules into a congruous system.

Currently, the National Center of Manufacturing Sciences (NCMS), Martin Marietta Corp., and the University of Texas at Arlington are under contract to redefine SOSAS [6]. The new document will be based on twin foundations of standards references for the hardware/software platform and object-oriented system analysis for application software. The applications are replaced by a component level architecture expressed in terms of responsibilities, requirements (inputs), and products (outputs). The primary architecture is a reference architecture; that is it contains no taxonomy - it is simply a pool of primitive responsibilities (with constraints) that may be organized application architecture. Interoperability is described at the source code level. Prototype systems using the NGC approach are currently in the early stages of development.

2.3: DICAM

CimFlex Teknowledge Corp. is undertaking a four year effort (begun in 92) to develop a new technology foundation and associated methodology for the rapid development of high performance intelligent controllers. These controllers will be employed in distributed intelligent control and management (DICAM) applications. This research is part of the Defense Advanced Research Projects Agency (DARPA) Domain-Specific Software Architecture (DSSA) initiative [7]. The company is attempting to fuse the best ideas from knowledge engineering and software engineering into the intelligent real-time control software being developed for an application and into the software development process itself. This generic control architecture combines a task-oriented domain controller with a meta-controller that schedules activities within a domain controller. The project has four elements. First, the formulation of a reference architecture for intelligent control. Second, the construction of applications in a development workspace in which system requirements are ultimately satisfied by choosing design components that specialize and particularize components of the generic reference architecture. Third, construction of reusable modules for an application building repository. Fourth, creation of a rich array of development tools that incorporate numerous techniques from both software engineering (control law specifiers, code generators, protocols, compilers, debuggers) and knowledge engineering (domain modelers, requirements managers, knowledge based design assistants). The company demonstrated the first working prototype of their development environment in August 1992 which showed how requirements can constrain and direct the software development process to produce software that specializes the generic controller reference architecture [8]. This methodology has not yet been applied to design a physical working system.

2.4: Theory of Intelligent Machines (Saridis)

An intelligent machines architecture has been advancing under professor George Saridis at the Rensselaer Polytechnic Institute since 1979 [9]. The functionality of the intelligent machine is implemented by intelligent controls using entropy as the metric. This approach is based upon a hierarchical control structure of three levels. First, the organization level is modeled after a Boltzmann machine for abstract reasoning, task planning, and decision making. Second, the coordination level is composed of a structure of Petri net transducers acting as coordinators that are supervised by a dispatcher, which also serves as an interface to the organization level. Lastly, the execution level consists of the sensory and motion control hardware that interacts one-to-one with the coordinators above. The entire architecture is based on the principle of increasing precision with decreasing intelligence. This means that the highest level tasks are assigned to the highest machine intelligence level and smallest complexity (database size), and the lowest level tasks are assigned to the lowest machine intelligence level and largest complexity.

The system is currently implemented on a robotic transporter designed for space construction. Professor Saridis claims the resulting software structure is extremely efficient, effective, versatile, and capable for remote operation as compared to other proposed architectures.

2.5: Port-Based Objects

Carnegie Mellon University (CMU) development has produced a software framework based on dynamically reconfiguring software for sensor-based control systems [10, 11]. The approach combines object-oriented design with port design of digital control systems. Thus, a port-based object is defined as an object that has various ports for communication. The internal states and methods are hidden from other objects. Only the ports of an object are visible to other objects. A global database of state information is the mechanism by which modules exchange information. The database is implemented with global shared memory. Every input/output port is a state variable. A global state table that contains all the modules' I/O port variables is stored in the shared memory. Tasks corresponding to each control module cannot access the table directly. Rather, each task has its own local copy of the table. Only the variables used by the task are kept up-to-date. At the beginning of every cycle of a task, the variables that are input ports are copied from the shared table. At the end of the task's cycle, the output ports are copied to the shared table. Furthermore, the global table is locked whenever a task is transferring data between its table and the global table. The global state variable table mechanism has been incorporated into the Chimera 3.0 Real-Time Operating System developed by CMU.

