NASREM
THE NASA/NBS STANDARD REFERENCE MODEL
FOR
TELEROBOT CONTROL SYSTEM ARCHITECTURE

James S. Albus
Ronald Lumia
John Fiala
Albert Wavering

Robot Systems Division
National Institute of Standards and Technology
(formerly the National Bureau of Standards)
Bldg. 220, Room B-124
Gaithersburg, MD 20899
United States of America

ABSTRACT

There are five major elements required for the development of an intelligent robot system.
Four of these are architectures:
1. Conceptual architecture
2. Functional architecture
3. Software architecture
4. Hardware architecture
The fifth element is the software development environment.

The original NASREM document describes the conceptual architecture, and suggests the outlines of a functional architecture.

Recent work at NIST is defining the functional architecture of the servo and primitive levels, and suggesting the outlines of software and hardware architectures, and software development environments.

NASREM: THE CONCEPTUAL ARCHITECTURE

NASREM represents the culmination of more than 15 years 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 [1,2,3,4,5]. Since 1986, RCS has been implemented for a number of additional applications, including the NBS/DARPA Multiple Autonomous Undersea Vehicle (MAUV) project [6], the Army Field Material Handling Robot (FMR)[7], and the Army TEAM (Technology Enhancement for Autonomous Vehicles) 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)[8].

The fundamental paradigm of the NASREM conceptual architecture is shown in Figure 1. The control system is represented as a three legged hierarchy of computing modules, 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 a database which contains the system's best estimate of the state of the external world. The world modeling modules keep the global memory database current and consistent.

TASK DECOMPOSITION

The first leg of the hierarchy 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.

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 a global memory database 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, 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 system modules. These recognize patterns, detect events, and filter and integrate sensory information over space and time. The sensory system 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 world model global memory database, and objects or relationships perceived to no longer exist are removed. The sensory system 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 has an operator interface at each level in the hierarchy. The 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 provides 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, 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 1 we can construct a timing diagram as shown in Figure 2. 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 subgoal 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 subtasks 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 2, and an average 10 subtasks. The planners have a planning horizon that extends about one average input command interval into the future. Figure 3 illustrates this principle.

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 subtask 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 subtask, the executor branches immediately to a preplanned emergency subtask. 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

