IEEE International Conference on Systems Engineering,
Pittsburgh, PA., Aug. 1990.

Configuration and Performance Evaluation of a Real-Time Robot Control System: The Skeleton Approach

Thomas Wheatley and John Michaloski

National Institute of Standards and Technology
Gaithersburg, MD

ABSTRACT

The use of a skeleton system to model a multi-processor robot control architecture offers the system designer a powerful tool to configure and evaluate system parameters such as process allocation, process cycle times, and communication links. The interactions between the system hardware, operating system, and compiler can be tested independently of the application code. This paper describes the skeleton approach as applied to the NASREM robot control architecture. The skeleton approach creates the shell of a functioning real-time control system utilizing the actual hardware and operating system code without using actual application code. This is done by replacing the processing part of the application code with time delays. Parameterization of time delays, communication paths, message buffer lengths, and process allocation provides for rapid prototyping of alternative system architectures. Actual system performance is measured to provide realistic data on computation and communication loads. The skeleton reporting facility provides quantitative assessments of system activity. To illustrate the use of this technique, the servo level of the NASREM hierarchy will be modeled using a 5.0 msec cycle time on a multiprocessor system, and compared with the actual system.

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. In [2], Michaloski et al. discuss some fundamental software design principles applicable to NASREM and other real-time control systems, and introduce the concept of a generic communication and control process (GCCP) as the basic software building block within the system. This paper introduces the concept of a distributed processing unit (DPU) as the basic hardware building block, and describes how a system architecture can be evaluated using these two building blocks in what is deemed the skeleton approach.

The purpose of the skeleton approach is to provide a quick and accurate appraisal of a system architecture. The skeleton reporting facility provides quantitative answers to system configuration functionality. Performance measures returned by the skeleton reporting facility currently include the computation and communication loads, the excess capacity measured as idle time, and GCCP input buffer counts. Fatal flaws like process lockout are monitored and reported to the operator.

Within the skeleton system, all major system architecture descriptors are interactively available for operator modification. Automatic regeneration of the skeleton architecture is produced by interactive changes to the descriptors that define the system architecture. Given the quantitative specifics and interactive nature of the skeleton system, a system designer can rapidly prototype and evaluate different architectural configurations.The system designer can reallocate processes, change execution times, relocate data buffers or alter communication links, all by changing a few descriptors.

Normally, small architectural changes can undermine functionality but the skeleton approach removes the mystery of such modifications. Testing these changes on a large system is difficult due to the confounding presence of application code.The skeleton system allows the system designer to focus on the system and not the effects of the application code on the system. Latent system problems such as processor lockout or OS protocol flaws can be tested and corrected. A high degree of confidence in the basic system architecture and real-time hardware and operating system environment can then be realized before the complexity of the application is added.

BACKGROUND

The complicated nature of robot control suggests a disciplined design architecture to manage the complexity. NASREM is a robot control architecture emphasizing the use of hierarchical decomposition into generic control levels. NASREM offers a consistent control methodology that lowers information density through the use of generic components and repeated superstructures. Each level uses a fundamental communication and control paradigm that features cyclic execution bounded by a response time [2, 3]. Figure 1 illustrates the time-bounded, input, compute, output cyclic process of the GCCP. The importance of the GCCP paradigm is that by strictly adhering to a few design principles, it provides for guaranteed metrics of performance [2].

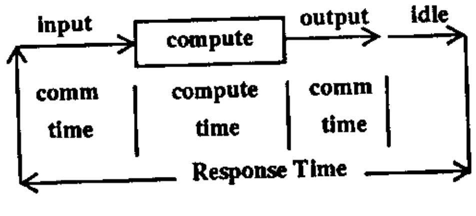

Figure 1. GCCP concept of a cyclic process

The skeleton approach produces a realistic analysis of a system architecture by using the actual hardware and operating system environment. The skeleton approach is concerned with analyzing the timing, utilization and interaction of processes in a hardware environment as opposed to functional correctness of any one process. Performance measures are then produced by

