NISTIR 5412

An Overview of NASREM: The NASA/NBS Standard Reference Model for Telerobot Control System Architecture

James S. Albus
Richard Quintero
and
Ronald Lumia

U.S. DEPARTMENT OF COMMERCE
Technology Administration
National Institute of Standards
and Technology
Robot Systems Division
Bldg. 220 Rm. B124
Gaithersburg, MD 20899

NIST

NISTIR 5412

An Overview of NASREM: The NASA/NBS Standard Reference Model for Telerobot Control System Architecture

James S. Albus
Richard Quintero
and
Ronald Lumia

U.S. DEPARTMENT OF COMMERCE
Technology Administration
National Institute of Standards and Technology
Robot Systems Division
Bldg. 220 Rm. B124
Gaithersburg, MD 20899

April 1994

AN OVERVIEW OF NASREM: THE NASA/NBS STANDARD REFERENCE MODEL FOR TELEROBOT CONTROL SYSTEM ARCHITECTURE

James S. Albus*
Richard Quintero**
Ronald Lumia***

Robot Systems Division
Bldg. 220, Room B-124
National Institute of Standards and Technology - NIST
(Formerly the National Bureau of Standards - NBS)
Gaithersburg, Maryland 20899

ABSTRACT

The NASA/NBS Standard Reference Model for Telerobot Control System Architecture (NASREM) was developed by the National Institute of Standards and Technology (NIST) for the National Aeronautics and Space Administration (NASA) to provide a software control system architecture guideline for use by development contractors charged with building the Flight Telerobot Servicer (FTS) control system as part of the Freedom Space Station project. The original NASREM document describes a conceptual or domain-independent architecture, and suggests the outline of a functional or domain-specific architecture for FTS. This paper presents an overview of the NASREM conceptual architecture and reviews subsequent work at NIST in defining a functional architecture for the servo and primitive levels. This work suggests outlines for software and hardware architecture specifications, and software development environments to complement the NASREM conceptual and functional architectures.

BACKGROUND

One of the more well known National Institute of Standards and Technology (NIST) architecture definition efforts was the development of the NASA/NBS Standard Reference Model for Telerobot Control System Architecture (NASREM).[1] NASREM was developed by NIST for the National Aeronautics and Space Administration (NASA) to provide a software control system architecture guideline for use by development contractors charged with building the Flight Telerobot Servicer (FTS) control system as part of the Freedom Space Station project.

NASREM represents the culmination of more than a decade of research at NIST on Real-time Control Systems (RCS) for robots and intelligent machines. The first version of RCS was developed for laboratory robotics and adapted for manufacturing control in the NIST Automated Manufacturing Research Facility (AMRF) during the early 1980's.[2,3,4,5,6] Since 1986, RCS has been implemented for a number of additional applications, including the Defense Advanced Research Projects Agency (DARPA) Multiple Autonomous Undersea Vehicle (MAUV) project,[7] the Army Field Material Handling Robot (FMR),[8] and the Army TEAM (Technology Enhancement for Autonomous Vehicles)[9] semi-autonomous land vehicle project. In 1987, RCS was adapted for use on the space station Flight Telerobotic Servicer, becoming the NASA/NBS Standard Reference Model Telerobot Control System Architecture (NASREM).[1]

WHAT IS A CONTROL SYSTEM ARCHITECTURE?

The Random House College Dictionary[10] defines ***architecture*** as "the character or style of building; the structure of anything." NASREM outlines a style of building real-time control systems for intelligent machines. These systems generally include software, hardware, machines, people, communications, information repositories, information/knowledge models and real-time execution models as shown in Figure 1. NASREM defines a highly structured, modular organization of these control system components which can serve as a standard reference model for an open-system architecture. These properties are discussed in detail by Quintero and Barbera.[11*]

---

* Division Chief, Robot Systems Division
** Group Leader, Unmanned Systems Group, Member AIAA
*** Group Leader, Intelligent Controls Group