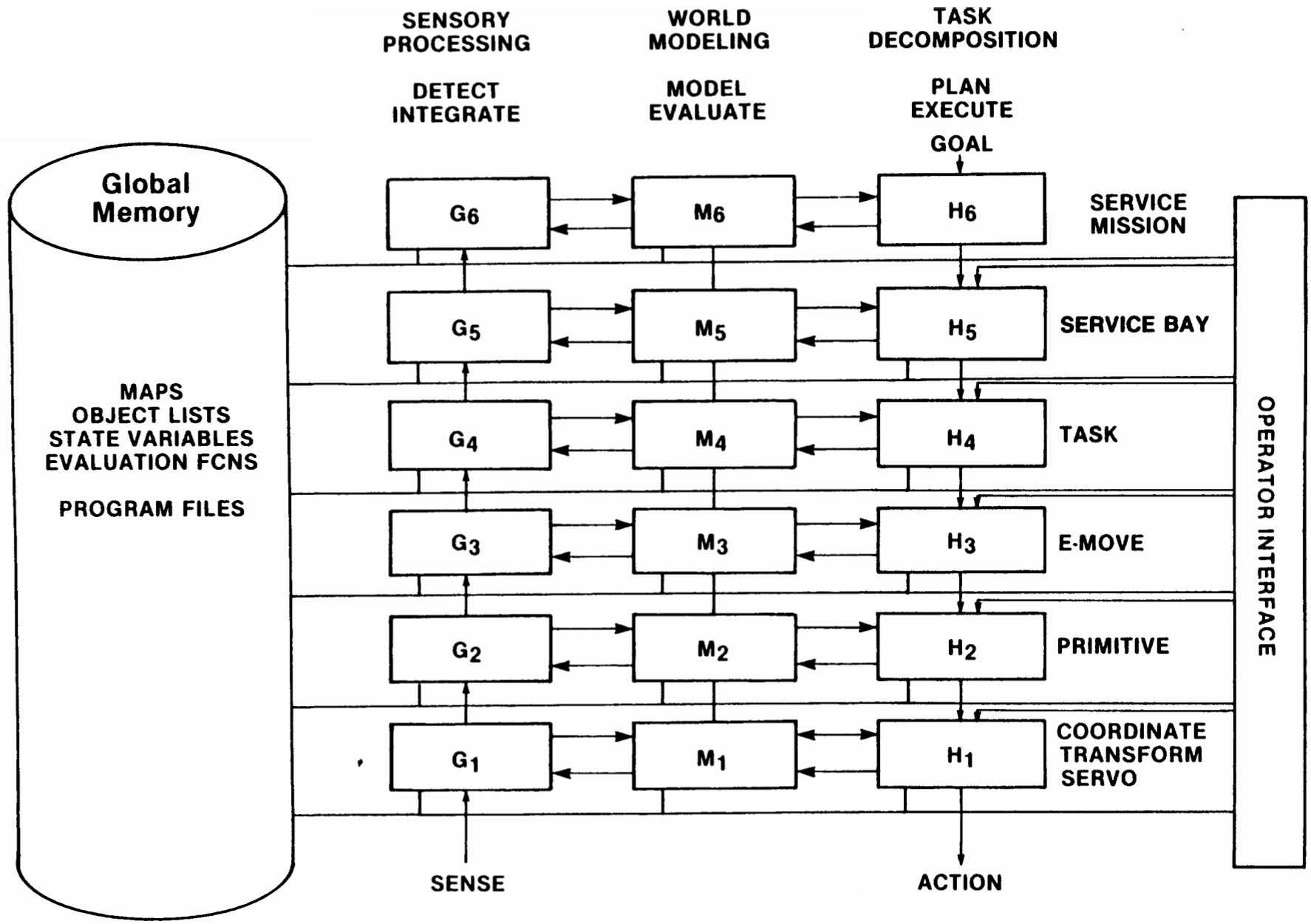
SENSORY PROCESSING
DETECT INTEGRATE
WORLD MODELING
MODEL EVALUATE
TASK DECOMPOSITION
PLAN EXECUTE
GOAL
Global Memory
MAPS
OBJECT LISTS
STATE VARIABLES
EVALUATION FCNS
PROGRAM FILES
G6
M6
H6
SERVICE MISSION
G5
M5
H5
SERVICE BAY
G4
M4
H4
TASK
G3
M3
H3
E-MOVE
G2
M2
H2
PRIMITIVE
G1
M1
H1
COORDINATE TRANSFORM SERVO
OPERATOR INTERFACE
SENSE
ACTION

