Using NASREM

for Telerobot Control System Development

Dr. Ronald Lumia[1]
Robot Systems Division
National Institute of Standards and Technology
Bldg 220, Rm B-127
Gaithersburg, Maryland 20899

Abstract

Using a functional architecture such as the NASA/NIST Standard Reference Model for Telerobot Control System Architecture (NASREM) to implement specific applications is helpful because much of the work resides in the infrastructure and the infrastructure is the same for all applications. Rather than recreating this infrastructure, our approach is to develop the additional 20% of the code which tailors the infrastructure for the specific application. This paper describes the process by which a system based on NASREM is developed.

1. Introduction

The Intelligent Controls Group of the Robot Systems Division at NIST has been implementing the NASREM architecture for several years. NASREM is a hierarchical system with six levels which is described in greater detail in [1-7]. While the implementation originally supported the development of NASA's Flight Telerobotic Servicer, the robot which would build and maintain the Space Station, our approach was to build a generic robot control system testbed. Consequently, the architecture itself should be independent of the software and hardware that implements it.

A generic architecture such as NASREM forms the basis for specific applications. Consequently, there are two fundamental types of modifications which must be supported. First, the architecture must be applicable to different application areas. Otherwise, it can be argued that the architecture is too focused on a particular application or industry. In the Robot Systems Division, the NASREM architecture has been used to develop the control system for underwater vehicles [8], mining vehicles [9], autonomous land vehicles [10], and as the basis for the Next Generation Controller, as well as for its original Space Station application. Second, the architecture must support

---

1. This work is a contribution of the National Institute of Standards and Technology and is therefore not subject to copyright.

evolution with technology. It must be easy to integrate new sensors into an existing application, to compare alternative algorithms, and to facilitate the integration of these modifications into the real-time control system. These two types of system modifications must be a natural consequence of the steps of system development. This paper describes how a system is developed using NASREM as the functional architecture, i.e., a NASREM methodology.

A *methodology* will be defined as the set of tools, techniques and evaluation criteria applicable to the system development steps. An effective methodology must be well-suited to the capabilities and limitations of humans as well as to those manifested by the current state of technology. The major human limitation is dealing with complexity; the major strength is reasoning. Humans cope with complexity by decomposing complicated systems into groups of simpler subsystems. To handle complexity, system developers often adopt top-down strategies for planning and design, bottom-up strategies to express experience-based knowledge, and pattern recognition to organize the collection of subsystems. We will define ***the NASREM Methodology*** as the collection of concepts and techniques that are geared toward improving the analysis, design, and implementation of real-time control applications.

The paper is organized as follows. First, an overview of NASREM and its philosophy is given. This is followed by a description of the system development methodology, the step by step approach to add the application dependent code to the NASREM infrastructure. Then, the servo level for a NASREM robot control system is described which will serve as an example of using the methodology.

2. NASREM Overview

2.1. NASREM Description

The NASA/NBS Standard Reference Model Telerobot Control System Architecture (NASREM) is outlined in [1]. The architecture emphasizes that the decomposition should maintain certain architectural features with respect to parallel processing and process communication such that the final design accurately represents system parallelism. This type of design provides advantages in building an evolvable system capable of meeting real-time performance requirements.

Figure 1 depicts the basic structure of the NASREM architecture applied to a telerobot control system. This figure is only meant to show the architecture at a conceptual level. The actual system has much additional structure not revealed at this level. As shown in the figure, the architecture is composed of levels. These levels are given names. The highest level is called the Service Mission level. The lowest level is called the Servo level. These names were chosen for the Space Station application. For a different application, the names of the levels may change but the design approach to populating the levels with code will remain the same.

At each "control" level there are sensory processing, world modeling and task decomposition components. Task decomposition components of the architecture are the behavior generators. These elements determine what is to be done and send commands to other system elements to affect that activity. Sensory processing components obtain and process data from system sensors. World modeling components utilize sensory processing data to maintain an internal model of the world. World modeling is the part of the system which interfaces sensory processing and task decomposition activities. Much of what world modeling does has to do with updating the global data system used by sensory processing and task decomposition.

As mentioned above, Figure 1 is conceptual. The figure depicts a system composed of 18 boxes. The figure does not show the level of detail needed for an actual system. For example, the figure does not show how the architecture decomposes around equipment. Figure 1 seems to indicate that the world modeling functions for camera and arm movements are more closely coupled than are the world modeling and task decomposition functions within each of these pieces of equipment. The figure shows one world modeling box and one task decomposition box at each level without showing any separation according to equipment. Clearly, world modeling functions related to arm movements interact much more with arm task decomposition than with eye movement world modeling. Thus, the level of detail of Figure 1 is not adequate for representing the actual system architecture.

