Design Methodology for NASREM Software Development

Dr. Ronald Lumia
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) when implementing 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, the 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. While the implementation was in support of NASA's Flight Telerobotic Servicer, the approach was to build a generic robot control system testbed. NASREM is a hierarchical system with six levels which is described in greater detail in [1-7].

Since the NASREM architecture has been developed in a technology independent manner, there are two types of modifications which must be allowed. First, the architecture must be able to support different application areas. The NASREM architecture has been use for underwater vehicles [8], mining vehicles [9], autonomous land vehicles [10], as a basis for the Next Generation Controller, as well as for its original space application. Second, the architecture must facilitate the integration of new sensors even if the application does not change so that the implementation may evolve with technology. 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 human capabilities and limitations as well as the technology capabilities and limitations. The major human limitation is dealing with complexity. Humans cope with complexity by decomposing complicated systems into groups of simpler subsystems. To handle complexity, system developers adopt bottom-up strategies to express experience-based knowledge, top-down strategies for planning and design, 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, the basic tenets of a NASREM implementation are given. This is followed by a description of the system development methodology. Then, a NASREM robot control system is described which is an example of using the methodology.

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, here, 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:

task (or process) oriented decomposition

controller node building blocks, a controller node C is a 4-tuple C = (SP, WM, BG, HI) consisting of a sensor processing component SP, a world modeling component WM, a behavior generation control component BG 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, only seven + or - two subordinates per controller node

human interface is possible at all controller nodes

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 execute as finite state machines exhibiting synchronous control at the lowest levels transitioning to asynchronous control at the highest levels

2.1. Task oriented decomposition

Every methodology seeks to devise a model of the problem domain which explicitly represents and emphasizes the most critical components of the problem while simplifying those aspects which have a lesser impact on the solution being sought. Every model is, after all, a simplification of the real world. The objective is to choose an abstraction (or a set of abstractions) that highlights the parts of the problem that make a difference in the quality and efficiency of the solution. That is why it is so important to choose a methodology which matches the problem domain. In the domain of intelligent machine control systems we believe tasks are the driving factor.

We characterize the domain of intelligent machine control systems to include the control of electro-mechanical devices, designed to perform some useful work, using computerized control. Intelligent machines are further assumed to possess the capability to respond to the physical environment in some intelligent way, in real-time. This definition is intended to cover automation systems, embedded systems and robotic systems.

Within this problem domain, practical control systems solutions are always defined within the context of the tasks to be performed (electro-mechanical actions to

be taken). The process of designing practical system solutions involves balancing trade-offs between general purpose capabilities and task specific requirements. Examples of such trade-offs can be seen in common human transportation systems. People have developed automobiles, trains, buses, aircraft, rockets, elevators and escalators all for transportation but each for a specialized class of transportation tasks. Recognizing this fact, this tenet suggests *that task decomposition should be explicitly represented in modeling intelligent real-time control systems.* Further it stipulates that task knowledge (control flow), task specific object knowledge and task specific data processing knowledge should be encapsulated within controller modules which respond with commands to their subordinates to accomplish system tasks when stimulated by task commands from their supervisors or other communications (task state knowledge) through global memory.

2.2. Hierarchical Organization

The primary objectives of the NASREM method is to manage software complexity, improve human understanding of the design and to provide for robust, efficient, coordinated, verifiable, real-time performance. Hierarchical organization has been found to be a key element in realizing these goals. Humans have used hierarchical organizations to manage complexity and real-time coordination problems throughout history.

There are also numerous examples of the power of hierarchical organization in the animal world. Humans, many insects and other animals instinctively form social hierarchies to build habitats, hunt for or produce food, reproduce, nurture their young and in general improve their quality of life and their chances of survival. People use hierarchies whenever complex real-time coordination of more than one individual is necessary. Notable examples are sports teams (managers, coaches, team captains/signal callers, player positions and specialized team groupings), enterprise structures (corporations, chief officers, divisions, plants, departments, sections, groups, worker specialties) and military organizations (Commander in Chief, Army, Theater, Corps, Division, Brigade, Battalion, Company, Platoon, Squad, Soldier). These organizational structures employ strategies and tactics in their procedures in order to deal with unstructured environments and uncertain information in an ordered way. Based on these structured plans measures of performance can be defined to help refine real-time performance. These highly organized groups typically spend a significant amount of time training and practicing their designated functions in order to maximize their real-time performance and the quality of the result. The NASREM analogy to these examples is a highly modular partitioning of process intelligence codified into a hierarchy of controller nodes. Each controller node has a bounded clearly defined range of authority and responsibility, each deals with a particular layer of abstraction and timing horizons within the problem domain and each has clearly defined vertical and horizontal channels of communication.