2.6: GISC

Sandia National Laboratories, along with the other U.S. Department of Energy Laboratories, is developing a Generic Intelligent System Controller (GISC) concept. This program is sponsored by the DOE's Robotic

Technology Development Program (RTDP). GISC is communication-based and follows the premise that sophisticated intelligent system performance is achieved by coordinating a collection of semi-autonomous subsystems, each with complementary capabilities [12]. Each subsystem has a well-defined command and control interface, and a supervisory control program coordinates the overall activities of the system through these subsystem interfaces. Individual subsystems may also possess real-time low-level control functions which can be performed autonomously and asynchronously. With the correct combination of supervisor and subsystem capabilities, such an approach supports the implementation of model-based control and sensor integration within reusable software structures. This approach also encourages the use of modularity, distributed multiprocessing environments, and standard commercial interfaces.

In order to build a GISC-based system, a set of tools is required for developing the supervisor and subsystems and also for integrating the pieces into an operational control system. Five software products have been developed to provide a range of capabilities required by an intelligent system. These include the Intelligent System Operating Environment (ISOE) which provides the low-level communication facilities. The General Interface for Supervisor and Subsystems (GENISAS) which provides the high-level communication framework needed for a distributed supervisor/subsystem arrangement [13]. GENISAS utilizes the client/server approach to break the system up into separate processes that communicate through messages. The Robot Independent Programming Environment and Language (RIPE/RIPL) enables development of generic subsystems by providing object-oriented interfaces to intelligent system devices [14]. The Sequential Modular Architecture for Robotics and Teleoperation (SMART) provides an underlying stable control system that has the performance and flexibility for sensor-based control and teleoperation, including force reflection [15]. The SMART component of GISC is also capable of shared and traded control modes with force reflection. Finally, the graphical programming environment, Sancho provides an easily reconfigurable menu-based operator interface and on-line simulation environment [16]. GISC has been built around the concept of on-line graphical simulation with subsequent operator approved robot motion and supervisory control with a graphical interface that is menu and mouse driven. In its current form, GISC is a loosely defined specification which allows more detailed interface specifications to be defined for each subsystem as the need arises. There is currently an effort to develop a detailed generic robot subsystem interface specification.

2.7: The Unified Telerobotic Architecture Project (UTAP)

The U.S. Air Force has begun a multi-phased project to develop a support infrastructure necessary to foster the creative development and innovative utilization of emerging telerobotic technologies for depot-level maintenance applications. NASA-JPL has completed phase 0 which prepared a study for defining a telerobotic architecture for

large aircraft maintenance. The report is a conceptual design for hazardous operations involving stripping, cleaning, finishing, and painting of large aircraft structures. The basis of the unified telerobotic architecture (UTA) focuses on the concept of a task sequence coupled with a world model [17]. There are three elements involved in system operation. First, the object modeling modules provide a model of the hardware and task environments. Second, the task description modules provide a means to build task programs. Third, the task execution modules perform simulations or drive devices. The hardware architecture is broken up into two locations: local and remote. The software is broken up into modules along functionality lines, which provides a means for easy system development, modification, and upgrades. For instance, the modules include an object knowledge base, object calibration, parent task program sequencer, subsystem task program sequencer, subsystem task level control, robot servo control, robot, etc. The report [17] concluded by recommending areas for additional study, one of which is a tradeoff analysis of the actual command/data types and structures with respect to module communication. The intent is to minimize the amount of internal data translation required.

Phase 1, underway this fiscal year, has tasked the National Institute of Science and Technology (NIST) to act as coordinator and prime contractor for the study of issues pertaining to the specification and validation of the architecture [6]. In a cooperative effort, NIST and JPL are examining the feasibility of implementing the initial JPL architecture and to develop preliminary interface specifications between all functional blocks in the unified telerobotic architecture (UTA). The output of this phase will be a working document that describes the interfaces and functional blocks of the UTA. Phase 2 will involve a systems integrator study that will analyze the UTA interface specification and determine how a UTA compliant system can be implemented to solve the representative application set, or suggest modifications to the portions where compliance is not possible. The contractor will then validate this analysis by designing a UTA compliant system and performing a validation test set. The design will be implemented on physical hardware using an Adept motion servo system and then demonstrate the interoperability and modularity of the architecture by replacing the Adept system with a Trellis motion servo system. The future phases of the UTAP are not sufficiently defined, but the UTA refinement process is planning to produce a request for proposals for an actual system in fiscal year 1997.