One important feature of the architecture not depicted in Figure 1 is that the architecture hierarchically decomposes around equipment. There is a single Task level for an entire robot. This Task level coordinates Elemental-move (E-move) levels for each major equipment subsystem. The E-move levels each coordinate a set of Primitive levels, which in turn coordinate Servo levels for each separate piece of equipment that must be controlled. The levels form a hierarchy (or tree) structure with a single Task level root node and Servo level leaf nodes. As an example of how the architecture decomposes around equipment consider the diagram shown in Figure 2. This diagram depicts a hypothetical robot system that consists of an arm with a simple gripper, and a camera with pan, tilt, zoom, focus and iris control.

There is a Servo level for controlling the manipulator arm. This Servo level contains sensory processing, world modeling and task decomposition components. The Servo level for the arm is commanded by a Primitive level for the arm. (For a more detailed description of the structure and functionality of manipulator arm Primitive and Servo levels see [2-4].) There is also a Primitive level for the gripper. These Primitive levels have sensory processing, world modeling, and task decomposition components. The gripper and arm Primitive levels are coordinated by an E-move level. If there were more than one arm, the second arm and its gripper would also have Primitive levels under this one E-move level.

There is no Servo level beneath the gripper Primitive level, and the Primitive level communicates directly with the sensors and actuators of the gripper. The gripper in

this example is assumed to be controlled open-loop. That is, discrete actions result in the gripper being opened or closed, such as with a pneumatic device. There is no closed-loop control for this device that provides a servo to a commanded position or gripping force. In cases such as this, where there is no closed-loop servo control, it is appropriate to command the device directly from the Primitive level. For the camera device in Figure 2, there are servo levels for the moveable components (pan & tilt, zoom, etc.) because these devices have closed-loop control in the example. If the iris mechanism, for example, were capable only of receiving open-loop commands of aperture settings, then this device would not have a Servo level.

The architecture is determined by the types of control system interfaces provided by the selected equipment. For example, if a simple gripper allowing only discrete open/close commands is used, then the architecture would not have a Servo level for gripper control. However, if a more complicated gripper accepts finger position commands and provides finger position feedback, then an implementation architecture should include a gripper Servo level to servo control gripper opening. The architecture design is dependent on the type of interface provided by the equipment, since this interface completely determines the ability of the control system to influence the behavior of the equipment.

As a further example, consider a gripper that has internal electronics such that it servos to a commanded position using internal position feedback. This feedback is not provided externally, it is only used internally to close a position loop. Since the interface to this equipment consists of command positions only, it can be considered an open-loop device for the purpose of architecture design. Again, the control system cannot specify the behavior of the local electronics; the behavior is a fixed attribute of the equipment which lies outside the control system architecture.

A requirement of the telerobot architecture for the Space Station is that it be able to evolve and incorporate new capabilities. This implies that individual components be easily modifiable, capable of executing new algorithms and communicating with new system elements. This is not generally a feature of hardware electronics of the type mentioned for the gripper servo above. Since the control system must exhibit many different capabilities, components must also be capable of performing many different algorithms, during normal operation. Again, this eliminates most hardwired control electronics from being included in the system architecture. Although one might label some collection of circuits as being a Servo level for, say, a gripper, this does not give those circuits the properties of multiple capabilities and evolvability that is required of a telerobot control system architecture. Thus, the architecture includes only the programmable components of the control system, the "software" components.

Note that the camera structure includes a Servo level for the image itself. This structure appears to be in conflict with the above mentioned criterion for the existence of a Servo level. Indeed, if the only task decomposition output were "on/off camera"

this would be a conflict, however task decomposition is also involved in planning for and selecting the sensory processing algorithms. Remember that task decomposition determines what is to be done by the system. This means that task decomposition configures the activities within a level, as well as commanding lower levels. Thus, there is a Servo level for configuring and coordinating low-level image processing activities. Due to the complexity of these activities, a level is required to "servo" thresholds and filter parameters for this image processing. (See [5] for details.)

As a final comment on equipment hierarchies it should be mentioned that they can be dynamic. That is, the hierarchy can be reconfigured while the system is in operation. As a simple example consider the case where, during normal operation, the manipulator's end-effector is replaced with a new one. The new end-effector may have completely different control requirements such that a new control architecture for this subsystem is mandated. This problem can be accommodated by dynamic reconfiguration of the control hierarchy.

