N90-29872

SYSTEM ARCHITECTURES FOR TELEROBOTIC RESEARCH

F. Wallace Harrison

Automation Technology Branch
NASA Langley Research Center
Hampton, Virginia

ABSTRACT

NASA currently supports several activities related to the definition and creation of telerobotic systems. The effort and investment required to create architectures for these complex systems can be enormous; however, the magnitude of process can be reduced if structured design techniques are applied. A number of informal methodologies supporting certain aspects of the design process are available. More recently, prototypes of integrated tools supporting all phases of system design from requirements analysis to code generation and hardware layout have begun to appear. This paper describes activities related to system architecture of telerobots at Langley Research Center including current activities which are designed to provide a methodology for the comparison and quantitative analysis of alternative system architectures.

1. INTRODUCTION

Much effort is presently being directed within NASA and other government organizations to creating architectures for telerobotic systems [1,2,3]; however, there remains a great deal of confusion over a precise definition of and the scope of the activities associated with the term system architecture. For example, how is system architecture related to computer and network architectures and operating, I/O and other systems? Does a system architecture define the organization of a system or does the organization of a system define a system architecture? We offer these definitions as a basis for the discussions in this paper. A system is an arrangement of things so related or connected as to form a complex or unitary whole. Further, we state that a system must possess a finite set of data, rules, facts, and principles organized and arranged in a regular, orderly manner so that a useful purpose is served. Architecture is the science and the art of construction and design; thus, system architecture is the term used to describe activities which ensure that systems are designed and constructed to meet these definitions. This paper presents a philosophy of telerobotic system construction as reflected in an approach to building telerobotic systems at Langley Research Center (LaRC) in the Intelligent Systems Research Laboratory (ISRL).

Section 2 of this paper is a general discussion of the activities related to system architecture in ISRL. Section 3 describes the Telerobotic System Simulation (TRSS), a real-time telerobotic simulation and run-time system for the investigation of telerobotic technologies. Section 4 discusses a Capability-based Architecture for Robotics (CBAR), a new architecture for building evolutionary, structured capabilities into telerobot systems. Section [5] briefly discusses current activities in system architecture designed to provide the tools for the comparison and quantitative analysis of alternative system architectures. We conclude by summarizing the lessons we have learned to date.

2. Intelligent Systems Research Laboratory

Telerobotics consists of a huge number of highly specialized and interrelated disciplines and technologies. The objective of automation research in the Automation Technology Branch (ATB) at LaRC is to advance technology in specific technology areas (mechanisms, controls, sensors, and operator interface) required for space-based assembly, servicing, and inspection systems [4]. However, meaningful progress in most of these areas depends on having at least a base-level capability in and understanding of the range of telerobotic technologies. The Teleoperator and Robotics System Simulation (TRSS) and the Intelligent Systems Research Laboratory (ISRL) have been developed to provide these base-level capabilities and is structured to allow and promote evolutionary development of telerobotic technologies. Development of TRSS began in 1981 by a small group of researchers and programmers with the objective of investigating the effects of transport delays on operator performance [5]. The graphical simulation ran on Control Data Corp., Cyber 175, used a simulated five degree-of-freedom manipulator, and used displays and controls in a general purpose aircraft simulator.

Parallel to the TRSS development, ISRL was established as a tool to investigate planning systems for robotic systems in a more realistic setting than a purely graphical simulation could provide. Early experiments were directed toward understanding the relationship between perception, reasoning, and manipulation in a simple blocksworld environment for autonomous robots as described in reference [6].

ISRL is organized as a distributed, hierarchical collection of teleoperator and robotic hardware. Each major subsystem consists of a user programmable controller and data communications hardware. In addition to their primary function each subsystem can be used as a program development system or as a general purpose computational element. Primary real-time communications occurs on a 250,000 byte per second packet-switching global bus conforming to the IEEE 488-1978 standard. Device drivers have been written to support demand-driven, priority-based network access and shared file system access. The current network configuration is illustrated in figure 1. Primary subsystems consist of two PUMA manipulators, a vision and laser ranging system, and an operator interface. Interfaces to a Symbolics 3670 computer, a Control Data Corporation Cyber 175 and a Redifusion Poly 2000 high-speed graphics subsystem are provided.