2.8: NOMAD

Trellis Software & Controls is developing NOMAD, a set of motion control software modules that can be combined with off-the-shelf computer hardware and software to build an open architecture robot or motion controller [18]. The base operating system for the product is LynxOS™, a POSIX compliant real-time version of UNIX. The system is programmable in C, with full UNIX compatibility, including X-Windows, Motif, NFS, and TCP/IP networking. A controller comprised of NOMAD and off-the-shelf computer components provides the full power of sensor-controlled robotic motion with user

replaceable kinematics. The current configuration contains three components: (1) a trajectory generator that provides sensor-controlled motion and a variety of industrial servo and I/O interfaces, (2) a user environment which consists of a C library for communicating with the trajectory generator along with system configuration tools and example programs, (3) several utilities such as the graphical servo tuning tool, machine simulator, command line interface, teach pendant support, and others.

2.9: ROBLINE

CIMETRIX, Inc. is developing an open architecture controller capable of controlling a diverse array of machines under a standards-based operating environment and built on industry standard hardware platforms [19]. This system also utilizes the Lynx Operating System. In addition to a main processor, each controller contains at least one digital signal processor for low level servo control and I/O monitoring. The controller uses a client/server model to isolate the application programming environment from the low level control functions. Client/server programming breaks the control system up into separate processes which communicate through messages. The server consists of multi-threaded processes for task sequencing, motion planning, and I/O. The server maintains a knowledge base for world models, control parameters, and other information. The clients can be other control system components, a general tool, or an application program. For off-line programming and simulation, the servo subsystem can be replaced with a graphics subsystem. A C library of utility functions is provided to the applications programmer. The library contains functions for motion control, I/O and process control, geometric modeling, user interface functions and many others. There is also the Non-Programmers Interface, which is a C interpreter Computer Aided Software Engineering (CASE) tool that allows users to select functions from an X-Windows program.

3: Conclusions and Recommendations

Ideally, a completely open architecture would be preferred. But because there is such a vast amount of hardware and software for the control of robot systems available and such rapid evolution of these systems, trying to standardize on a specific controller architecture will be difficult. An alternative approach is to use an incremental method by which a common, yet adaptable architecture could be realized [2]. This is the same methodology as the hybrid approach. The first step is to address standard requirements for individual technologies and the interfaces required to make them work together effectively. One method which has proven effective in many areas ranging from computer networking to robotic systems is a modular communications-based philosophy that provides building blocks for intelligent systems. As the number and type of modules increase and are made more robust and easier to use, then de-facto standards may evolve that are based on these common interfaces. This will also ensure adaptability. A market driven standardization similar to the IBM-PC could result.

Furthermore, if one compares most of the reviewed architectures, one conclusion is that they are overly

restrictive because they define individual module functionality. In other words, a vendor or a system integrator wouldn't have much flexibility to develop new or different modules as the need arose. A better method is to define the means of communication and the module interfaces, and let the system integrator choose or invent modules to fulfill the required functionality. Additionally, the architecture wouldn't become obsolete overnight due to technology improvements or new functionality required of the system. This all points to a communication-based philosophy such as used in the NGC approach, the GISC approach, or the CMU port-based object approach. Other control approaches that are similar to the above approaches are promising, but have not been tested in an advanced control architecture with multiple robots and multiple sensor systems because they are generally in the early stages of development.

Because of the rapid evolution of technology, a completely open architecture would be difficult to implement in practice. Thus, an effective alternative would be to adopt a hybrid architecture that will offer the following:

- A multiprocessing environment that is scaleable and based on a standard bus architecture.
- A multitasking/multi-threaded real-time operating system.
- The system software code should be written with a data abstraction capable language with object-oriented features that is structured in a modular fashion. The software should be also developed with the requirement of real-time operation.
- Standardize interfaces rather than subsystems. A communication-based philosophy should be required such that interfaces to the subsystems offer easily callable well-behaved subroutines or functions with source code delivered to the buyer.
- Adequate resources should be available for maintaining existing code and for developing new functionality.

4: Acknowledgments

I would like to thank Dave Miller for providing his paper regarding standards for intelligent robotic architectures [2]. It served as the basis for this paper.