2.2. NASREM Tenets

NASREM provides fundamental real-time control system analysis, design and implementation strategies and heuristics that we will term tenets. We will use the word tenet to mean guidelines and rules of thumb which characterize the philosophy of the NASREM methodology. The NASREM Methodology is based on the following empirically established method tenets:

- Controller node building blocks. A controller node C is a 4-tuple C = (SP, WM, TD, HI) consisting of a sensor processing component SP, a world modeling component WM, a task decomposition control component TD and a human interface component HI. These functions are assumed to exist in each controller node.
- Each controller node C has one supervisor at a time, and to limit complexity, a maximum of only seven + or - two subordinates per controller node.
- System controller organized as a hierarchical collection of controller nodes C.
- Control hierarchy organized around tasks top-down, and equipment bottom-up.
- Order of magnitude differences in spatial and temporal resolution partitions controller nodes into adjacent hierarchical levels.
- Design emphasis on concurrency vis a vis real-time processing to handle computationally intensive control and sensor operations - especially as hardware costs continue to fall.
- Controllers exhibit synchronous control at the lowest levels and transition to asynchronous control at the highest levels.

Based on these tenets, we will describe the NASREM system development method-

ology and provide a specific example of using it.

3. System Development Methodology

The NASREM project development steps are intended to provide a formal and systematic approach to analyzing, designing, and implementing practical intelligent machine systems which would typically perform some physical work in a given environment (e.g., operate a vehicle, assemble parts, perform construction or mining tasks, etc.). If several competing conceptual designs exist, the method could be used to evaluate the feasibility and expected performance of each design, as part of the engineering analysis and selection process. The method might also be used to analyze the potential benefits and trade-offs which might result from selecting among different system components within a particular conceptual design (e.g., selecting different types or sets of sensors and actuators or integrating "black box" off-the-shelf subsystems). The NASREM methodology described in this paper should be interpreted as an iterative, real-time software development method. The steps listed are roughly in the sequential order, however the developer(s) should allow iteration within and between steps.

The method describes integration of executable controller modules in a bottom-up process. As these controller modules are integrated, their behavior can be studied and their performance measured to further refine the control hierarchy. This process should involve revisiting and revising the original problem description and requirements as well as the organizational structure and definition of the NASREM controller modules. As the running system evolves it should be used as a tool to enhance the dialogue with the domain experts and sponsors. By demonstrating the evolving system, developers, experts and customers are better able to refine requirements and explicitly capture the domain expert's knowledge.

The process of developing a NASREM system is shown in Table 1. A more complete description of the entire system development process is provided in [11, 12]. The process starts with a definition of the task goals, and the performance requirements of the control system in order to develop a complete problem description, task scenarios, and expectations.

In Step 2, the system developer gathers knowledge of the problem domain to define the information which is fundamental to the problem. While Step 1 was concerned with the goals for the system, Step 2 is concerned with the tools which will help achieve these goals. Consequently, the processes must be defined. This clearly depends on the specific equipment and problem but could include information such as the object and workspace geometry, machine and load kinematics, dynamics and coordinate systems, etc.

A systems analysis is performed in Step 3 to conceptualize the application in terms of Resource Management (NASREM Hierarchy), Data Management, Human

Interface Management, and Communications Management requirements. The architectural design analysis of the Resource Management requirements defines and models the set of resources bottom up. Then, the controller framework, i.e., NASREM hierarchy is built to establish the relationship between resources which exhibit common spatial and temporal characteristics. This is followed by a mapping of the capabilities into NASREM hierarchial control nodes. Capabilities may be distributed across individual controller boundaries. The NASREM systems analysis continues by looking at the data management. One defines a set of objects which can be acted upon and lists their relevant attributes in terms of their hierarchical structure, e.g., points, linear features, surfaces, objects and groups.

Each application needs a vocabulary which is typically unique to the application. In Step 4, NASREM task analysis, the task tree vocabulary for each level of the hierarchy is defined. This includes the tasks, the task frames, procedures, etc. This step balances the system goals with the capabilities of the system. One iteratively refines the system controller and/or the task descriptions based on either deficiencies in the task objectives, resource capabilities, information modeling, or communication interfaces until the goals of the tasks are met by the specified system.

In Step 5, software modules are coded and tested. The code for each controller node is developed using generic NASREM coding templates in a bottom up fashion. Simulators are developed to drive each controller in a closed-loop fashion and the human interface and display simulators are developed as required. Each software module is tested and measured to determine the expected performance of each controller node. This may involve subdividing controller nodes into concurrent processes to meet real-time performance goals.

Step 6 integrates and tests the controller nodes. One maps the concurrent processes onto actual computer hardware and assesses the performance. First, controller nodes have only their communication interfaces. When these interfaces are verified, code is added module by module to test controller node functionality. As more modules are integrated into the system, it is possible that more controller nodes may be necessary to meet the design criteria.