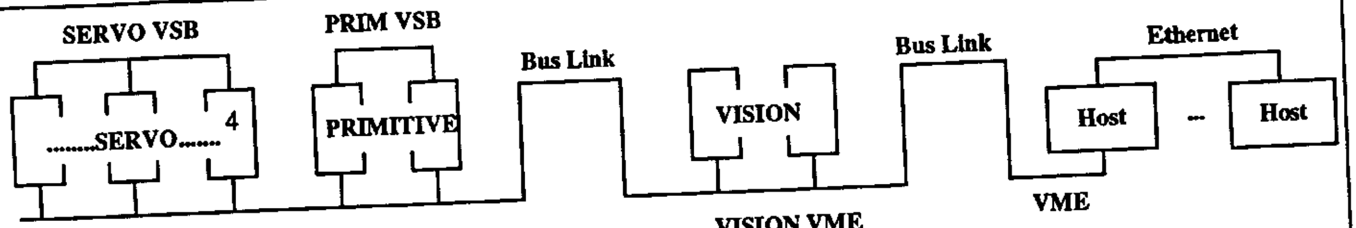

Figure 3 Laboratory Multi-Processor Architecture

employing a skeleton (or basic elements) emulation of the compute and communication GCCP steps. For processing, the skeleton emulation is concerned with "how long" computation takes independent of processor speed. For communication, the skeleton emulation is concerned with "how much" and "where to" data transmission, rather than the actual data. The frequency of communication is mandated to occur every cycle by the GCCP paradigm. The length and final destination of transmission defines the skeleton communication activity.

The skeleton approach emulates both GCCP computational processing and GCCP communication with straightforward substitution concepts. The computational emulation is achieved by replacing the actual code of the GCCP compute step with time delays. The communication emulation is achieved by moving dummy buffers of specified length across designated transport links every cycle. Given this generic emulation model of a GCCP, it is repeated throughout the system and will be termed a Skeleton Communication and Control Process (SCCP). To build an actual skeleton system, it is the responsibility of the system designer to distribute the SCCPs across the hardware architecture and supply such SCCP parameters as cycle timing values, buffer lengths, and destinations.

The skeleton approach uses the generic SCCP as the fundamental software building block to describe the skeleton software environment. In order to model the accompanying skeleton hardware environment, the concept of a distributed processing unit (DPU) is introduced. Figure 2 describes a DPU as the union of a CPU and memory, connected by one to n communication links to other DPUs.

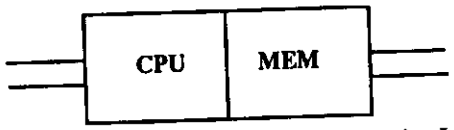

Figure 2. DPU with four Communication Links

A DPU without any CPU is simply a common memory board. Figure 3 illustrates the DPU concept as applied to our laboratory system. The hardware environment consists of CPUs, common memory boards, VME major bus system, VSB sub-bus system, multiple backplanes, and Ethernet connections to host computers.

A skeleton architecture must satisfy three configuration definitions: a software SCCP configuration, a hardware DPU configuration, and a SCCP-onto-DPU configuration mapping.

The SCCP software parameters are specified by supplying a process cycle time, buffer lengths, and linkages. A typical SCCP buffer mapping would contain command, status, and parameter buffers. Additionally, several SCCP synchronization primitives are available. These include basic time cycle synchronization, blocked execution awaiting external event, and a awaiting new data synchronization.

The hardware configuration must define the backplanes, CPU's, memory boards, bus connections and physical locations of all hardware. The CPU description includes each CPU by ID, physical system location within the several possible backplanes, mailbox address, and memory address within the system. Memory boards are described by ID, size, and address with respect to the different communication links. Each backplane has a table describing installed CPUs and common memory boards. Communication paths are specified as either main or sub-bus linkages.

The final SCCP-onto-DPU configuration specifies the partitioning of processes across the multiple processors in the system. Each DPU is self-configuring given this global definition of the system architecture.

USER SESSION

A typical skeleton system session consists of several phases: configuration definition and recompilation, downloading, execution, evaluation, and reconfiguration. Once the initial configuration has been defined and the skeleton code downloaded, iterative reconfiguration and subsequent evaluation can be achieved within the run-time environment.