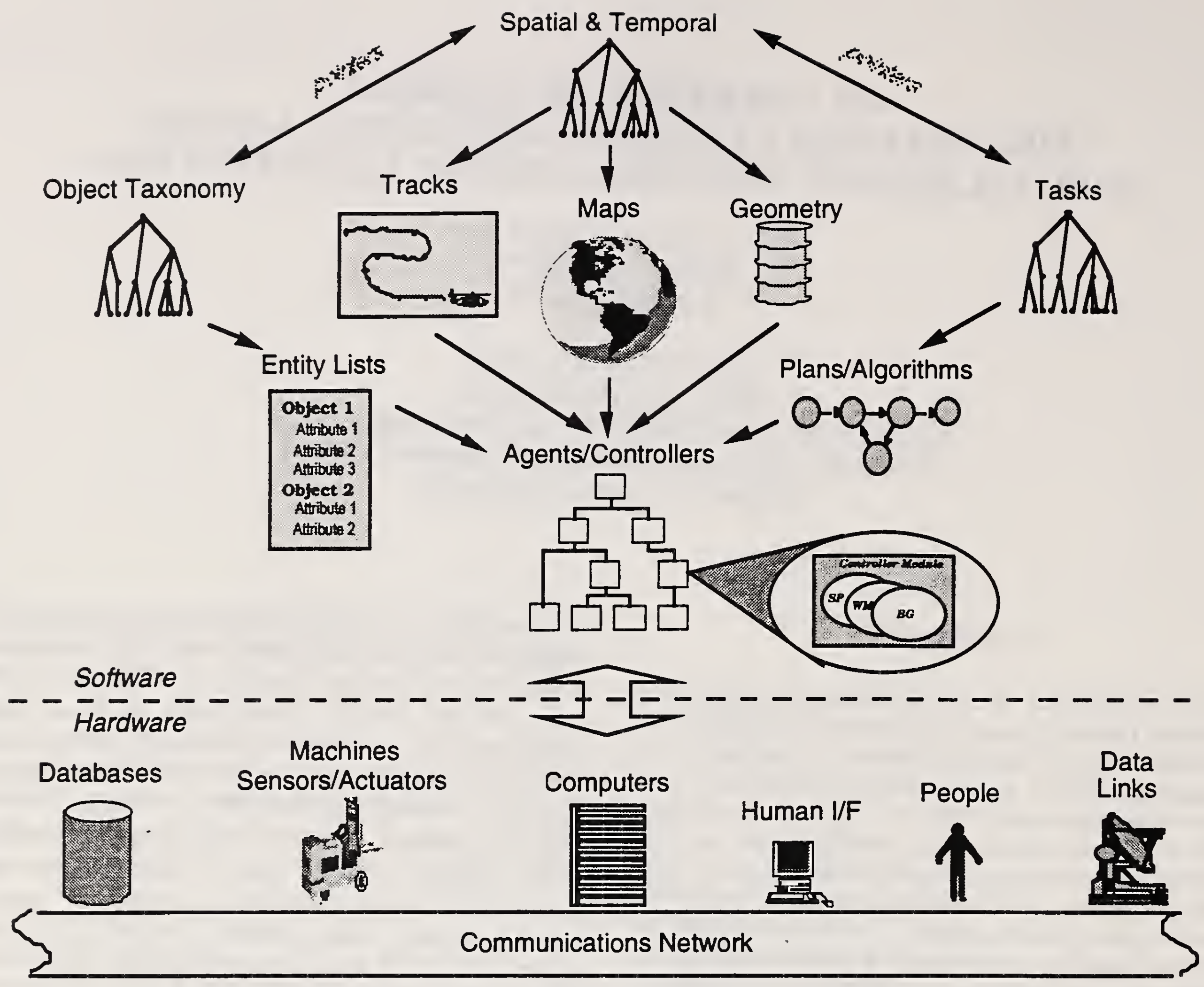

Figure 1. Architecture Components

THE PROBLEM DOMAIN

The NASREM conceptual architecture addresses the broad domain of intelligent machine control systems problems. We define ***intelligent machines*** to be machines designed to perform useful physical work while employing in situ knowledge (sensory input data), and a priori knowledge, tactics and strategy. Intelligent machines use feedback from the physical environment to manifest "intelligent behavior" in real-time via computerized real-time control of the machine's electro-mechanical actuators and sensors. In addition, we believe that practical intelligent machines almost always require some level of human interaction. The definition given above is intended to include: ***automation systems, embedded systems,*** and ***robotic systems*** ranging from factory floor robots to space vehicles and planetary exploration robots.

Kramer and Senehi have defined several tiers of architectural definition.[12] A conceptual architecture model is a high tier definition which is typically domain and application independent. A conceptual architecture can be applied to a broad range of application domains and typically does not specify any particular hardware, software, or communications mechanisms. A functional architecture, as defined here, is a mid-level model which is domain-specific. At the lowest tier of definition an architecture is fully implemented for a particular application with all elements completely specified (domain, application, and implementation specific).

In order to actually develop an intelligent control system implementation, systems engineers need a software development environment with an embedded set of methods or guidelines (and software libraries) to allow them to easily evolve a control system design. The process involves specifying at least four architectures:

1. Conceptual architecture
2. Functional architecture
3. Software architecture
4. Hardware architecture

The original NASREM document describes the conceptual or domain-independent architecture, and suggests the outline of a functional or domain-specific architecture for FTS.

NASREM: The Conceptual Architecture

The NASREM conceptual architecture is founded on the RCS view of intelligent machine architectures. The elements of an intelligent machine are modeled as a closed-loop control system as shown in Figure 2. A closed-loop system is formed in the machine by inputting sensory data to ***Sensory Processing (SP)***, passing the processed information off to ***World Modeling (WM)***, which maintains the machine's best estimate of the state of its world, and finally closing the loop through ***Behavior Generation (BG),*** also referred to as ***Task Decomposition (TD),*** which plans and executes actions to be performed through the machine's actuators.

An intelligent machine may also utilize a value system in order to judge the "goodness" of the results of its actions within the context of the tasks it is expected to perform. The value system, or the ***Value Judgment (VJ)*** function, is used in goal selection to direct Behavior Generation in selecting alternative plans and actions.

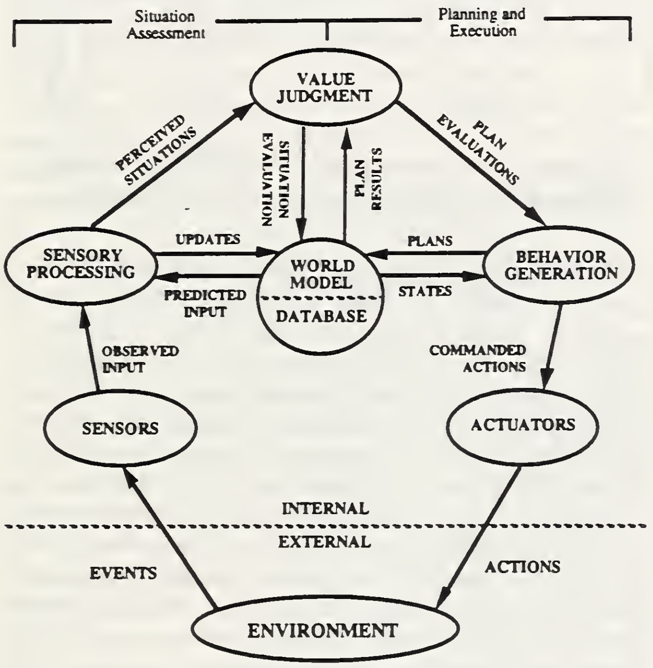

Figure 2. An Intelligent Machine System

The terms, ***World Model*** or ***World View,*** are used to describe the intelligent machine's collective capability to perceive the world in which it functions (both external and internal). When we use these terms we are referring to algorithms for understanding the world, WM server functions, and the information stored in the Global Memory.