The last step is maintenance. The above six steps are iterated in order to fix, improve, or extend the controller nodes and the application system.

4. System Development Methodology for a NASREM Robot Controller

4.1. Description of the General Approach for Real-time System Development

It is assumed that the hierarchy for the desired equipment has been established for step 3 of the methodology as shown in Section 2.1. This section will concentrate on steps 5 and 6 of Table 1, from code development toward integrating the code into the real-time robot control system. First, atomic units are defined [6]. These atomic units

are the smallest chunk of software, which if broken into smaller chunks, would result in a less efficient implementation. For example, robot inertia compensation should be done for all joints together. If each degree of freedom is implemented in parallel, the communication between the degrees of freedom would take more time than that gained by the parallelization of the processes. The implementation of the atomic unit, which is called a process, has five properties:

- Continuous cyclic execution
- Read-compute-write execution cycle
- Concurrency
- Interfaces through global data system
- Activation/inactivation

First, a process repeatedly performs its designated function. Secondly, each execution cycle of a process consists of reading inputs, performing computation, and writing outputs. A process that repeatedly performs this cycle is said to be cyclically executing. Since a process can run concurrently and asynchronously with the other processes in the system, a process should be capable of retaining some context from cycle to cycle in the form of process variables. Process variables are not globally defined. The fourth property, however, states that the inputs and outputs of a process communicate via the global data system, and therefore must be globally defined. Finally, a process can be made active or inactive. For example, there is no need to compute inverse kinematics if the current control algorithm does not use the information. This allows the computing resources to be shared.

Once each process has been programmed and debugged as well as possible on the host system, it is cross compiled for use in the target system. Then, each process is assigned to a processor. For economic reasons, there will be fewer processors than processes. The process to processor assignment is critical to achieve good runtime performance. This assignment is revised often to hone the real-time performance of the system. Consequently, it should be easy to modify the assignments. At this point in the design process, load modules for each processor are available.

The load modules are downloaded to the computers in the target system. Depending on what is downloaded, the goal may be to obtain timing measurements for a particular algorithm, determine the performance of the algorithm during a task, or measure the performance of the entire system. At this point the testbed can be used for any research and development needs of the investigator.

4.2. Atomic Unit to Processor Assignment for the Real-time System

The result of process to processor assignment and the resulting system will be illustrated by looking at the lowest NASREM level, the Servo level. Figure 1 shows the atomic units associated with the Servo level. The square boxes depict the processes

while the ovals denote the data buffers which communicate through the global data system. A diagram similar to this is developed for each part of the system so that all of the processes are identified. Then, the system designer assigns each process to a particular processor for the real-time execution phase. Figure 2 shows the current process to processor assignment for the current NIST implementation of NASREM. The manipulation backplane is shown in the upper half of the diagram. Seven processors perform the Prim and Servo functions associated with manipulation. One interface card connects the manipulation backplane to the RRC controller (power electronics for the robot). Also, two BIT-3[1] parallel interfaces reside in the backplane, one to connect to the SUN host, the other to connect the Manipulation and Perception backplanes. The lower half of Figure 2 shows the process to processor assignment for the perception backplane. With the current configuration, the manipulation Servo loop, which is the most time critical part of the system, supports a 2.5 ms sampling rate and a 5 ms loop time for the critical path of most control algorithms.

5. Conclusion

Using a functional architecture helps to improve efficiency in developing and integrating a system for a particular application. The contribution of this paper is to outline a methodology for system development in a way that makes it possible to implement a system.

6. References

[1] Albus, J.S., McCain, H.G., Lumia, R., "NASA/NBS Standard Reference Model Telerobot Control System Architecture (NASREM)," NIST Tech. Note 1235, NIST, Gaithersburg, MD, July, 1987.

[2] Fiala, J., "Manipulator Servo Level Task Decomposition," NIST Technical Note 1255, NIST, Gaithersburg, MD, October, 1988.

[3] Wavering, A. "Manipulator Primitive Level Task Decomposition," NIST Technical Note 1256, NIST, Gaithersburg, MD, October, 1988.

[4] Kelmar, L. "Manipulator Servo Level World Modeling," NIST Tech. Note 1258, NIST, Gaithersburg, MD, December, 1989.

[5] Chaconas, K. J., Nashman, M., "Visual Perception Processing in a Hierarchical Control System: Level 1," NIST Tech. Note 1260, June, 1988.

[6] Fiala, J., "Note on NASREM Implementation," NISTIR 89-4215, NIST, Gaithersburg, MD, December, 1989.

---

1. Certain commercial equipment is identified in this article to describe the work adequately. Such identification does not imply recommendation or endorsement by NIST, nor does it imply that the equipment is necessarily the best available for the purpose.

