THE FLIGHT TELEROBOTIC SERVICER: FROM FUNCTIONAL ARCHITECTURE TO COMPUTER ARCHITECTURE

RONALD LUMIA
JOHN FIALA
ROBOT SYSTEMS DIVISION
NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY

ABSTRACT

The NASA/NBS Standard Reference Model for Telerobot Control System Architecture (NASREM) has been adopted by NASA for use in the Flight Telerobotic Servicer (FTS), a two armed telerobotic manipulator which will build and maintain the Space Station. NASREM is technology independent; the same functions must be performed by all controllers. NASREM provides the paradigm which allows the FTS to evolve with technology because standard interfaces can be defined so that functionally equivalent software and hardware modules may be interchanged. After a brief tutorial on the NASREM functional architecture, the approach to its implementation will be shown. First, interfaces must be defined which are capable of supporting the known algorithms. This will be illustrated by considering the interfaces required for the SERVO level of the NASREM functional architecture. After interface definition, the specific computer architecture for the implementation must be determined. This choice is obviously technology dependent. An example illustrating one possible mapping of the NASREM functional architecture to a particular set of computers which implements it will be shown. The result of choosing the NASREM functional architecture is that it provides a technology independent paradigm which can be mapped into a technology dependent implementation capable of evolving with technology in the laboratory as well as in space.

INTRODUCTION

The requirements of the Flight Telerobotic Servicer (FTS) of the Space Station are driving the development of robot systems for space applications. One of the key requirements is that the telerobot should be able to evolve with technology. This requirement implies the need for a reference model or functional architecture for the control system. A functional architecture is essential for several reasons. The control system cannot be developed as a static system but must be conceived to be able to evolve over time in order to benefit from advances in technology. Consequently, the architecture must be sufficiently flexible to support telerobotics in the beginning of the program and to

gradually support more autonomy of robot tasks. NASREM provides this functional architecture. Another aspect compelling the use of NASREM is that it provides a common reference model to which all designs must interface. Previous work in the Automated Manufacturing Research Facility (AMRF) at the National Institute of Standards and Technology (formerly NBS) has shown that system integration is the most difficult challenge [1]. The value associated with such a standard means that there is a common basis for the comparison of different design approaches for solving technical problems.

While the choice of a functional architecture is a crucial decision for the evolvability of the FTS, it does not contain all of the information required for a complete design. The purpose of this paper is to illustrate how a designer would proceed from the functional architecture to the functioning FTS. This paper is organized in the following manner to delineate the design process from conception to realization. First, a description of the NASREM reference model is presented. Then, the method used to define the interfaces is presented. This is followed by an example of how a particular computer architecture can realize the functional architecture. Finally, the potential impact of NASREM on both space and terrestrial applications of robots is assessed.

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

The fundamental paradigm of the control system is shown in Figure 1. The control system architecture is 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.

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.

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.

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.

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.

INTERFACE DEFINITION

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 [2], 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.

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) y_d, and control gains. In this approach, the gains are coefficients of a linear combination of state errors $(y-y_d)$. 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 [3]. However, this basic algorithm is inadequate for controlling the gross aspects of manipulator motion, as described in [8]. 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 [3,4] or gain values [8].

Figure 2 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 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 (z_d, z_d, z_d) for the manipulator, and the desired force and rate of change of force vectors (f_d, 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 2, 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 [3] 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 [4]. 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.

EXAMPLE OF A COMPUTER ARCHITECTURE TO IMPLEMENT NASREM

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 [5].

The NIST implementation considers two aspects of the process: the development environment on 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 3 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 [6] 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 [7].

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 FTS to evolve with technology, both for space as well as for terrestrial applications.

REFERENCES

[1] J.A. Simpson, R.J. Hocken, J.S. Albus. "The Automated Manufacturing Research Facility of the National Bureau of Standards," Journal of Manufacturing Systems, 1, 1, 1982, 17.

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

[3] J. Fiala, "Manipulator Servo Level Task Decomposition," NIST Technote #1255, April 20, 1988.

