Proc. of IEEE International Conference on Systems Engineering; August 1990; Pittsburgh, PA

Design Principles for a Real-Time Robot Control System

John Michaloski and Thomas Wheatley

National Institute of Standards and Technology
Gaithersburg, MD

ABSTRACT

A reliable robot control system must exhibit deterministic performance under all conditions. To achieve determinism, the architecture must account for both functional and timing requirements. The job of the design process is to formulate a solution to fulfill such a set of functional and timing requirements. Ideally, a more formalized design methodology and better design tools will result in a better design. Yet, even with good design methodologies and design tools, flaws often occur due to the sheer complexity of requirements, mitigated by the need for a multiprocessor architecture. The goal of this paper will be to examine the finer subtleties in the design of multi-processor hierarchical robot control systems from the application viewpoint. A concurrent communication and control algorithm will be presented as the generic software component of the control system in an attempt to reduce the complexity and increase the reliability of the overall design. We will propose some design principles that informally guarantee the correctness of the algorithm. These principles help at a lower level of design abstraction where the actual components that comprise the computer system architecture need to avoid functional malfeasance such as deadlock, starved processes, poor performance, and other design shortcomings.

1. INTRODUCTION

Real-time systems satisfy real-world timing obligations with performance characterized by high data communication bandwidth, maximized throughput, and fast execution. To realize these requirements, real-time target systems use: 1) fast context switching, 2) small interrupt latency, 3) priority scheduling, 4) high-priority tasking model, 5) grant low-priority tasks non-preemptable status, and 6) synchronization of tasks. Real-time response alone does not define a functionally correct system and leads to the prima facie real-time design principle: *time-bounded reliable and deterministic performance is a necessary requirement to define a real-time control system.*

The primary robot control requirement is deterministic real-time response. Smooth motion imposes an intrinsic real-time response on robot control. Real-time response leads to the subsequent commitment (which can be viewed as a requirement) to multicomputer architectures to satisfy timing constraints. Distribution of the multiprocessor control system architecture may be loosely or tightly-coupled across a network or shared backplane but must satisfy the deterministic real-time response requirement.

Given these basic system requirements, this paper presents the NIST[1] NASREM architecture as a view of the control system from "programming in the large," and reviews the impact of timing and multicomputer requirements of "programming in the small" of a robot control system. The "programming in the small" software component discussed in this paper is the *generic communication and control process* (GCCP). Developmental experience with GCCP has led to principles for improved design and implementation. This paper will use these principles to prove GCCP correctness. The paper is organized as follows: 1) a background description of the NASA/NBS Standard Reference Model for Telerobot Control System Architecture (NASREM) architecture, 2) a presentation of the basic control system software component called GCCP, 3) a discussion of the software design principles used to informally prove the correctness of the GCCP algorithm, and 4) a review of algorithm performance.

1. National Institute of Standards and Technology, formerly the National Bureau of Standards (NBS).

2. NASREM DESIGN ARCHITECTURE

The integration of manipulator, control and sensor system, and user-interface forms the design of a robot control system. This section presents a software control methodology aimed at reducing the impact of timing and software complexity on the design. To achieve a robust design, the requirements of real-time environments as well as functional correctness must be met and derived from appropriate design principles. Sequential design methodologies propose small module size, modular independence, "black box" implementation hidden from specification, and levels of logical abstraction. Real-time control software is well-served by these design goals, but timing constraints impose more rigorous accountability. Real-time reliability demands a stronger threshold of assurance than just logical program correctness. Much as structured programming improves the quality of sequential software, the GCCP algorithm to be presented, plus the accompanying design principles, improves the quality and reliability of real-time control.

NASREM employs a real-time control design methodology that is well-structured to simplify robot control system design. NASA has adopted the NASREM architecture for use in the Flight Telerobotic Servicer (FTS), a two armed telerobotic manipulator which will build and maintain the Space Station. The NASREM architecture is based on real-time hierarchical control derived from basic task decomposition. This methodology has led to a standard set of control levels with well-defined interfaces [7]. The overall NASREM design architecture is more fully explained in Albus et al [1].

A control system organized into the well-defined hierarchical structure results in a more comprehensible system. The hierarchical structure imparts a systematic approach much like a road map or file cabinet; it simplifies finding a certain process in order to make additions, modifications or diagnostic analysis to that process [6]. The "programming in the small" model repeated throughout the control system is the virtual control process [9]. Virtual control operates cyclically, sampling command and status much like a feedback control loop. The GCCP algorithm implements virtual control.

Concurrency is the main benefit of virtual control. NASREM partitions the control hierarchy into concurrent GCCPs to improve performance. GCCP supports a variety of concurrent uniprocessor or parallel multiprocessor architectures. Concurrent NASREM levels communicate with adjoining levels through predefined command and status buffers. In this context, each level within the hierarchy samples as inputs the command from the adjoining upper level and status from the adjoining lower level, comparing the command input to the sensory sampled status environment, computing a goal-directed action, and outputting this action as a command to the adjoining lower level and a status to the adjoining higher level within a given response time. Figure 1 illustrates the input, process, output sequence over time for a level n.