5: References

- [1] Harrigan, R.W., "Automating the Operation of Robots in Hazardous Environments," *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Yokohama, Japan, July 26-30, 1993.
- [2] Miller, D.J., "Standards and Guidelines for Intelligent Robotic Architectures," *Proc of AIAA Space Programs and Technologies Conference and Exhibit*, September 21-23, 1993.
- [3] Red, W.E., Dixon, M., "Survey of Robotic Enabling Technologies for WINCO/DOE," *Study sponsored by WINCO/DOE and conducted for Idaho Nuclear Engineering*

Laboratories (INEL) under contract C85-110766 (HRS-200-90), September 1990.

[4] Albus, J.S., McCain, H.G., Lumia, R., "NASA/NBS Standard Reference Model for Telerobot Control System Architecture (NASREM)," *NIST Technical Note 1235*, April 1989.

[5] Martin Marietta, "Draft Volume I of Next Generation Workstation/Machine Controller (NGC) Specification for an Open System Architecture Standard (SOSAS)," *Document No. NGC-0001-13-000-SYS*, March 1992.

[6] Leahy, M.B., Petroski, S.B., "Unified Telerobotic Architecture Project Program Overview," *Submitted to the 1994 IEEE International Conference on Intelligent Robots and Systems (IROS)*.

[7] Hayes-Roth, F., Erman, L.D., Terry, A., Hayes-Roth, B., "Domain-Specific Software Architectures: Distributed Intelligent Control and Communication," *IEEE Symposium on Computer-Aided Control System Design*, pp. 117-128, Napa, CA, March 1992.

[8] Hayes-Roth, F., Erman, L.D., London, R., Terry, A., "Tools and Methods for Developing Distributed Intelligent Control and Management (DICAM) Application Systems," *Position Paper for the Joint Services Guidance and Control Committee (JSGCC) Software Initiative Workshop*, October 13, 1992.

[9] Saridis, G.N., "Architectures for Intelligent Controls," *IEEE Symposium on Implicit and Nonlinear Systems*, Ft. Worth, TX, December 14-15, 1992.

[10] Stewart, D.B., Volpe, R.A., Khosla, P.K., "Integration of Real-Time Software Modules for Reconfigurable Sensor-Based Control Systems," *Proceedings 1992 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Raleigh, NC, July 7-10, 1992.

[11] Stewart, D.B., Volpe, R.A., Khosla, P.K., "Design of Dynamically Reconfigurable Real-Time Software using Port-Based Objects," *Submitted to IEEE Transactions on Software Engineering*, July 1, 1993.

[12] Griesmeyer, J.M., McDonald, M.J., Harrigan, R.W., Butler, P.L., Rigdon, B., "Generic Intelligent System Controller (GISC)," *Sandia Internal Report, SAND92-2159*, Sandia National Laboratories-New Mexico, October 1992.

[13] Griesmeyer, J.M., "General Interface for Supervisor and Subsystems (GENISAS)," *Sandia Internal Report*, Sandia National Laboratories-New Mexico, October 1992.

[14] Miller, D.J., Lennox, R.C., "An Object-Oriented Environment for Robot System Architectures," *IEEE Control Systems*, Vol. 11, No. 2, February 1991.

[15] Anderson, R.J., "SMART: A Modular Architecture for Robotics and Teleoperation," *IEEE International Conference on Robotics and Automation*, Atlanta, GA, May 2-6, 1993.

[16] McDonald, M.J., Palmquist, R.D., "Graphical Programming: On-line Robot Simulation for Telerobotic Control," *Proceedings of the International Robots and Vision Automation Conference*, Detroit, MI, April 5-8, 1993.

[17] NASA JPL, "A Generic Telerobotics Architecture for C-5 Industrial Processes: Final Report," *Prepared for Air Force Material Command (AFMC) Robotics and Automation Center of Excellence (RACE) San Antonio Air Logistics Center, Kelly AFB, TX, 78241*, August 1993.

[18] Rossol, L., "Nomad Open Architecture Motion Control Software," *Proceedings of the International Robots and Vision Automation Conference*, Detroit, MI, April 5-8, 1993.

[19] Sorensen, S., "Overview of a Modular, Industry Standards Based Open Architecture Machine Controller," *Proceedings of the International Robots and Vision Automation Conference*, Detroit, MI, April 5-8, 1993.

DATE

FILMED

7/7/94

END