Global Memory (GM) is the complete collection of globally defined variables in a NASREM application. It may be thought of as the repository or knowledge base where shared information is stored. In many applications Global Memory is implemented in a distributed fashion. GM can also be viewed as a combination of the communications mechanisms and the repository (or simply the interface buffers) necessary for implementing a NASREM application.

The NASREM Hierarchy

NASREM extends the notion of an intelligent machine model containing the basic SP, WM and BG by creating a hierarchy (see Figure 3.). The basic SP, WM, and BG functions are grouped as ***controller nodes*** and distributed in a hierarchically organized, integrated set of nodes. In a NASREM implementation a node is a collection of one or more software modules. Each controller node is assigned a set of tasks at an appropriate level of abstraction and each has a limited range of authority and responsibility within the chain-of-command formed by the hierarchy (much like a human military command structure would be organized). It should be noted that only commands and status are constrained to flow in a hierarchical fashion between supervisory and subordinate modules in NASREM. Global memory data transfers are triple buffered[11] and may be communicated in any appropriate manner (e.g., point-to-point, multi-cast, broadcast, etc.), as dictated by the design requirements.

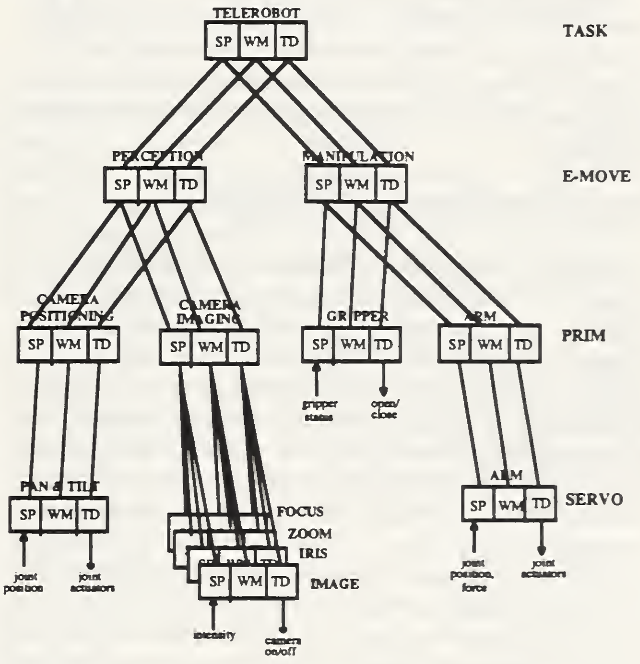

Figure 3. NASREM Hierarchy Tree

The ***Task Decomposition (TD)*** function is further decomposed into ***Job Assignment (JA), Planner (PL), and Executor (EX)*** functions. ***Job Assignment*** involves commanding subordinates to carry out concurrent tasks. Stored or generated plans temporally decompose tasks into sequences of sub-goals to be accomplished, to the limit of the appropriate planning horizon for a level. ***Planners*** are responsible for selecting pre-stored plans and/or generating new plans to be instantiated by the Executors. ***Executors*** instantiate the next step in the current plan based on the current state of the world as viewed via the World Model. Executors pass instantiated task commands to the next lower level JA, where this pattern is repeated, down the hierarchy, in a pipelined refinement of task detail. In general, subordinate levels deal with less abstract task details, at faster sub-goal completion rates.

Temporal decomposition in a NASREM design deals with planning horizons, memory spans and goal completion rates which are subdivided by roughly an order of magnitude in time between levels. Planning horizons and memory spans increase as we move to higher levels of the architecture while sub-goal accomplishment rates increase as we traverse the hierarchy towards the lower levels.

In NASREM, there is a notion of decomposing the control system design into layers or levels of abstraction. NASREM specifies the types of tasks carried out at each level of abstraction, starting at the bottom of the hierarchy.[1,11] For the Flight Telerobot Servicer (FTS) application these levels have been labeled as follows: **Level 1 - Servo, Level 2 - Primitive or Prim, Level 3 - Elementary Move or E-Move, Level 4 - Task, Level 5 - Service Bay, and Level 6 - Service Mission.**

There is no upper limit on the number of levels in a NASREM hierarchy. The number of levels of coordination and abstraction are strictly a function of the demands of the application (i.e., the organization of people, machines, communications links and tasks to be coordinated).

The Three Legs of NASREM

Another view of the NASREM architecture is shown in Figure 4. The control system is represented as a hierarchy of computing modules in sets of three, forming nodes in the hierarchy. These nodes are serviced by a communications system and a global memory. The task decomposition modules perform real-time planning and task monitoring functions; they decompose task goals both spatially and temporally. The sensory processing modules filter, correlate, detect, and integrate sensory information over both space and time in order to recognize and measure patterns, features, objects, events, and relationships in the external world. The world modeling modules answer queries, make predictions, and compute evaluation functions on the state space defined by the information stored in global memory. Global memory is the repository which contains the system's best estimate of the state of the external world. The world modeling modules keep the global memory interface buffers current and consistent.