Manipulator and controller. The two digitally-controlled PUMA robotic arms driven by direct current servo systems, provide manipulator functions in ISRL. The PUMA, typically used in "pick-and-place" industrial applications, is a six degree-of-freedom (DOF) anthropomorphic manipulator. It has been augmented with a parallel jaw gripper and a six DOF force torque sensor. In its factory configuration, control is provided by a hierarchical controller composed of a master and six slave microprocessors, each slave providing low-level proportional, integral, derivative (PID) control of one joint of the manipulator. The master controller is an LSI 11-03 microprocessor with 4000 to 32000 bytes of random access memory for user program storage and 28000 to 30000 bytes of read-only memory containing the VAL operating system.

While the controller and VAL are adequate for many industrial applications, its structure and programming do not allow the flexibility necessary in a research environment. To obtain this flexibility, the master controller was replaced with a general purpose computer (LSI 11-73); and, using VAL as a reference, new software was generated with the necessary capability. Computational and input/output facilities are enhanced with the addition of a special-purpose peripheral device drivers, while more general capabilities, such as manipulator initialization, position and rate control, coordinate transformations, and extended I/O are provided by a library of FORTRAN functions.

End Effector. Grasping functions are provided by a microprocessor-controlled parallel jaw end effector having integral force, proximity, base overload, and crossfire detectors. Finger force and torque about the X and Y axis can be sensed. Proximity sensors are simple binary, infrared reflectivity-based emitter-detectors. Cross-fire detectors detect the presence of a work piece between the jaws by interruption of a light beam.

One of the first telerobotic studies in ISRL was an active compliance task 5]. The finger force/torque sensors sensed constraint forces during close tolerance peg insertion and fed this data to control and display modules via the data acquisition system. A simple computer graphics display indicated the magnitude and direction of the binding forces and torques. Using the display the operator could readily command the arm to move to null any disturbing forces. A subsequent modification allowed the operator to select a mode in which the force and torque data were fed directly to the control system to null the force and torques automatically.

Vision. Current ISRL image acquisition and analysis approaches partition this problem area between man and machine, giving each responsibility for that which it does best. Man serves as the basic mechanism for image interpretation and understanding, while the machine vision system performs image acquisition/enhancement/compression and determines the location of objects. Image acquisition is provided by a Data Cube imaging system interfaced to a Micro-Vax II general-purpose processor. Current efforts have centered on determining the three-space location and orientation of labeled objects using a single camera, and research on a multifunction recognition operator for telerobotic vision [7].

The operator is responsible for moving the camera to acquire a labeled object in the field of view. Once acquired, simulation modules are provided for vision-based control of the manipulator. Vision-based control was demonstrated in a recent satellite servicing simulation. After acquiring a label (in this case, a pattern of light emitting diodes) associated with the simulated experiment module, the vision system determined the relative location of the module with respect to the camera and the end effector, and then commanded the manipulator to move to the module under control of the vision system [8].

A coherent laser scanning system has recently become available in ISRL. Laser scanning systems promise high accuracy ranging with television-like displays over wide ranges of ambient illumination [9]. Current efforts are concentrated on generating accurate representations of the manipulator environment for path planning and collision avoidance.

In 1983, TRSS was interfaced to ISRL. Kinematics of the simulation were converted to those of a PUMA 560 and the capability for force control was added. In 1985, automatic control based on vision sensing was added. All TRSS control strategies, based on resolved-motion rate control, are developed as shared control. That is, all automatic control schemes must continuously share control with the operator; thus the operator is free at any time to "help" the automatic system. Conversely, the automatic system is free to aid the operator in task completion. The system requires no complicated semaphores, signaling mechanisms, or logic to switch between automatic and teleoperator control. The most significant feature of the TRSS control strategy is that the reference signal inputs of the manipulator control system is formed as linear combination of outputs from any number of control/sensor modules. The most obvious advantage of this approach is that sensors and associated control modules may be distributed, both spatially and temporally, across multiple processors. Multirate control is almost trivial to implement and the system has some intrinsic fault tolerance. If sensor communications fails, motion based on that sensor output stops.