The first phase includes analysis of possible configurations and definition of an initial skeleton architecture. Typically, most sessions will involve only the software parameters to define new GCCPs and their allocation. The skeleton code is then downloaded to the target CPUs. Assuming each CPU can determine its ID from dipswitches or geographical addressing, the code is identical for all boards. One of the target boards or the host computer serves as the human interface, where the skeleton system variables can be modified.

The iterative execution, evaluation, and reconfiguration sequence begins with each CPU determining its system and GCCP parameters, initializing buffers, and waiting for a start signal from the operator. Any additional monitoring equipment is also enabled at this time.

Upon receipt of the start signal, each CPU executes the GCCPs in the proper sequence for a pre-determined number of times. During each GCCP execution cycle, a running sum is kept on the elapsed communication, computation and idle times, and the number of times any new data has arrived for each input buffer of the GCCP. Data collection time was kept to a minimum to avoid compromising the overall timing results.

At the end of the run, the total percentage of communication and computation times are calculated for each GCCP and DPU,

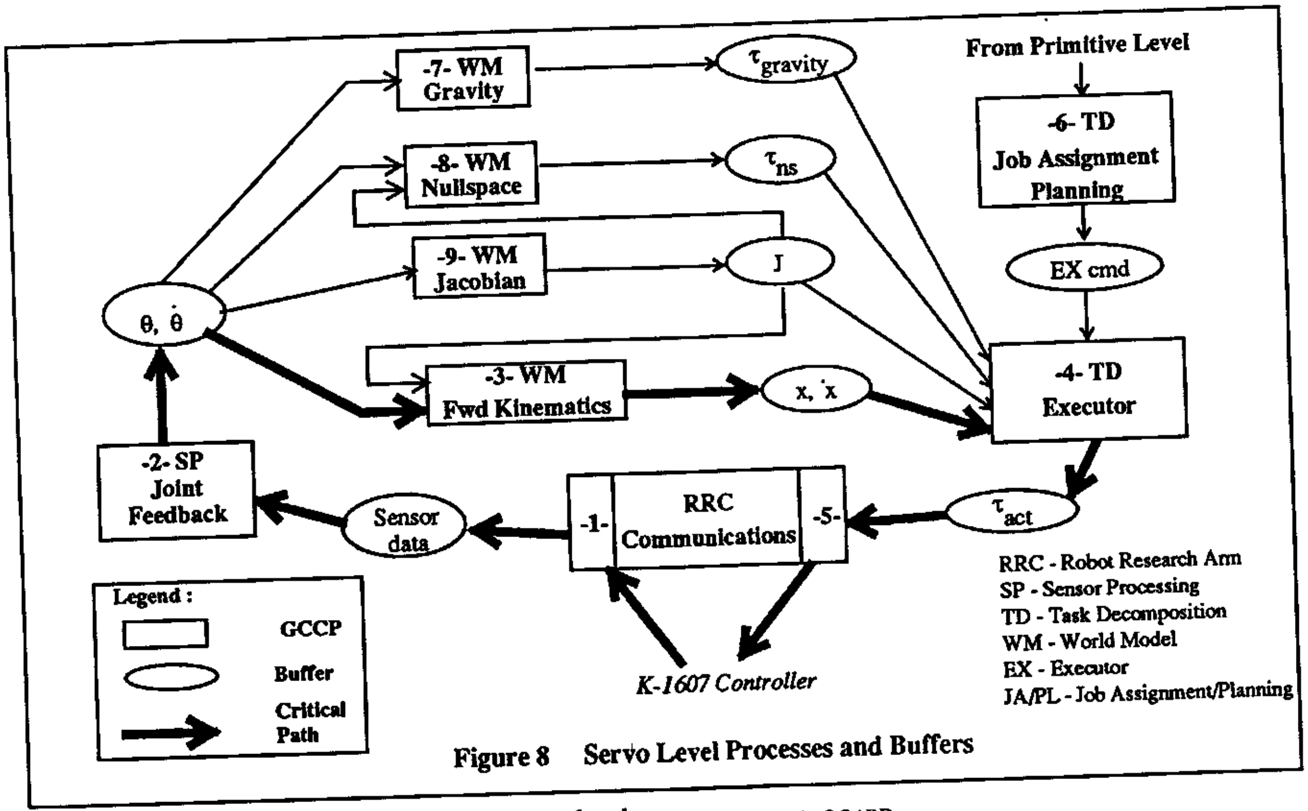

Figure 8 Servo Level Processes and Buffers

and presented back to the operator. Based upon this results, the system designer then reallocates processes, changes time delays, buffer sizes or locations, or whatever is deemed necessary for the task at hand. The modified skeleton system variables are broadcast to the target CPUs, and the next execution cycle started. No recompilation, linking, or redownloading of programs is necessary due to the parameterization of the system variables, resulting in rapid turn-around time for testing new system configurations or application code.

EXAMPLE

To verify the use of this technique to evaluate system configurations, it must be shown that it can correctly model an existing system. The SERVO level of our system was chosen as an initial test case, since it is both well documented and understood, and sufficiently complex to exercise most of the features built into the skeleton system. First, the hardware DPU configuration for the overall skeleton architecture must be defined.

The hardware DPU configuration is described in a straightforward ascending manner. The physical parameters and location of individual DPUs are described first, followed by parameters for each individual backplane, and finally parameters to describe collections of backplanes for the overall system.