Task Decomposition. The first leg of the hierarchy (in Figure 4) consists of task decomposition modules which plan and execute the decomposition of high level goals into low level actions. Task decomposition involves both a temporal decomposition (into sequential actions along the time line) and a spatial decomposition (into concurrent actions by different subsystems). Each task decomposition module at each level of the hierarchy consists of a job assignment manager, a set of planners, and a set of executors.

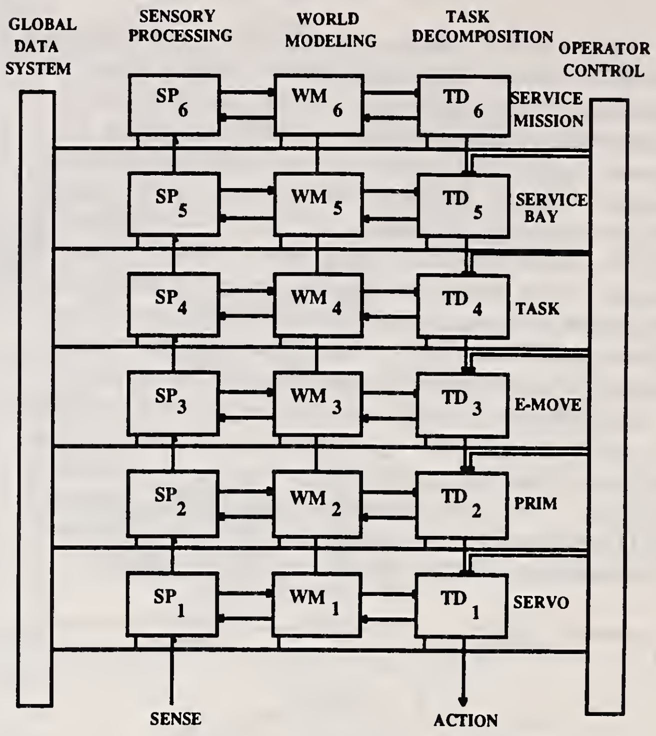

Figure 4. Layers of NASREM Nodes formed by Modules in Sets-of-Three

World Modeling. The second leg of the hierarchy consists of world modeling modules which model and evaluate the state of the world. The "world model" is the system's best estimate and evaluation of the history, current state, and possible future states of the world, including the states of the system being controlled. The "world model" includes both the world modeling modules and a knowledge base stored in global memory where state variables, maps, lists of objects and events, and attributes of objects and events are maintained. The world model maintains the global memory knowledge base by accepting information from the sensory system; it provides predictions of expected sensory input to the corresponding sensory system modules; based on the state of the task and estimates of the external world, answers "What is?" questions asked by the executors in the corresponding task decomposition modules; and answers "What if?" questions asked by the planners in the corresponding task decomposition modules.

Sensory Processing. The third leg of the hierarchy consists of sensory processing modules. These recognize patterns, detect events, and filter and integrate sensory information over space and time. The sensory processing modules at each level compare world model predictions with sensory observations and compute correlation and difference functions. These are integrated over time and space so as to fuse sensory information from multiple sources over extended time intervals.

Newly detected or recognized events, objects, and relationships are entered by the world modeling modules into the global memory, and objects or relationships perceived to no longer exist are removed. The sensory processing modules also contain functions which can compute confidence factors and probabilities of recognized events, and statistical estimates of stochastic state variable values.

Operator Interface. The control architecture allows an operator interface at each level in the hierarchy. For the FTS functional architecture, operator interface provides a means by which human operators, either in the space station or on the ground, can observe and supervise the telerobot. Each level of the task decomposition hierarchy may provide an interface where the human operator can assume control. The task commands into any level can be derived either from the higher level task decomposition module, from the operator interface, or from some combination of the two. Using a variety of input devices, a human operator can enter the control hierarchy at any level, at any time of his choosing (if so designed), to monitor a process, to insert information to interrupt automatic operation and take control of the task being performed, or to apply human intelligence to sensory processing or world modeling functions.

The sharing of command input between human and autonomous control need not be all or none. It is possible in many cases for the human and the automatic controllers to simultaneously share control of a telerobot system. For example, in an assembly operation, a human might control the position of an end effector while the robot automatically controls its orientation.

Timing. For the control hierarchy shown in Figure 4 we can construct a timing diagram as shown in Figure 5. The range of the time scale, and hence the planning horizon and event summary interval, increases exponentially by an order of magnitude at each higher level. The loop bandwidth and frequency of sub-goal events decreases exponentially at each higher level.

The origin of the time axis is the present, i.e. t=0. Future plans lie to the right of t=0, past history to the left. The open triangles in the right half-plane represent task goals in a future plan. The filled triangles in the left half-plane represent task completion events in a past history. At each level there is a planning horizon and a historical event summary interval.

This timing diagram suggests a duality between the task decomposition and the sensory processing hierarchies. At each hierarchical level, planner modules decompose task commands into strings of planned sub-tasks for execution. At each level, strings of sensed events are summarized, integrated, and "chunked" into single events at the next higher level. At each level, planning horizons extend into the future about as far, and with about the same level of detail, as historical traces reach into the past.

At each level, plans consist of at least two, and an average ten sub-tasks. The planners have a planning horizon that extends about one average input command interval into the future.