3. Telerobotic System Simulation

In 1986, a requirement for task-referenced control of multiple manipulators was generated. To facilitate the implementation of these capabilities, the simulation was ported from the CYBER

175 to a VAX/11-750 located in ISRL and a critical examination of existing TRSS code (a single thread of code executed once each clock cycle) and system organization was conducted as part of this activity. The major conclusion of this examination was that the simulation needed to be more modular in organization. An objective in TRSS is that it provide basic capabilities in specific technology areas that are of marginal interest to a researcher. Modularization hides the implementation details of these uninteresting, but interacting, modules so that a researcher need only to understand its behavior and interface to utilize its capabilities. Another conclusion was that top-down design was inappropriate for the target environment. Providing capabilities where none existed before is often the prime objective of a research project. This implies that fixed requirements specifications and hardware architectures are often not available and that even the high-level organization of the simulation may change frequently. TRSS should provide mechanisms to make the reorganization of the simulation possible. A third conclusion was an extraordinarily high percentage of development time was being spent by research personnel on hardware interfaces and communications software. The Teleoperator and Robotics Testbed (TART) was conceived and implemented in response to these limitations. TART consists of a baseline of standard modules and interfaces, termed capabilities, and a logical organization which defines the operating characteristics of a telerobotic system. Capabilities facilitate design, coding, and testing of algorithms and aid the integration of alternative algorithms and software from other sources.

The TART system architecture, as illustrated in figure 2, consists of six layers (L1 through L6), each supported by the functions and capabilities of the layer below. To the programmer, each boundary defines an abstract machine on which the functions of the next higher level are defined. Layers accept commands from and provides feedback to the layer above. The lowest level of TART, the sensor/actuator layer L6, is defined and fixed by the communications protocols and physical characteristics of the collection of devices available in ISRL (see discussion above). Servo level control of manipulators, camera systems, and end effectors and processing of raw sensor data is provided at L5. In some cases, this processing at this layer is provided by vendor-supplied equipment. In other cases, such as machine vision, algorithms are provided by ATB researchers. Inputs to the sensor processing and servo control layer from L4 are the commanded positions and orientations and/or their derivatives for manipulators, grippers, pointing systems, and other devices in their local coordinate system. L5 provides preprocessed sensor and control states to the next higher layer. The electrical voltages and currents to drive motors and actuators are output to L6. If data rates require the interpolation of set points, this is provided in L5.

The communications layer, L4, maps the input and output from the servo/sensor processes to a consistent unambiguous representation. The data structures for all similar devices are required to be identical in form with data scaled to the same units. Thus L4 isolates higher levels from the eccentricities of the underlying hardware and communications protocol. For example, all Cartesian force/torque sensors have a uniform representation which includes a signal indicating overload conditions. If a particular force/torque sensor does not generate this signal in hardware, it must be synthesized in software. Data conversion from device measurement units to laboratory units is also done at L4. The communications layer is maintained by programming support personnel who have experience in dealing with real-time communications. The virtual telerobot architecture provided by L4 functions much like the set of registers available to machine language programmers of computer systems. Just as one does not have to understand the microprogramming and internal data paths of a computer to program it, L4 isolates the researcher from much of the low level programming of the telerobot.

The capabilities required to make the devices in ISRL perform as a system are provided in the coordination and control layer L3. Commands received from L2 are distributed among a number of control capabilities (vision, force, etc.) based on the task to be performed. Tasks that require more than one resource, ie. multiarm control or a compliant grasp, are coordinated in L3 by simultaneously enabling multiple control capabilities with appropriate gains. Coordinate transformations from world to local device coordinates are performed here. Since the capabilities at this level, such as force or vision control, are defined with respect to the TART virtual architecture, the programmer can focus on algorithm development.