[7] Kelmar, L. "Manipulator Primitive Level World Modeling," NIST Tech. Note 1273, NIST, Gaithersburg, MD, December, 1989.

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

[9] Huang, H.M., "Hierarchical Real-Time Control System for Use with Coal Mining Automation," Fourth Conference on the Use of Computers in the Coal Industry, Morgantown, WV, June 1990.

[10] Szabo, S., Scott, H.A., Murphy, K.N., Legowik, S.A., "Control System Architecture for a Remotely Operated Unmanned Land Vehicle," Proceedings of the 5th IEEE International Symposium on Intelligent Control, Philadelphia, PA, September, 1990.

[11] Michaloski, J., "Elements of Real-time Intelligent Control System Design," NIST Technical Note, NIST, Gaithersburg, MD, March, 1992

[12] Quintero, R., Barbera, A.J., "A Real-Time Control System Methodology for Developing Intelligent Control Systems," NISTIR 4936, Gaithersburg, MD, October, 1992.

Table 1. NASREM System Development Methodology

1. Develop problem description, task scenarios and expectations.

1.1.Define the task goals, and performance requirements of the control system.

2. Gather domain knowledge.

2.1.Define processes, object and workspace geometry, machine and load kinematics, dynamics, and coordinate systems

3. NASREM Systems Analysis. Conceptualize the application in terms of Resource Management (NASREM Hierarchy), Data Management, Human Interface Management, and Communications Management requirements

3.1.Perform architectural design engineering analysis of the Resource Management requirements

3.2.Perform Data Management Engineering Analysis

3.3.Perform Communications engineering analysis

3.4.Perform Human Interface engineering analysis

4. NASREM Task Analysis. Perform Task Decomposition vis a vis NASREM Systems Analysis.

4.1.Develop a task tree vocabulary for communication interfaces based on functional capability of controller node.

4.2.Define tasks, task frames, and procedures (scripts, process plans, or state graphs/tables), including timing and synchronization at each controller node.

4.3.Iteratively refine System Controller and/or Task Descriptions based on either deficiencies in task objectives, resources capabilities, information modeling, or communication interfaces.

5. Code and Test Software Modules.

5.1.Develop code for each controller node using generic NASREM coding templates in a bottom-up fashion.

5.2.Develop simulators to drive each controller in a closed-loop fashion.

5.3.Develop human interface and display simulators required.

6. Integrate and Test Controller Nodes

6.1.Map the concurrent processes onto actual computer hardware and assess projected performance expectations.

6.2.Recursively add controller nodes. First, test controller nodes in stand-alone operation. Second test intra-controller process communication interfaces. Third test inter-controller communication interfaces. Finally, test the suite of controller node functionality.

7. Maintain

7.1.Iterate on above steps to fix, improve, modify, extend the controller nodes and application system.