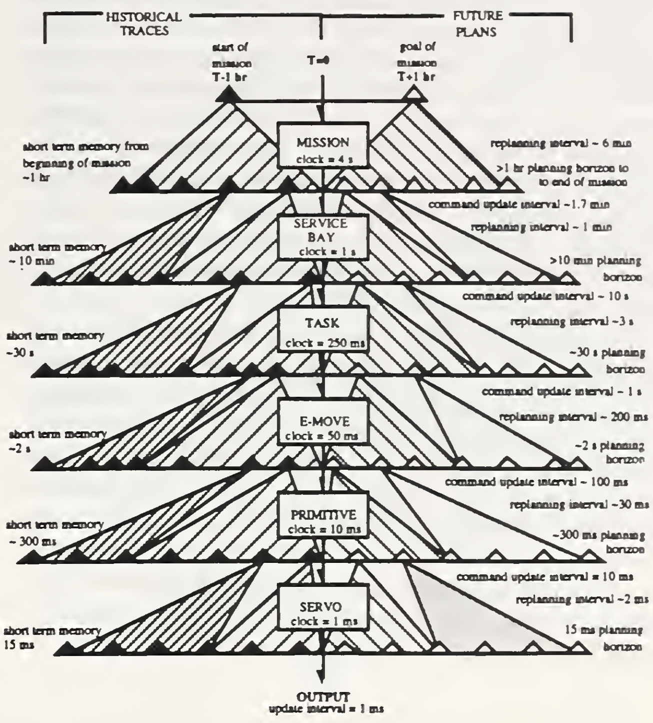
HISTORICAL TRACES
FUTURE PLANS
start of mission T-1 hr
T=0
goal of mission T+1 hr
short term memory from beginning of mission ~1 hr
MISSION clock = 4 s
replanning interval ~ 6 min
>1 hr planning horizon to to end of mission
command update interval ~1.7 min
replanning interval ~ 1 min
short term memory ~ 10 min
SERVICE BAY clock = 1 s
>10 min planning horizon
command update interval ~ 10 s
replanning interval ~3 s
short term memory ~30 s
TASK clock = 250 ms
~30 s planning horizon
command update interval ~ 1 s
replanning interval ~ 200 ms
short term memory ~2 s
E-MOVE clock = 50 ms
~2 s planning horizon
command update interval ~ 100 ms
replanning interval ~30 ms
short term memory ~ 300 ms
PRIMITIVE clock = 10 ms
~300 ms planning horizon
command update interval = 10 ms
replanning interval ~2 ms
short term memory 15 ms
SERVO clock = 1 ms
15 ms planning horizon
OUTPUT update interval = 1 ms

Figure 5. NASREM Timing Diagram

Replanning may be done at cyclic intervals, or whenever necessary. Emergency replanning begins immediately upon the detection of an emergency condition. Under full alert status, the cyclic replanning interval should be about an order of magnitude less than the planning horizon (or about equal to the expected output sub-task time duration). This requires that real-time planners search to the planning horizon about an order of magnitude faster than real time.

Plan executors at each level have the task of reacting to feedback every control cycle interval. If the feedback indicates the failure of a planned sub-task, the executor branches immediately to a preplanned emergency sub-task. The planner simultaneously selects or generates an error recovery sequence which then can be substituted for the former plan which failed.

When a task goal is achieved at time t=0, it becomes a task completion event in the historical trace. To the extent that a historical trace is an exact duplicate of a former future plan, the plan was followed, and every task was accomplished as planned. To the extent that a historical trace is different from the former plan, there were surprises.

At each level in the control hierarchy, the difference vector between planned and observed events is an error signal, that can be used by executor sub-modules for servo feedback control.

THE FTS FUNCTIONAL ARCHITECTURE

In order to implement a functional architecture, especially one like NASREM which allows evolution with technology, the interfaces must be carefully defined. Although the NASREM conceptual architecture specifies the purpose of each module in the control system hierarchy, it does not completely specify the interfaces between modules. This section will describe the method by which the interfaces for the SERVO level of the FTS hierarchy have been defined. The method involves gathering all of the algorithms available for SERVO level control, dividing each algorithm into the parts which inherently belong to task decomposition, world modeling, and sensory processing, and then deriving the interfaces which will support these algorithms.

The NASREM architecture, as presented in,[1] defines the basic architecture for a robot control system capable of teleoperation and autonomy in one system. Recently, efforts have been directed at specifying in detail the architecture requirements for robotic manipulation. An important criterion for the design is that it support the algorithms for manipulator control found in the literature. This assures that the control system can serve as a vehicle for evaluating algorithms and comparing approaches. Any design, however, must constrain the problem sufficiently so that detailed interfaces can be devised.

Servo Level

With this in mind, the Servo Level design was based on a fundamental control approach which computes a motor command as a function of feedback system state y, desired state (attractor) yd, and control gains. In this approach, the gains are coefficients of a linear combination of state errors (y-yd). The system state and its attractor are composed from the physical quantities to be controlled, (i.e. position, force, etc.,) and can be expressed in an arbitrary coordinate system. This type of algorithm is the basis for almost all manipulator control schemes.[13] However, this basic algorithm is inadequate for controlling the gross aspects of manipulator motion.[14] The servo algorithm can provide "small" motions so that the algorithm's transient dynamics are not significant in shaping the gross motion. This means that the Primitive Level must generate the gross motion through a sequence of inputs to the Servo Level. This can be achieved through an appropriate sequence of either attractor points[13,15] or gain values.[14]

Figure 6 depicts the detailed Servo Level design. The task decomposition module at the Servo Level receives input from Primitive in the form of the command