A popular alternative software organizational structure often used in software engineering is the notion of "independent cooperating agents." In this technique software agents are created which are roughly equal in authority and responsibility. These agents must then negotiate for a share of control over a finite pool of resources (computing time, actuators, sensors, fuel, power, goal designation, direction and speed of

movement, etc.) in order to achieve a desired set of system goals. They negotiate according to some established set of arbitration rules (priorities, voting, ordered sets of constraint conditions, etc.) and usually use message passing over a computer network as a communications scheme. Some have suggested applying these techniques to real-time intelligent machine applications. This technique is analogous to the human parliamentary organizational structure. Humans use this type of structure to establish policy, to define arbitration rules, to divide resources and to create or modify rules or laws of social interaction (e.g., government, civic organizations, standards bodies, etc.). Such human organizations are not known for there efficient real-time performance. The results achieved using this structure are also highly unpredictable. The principle utility of such structures is to insure some measure of "fairness" in the result as well as a social sense of "due process." In short, we don't believe this technique alone (without hierarchical organization) is a better choice for achieving robust, efficient, coordinated, verifiable, real-time performance. On the other hand, a NASREM implementation could certainly benefit from the judicious use of these techniques when the application warrants them.

2.3. Hierarchy organized around tasks top-down, and equipment bottom-up

The process of developing a NASREM application and organizing its control components is an iterative one which begins with a top-down decomposition of tasks to be performed (forming a task tree structure) and the organization of a hierarchy of controller nodes, which will be responsible for coordinating the tasks to be performed using a bottom-up design procedure. The order in which these two activities take place is not specified and is less important than realizing that the objective of this iterative design process is to map the task tree onto the controller hierarchy.

By organizing a NASREM hierarchy around the equipment to be controlled (machines, actuators and sensors) in a bottom-up process we can minimize and simplify the difficult real-time problems of resource contention, conflict resolution and scheduling. Every man-made machine ultimately has a finite number of actuators and sensors through which it can influence the problem environment. We can capitalize on this fact by structuring a supervisor-subordinate hierarchy of intelligent controller nodes using a bottom-up approach starting from the actuators and sensors that the intelligent machine will be directly controlling. We can ensure that every subordinate is directed by only one supervisor at any instant in time (one software read/compute/execute cycle) and that there is a clearly defined supervisory controller for each individual actuator and sensor at any instant in time. Using this structure it is then possible to define coordination and scheduling plans to be used by each controller within the scope of its level of authority and responsibility. Furthermore, because the hierarchical structure predefines the responsibility and authority of each controller and the resources controlled at any instant in time, the problems of resource contention, conflict resolution and scheduling are dealt with locally within each supervisory controller (as defined by its local library of plans) and are independent of the concurrent actions of all other controllers at the same level of authority.

Whenever practical the designer should group hardware components (actuators and sensors) and software components (NASREM controllers) so as to minimize the number of interfaces necessary in order to implement closed loop control through the

NASREM hierarchy at the lowest level practical. The intent of this guideline is to localize the design of closed loop control within NASREM (into modular subsystems) in order to minimize the potential for "ripple effect" when evolving, expanding or maintaining the implementation and to minimize the need for and extent of high bandwidth feedback loops. Communications bottlenecks can also be eliminated by judicious grouping of closely coupled components. Physical constraints within the problem domain often will conflict with this guideline thereby imposing engineering trade-off decisions. Other NASREM tenets listed here might also conflict with this guideline. For example the "seven + or - two" tenet suggests adding hierarchy (and therefore additional interfaces) in order to manage complexity and improve human understanding. The "order of magnitude" tenet can also suggest the addition of more levels in the hierarchy and additional interfaces again in the interest of complexity management and understandability. As always, the engineer's function is to perform the engineering analysis necessary to arrive at a viable engineering design trade-off.

2.4. Order of magnitude between levels

In dealing with the issues of minimizing the impact of complexity and providing for human understanding of the design result we have devised a rule of thumb that suggests limiting the scope of the problem domain for any individual controller node or NASREM level to roughly one order of magnitude in terms of the resolution of the maps and the geometric models it uses for planning and in terms of the timing horizon it uses for planning and scheduling. This rule of thumb should not be strictly interpreted especially at the higher NASREM levels since its intent is to limit the number of planning steps any one planner needs to plan into the future. In some cases the number of required planning steps is event driven rather than strictly related to the passage of time. Good engineering judgement needs to be exercised when applying this rule of thumb since the objective here is to limit the scope of complexity in the coordination of tasks and to limit the number of planning steps (or decision points in a program) to be planned into the future to roughly less than ten. Some individual steps may have very long durations.