Task primitives (move arm(s), close end effector, etc.) are executed by the task level, L2. Capabilities in L2 are activated by commands from L1 and enable appropriate control modules in L3 by setting variables in global memory. L2 then monitors the system as the task is performed and reports success or failure to L1. Each task primitive is implemented as a subroutine and maintained in a library which is directly linked with the TRSS operator interface program or other applications. Levels L3 and L2 taken together provide a capability similar to the intrinsic functions of a compiler.

L1 provides the interface between external systems, both man and machine, and the remainder of TART. Displays for monitoring system status and a command interpreter to decompose high level commands (move to or grasp an object) into sequences of task primitives are provided. L1 provides the remainder of the capabilities found in the typical compiler or interpreter. New task primitives and sequences of commands can be executed, debugged, and evaluated in the operator interface command language before being programmed as an L2 module. For a discussion of the use of TART by external system see [6].

These six levels, their functional descriptions and their logical organization, define the TART system architecture. It should be noted that any implementation is a compromise among a number of competing requirements (speed, ease of use, cost, etc.). The principle function of the TART implementation architecture is to support telerobotics research, and ease of use and reduced cost are emphasized at the expense of performance. In general, layers 4 and above are implemented on a single MicroVax II microprocessor under the VMS operating system. The virtual machine architecture is implemented as a installed shareable image making its data structures global to all processes. Each layer of the architecture is implemented as one or more VMS processes and each process, once started, is responsible for its own scheduling using the mechanisms offered by the VMS operating system. Every capability is implemented as a separate process. To the researcher, this means that refinements and alternatives to algorithms can easily be investigated by stopping the current process and starting a new. However, the requirement that each capability in the TART architecture be a separate process is a major source of overhead and the global accessibility of the virtual architecture can lead to abuse of the TART design philosophy and subtle programming errors. The Capability-Based Architecture for Robotics (CBAR), is being developed in response to these problems.

4. A Capability-Based Architecture for Robotics (CBAR)

The system architecture of CBAR is based on a more formal definition and representation of a capability. The motivation for creating the abstraction of the capability is related to human limitations in the ability to manage and understand information and control flow in the design of complex systems. To overcome this limitation, capabilities have clearly defined function and operate on a small set of interface data structures with control flow limited to that implied by changes in each capability's data structures. No explicit command/response mechanism is required. CBAR encourages the use of modularization in the design process and is amendable to both top-down (recursive application of the abstraction) and bottom-up (utilization of existing abstractions) design practices. At all levels of the system design, both function and interface are represented in only sufficient detail to understand the immediate design and implementation problem. Both the designer's and implementer's ability to efficiently create systems that behave properly and are easy to maintain and document is enhanced.

A capability, C, is defined by its set of data structures and an associated transformation and is represented in figure 3 where P is called the planning data structure, Q is the query data structure, R is the output data structure, S is the sensor data structure, T is a capability transformation, and A is a set of attributes. Functionally, a capability transforms S, the world as seen through its sensor data structure, to P, the world as we wish it, by transitioning through a series of states Q while producing output R to effect changes on its environment.

To better understand the representation, consider a control engineer's model of a simple feedback controller capability (see figure 4). The controller has some state (Q) and an algorithm (T) which forms an output (R) to control a plant based on a set point (P) and feedback from sensors (S). A more general interpretation can be taken from planning systems. Here a particular situation or configuration is a problem state (Q) which is derived from sensor data (S) and previous states. An operator (T) transforms a given state into another state while producing output (R). A solution to a problem is a sequence of operators that transforms an initial state into a goal state (P). Complex functional capabilities can be generated by repeated application of the capability abstraction as illustrated in figure 5. The behavior of a system at any given time is described by the set of active capabilities and their interconnections and can be represented as a lattice.

The flow of data in the CBAR lattice is bidirectional and nonstationary. Goals flow from the top of the lattice to the bottom through the P data structures with increasing levels of detail. State information flows from bottom to top through the Q data structures with decreasing detail. The form of the lattice is determined by the active set of goals and states.