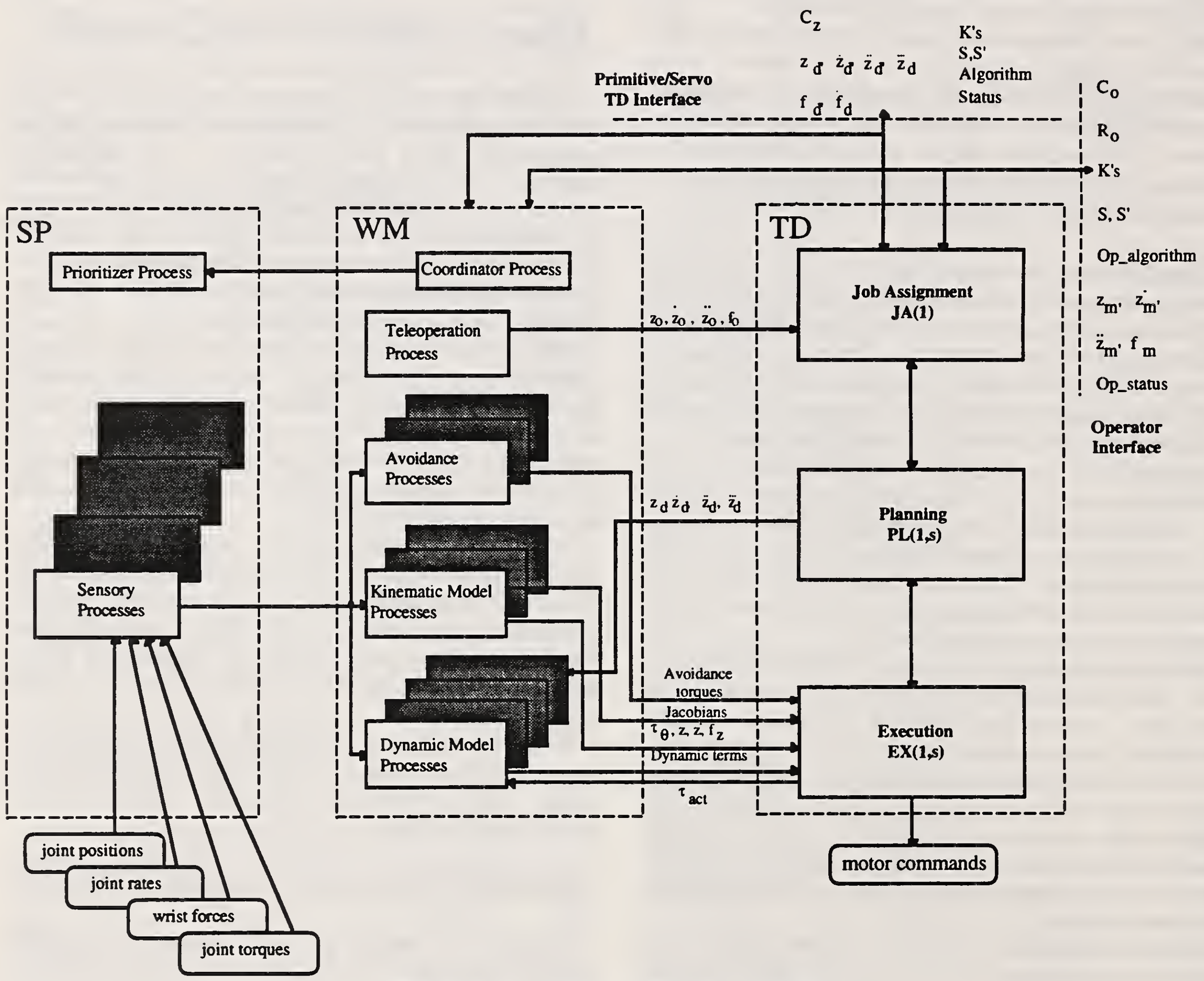

Figure 6. Servo Level Design

specification parameters. The command parameters include a coordinate system specification C_Z which indicates the coordinate system in which the current command is to be executed. C_Z can specify joint, end-effector, or Cartesian (world) coordinates. Given with respect to this coordinate system are desired position, velocity, and acceleration vectors ($\mathbf{z}_d$, $\dot{\mathbf{z}}_d$, $\ddot{\mathbf{z}}_d$) for the manipulator, and the desired force and rate of change of force vectors ($\mathbf{f}_d$, $\dot{\mathbf{f}}_d$). These command vectors form the attractor set for the manipulator. The K's are the gain coefficient matrices for error terms in the control equations. The selection matrices (S,S') apply to certain hybrid force/position control algorithms. Finally, the "Algorithm" specifier selects the control algorithm to be executed by the Servo Level.

When the Servo Level planner receives a new command specification, the planner transmits certain information to world modeling. This information includes an attention function which tells world modeling where to concentrate its efforts, i.e. what information to compute for the executor. The executor simply executes the algorithm indicated in the command specification, using data supplied by world modeling as needed.

The world modeling module at the Servo Level computes model-based quantities for the executor, such as Jacobians, inertia matrices, gravity compensations, Coriolis and centrifugal force compensations, and potential field (obstacle) compensations. In addition, world modeling provides its best guess of the state of the manipulator in terms of positions, velocities, end-effector forces and joint torques. To do this, the module may have to resolve conflicts between sensor data, such as between joint position and Cartesian position sensors.