FIGURE:1

NASREM TIMING DIAGRAM

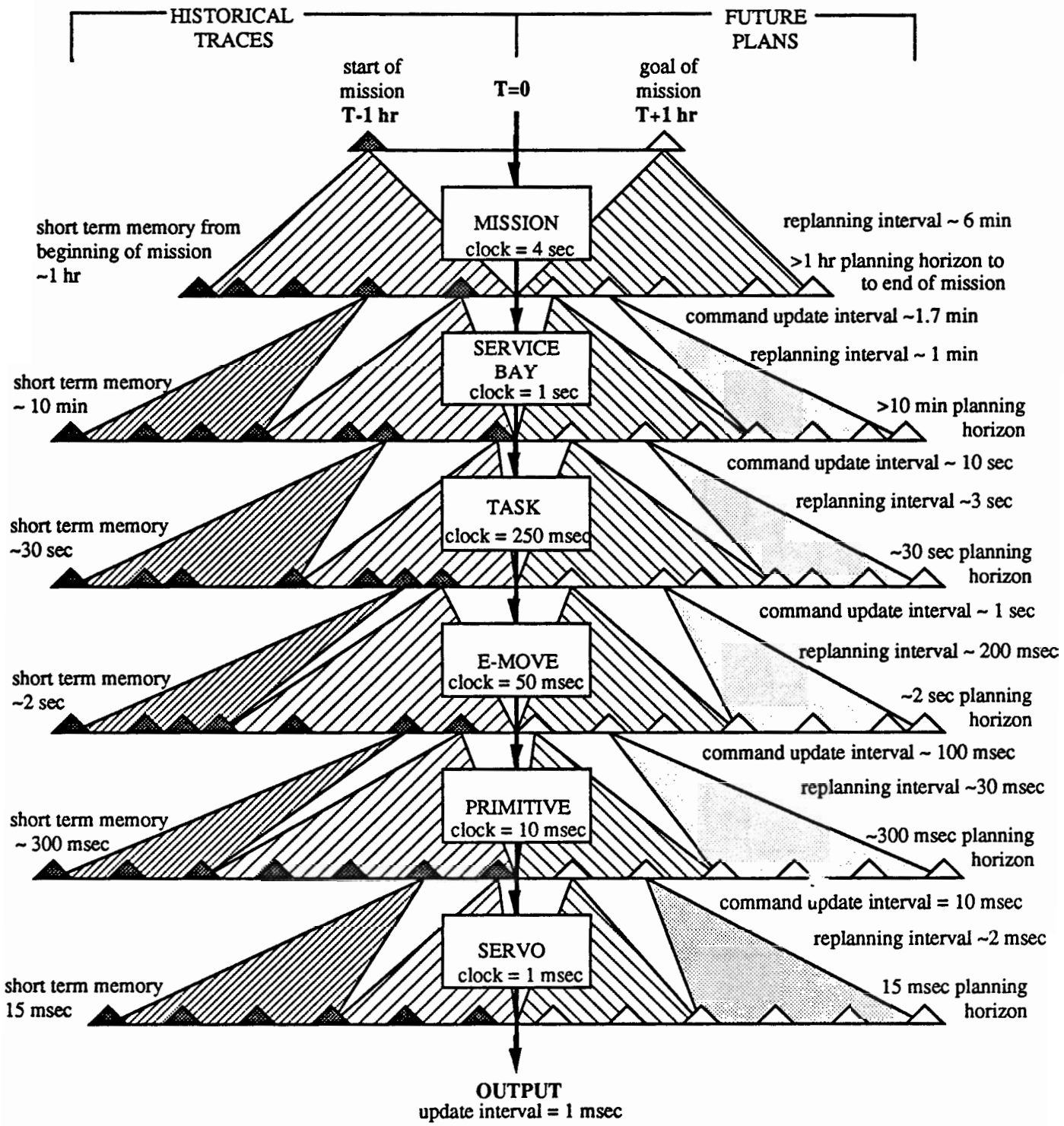

FIGURE :2

Hierarchical Planning

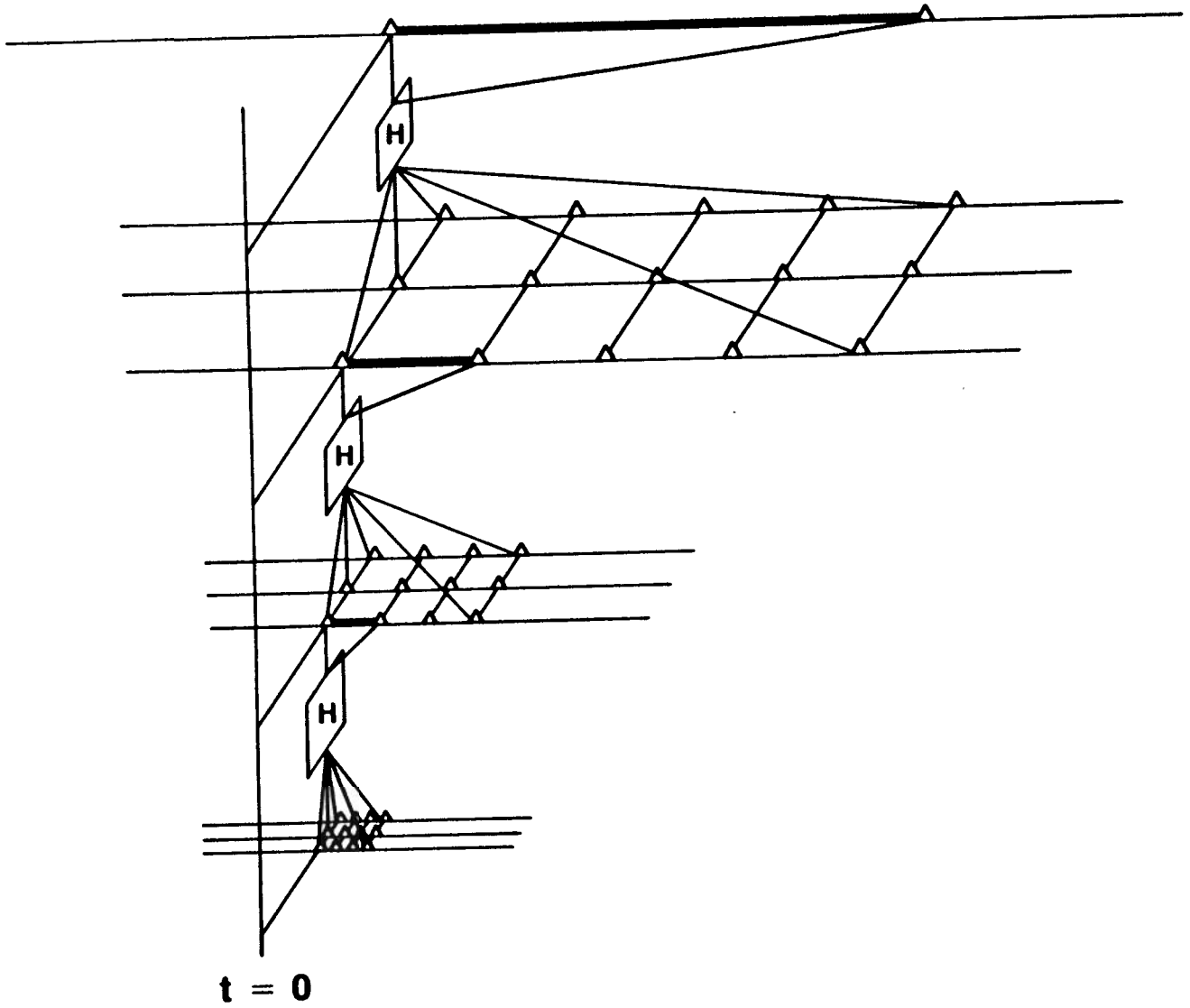

FIGURE: 3

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 submodules for servo feedback control.

INTERFACES

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 functional 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 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 NASA/NBS Standard Reference Model (NASREM) Telerobot Control System Architecture, as presented in [8], 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 [9]. However, this basic algorithm is inadequate for controlling the gross aspects of manipulator motion, as described in [10]. 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 [9,11] or gain values [10].

Figure 4 depicts the detailed Servo Level design. The task decomposition module at the Servo Level receives input from Primitive in the form of the command specification parameters. The command parameters include a coordinate system specification Cz which indicates the coordinate system in which the current command is to be executed. Cz can specify joint, end-effector, or Cartesian (world) coordinates. Given with respect to this coordinate system are desired position, velocity, and acceleration vectors (zd, zd, zd) for the manipulator, and the desired force and rate of change of force vectors (fd, fd). 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 4, 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 [9] 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 [11]. 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.