The SERVO level is currently implemented on four DPUs residing on a common VSB backplane. Figure 3 depicts our current laboratory system spread across two VME backplanes connected by bus links, a host computer connected by bus link, and additional hosts connected by Ethernet. The hardware description for DPU 4 is in Figure 4, its VSB backplane (SERVO) described in Figure 5, and its VME backplane (ROBOT) in Figure 6. The common memory board for the SERVO level is described in Figure 7. Each description contains cross-references to one another to allow other DPUs access to information about itself and its neighbors. Similar descriptions are required for the remaining components of the design.

```
BOARD_4 : BOARD :=
 (cpu_id             => 4,          -- 1 Mbyte, 20 Mhz CPU
  loc_vsb            => 1,          -- SERVO VSB backplane
  loc_vme            => 1,          -- ROBOT VME backplane
  node_id            => 0,          -- Not used
  dip_switches       => 16#FF90#,  -- used to determine CPU ID
  ext_vme_addr       => address'ref (16#0120_0000#),
  ext_vsb_addr       => address'ref (16#0000_0000#),
  rem_mail_box       => address'ref (16#0000_8240#)) ;
```

Figure 4 DPU 4 Physical Description

```
VSB_1 : VSB_BACK_PLANE :=
 (back_plane_id      => 1,          -- SERVO VSB backplane
  cpus_on_it         => (1,2,3,4,0,0),   -- DPUs by ID on it
  mems_on_it         => (1,0,0,0,0,0)) ;  -- Memory on it by ID
```

Figure 5 SERVO VSB Backplane Description

```
VME_1 : VME_BACK_PLANE :=
 (back_plane_id      => 1,          -- ROBOT VME backplane
  cpus_on_it         => (1,2,3,4,5,6,0,0,0,0),  -- DPUs by ID on it
  mems_on_it         => (1,2,0,0,0,0)) ;      -- Memory on it by ID
```

Figure 6 ROBOT VME Backplane Description

```
MEM_1 : COM_MEMORY :=
 (mem_id             => 1,          -- SERVO common memory
  vme_addr           => address'ref (16#0820_0000#),
  vsb_addr           => address'ref (16#1000_0000#),
  mem_size           => 16#0010_0000#) );
```

Figure 7 SERVO Common Memory Board Description

```
Executor :  GENERIC_PROCESS  :=
  (time_delay        => 3500,     -- delay time in usec
   buffer_wait       => 0,        --  >= 1  = wait
   ext_event_sync    => 0,        --  = 1, wait
   num_buf_in        => 5,        -- loop index for buf_in
   buf_in            => (3,5,6,7,8,0),  -- array of input buffers
   num_buf_out       => 1,          -- loop index for buf_out
   buf_out           => (4,0,0,0,0,0)); -- array of output buffers
```