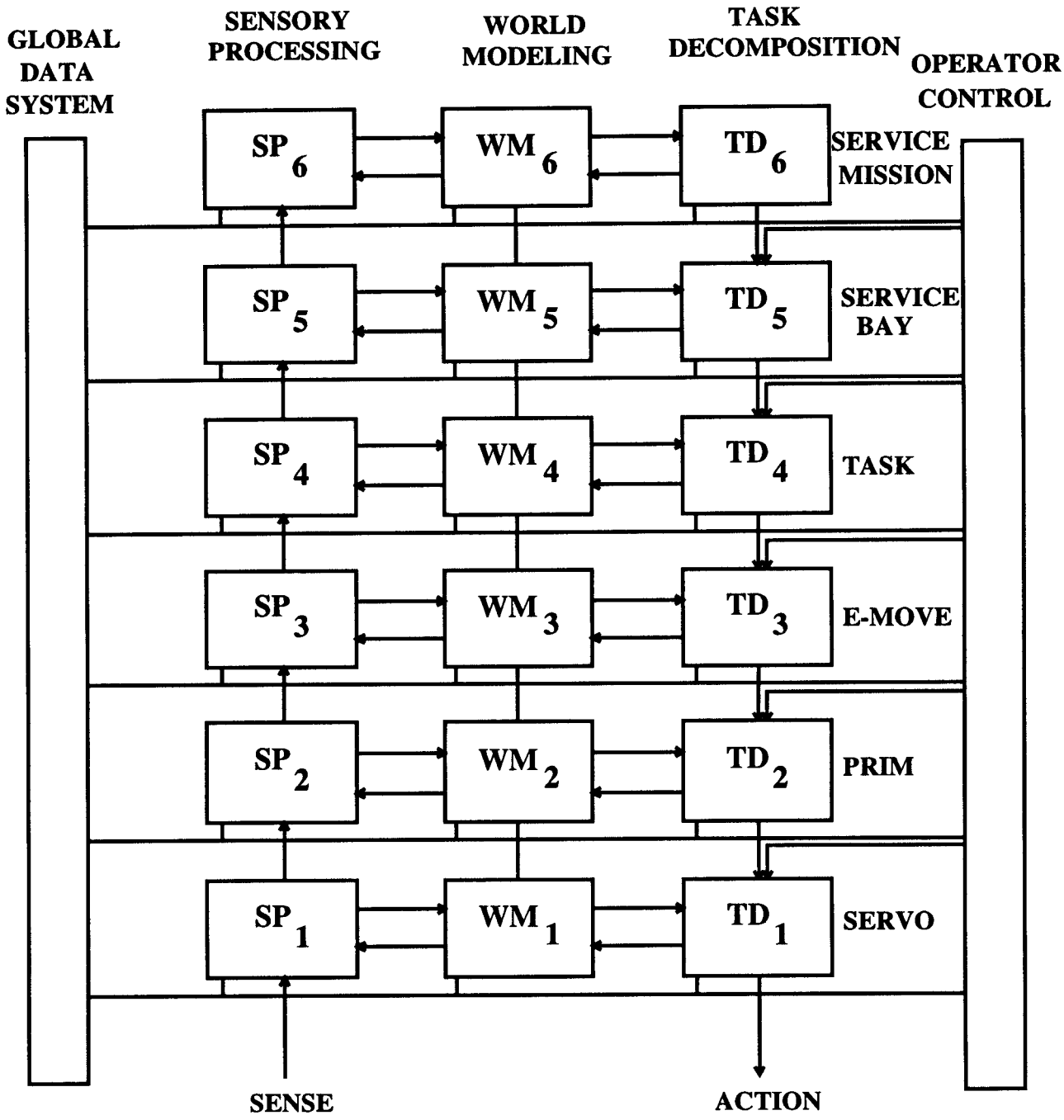

Figure 1. NASREM control architecture

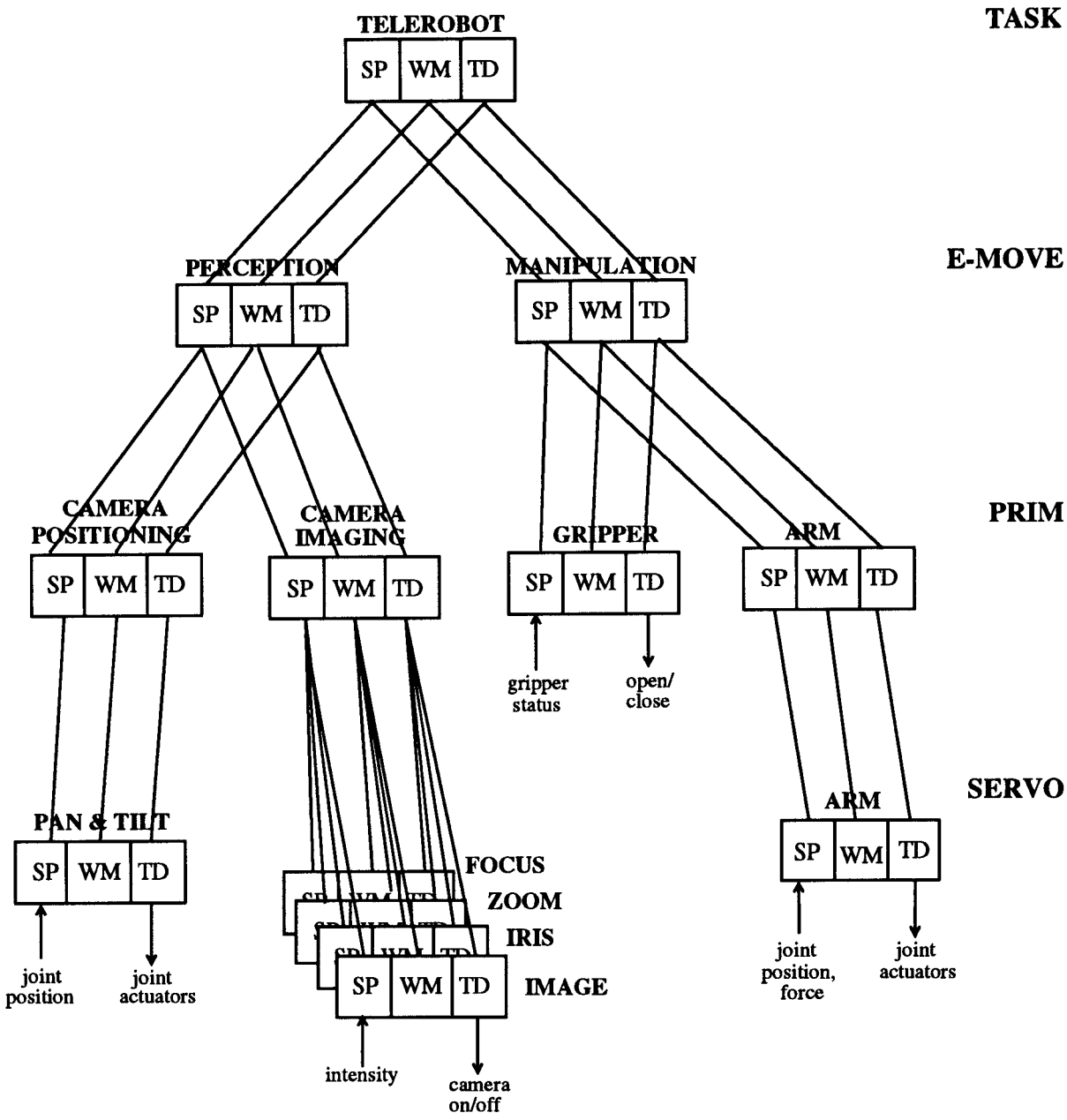

Figure 2. Hierarchical Decomposition around Equipment.