Two degenerate capabilities that are of interest are best explained by an example. Consider the feedback controller discussed above used to control a plant consisting of an amplifier-driven motor and position encoder. The problem, depicted in figure 6, is to generate a set of motor currents to drive the motor through amplifier A to a state sensed through encoder E. C3, called an input capability, transforms (not necessarily linearly) from sensor E internal units to a more convenient representation (angular displacement, velocity, etc.) and makes the coding of C1, the control algorithm, much more tractable. C2, an output capability, functions in a similar manner but generates actuator control signals.

Dynamic reconfiguration of the lattice is accomplished through contact and disconnect mechanisms. A capability ready to execute is said to be contacted. Capabilities are contacted by a name, C, assigned as they are created (currently the file name qualified by subroutine name) with a desired set of access rights. The access rights are compared with the current accessibility attributes and the contact is either allowed or denied. If a contact is allowed, the capabilities exchange information regarding the location of their data structures and a count of the number of contacts is incremented.

Links in the lattice are broken when a capability is no longer required via a disconnect mechanism and a capability with no contacts is said to be disconnected. A capability receiving a disconnect signal decreases its contact count and, if zero, terminates execution.

Accessibility attributes refer to the permissible read and write access modes applied to the data structures of a capability and are placed to the right of the directed line segments. Any capability always has read access to its own planning data structure (P) and write access to its query data structure (Q); however, external access to P and Q are strictly enforced. Read access (RA) means that only a single external routine can read a capability's Q data structure and implies that a capability can be contacted by only a single external source. For example, in figure 6 capability C1 has exclusive ownership of C2 preventing other processes from controlling the amplifier drive signal. Write access (WA) means that only a single external process can write to a capabilities P data structure, but is not sufficient in itself to insure sole ownership. Read shared (RS) and write shared (WS) access means that multiple external routine have access to a capabilitiy's P and Q data structures respectively. In figure 6, capability C3 is contacted by C1 with RS access implying that other external routines may access and monitor the position and rate of the motor.

Consider an expanded three level implementation of the previous motor controller example in figure 7. Levels L2 and L3 are identical in form and function to figure 6; however, an additional layer L1 has been added to monitor the controller and provide a status indication to external routines. The sensor capability, C3, has been contacted with RS access to allow simultaneous access to its Q data structure by both C1 and C4. Note that C4 could have monitored position and rate through Q3 but this results in a communications delay, less reliability, and more compute overhead.

5. CURRENT WORK

Predicting the performance of architectures for complex systems during early design and development is a difficult and demanding task. Typically, accurate quantitative measures are unavailable until late in the design process leaving many key decisions to rest on the experience and prejudices of the designer. Until the subjective elements of the process can be significantly reduced, the design and implementation of system architectures will remain more an art than a science. Today, it would be unthinkable to attempt the design of a computer system without appropriate computer-aided design tools. The quality and quantity of tools for the VLSI design have increased dramatically since their introduction in the 1960's. More recently, similar tools have been introduced for software design; however, few tools to aid the integration of both hardware and software in the creation of complex system (such as telerobots) exist. The Architecture Design and Assessment System (ADAS), a set of computer-aided design tools supporting system design from initial concept through hardware/software implementation, has been developed by Research Triangle Institute (RTI) [10]. ADAS is a collection of tools that can be used to identify pathologies early in the design process so that alternatives can be created and analyzed. LaRC and RTI are currently modeling TRSS and the TART architecture using ADAS with the objective of developing a methodology for the comparative analysis of telerobotic architectures.

Using the ADAS graphical interface, the hardware and software designs of three systems: TRSS/TART, TRSS/CBAR, and the FTS/NASREM are being modeled and analyzed. TRSS/TART, because of the availability of detailed performance data, will be used to validate the modeling and analysis techniques. ADAS is also an integral element of the CBAR design and implementation. The design will be captured as data flow graphs and mapped to hardware. All design decisions will be evaluated via simulation to pinpoint bottlenecks and determine the best partition between hardware and software. As details of the NASREM/FTS architecture are made available, functional simulations will be conducted to validate the design.