2.5. Limit control node complexity

This tenet seeks to manage task knowledge complexity and human understanding of the design result by adding hierarchy when more than roughly nine subordinates must be supervised by a controller. This rule takes its name from George Miller's work. He empirically found that human beings are able to manage seven + or - two things at a time without losing understanding. This guideline suggests adding a supervisory controller whenever more than nine subordinates must be controlled and that the optimum number of subordinates per supervisor is roughly five. Again software engineering judgement is always called for in applying these guidelines. It is possible that in a particular application only one supervisor may be needed for a much greater number of subordinates (e.g., twenty or more) if all of the subordinates are performing very similar tasks. We sometimes see this pattern in human organizations such as in assembly line production.

2.6. SP/WM/BG/HI functions assumed to exist in each node.

The NASREM architecture has evolved at NIST from a controls system point of view. It is grounded in the notion that closed-loop control is a fundamental construct of the architecture style. With this in mind we have defined a generic controller node as the basic processing entity through which closed-loop control can be implemented. Therefore, a generic controller node must be capable of performing the basic closed-loop control functions of Sensory Processing (SP), World Modelling (WM), Behavior Generation (BG), and Human Interface (HI). A NASREM architecture is then built of NASREM controller modules interconnected to form a hierarchical tree structure of controllers with each controller forming a node in the hierarchy. Furthermore the controllers are designed to be interconnected to form nested closed-loop control with each controller node contributing to one layer of problem abstraction and decomposition.

While the NASREM architecture philosophy focuses on highly modular closed-loop control, it does not require closed-loop control within every implemented controller. In fact it would be possible to implement an entire NASREM compatible architecture without including real-time sensory input for closed-loop control. This implies that while SP, WM, BG and HI functions may exist within every controller node, there is no requirement that more than one exist in any implemented controller node. In practice we often see open-loop controllers implemented at the lowest level of NASREM especially when implementing a simple on/off control function. In this case closed-loop control might be implemented at the next higher level through a supervisory controller node. One or more subordinates could be dedicated to SP functions and others might be dedicated to performing BG functions. The supervisory node in this example might perform simple SP functions and more complex WM and BG functions.

2.7. Human Interface is possible at all nodes.

Recognizing that every practical system must have a human interface at some level and that it is usually desirable to develop intelligent machine systems in an evolutionary manner, we believe it is prudent to build in the possibility for human interface to every controller node as part of the system design philosophy as opposed to adding the human interface capability in an ad hoc manner as the need arises.

2.8. Design emphasis on concurrency

This tenet suggests that it is generally more cost effective to add microcomputer hardware (single board computers and memory) to a multiprocessor environment than to increase the computing power of a central processing unit (CPU) and the complexity of its operating system and the control system application programs to dynamically manage the central processing unit's computing power. As a consequence of this tenet, the NASREM methodology differs from traditional procedural programming methods, which are aimed at producing sequential code to efficiently execute on a single CPU, in that it is an inherently parallel programming method which seeks to develop software processes (controller modules) which will execute in a cyclical and concurrent manner on multiple CPUs. These software processes can then be eas-

ily distributed across a multiprocessor hardware computing environment in order to meet real-time execution constraints. In general this method is less concerned with optimizing the utilization of any one CPU and more concerned with minimizing software complexity and providing a means to ensure deterministic and verifiable real-time performance.

2.9. Controllers as finite state machines

In keeping with the controls systems origins of the NASREM philosophy as well as the emphasis placed on programming for concurrent multiprocessor environments, this tenet defines a NASREM controller module as a finite state machine. Finite state machines are characterized as functioning on a read/compute/write execution cycle. Such machines divide time into discrete increments based on their execution cycle. An instant of time in this context is considered to be one read/compute/write execution cycle. Finite state machines deal with external stimuli as snap-shots in time and have a deterministic and verifiable response which depends solely on the internal state of the machine and the event or external stimulus being presented to the machine (through global memory) at any given instant in execution time. Finite state machines can exhibit amazingly complex and intelligent dynamic responses to uncertain stimuli even though they are completely deterministic and verifiable. These properties make them an excellent choice as the basic execution model for NASREM applications.