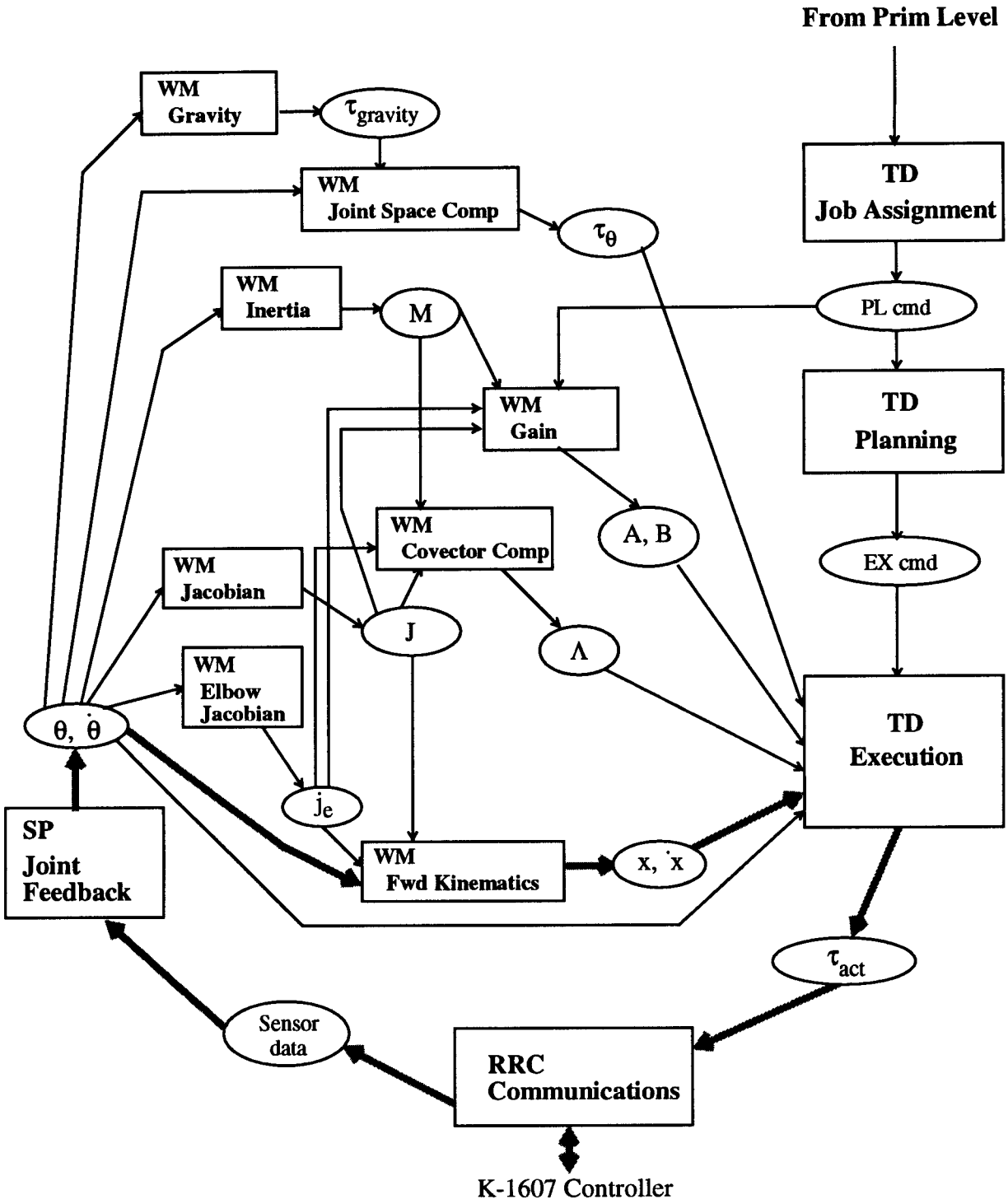

Figure 3. Servo Level Boxes Representing Atomic Units