The CBAR implementation architecture is currently under development. A simplified representation of TRSS control architecture in CBAR form is being used as a model for the first prototype. Major design decisions are being evaluated using the Architectural Design and Assessment System to provide quantitative performance analysis before implementation.

6. CONCLUDING REMARKS

At LaRC, architectural endeavors have evolved into two distinct activities. System architecture activities are concerned with the functional characteristics and the logical organization of a system. As such, it is principally a conceptual and philosophical activity that results in a system design specification, a detailed understanding of what the system is to do and how it is logically organized. The implementation architecture phase translates the system design specification into a detailed implementation plan by mapping the system architecture onto the hardware and software subsystems. The partitioning has important practical implications. Partitioning the architectural design makes each phase more tractable by limiting the number of degrees of freedom in each step; and, properly done, new device technologies and algorithms can be incorporated into the system without scrapping or compromising the entire design specification. For example, several families of computers have been designed which share common system architecture with performance ranges of 1-100 and yet most software written and compiled and linked on one machine can run on any other family member.

The role of the system architect in a research environment is at best difficult. The system architect is typically concerned with topics such as requirements engineering, design specification, implementation architectures, testing, validation, and maintenance. It is sufficient to say that to most researchers, these topics are not of paramount importance, yet the difference between a project's success and failure is often correlated with the quality of its supporting systems engineering. Certainly, the utility of the project's results to others is enhanced when sufficient attention is given.

What are some of the attributes of a "good" system architecture? First, there is a consistent design philosophy. Well-defined software and hardware structures with a carefully designed minimized rule set are the result and enable programmers to create efficient and maintainable applications. High-level languages and adequate documentation are absolute requirements. We maintain that the current debate over the need for a "standardized" telerobot control system architecture is unnecessary and distracting. With imagination, thought, and commitment we can build systems which instead of stifling creativity, will encourage it; instead of locking us into proprietary systems, will facilitate their introduction; and instead of being rigidly confining, will be easily extendable.

BIBLIOGRAPHY

[1] Albus, James S., McCain, Harry G, Lumia, Ronald: "NASA/NBS Standard Reference Model for Telerobot Control System Architecture (NASREM)," National Bureau of Standards, NBS Technical Note 1235, July 1987.

[2] Matijevic, J. and Dolinsky, S.: "Functional Requirements for the Telerobotic Testbed Project," Jet Propulsion Laboratory, JPL internal document D-3693, December 1988.

[3] Harrison, F. Wallace, and Pennington, Jack E.: "System Simulations Supporting NASA Telerobotics," presented at the Workshop on Space Telerobotics, Pasadena, California, January 1987.

[4] Sliwa, Nancy E.: "Telerobotic Research at Langley Research Center," presented at the Space Operations Automation and Robotics Conference, Houston, Texas, August 1987.

[5] Pennington, Jack E.: "A Rate Controlled Teleoperator Task with Simulated Time Delays,"NASA Langley Research Center, TM-85653, September 1983.

[6] Orlando, Nancy E.: "An Intelligent Robotics Control Scheme," presented at the American Controls Conference, San Diego, California, June 1984.

[7] Goode, Plesent W, and Cornils, Karin: "Monovision Techniques for Telerobots," presented at the Workshop on Space Telerobotics, Pasadena, California, January 1987.

[8] Cornils, Karin and Goode, Plesent W.: "Location of Planar Targets in Three Space from Monocular Images," presented at the Goddard Conference on Space Applications of Artificial Intelligence and Robotics, May 1987.

[9] Goodwin, F. E.: "Coherent Laser Radar 3-D Vision Sensor," Proceedings of Sensors '85, November 1985.

[10] Ingogly, W. F., McLin, D. M., Morrill, R. R., Frank, G. A., and Franke, D. L.: "The Evaluation of 1750A Hardware Implementation Using ADAS," NASA Langley Research Center, CR-178247, January 1987.

