MAPPING PROCESSES TO PROCESSORS FOR SPACE BASED ROBOT SYSTEMS

Thomas E. Wheatley

National Institute of Standards and Technology
Gaithersburg, MD

ABSTRACT

The NASA/NBS Standard Reference Model for Telerobot Control System Architecture (NASREM) [1] has been adopted by NASA for use in the Flight Telerobotic Servicer, a two-armed telerobotic manipulator which will build and maintain the Space Station. NASREM provides the paradigm that allows standard interfaces to be defined so that functionally equivalent software and hardware modules can be interchanged. This paper examines the mapping of these logical modules onto a functional computer architecture. Interfaces must first be defined which are capable of supporting the algorithms in the literature. After interface definition, the specific computer architecture for the implementation can be determined. An example is shown mapping the SERVO level of NASREM onto a set of computers utilizing the knowledge of the dominant response time required to aid in the selection process.

INTRODUCTION

The NASA/NBS Standard Reference Model for Telerobot Control System Architecture (NASREM) [1] defines a logical computer architecture for the NASA Space Station Flight Telerobotic Server. The overall architecture is hierarchically structured with specific operations performed at each level. The lowest level is SERVO, where individual motors are servoed to achieve input commanded positions. Fiala [2] describes the functionality of the three SERVO Task Decomposition modules in NASREM (Job Assignment (JA), Planner (PL), and Executor (EX)), and defines interfaces internal and external to these modules to support algorithms in the current literature. Kelmar [3] describes the functionality and interfaces of the World Model (WM) module for the SERVO level, and Nashman and Chaconas [4] describe the same for the Sensory Processing (SP) module. In [5], Wheatley et al. introduce the notion of a virtual control loop to describe the interaction of these three modules within any particular level of NASREM and examine the various communication tools available to the system designer. Michaloski et al. [6] expand on this idea by exploring the use of the response time of each virtual control loop as a guiding factor in the mapping of processes to processors. Zhang and Paul [7] examine various servo control schemes in terms of their computational cost. This paper draws from the above concepts to demonstrate the stepwise process of mapping the logical modules of the SERVO level of NASREM onto a functional computer architecure utilizing a hybrid mix of multiprocessing and multitasking.

DISCUSSION

The assignment of processes to processors is typically an iterative one. There are four considerations that need to be examined to determine an allocation that can achieve the desired results: 1) the amount of data to be exchanged; 2) the execution time of each process; 3) the communication paths chosen for the data; and 4) the communication costs in terms of operating system support of moving the data. The first one is dictated by the interfaces of the processes and the specific application at hand, whereas the last three are determined by the current technology available.

The desired response time is the driving force in determining the tradeoffs between these four factors. It can be considered the sum of the execution time and the communication time [6]. The execution time is defined as the sum of the execution times of the individual processes. Data transfer internal to a process or I/O is considered part of its execution time. The communication time is defined as the sum of the times for operating system communication calls for moving the data between processes. In [5,6], ten percent of the response time for a particular control loop is suggested as a reasonable amount of time for communication. A 200 Hz update rate at the SERVO level would then have a 5 msec response time with 4.5 msec for execution and 0.5 msec for communication.

Although this time interval is very stringent, it is feasible with current technology. Table 1 details some typical times for common communication calls executed on a 20 MHz 68020 VME-based CPU board using ADA. These calls transfer "ownership" of the data without physically

Communication Call	Time in usec	Normalized
1. Subroutine	15	1.0
2. Task Signal	153	10.2
3. Task Swap	67	4.5
4. Send Datagram	90	6.0
5. Receive Datagram	68	4.5
1000 Data Transfers	**Time in usec**	**Normalized**
1. Direct Word	2027	1.0
2. Explicit Array	3533	1.7
3. Implicit Array	3637	1.8
4. Direct Word, VME	2784	1.4
5. Explicit Array, VME	3999	2.0
6. Implicit Array, VME	5480	2.7
7. Direct Word, VSB	2585	1.3
8. Explicit Array, VSB	3779	1.9
9. Implicit Array, VSB	4634	2.3