A state-driven NASREM implementation would typically employ periodic "servoing" or data sampling as opposed to "event-driven" interrupt processing in order to cut down on processing overhead and at the same time ensure deterministic and verifiable behavior (in terms of execution time and response time) particularly at the lowest levels of the architecture. At higher levels of an implementation - and lower performance bandwidths - it is often convenient to transition to an asynchronous control technique.

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, produce parts, perform construction or mining tasks, operate a sensor, 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 Figure 1. A more complete description of the entire system development process is provided in [11]. 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

This section will concentrate on steps 5 and 6 of Figure 1, from code development toward integrating the code into the real-time robot control system. First, atomic units are defined. 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. Depend-

ing 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 2 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 3 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 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 3 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] Kelmar, L. "Manipulator Servo Level World Modeling," NIST Tech. Note 1258, NIST, Gaithersburg, MD, December, 1989.

[4] Fiala, J., "Note on NASREM Implementation," NISTIR 89-4215, 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] Wavering, A. "Manipulator Primitive Level Task Decomposition," NIST Technical Note 1256, NIST, Gaithersburg, MD, October, 1988.

[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

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 System Development Steps

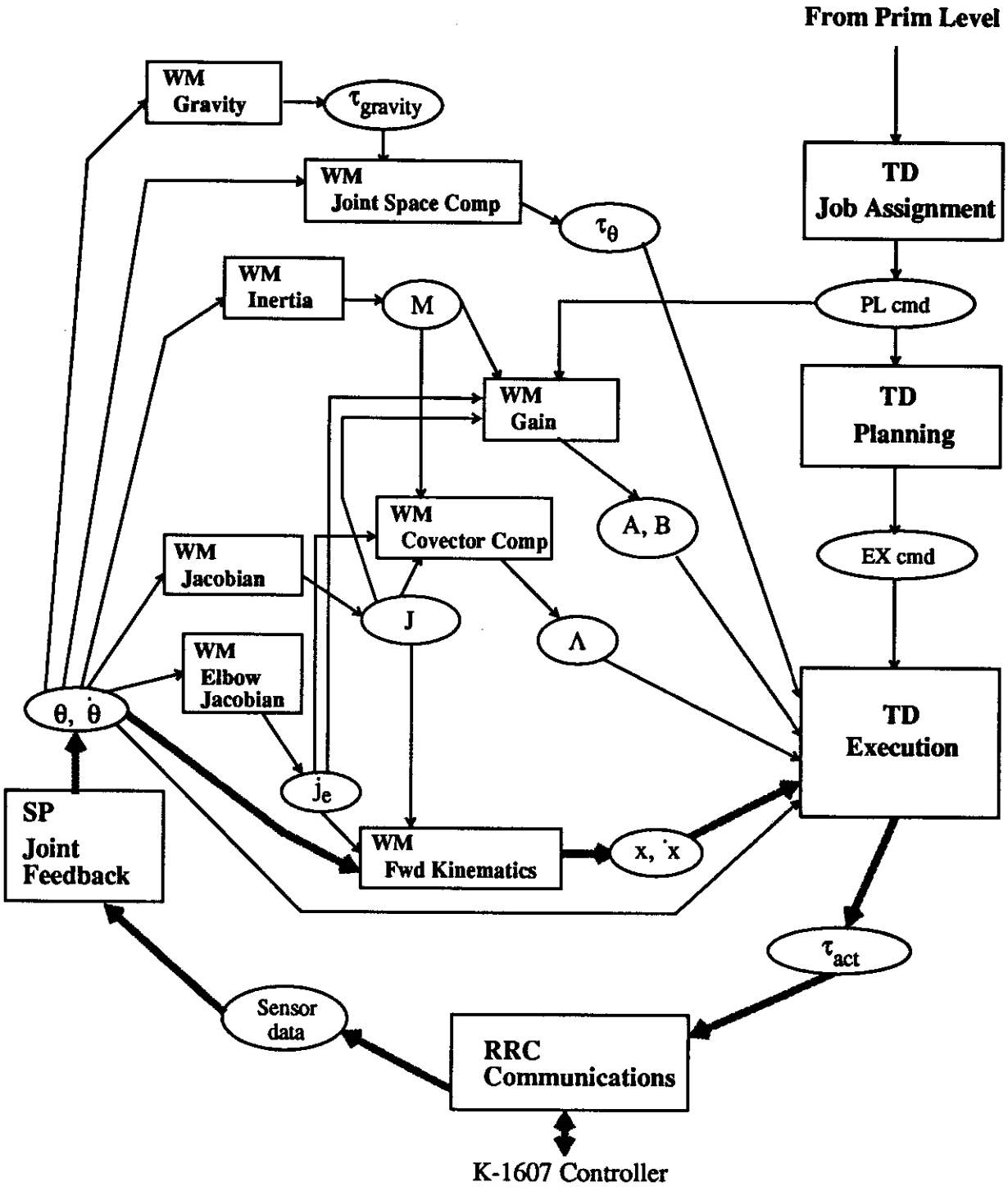

Figure 2. Servo Level Boxes Representing Atomic Units

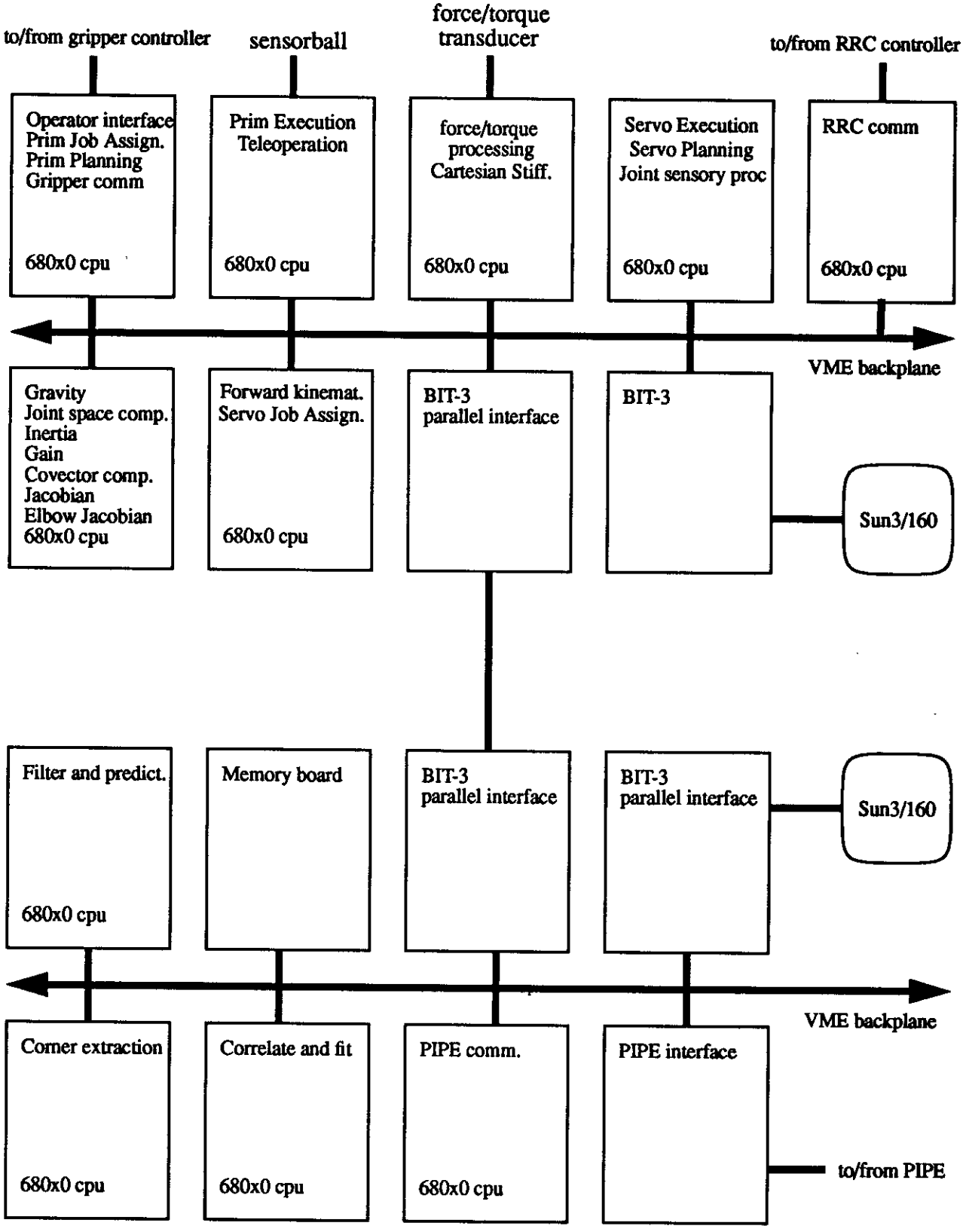

Figure 3. Process to Processor Assignment for the NIST NASREM Implementation