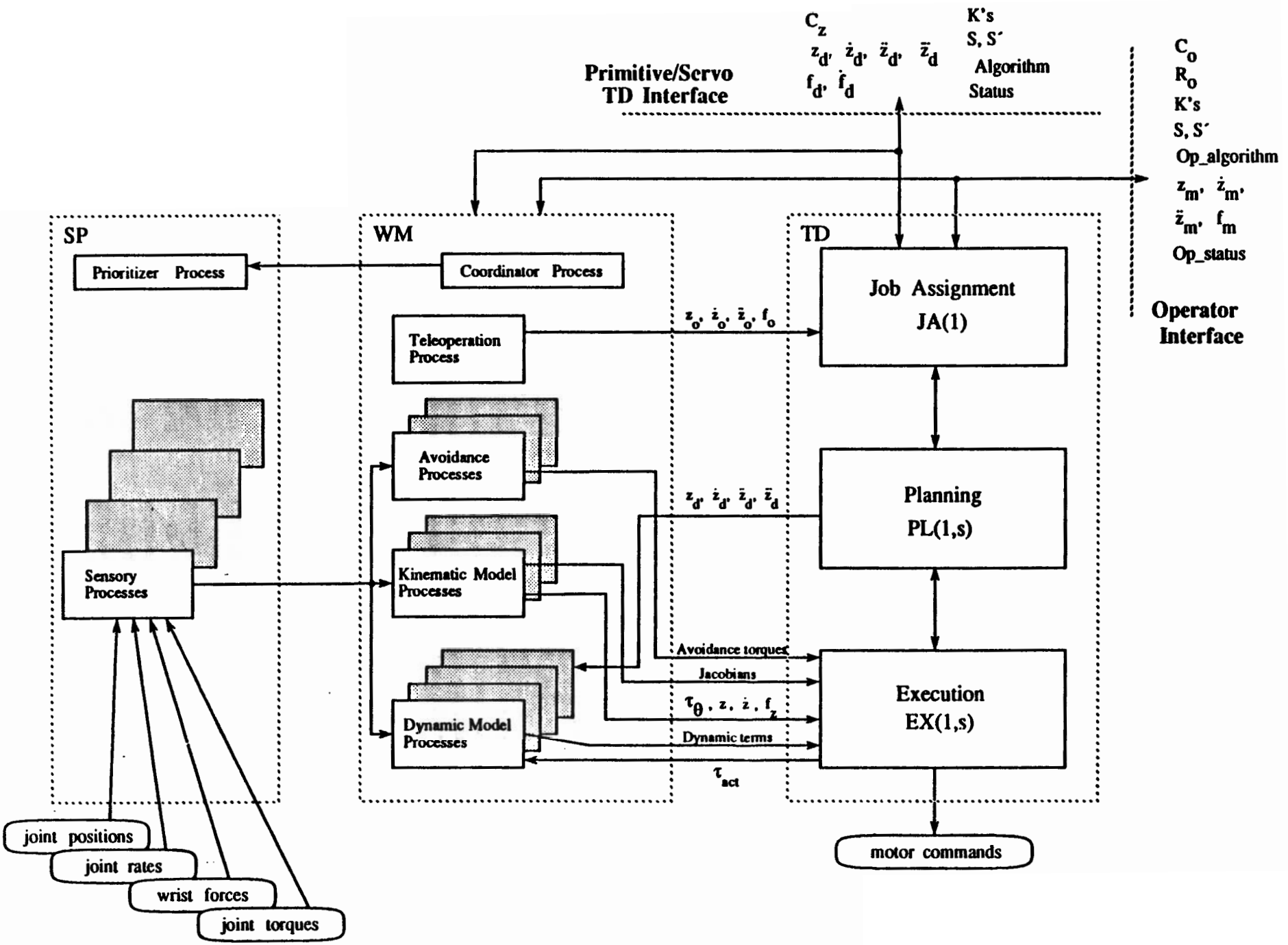
Primitive/Servo
TD Interface
C_z
$z_{d'}$ $\dot{z}_{d'}$ $\ddot{z}_{d'}$ $\bar{z}_d$
$f_{d'}$ $\dot{f}_d$
K's
S, S´
Algorithm
Status
C_o
R_o
K's
S, S´
Op_algorithm
z_m, $\dot{z}_m$,
$\ddot{z}_m$, f_m
Op_status
Operator
Interface
SP
WM
TD
Prioritizer Process
Coordinator Process
Job Assignment
JA(1)
Teleoperation
Process
z_o, $\dot{z}_o$, $\ddot{z}_o$, f_o
Avoidance
Processes
Planning
PL(1,s)
$z_{d'}$ $\dot{z}_{d'}$ $\ddot{z}_{d'}$ $\bar{z}_d$
Sensory
Processes
Kinematic Model
Processes
Avoidance torques
Jacobians
Execution
EX(1,s)
τ_θ , z, $\dot{z}$, f_z
Dynamic Model
Processes
Dynamic terms
τ_{act}
joint positions
joint rates
wrist forces
joint torques
motor commands