[4] A.J. Wavering, "Manipulator Primitive Level Task Decomposition," NIST Technote #1256, January 5, 1988.

[5] J. Michaloski,T. Wheatley,and R. Lumia, "Timing Analysis for a Parallel Pipelined Hierarchical Control System", 9th Real-Time Systems Symposium, (submitted).
[6] E.W. Kent, M.O. Shneier, and R. Lumia, "PIPE," Journal of Parallel and Distributed Computing, Vol. 2, 1985, pp. 50-78.
[7] 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.
[8] J. Fiala, "Generation of Smooth Trajectories without Planning," 1989 IEEE Robotics and Automation Conference, (submitted).

NASREM: NASA/NBS STANDARD REFERENCE MODEL

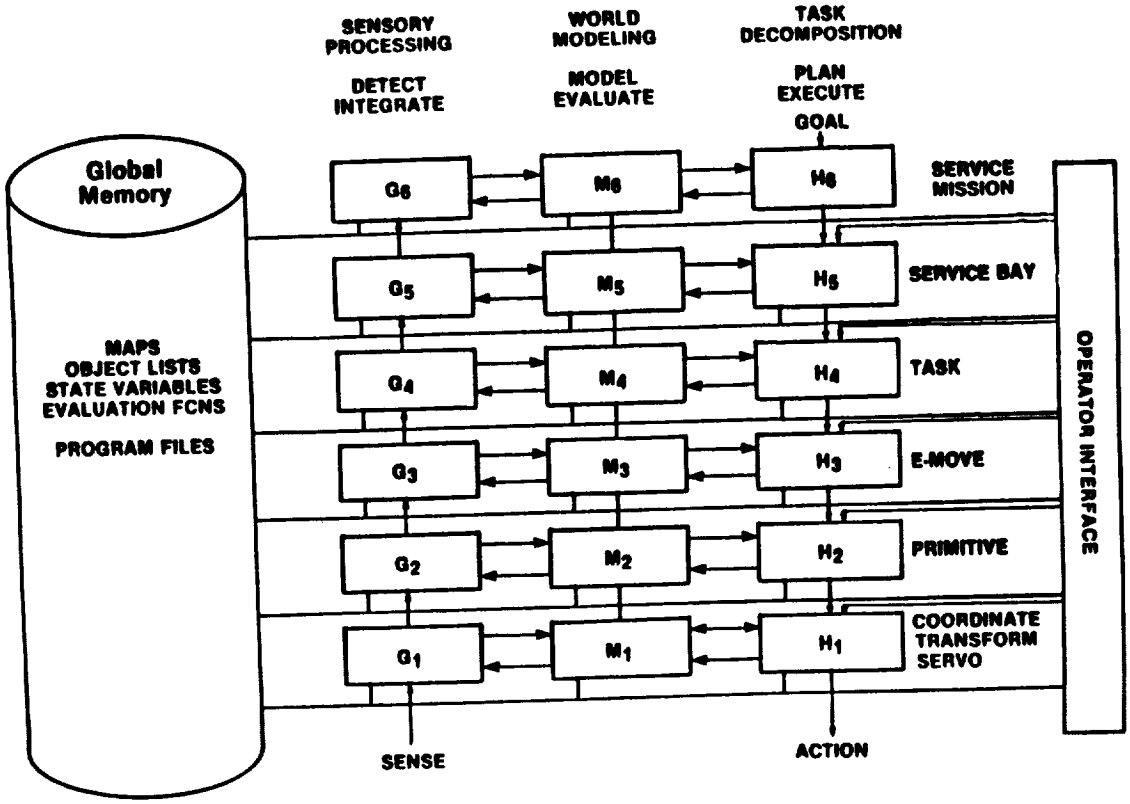

FIGURE 1

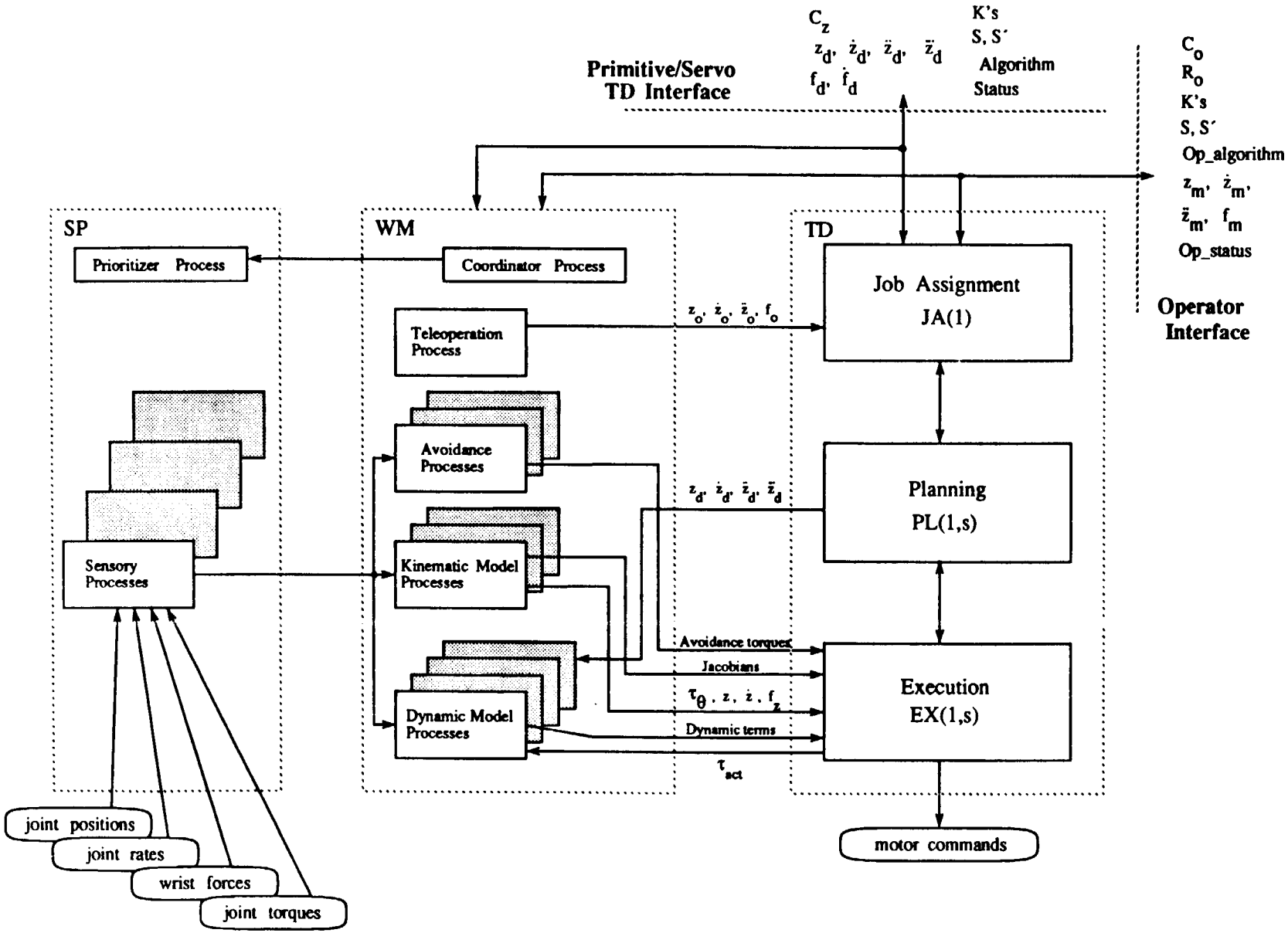
Primitive/Servo
TD Interface
C_z
z_d, $\dot{z}_d$, $\ddot{z}_d$, $\dddot{z}_d$
f_d, $\dot{f}_d$
K's
S, S´
Algorithm
Status
C_o
R_o
K's
S, S´
Op_algorithm
z_m, $\dot{z}_m$,
$\ddot{z}_m$, f_m
Op_status
Operator
Interface
SP
WM
TD
Prioritizer Process
Coordinator Process
Job Assignment
JA(1)
Teleoperation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