Table 1 - Communication Calls and Data Transfer Timing

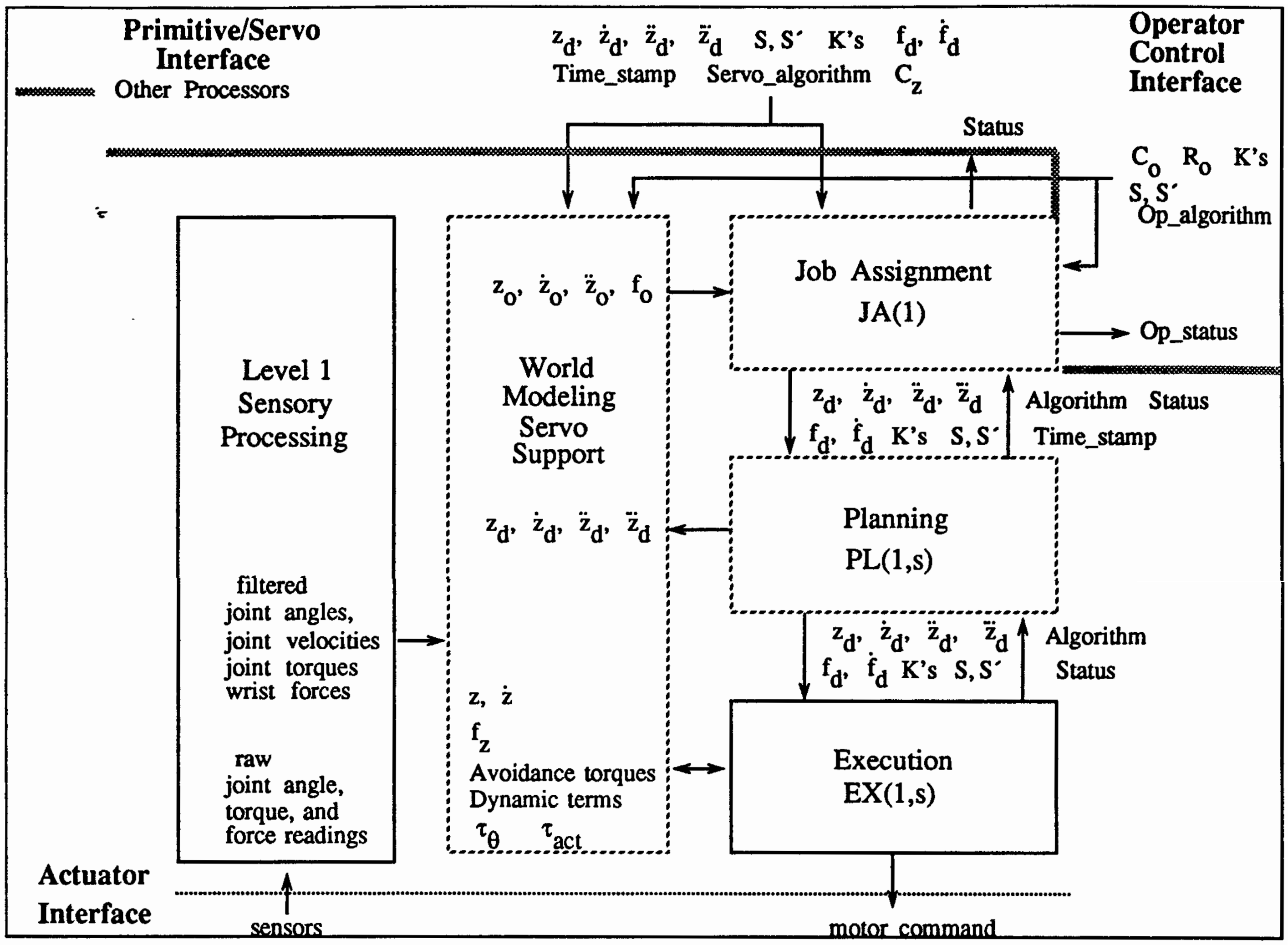
Primitive/Servo Interface
Other Processors
$z_d, \dot{z}_d, \ddot{z}_d, \dddot{z}_d$ S, S′ K's $f_d, \dot{f}_d$
Time_stamp Servo_algorithm C_z
Operator Control Interface
Status
C_o R_o K's
S, S′
Op_algorithm
Job Assignment JA(1)
Op_status
$z_o, \dot{z}_o, \ddot{z}_o, f_o$
Level 1 Sensory Processing
World Modeling Servo Support
$z_d, \dot{z}_d, \ddot{z}_d, \dddot{z}_d$ $f_d, \dot{f}_d$ K's S, S′
Algorithm Status Time_stamp
Planning PL(1,s)
$z_d, \dot{z}_d, \ddot{z}_d, \dddot{z}_d$
filtered joint angles, joint velocities joint torques wrist forces
$z_d, \dot{z}_d, \ddot{z}_d, \dddot{z}_d$ $f_d, \dot{f}_d$ K's S, S′
Algorithm Status
z, ż
f_z
Avoidance torques
Dynamic terms
τ_θ τ_{act}
raw joint angle, torque, and force readings
Execution EX(1,s)
Actuator Interface
sensors
motor command