FIGURE: 4

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 the specific implementation under construction at NIST. While every effort is being made to do the job properly, there is no reason to assume that this implementation is optimal in any way. It is 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-processing communication requirements, tasking within certain processors, etc. For a more detailed description of this methodology, see [12].

SOFTWARE 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 5 shows the approach. A network of SUN workstations running UNIX is used for the development environment, sacrificing the speed of the developed code for the ease of development. Once the code is tested as well as possible, it is downloaded to the target system. The target system consists of a VME backplane of several (currently 6) Motorola 68020 processors. For rapid iconic image processing, the PIPE system [13] is interfaced. The target hardware drives the Robotics Research Corporation arm.

From the software side, the multiprocessing operating system used for the target is required to be as simple as possible so that the overhead is minimized. The duties of the operating system are limited to very simple actions such as downloading and starting up the processors and interprocessor communication. Tasking is not performed at the lower levels of the hierarchy because of the overhead associated with context switches. NIST researchers are currently investigating three alternatives for tasking: tasking provided by the native compiler, pSOS tasking, and ADA tasking. Interprocessor communications alternatives including pRISM, sockets, etc., must also be evaluated empirically. The actual application code is written in ADA. Although ADA compilers usually cannot currently produce code as efficient as other languages such as C, NIST researchers have shown that the gap is steadily decreasing [14].

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, such as those shown in Figure 2, 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 the additional communication requirements of 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 used by the designer in order to improve the performance of the system. This allows a particular configuration which implements the functional architecture to change with time as improvements in technology are realized.

CONCLUSION

The NASREM functional architecture provides the technology independent paradigm which serves as the foundation from which any NASREM implementation can be derived. Interfaces may be developed for the NASREM architecture which will 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, since the interfaces are carefully specified, alternative software and hardware solutions may easily be tested and integrated. This will allow the Flight Telerobotic Servicer (FTS) to evolve with technology, both for space as well as for terrestrial applications.

REFERENCES

[1] 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.

[2] 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.

SYSTEM DEVELOPMENT
(View at Hardware)