FIGURE 2

SYSTEM DEVELOPMENT
(View at Hardware)

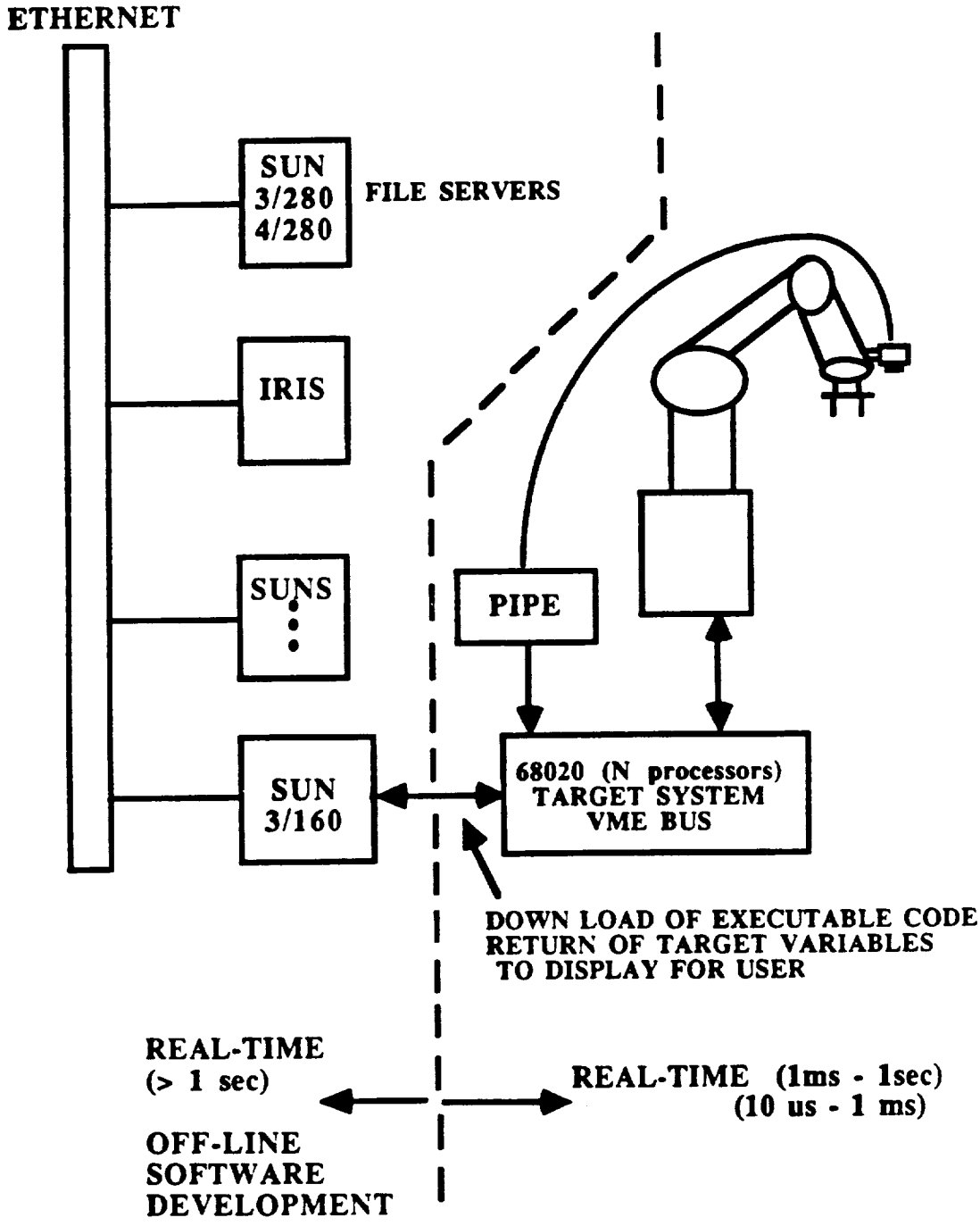

FIGURE 3