Figure 1 - Interfaces for SERVO Level of NASREM

copying it. They are normalized to the subroutine call for comparison purposes. Table 1 also depicts some typical times for data transfers of 1000 32-bit words, and normalizes these values to the direct on-board data transfer. The destination in each case was on-board memory. The times for the array transfers incorporate the array addressing and/or loop overhead, whereas the direct transfers include just the loop overhead.

These were timed directly from the ADA code instead of from the raw bandwidth numbers possible on these communication paths. In this fashion, one obtains a truer picture of the interaction between the compiler and the hardware. In particular, the explicit array transfer using a "for loop" compiled into long moves, and the implicit transfer (equating the two arrays) compiled into a string move of bytes. The implicit move on-board was similiar in speed to the explicit move due to the efficiency of the string move and the absence of on-board arbitration for memory. The two off-board transfers show the effects of the additional bus arbitration due to the byte transfers, even though there was no other bus traffic.

A casual examination of these times highlights the costs involved by utilizing high-level system calls for a low-level function [5,6]. At an average of ~80 usec each, one could use only six high-level calls before exceeding our target limit of 500 usec. This by itself does not offer much guidance in the assignment of processes to processors. We must first examine the interfaces and the specific application at hand.

Figure 1 depicts the interfaces between the Sensory Processing, World Model, and Task Decomposition modules in the SERVO level of NASREM [1-4]. Here, z denotes the coordinate system of choice such as joint or Cartesian space, z_0 denotes the origin of that space, and z_d denotes the desired final position. The SP and EX modules run at the desired servo rate, and the JA and PL run at some sub-multiple of this rate [2,4]. The WM contains dynamic terms and Jacobians which can be computed at the servo rate, however, most control systems compute these also at some sub-multiple of the servo rate [2,8,9]. For purposes of example, we choose to define the response time required for the non-servo rate functions as five times the servo response time, or 25 msec. Based on this, we shall examine two potential configurations. Configuration 1 attempts to put all the SERVO level modules onto one processor, and Configuration 2 separates the non-servo rate functions (denoted by the dotted lines around JA, PL, and WM) from the servo rate functions onto two processors.

For Configuration 1, there are two incoming datagrams from the Primitive (PRIM) and Operator Control Interface (OCI) processors. Copies of these datagrams reside in both the WM and the JA modules. The burden of this copying is placed on the sending processors. Examining Table 1, the two datagram reads take 136 usec, leaving the data in on-board memory. For our example of 5 msec, this leaves 364 usec for the communication time.

It is often desirable to utilize a tasking model to separate processes based on either function and/or time [9]. Here, we separate the servo rate processes from the non-servo rate functions. For a simple two-task system, this adds 134 usec for two swaps, reducing the remaining time to 230 usec. For a three task system, the use of task signals would be required which adds 459 usec to the 136 usec of the two datagrams. This exceeds our limit of 500 usec for the communication time. Therefore, Configuration 1 is limited to a simple two task system with the servo rate modules running to completion in the foreground and the non-servo rate modules running in the background. This allows the background task to execute over five servo cycles in our example, using the remaining time in each 4.5 msec interval. The limiting factor now becomes the execution time of each task, and will be discussed later.

Configuration 2 physically separates the two tasks of Configuration 1 onto two processors. The servo rate processor A would have two interfaces to processor B (the non-servo rate functions) at the WM and PL interfaces. As in Configuration 1, this takes a total of 136 usec for A, leaving 364 usec for any internal communications. Since all the functions within A operate at the same rate, any tasking would be based on separation of functions. As the SP module is fairly simple and must be run first to supply the EX module with new data, it is an appropiate choice for a subroutine call by EX [5,6]. Therefore, the limiting factor for Processor A is the execution time of the servo control algorithm.