Sensory processing, as shown in Figure 6, reads sensors relevant to Servo and provides the filtered sensor readings to world modeling. In addition, certain information is transmitted up to the Primitive Level of the sensory processing hierarchy. Primitive uses this information, as well as information from Servo Level world modeling, to monitor execution of its trajectory. Based on this data, Primitive computes the stiffness (gains) of the control, or switches control algorithms altogether. For example, when Primitive detects a contact with a surface, it may switch Servo to a control algorithm that accommodates contact forces.

A more complete description of the Servo Level is available in[13] where the vast majority of the existing algorithms in the literature are described. The same process for developing the interfaces based on the literature has also been performed for the Primitive level and is available in.[15] While the procedure is planned for each level in the hierarchy, the amount of literature support tends to decrease as one moves up the hierarchy.

Hardware and Software Architectures

Once the interfaces are defined, it is possible to choose a computer architecture and begin to realize the system. This section will describe a specific implementation under construction at NIST. While every effort was made to do the job properly, there is no reason to assume that this implementation is optimal in any way. It simply illustrates one realistic method to implement the NASREM architecture.

While a functional architecture is technology independent, its implementation obviously depends entirely on the state-of-the-art of technology. The designer must choose existing computers, buses, languages, etc., and, from these tools, produce a computer architecture capable of performing the functions of the functional architecture. The system must adequately meet the real-time aspects of the controller so that adequate performance is achieved through careful consideration of computer choice, multiple processor real-time operating system, inter-processor communication requirements, tasking within certain processors, etc. For a more detailed description of this methodology see reference.[16]

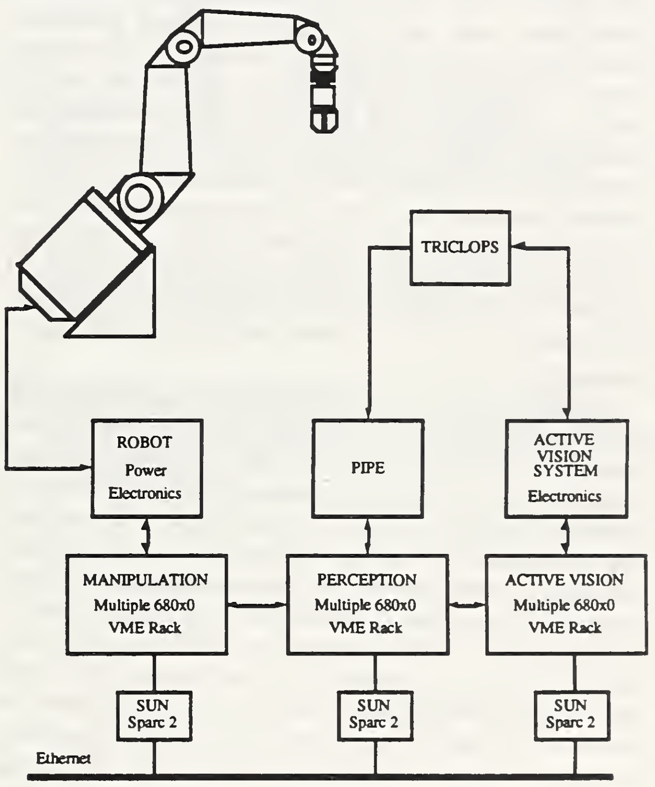

Figure 7. NASREM Testbed for Integrating a Robot Manipulator and an Active Vision System called "TRICLOPS"

System Development Environment

The NIST implementation considers two aspects of the process: the development environment in which the code is developed, debugged, and tested as well as possible, and the target environment where the code for the real-time robot control system is executed. Figure 7 shows the equipment configuration we used in implementing a NASREM testbed for a machine vision research project called "TRICLOPS." A network of SUN* workstations running UNIX was used for the development environment, sacrificing the execution speed of the code for ease of development. Once the code was tested as well as possible, it was downloaded to the target system, reclaiming code execution speed. The target system consisted of a VME backplane and six Motorola 68020 processors. For rapid iconic image processing, the PIPE system[17] is used. The target hardware drives a Robotics Research Corporation robot arm.

From the software side, the multiprocessing operating system used for the target environment should be as simple as possible so that the operating system overhead is minimized. The duties of the operating system are limited to very simple actions such as downloading and starting up the processors and inter-processor communication. Software multi-tasking is not used at the lower levels of the hierarchy because of the execution-time overhead associated with operating system context switching. For the FTS project NIST researchers investigated three alternatives for software execution scheduling: the tasking model provided by the native compiler, the pSOS tasking model, and the ADA tasking model. Inter-processor communications alternatives including pRISM, sockets, etc., were also evaluated empirically. A shared memory mechanism was finally selected because of real-time computing constraints. The actual application code is written in ADA. Although ADA compilers typically cannot produce code that is as efficient as code produced in other languages such as C, NIST researchers have shown that the gap is steadily decreasing.[18]

The application code is developed by programming the processes which achieve the functions associated with the boxes in the functional architecture. The problem then becomes one of assigning each of the processes to a particular processor. There is a clear trade-off between the cost of the solution and the performance of the system. There are currently no software tools which automatically perform this assignment based on an arbitrary index of performance. The approach at NIST is step-wise refinement of the performance of the system. Given the particular hardware being used, a certain number of processors is chosen arbitrarily. For that configuration, the processes are assigned to the processors. Then, the system is evaluated in terms of its performance. If the performance is unacceptable, the designer has several options. The first option is to add more processors. This alternative is balanced against the possibility of additional communication requirements between the processors. Another alternative is to add faster processors or special purpose processors, such as dynamics chips, which optimize particularly compute intensive operations. This trade-off clearly relates to cost. Another alternative is to reassign the processes to the processors in order to balance the workload of each processor. Each of the alternatives can be evaluated by the designer in order to balance system costs against improved performance of the system. This technique also allows a particular configuration which implements the functional architecture to change with time as improvements in technology are realized.