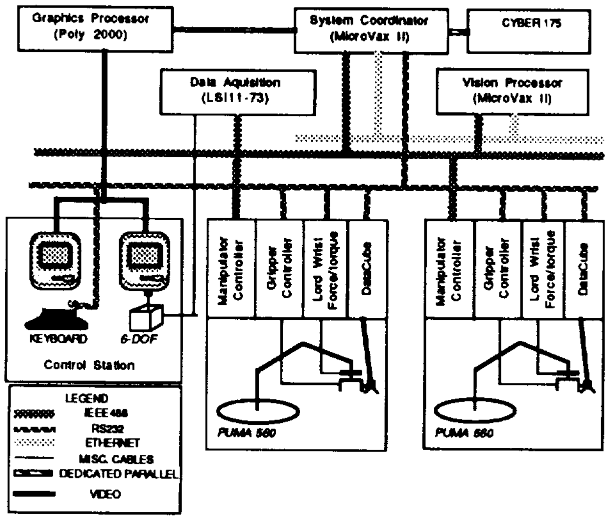

Figure 1. ISRL Physical Layout.

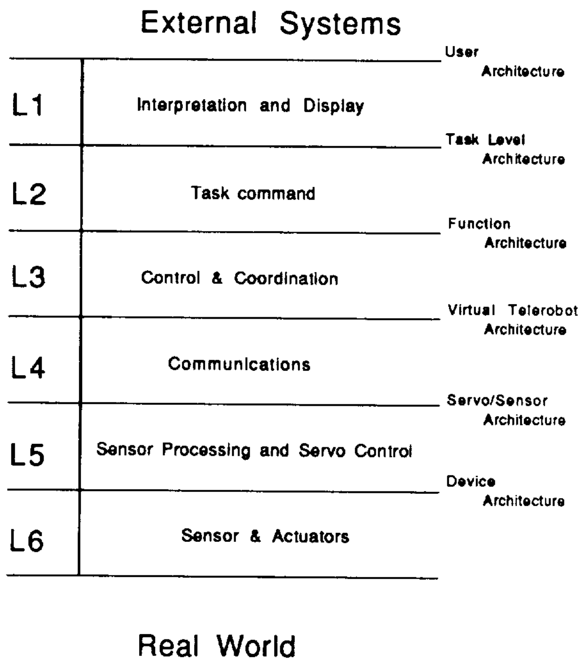

Figure 2. The TART System Architecture.

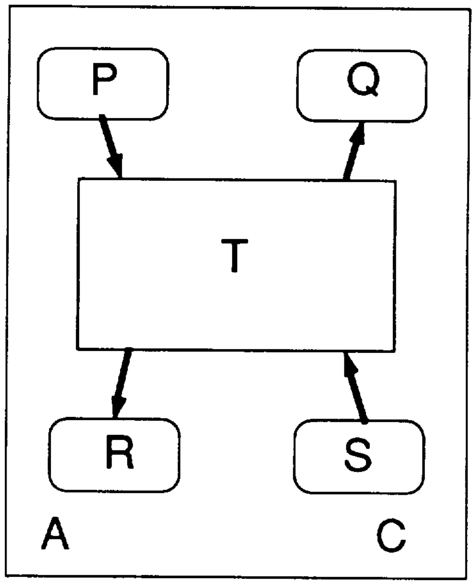

Figure 3. Capability Representation.

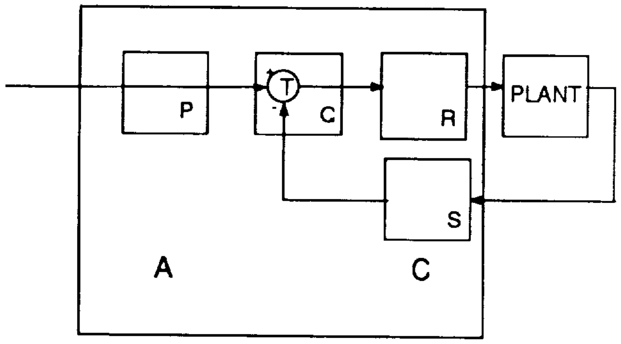

Figure 4. Control system interpretation of a capability.