Processor B has a response time of 25 msec, leaving 2.5 msec for communications. The two datagrams to A total 180 usec. Since B is the sending processor, the data transfer must be accounted for. From [2], we determine the amount of data to be traded between the WM and EX processes for a six degree-of-freedom (DOF) arm to be 126 words. Of these, 24 words are from servo rate functions which are on A. This leaves 102 words coming from the functions on B to A. There are an additional 118 words passing across the interface between PL and EX, for a total of 220 words from B to A. Using the explicit array transfer time for the VSB bus from Table 1, this adds 829 usec to the communication time. The two incoming datagrams from PRIM and OCI processors bring the total to 1.14 msec. This leaves ample time for tasking among and/or within the three modules of JA, PL, and WM on processor B.

The use of the VSB bus is preferred here for both the increased speed and the tight coupling of A and B. The VME transfer takes 878 usec. The VME bus should be viewed as a global communication path with local bus traffic like this located on VSB clusters or other private communication paths. The use of dual-ported memory between these paths and the main communication path allows the notion of global memory to be preserved without sacrificing the bandwidth of the main communication path.

We now have two potential configurations for the process allocation based on the defined interfaces and the desired response time. In order to determine which algorithms can run on these or other configurations, we must examine the execution time. If the execution times of the individual processes are already known, they are simply added and compared to our target execution time. However, it is often desirable to estimate the execution time of an algorithm before actually coding it. This is fairly straightforward to do, especially for compute-bound algorithms such as servo control. To best illustrate the use of this technique, consider the simple proportional-derivative-integral (PID) servo control algorithm in [2]

$$K_p(\theta_d - \theta) + K_v(\theta_d - \theta) + K_i \int (\theta_d - \theta) + \dot{\theta}_d = \tau_{act.}$$

The integral is performed digitally by the following

$$\int_{t_0}^{t} (\theta_d - \theta) \Rightarrow \quad sum = sum + \Delta T(\theta_d - \theta)\text{ , where}$$

ΔT is the sample period. Here, the values θ and θ are calculated by the SP module, and supplied to the WM module. These are then passed to the EX module. The remaining terms are supplied directly by the PRIM module. An estimate of the calculations required to achieve this control algorithm for a six DOF arm are as follows:

1) θ , $\dot{\theta}$		6 adds	(SP)
2) $K_p(\theta_d - \theta)$	6 multiplies	6 adds	(EX)
3) $K_v(\theta_d - \theta)$	6 multiplies	6 adds	(EX)
4) $K_i \int (\theta_d - \theta)$	12 multiplies	12 adds	(EX)
5) remaining adds		18 adds	(EX)

This yields a total of 24 multiplies and 48 adds. Table 2 details the times for various integer and floating-point operations on the 20 MHz 68020/68881 system, again timed directly from the ADA code. Using the array operations on floats from on-board memory, we determine that the calculations would take 1024 usec. This time takes into account the reading of the values, the actual operation, and the storing of the values back to on-board memory. There is some additional program overhead that is difficult to estimate [6], however an additional 5% is a reasonable estimate. There is an additional time of 66 usec to transfer 24 words of I/O, bringing the total to 1.14 msec for this simple PID control, well within our self-imposed limit of 4.5 msec. This then could be implemented with ease on either Configuration 1 or 2, with ample room for growth.

Up to this point, the WM has been mostly passive. It maintains an internal model of the world and is designed to deal with different coordinate systems as well as dy-

Operation	Integer	Short Float	Float
1. Direct Add	2281	12752	13748
2. Array Add	3680	13927	15396
3. Direct Multiply	4681	14015	14734
4. Array Multiply	5791	14947	16385
5. Direct Sine	-	-	48996
6. Array Sine	-	-	50057
All times are in usec for 1000 data operations			

Table 2 - Integer and Floating Point Operation Timing

namic modeling [3]. Consider the same PID servo algorithm in Cartesian coordinates

$$J^t(\theta)[\, K_p(x_d-x) + K_v(\dot{x}_d-\dot{x}) + K_i \int(x_d-x) + \ddot{x}_d \,] = \tau_{act.}$$

The x vectors correspond to Cartesian space [2], and are available from the PRIM module. The transpose of the Jacobian is supplied by the WM module. Following the methods described above, we see that the computations of the EX module increase by 36 multiplies and 30 adds in determining the product of the Jacobian and the inner term. This increases the total time spent in the SP and EX modules to 2.14 msec. However, the WM module must calculate the transpose of the Jacobian. In [10], Orin and Schrader use the rough estimate of 100-150 multiplications, 60-80 additions, and 10-12 sine/cosine operations for a 6 DOF arm. Assuming the upper limit on each of these and adding program overhead, the 20 MHz 68020/68881 system would take ~ 4.1 msec to calculate the Jacobian.