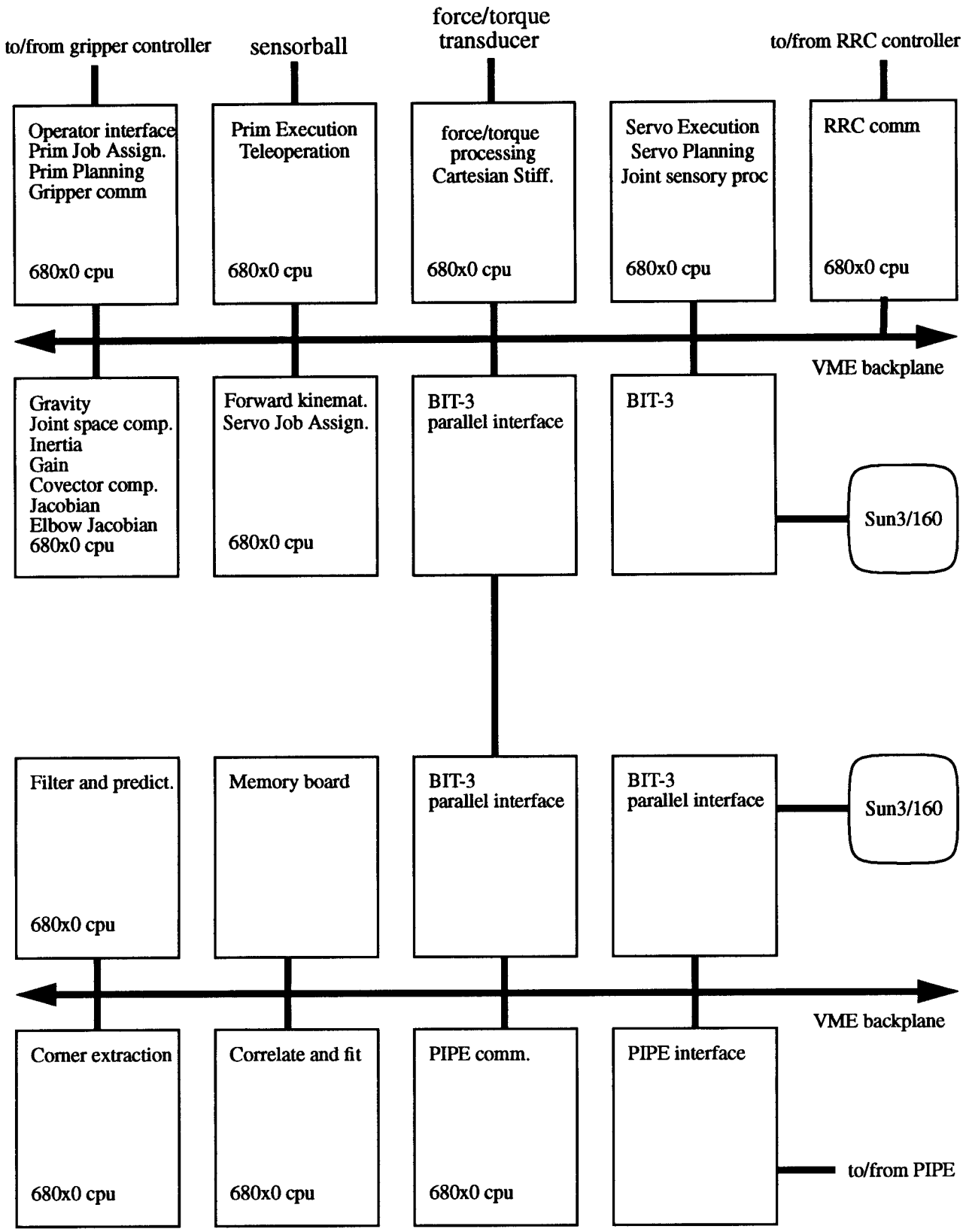

Figure 4. Process to Processor Assignment for the NIST NASREM Implementation