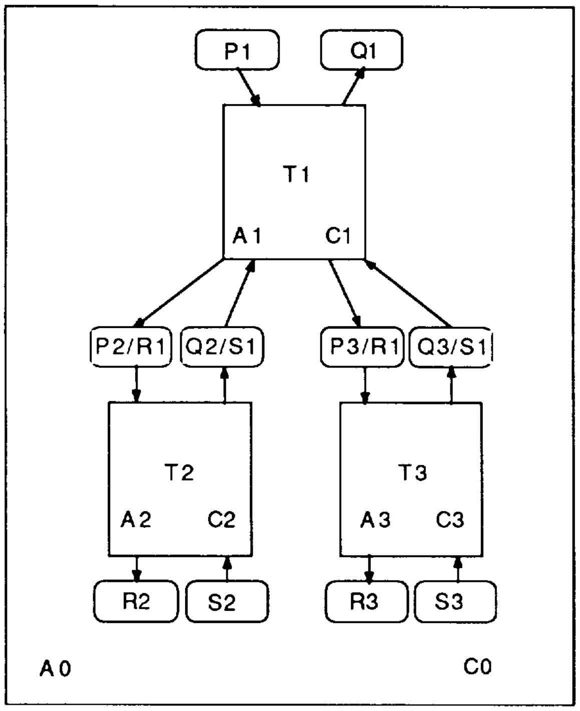

Figure 5. Hierarchical Capability Representation.

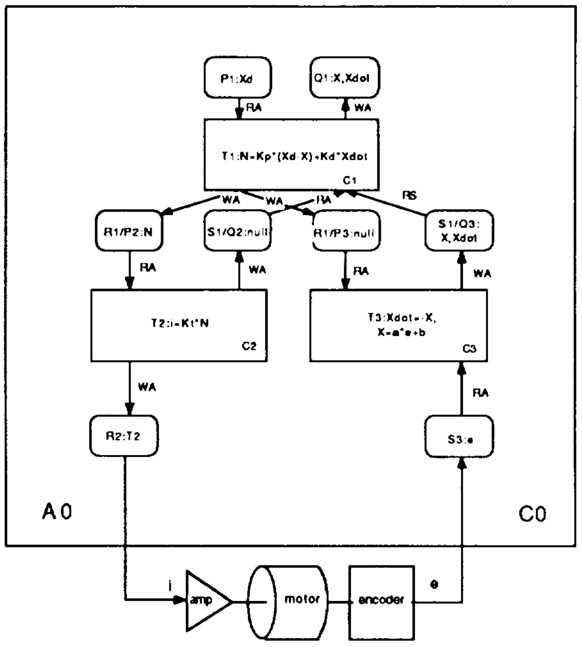

Figure 6. Simple control system.

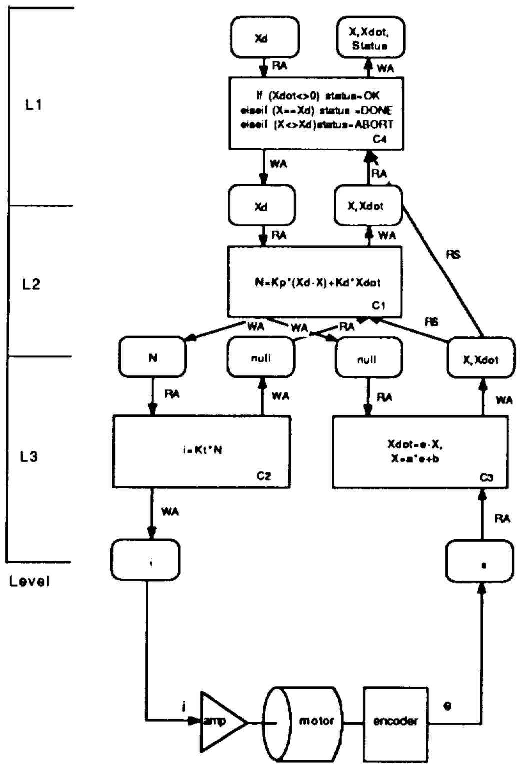

Figure 7. Expanded control system design.