This execution time is "spread out" over five servo updates for Configuration 1, along with the time for the JA and PL modules. The SP and EX modules require 2.14 of the allotted 4.5 msec execution time, leaving 2.36 msec in each cycle, or 11.8 msec in total, for the background tasks of WM, JA, and PL. Therefore, this algorithm could also be realized on either of the two configurations.

Configuration 1 is self-limiting in that as the execution time of the servo rate task increases, the remaining time for the non-servo rate task decreases. This is best shown by considering an example using the computed torque control algorithm in Cartesian space [2,8]

$$J^t(\theta)M_x(\theta)[\, K_p(x_d-x) + K_v(\dot{x}_d-\dot{x}) + \ddot{x}_d \,]$$
$$+ \tau_{cent.x}(\dot{\theta},\theta) + \tau_{gravity}(\theta) = \tau_{act}.$$

The relationship of the terms $M_x(\theta)$ and $\tau_{cent.x}(\dot{\theta},\theta)$ to the joint space terms used above is [8]

$$M_x(\theta) = J^{-t}(\theta)M(\theta)\, J^{-1}(\theta)$$

$$\tau_{cent.x}(\dot{\theta},\theta) = \tau_{cent.}(\dot{\theta},\theta) - M(\theta)\, J^{-1}(\theta)\;\; \dot{J}(\theta)\, \dot{\theta}\, .$$

Here, $J^{-t}(\theta)$ is the transpose of the inverse Jacobian, $M(\theta)$ is the matrix of inertia coefficients, $\tau_{gravity}(\theta)$ is the vector of gravity loading on each joint, and $\tau_{cent.}(\dot{\theta},\theta)$ is the vector of centrifrugal and Coriolis effects. The servo rate task execution time increases to 3.31 msec for the additional matrix multplies and adds leaving a total of only 5.95 msec over the five cycles to invert the Jacobian and calculate all the above dynamic terms for Configuration 1. Configuration 2, with possibly additional processors for the WM, would be required to run this control algorithm.

CONCLUSION

A stepwise process has been shown that maps a logical description of a system architecture onto a functional computer architecture based on a description of the application at hand and the response times required for that application. The response times are broken into execution and communication times to partially separate the application dependent issues from the technology dependent issues. Coupled with knowledge of the interfaces required and times of key operations taken from actual code on the target hardware, the system architect is provided with a powerful set of tools in the iterative procedure of assigning processes to processors. The use of new algorithms or the addition of redundancy to the system can be examined before any of the system is actually built. This technique can be expanded to characterize an entire system for examining such issues as processor utilization, bus bandwidth, and dynamic process allocation. In this fashion, one can determine the maximum and minimum number of processors required for the range of tasks. This allows processors to be turned on and off to conserve power depending on the complexity of the task at the given moment.

REFERENCES

[1] Albus, J. S., McCain, H. G., Lumia, R., "NASA/NBS Standard Reference Model for Teleboot Control System Architecture (NASREM)," NBS Technical Note 1235, July 1987.

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

[3] Kelmar, L., "Manipulator Servo Level World Modeling," NIST Technical Note 1258, February 1989.

[4] Nashman, M., Chaconas, K., "Visual Perception Processing in a Hierarchial Control System - Level 1", NIST Technical Note 1260, March 1989.

[5] Wheatley, T. E., Michaloski, J. L., Lumia, R., "Requirements for Implementing Real-time Functional Control Modules on a Hierarchial Parallel Pipelined System", Proceedings of the NASA Conference on Space Telerobotics, TBP, January 1989.

[6] Michaloski, J. L., Wheatley, T. E., Lumia, R., "Exploiting Computational Parallelism with a Hierarchial Control System", TBP.

[7] Zhang, Y., Paul, R., P., "Robot Manipulator Control and Computational Cost", Report to NIST unser Dept. of Commerce Grant No. 60NANB7D0749, 1988.

[8] Khatib, O., "A Unified Approach for Motion and Force Control of Robot Manipulators: The Operational Space Formulation," IEEE Journal of Robotics and Automation, Vol. RA-3, No. 1, February 1987.

[9] Software Components Group, "pSOS+/68K User's Manual", Version 1.0, September 1988.

[10] Orin, D. E., Schrader, W.W., "Efficient Jacobian Determination for Robot Manipulators", Robotics Research: The First International Symposium, MIT Press, 1984.