Figure 9 Executor GCCP Parameters

DPU ID	Time in msec	Modeled Time	Processes	% compute	% communicate	% wait
1	5.0	5.0	1,5	90.3	9.0	0.0
2	4.0	5.0	2,4	73.4	21.3	4.9
3	4.6	5.0	3,6	86.0	9.0	5.0
4	24.4	23.8	7,8,9	96.5	3.4	0.0

Table 1 - Processor allocation and SERVO evaluation results

After the hardware DPU configuration is defined, a software GCCP configuration for the application is built. Fiala et. al. [4] describes the implementation of a Jacobian-Tranpose algorithm in detail for the SERVO level. Figure 8 illustrates the connectivity of the nine GCCPs used in constructing the skeleton system for modeling this algorithm at the SERVO level. Figure 9 describes the GCCP parameters for the Executor process.

The final GCCP-onto-DPU configuration specifies the partitioning of processes across the multiple processors in the system. The nine GCCPs of the SERVO level are spread across four DPUs in the actual system to achieve a 5 msec update rate to the robot controller.

The SCCP processor allocation and the resulting percentages of the DPU utilization are given in Table 1. The percentages show a well-tuned SERVO architecture, with the percentage wait result reflecting the fact that the system was running at approximately maximum capacity. Any increase in algorithm complexity or robot update rate may require additional or faster DPU's.

We experimented with the skeleton system capability to detect design flaws. The critical path cycle time of SERVO was reduced to 2.5 msec, while maintaining the same hardware architecture and process allocation. As expected, the skeleton system detected a pipeline jam resulting in a process lockout.

DISCUSSION

The use of a skeleton system to create the shell of a functioning control system has been demonstrated. A multiprocessor application was successfully modeled and results were verified with those of the actual running system. The parameterization of time delays, communication paths, message buffer lengths, and process allocation provided for rapid turn-around time for modelling new applications or system architectures. Given this iterative approach to building a system, one can readily assess various system configurations. Experience with the skeleton system has led to some general observations.

Data collection can be memory intensive and the data collection times may affect performance measures. For most processes, the data collection time had no effect on performance. However, a minor effect on the results was observed when modeling GCCPs of short duration. An external data collection system similar to LTRAMS [5] may be required to collect large amounts of data or reduce the collection time.

Consistency of timers across processor is troublesome. A 32-bit,1 μsec resolution timer was built to measure the individual GCCP computation, communication, and wait times. The timer was available to all DPUs to guarantee consistency across various DPUs.

Mismatched execution frequencies in pipelined processes can cause data dropout or cycle skipping. This was evident in the first use of the skeleton system since it expected a fixed number of data buffers to be available before proceeding to the data collection phase. A termination condition parameter was added to prematurely stop the process and allow the operator to evaluate the results.

Current enhancements underway include the use of a tasking model to allow single-processor concurrent modeling as opposed to the strictly procedural method currently used, a calibration technique for CPUs with different clock speeds, an improved method of data collection, and an operator interface for graphic visualization of the performance measures and improved user reconfiguration control. Future work will include the development of an automated process allocation and scheduler to aid in the initial selection of the GCCP to DPU allocation. Given a system definition comprising of a set of GCCPs, times, and data interfaces, a System Architecture Management System (SAMS) could be built from the combination of the skeleton system and the automated process scheduler. A function of SAMS would then be the automated program generation of the system using the actual application code.

REFERENCES

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

[2] Michaloski, J.L., Wheatley, T.E., "Design Principles for a Real-Time Robot Control System," to be presented at the IEEE International Conference on Systems Engineering, Pittsburgh, PA., Aug. 1990."

[3] Fiala, J.C. "Note on NASREM Implementation" NISTIR 89-4215, National Institute of Standards and Technology, Gaithersburg, Md., December 1989.

[4] Fiala, J., Wavering, A., "Implementation of a Jacobian-Transpose Algorithm", NIST IR 90-4286, National Institute of Standards and Technology, Gaithersburg, Md., January 1990.

[5] Mink, A., Draper, J., Roberts, J., Carpenter, R., "Hardware Assisted Multiprocessor Performance Measurements", Proc. of the 12th IFIP WG 7.3 International Symposium on Computer Performance: Performance 87, Brussels, Belgium, Dec. 1987, pp. 151-168.