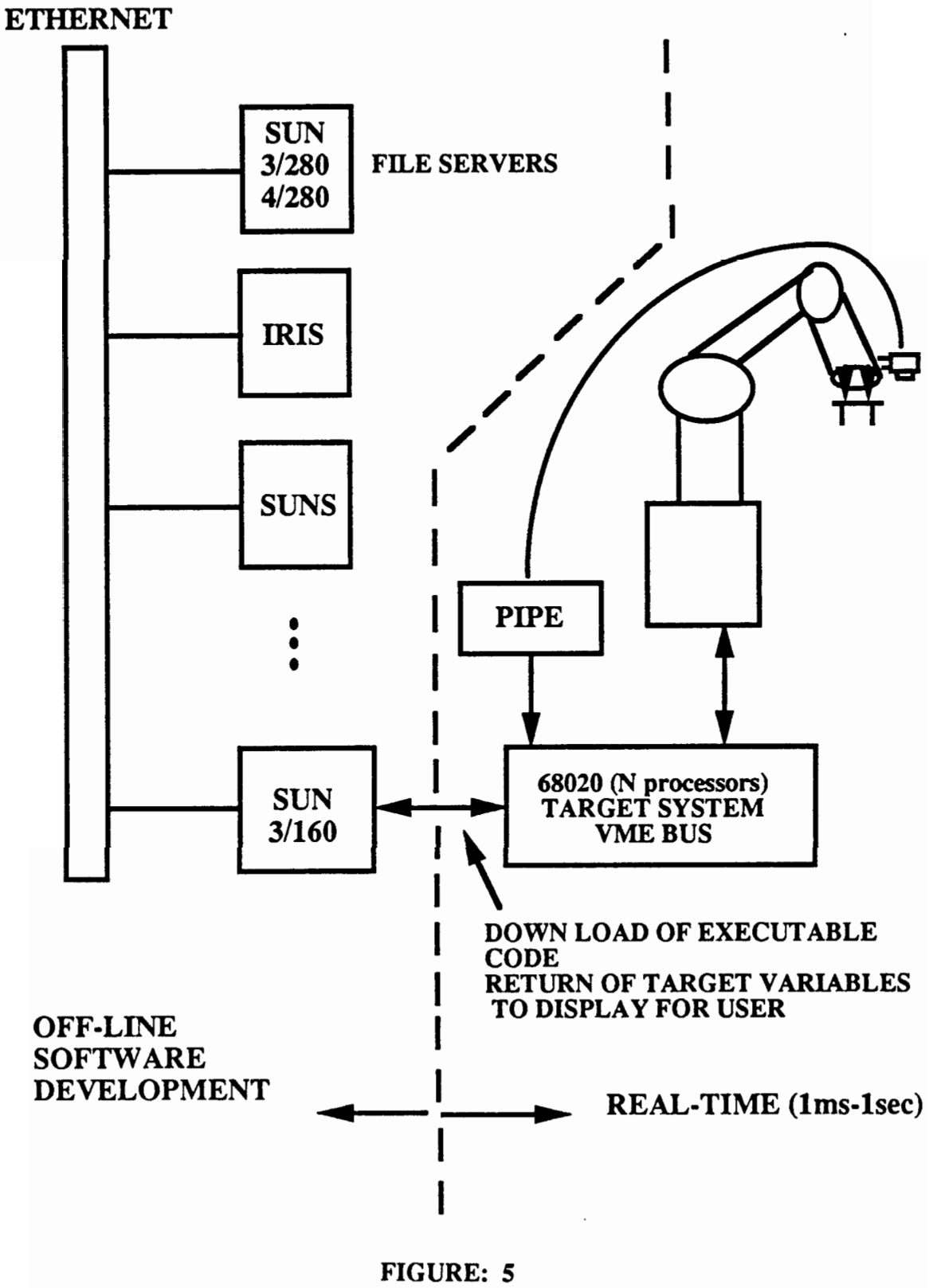

FIGURE: 5

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

[4] 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.

[5] 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.

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

[7] 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.

[8] J.S. Albus, R. Lumia, H.G. McCain, "NASA/NBS Standard Reference Model For Telerobot Control System Architecture (NASREM)," NBS Technical Note #1235, also as NASA document SS- GSFC-0027.

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

[10] J. Fiala, "Generation of Smooth Trajectories without Planning," (in progress).

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

[12] J. Michaloski, T. Wheatley, and R. Lumia, "Timing Analysis for a Parallel Pipelined Hierarchical Control System", 9th Real-Time Systems Symposium, (submitted).

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

[14] 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.