* References to specific brand names, equipment, or trade names in this paper are made to facilitate understanding and do not imply endorsement by the National Institute of Standards and Technology.

Methodology Tenets

We use the word tenet, here, to mean guidelines and engineering rules of thumb which characterize our NASREM methodology approach. Together the NASREM conceptual architecture and these tenets form a basic set of rules or systems integration standards for building real-time control systems. An in-depth discussion of these tenets is presented in.[11]

1) Use task oriented decomposition (driven by scenarios)
2) Use hierarchical organization and assign responsibility and authority
3) Organize the control hierarchy around tasks top-down and equipment bottom-up
4) Partition by an order of magnitude between levels (spatial and temporal resolution) and roughly ten decisions or less per plan
5) Use seven + or - two subordinates per supervisor and only one supervisor at a time
6) SP/WM/BG functions are distributed throughout the hierarchy and assumed to exist in each node
7) Allow human interface at any node
8) Controller modules are finite state machines communicating through Global Memory
    * Use a controller template as the basic building block
    * Use cyclic sampling rather than interrupts for context switching
    * Surround all modules with data buffers
    * Use non-blocking input/output (I/O)
    * Implement Global Memory using a One Writer, Many Readers Paradigm
    * Match the control cycle time to the demands of the control application
9) Design for concurrent processing
    * Measure execution time performance
    * Allocate sufficient computing resources
10) Use synchronous control at the lowest levels, transitioning to asynchronous control at the highest levels

Conclusion

The NASREM functional architecture developed for the Flight Telerobotic Servicer (FTS) can serve as a technology independent paradigm foundation from which any NASREM telerobot implementation can be derived. For the FTS project we developed interfaces for the NASREM architecture modules which take into account the research already published in the literature. When a NASREM implementation is desired, the result is, by necessity, a reflection of the current state-of-the-art.

However, when the interfaces are carefully specified, alternative software and hardware solutions may easily be tested and integrated. This will allow the FTS control system we developed to evolve with technology, both for space as well as for terrestrial applications.

REFERENCES

[1] J.S. Albus, H.G. McCain, and R. Lumia, "NASA/NBS Standard Reference Model for Telerobot Control System Architecture (NASREM)," NIST (formerly NBS) Technical Note 1235, April 1989 Edition, also as NASA document SS- GSFC-0027.

[2] J.S. Albus, A.J. Barbera, R.N. Nagel, "Theory and Practice of Hierarchical Control," Proceedings of the 23rd IEEE Computer Society International Conference, September 1981.

[3] A.J. Barbera, J.S. Albus, M.L. Fitzgerald, L.S. Haynes, "RCS: The NBS Real-Time Control System," Robots 8 Conference and Exposition, Detroit, MI, June 1984.

[4] S. Leake, R.D. Kilmer, "The NBS Real-Time Control System User's Reference Manual," NBS Technical Note 1258, June 1988.

[5] M.O. Shneier, E.W. Kent, J.S. Albus, P. Mansbach, M. Nashman, L. Palombo, W. Rutkowski, T.E.Wheatley, "Robot Sensing for a Hierarchical Control System, "Proceedings of the 13th ISIR/Robots 7 Symposium, Chicago, IL, April 1983.

[6] E.W. Kent, J.S. Albus, "Servoed World Models as Interfaces between Robot Control Systems and Sensory Data," Robotica (1984) volume 2, pp. 17-25.

[7] J.S. Albus, "System Description and Design Architecture for Multiple Autonomous Undersea Vehicles," NIST Technical Note 1251, September 1988.

[8] H.G. McCain, R.D. Kilmer, S. Szabo, A. Abrishamian, "A Hierarchically Controlled Autonomous Robot for Heavy Payload Military Field Applications," Proceedings of the International Congress on Intelligent Autonomous Systems, Amsterdam, The Netherlands, December 8-11, 1986.

[9] S. Szabo, H. A. Scott, R. D. Kilmer, "Control System Architecture for the TEAM Program," Proceedings of the Second International Symposium on Robotics and Manufacturing Research, Education and Applications, Albuquerque, NM, November 16-18, 1988.

[10] Random House College Dictionary, 1982, Revised Edition

[11] R. Quintero and A. J. Barbera, "An RCS Methodology for Developing Intelligent Control Systems," NISTIR 4936, October 1992.

[12] T. R. Kramer and M. K. Senehi, "Feasibility Study: Reference Architecture for Machine Control Systems Integration," NISTIR 5297, November 1993.

[13] J. Fiala, "Manipulator Servo Level Task Decomposition," NIST Technical Note #1255, April 20, 1988.

[14] J. Fiala, "Generation of Smooth Trajectories without Planning," NISTIR 4622, June 1991.

[15] A.J. Wavering, "Manipulator Primitive Level Task Decomposition," NIST Technical Note #1256, January 5, 1988.

[16] J.L. Michaloski, T.E. Wheatley, R. Lumia, R., "Analysis of Computational Parallelism with a Concurrent Hierarchical Robot Control System," NIST Technical Report NISTIR 90- 4251, March 1990.

[17] E.W. Kent, M.O. Shneier, and R. Lumia, "PIPE," Journal of Parallel and Distributed Computing, Vol. 2, 1985, pp. 50-78.

[18] S. Leake, "A Comparison of Robot Kinematics in ADA and C on Sun and microVAX," Robotics and Automation Session, IASTED, Santa Barbara, CA., May 25-27,1988.