We will present the virtual control GCCP algorithm in progression

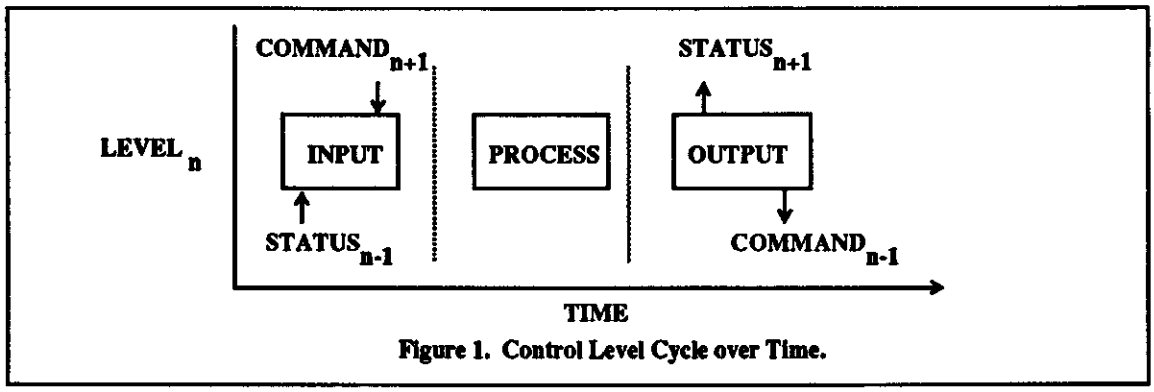

Figure 1. Control Level Cycle over Time.

from the simplest input-process-output sequence to a more sophisticated model in order to explore various design issues. Rather than a completely general approach, the algorithm presented in figure 2 shows communication of commands and status for levels in the hierarchy. Critical sections and mutual exclusion are modeled in the algorithms with the semaphoring operations lock, and unlock. The variable N of type level_number refers to the level number within the hierarchy; with a smaller N referring to a level lower in the hierarchy, hence closer to the robot. Comments are delimited by double quotes.

The GCCP algorithm comprises a basic structuring mechanism of NASREM. Task decomposition to level structuring is used before any further refinement is undertaken. Defining the data interfaces between levels completes the basic outline of the control system. Depending on timing constraints, this basic set of communicating processes can be time-shared tasks on one processor or partitioned across processors. (Fiala more fully explains an implementation in ADA using this software structuring methodology [5].) For NASREM, the level process step is further refined into additional GCCP that communicate through well-established interfaces.

3. ALGORITHM TIME- CRITICAL CORRECTNESS

The initial GCCP algorithm demonstrates communication between concurrent processes. Proving the correctness of this algorithm depends on showing that the functional and time-dependent features are correct. We will use an informal, intuitive approach to show the correctness of the GCCP algorithm. (References [4, 11] present a more formal approach to general task scheduling and communication.) Functional correctness will be assumed so that time-dependent aspects of the algorithm can be evaluated. The algorithm will be considered *time-critical correct* if every real-time process executes deterministically bounded by fixed response time and no processes deadlock.

In order to guarantee execution, we must establish that a process executes within a predetermined response time. This requires strict accountability of timing side-effects. Some side-effects are secondary,

```
procedure level(N)
 N: level_number;
begin
    initialization_procedure;
    repeat
        lock, read_command(N+1), unlock;      " read command "
        lock; read_status(N-1) unlock ;       "read status "
        process;                              "execute level "
        lock, write_command(N-1), unlock;     "write command "
        lock, write_status(N+1), unlock;      "write status "
    until termination_condition;
    termination_procedure;
end level;
```

Figure 2. Generic Communication and Control Processes (GCCP) Algorithm

but have an important impact on guaranteed performance in a real-time environment. For example, an operating system supporting virtual-memory management is not required since disk access is relatively slow. More important for guaranteed performance is operating system scheduling. To prove correctness, we must verify that process existence and scheduling is deterministic guaranteeing predetermined processor access.

To establish process existence, a simplifying design principle is that *every process is statically allocated to memory*. The dynamic creation of robotic processes is of limited usefulness because the large overhead required to replace unreliable or crashed processes by new processes [10]. The use of statically defined processes insures process existence resulting in improved performance and simplified design. For single CPU dedicated processes, the existence is understood. For concurrent processes sharing a CPU, the operating system plays a major role and one must establish deterministic scheduling.

To guarantee processor access, secondary scheduling must not interfere with predetermined processor allocation. This scheduling edict leads to the principle that *fairness is not an issue in real-time control*. In a general purpose operating system, every process is given an equal opportunity at the CPU. Real-time control needs priority based scheduling where each process has a basic execution time and a period performance deadline. All real-time processes must meet periodic deadlines while sharing the processor. Scheduling must be known for guaranteed execution.

The second scheduling principle requires *tasking to be deterministic and user-controllable, implying that time slicing is of limited importance*. In general, time-sliced round-robin scheduling is difficult to characterize and verify. Time quantum round-robin scheduling can overcome non-determinism but spurious context switching relegates round-robin to low speed applications where response time is not critical. Instead, a priority based system is preferred where processes execute in known sequence.

Having described the correct computing environment with no adverse side-effects, we must establish the determinism of the algorithm. A GCCP will be defined as *time-critical correct* if the following requirements are maintained:

- process step executes deterministically with bounded response time,
- process step avoid deadlocks,
- communication avoids deadlocks,
- interprocess-communication obeys timing bounds.

The following sections will present principles to achieve time-critical correctness.

3.1 Processing Time-Critical Correctness and Deadlock Avoidance.

The "information hiding" structured design principle is a widely accepted practice. In a sequential environment, information hiding is commonly done with the "black box" metaphor. Unfortunately, a real-time environment imposes additional constraints on "black box" defi-

nition. To establish that the processing step of the GCCP algorithm does not deadlock, it must be shown that the process "black box" always completes one cycle by the bounded response time. The following "black box" correctness principles are founded on the notion that the functional specification must acknowledge timing constraints and guaranteed performance.

Deterministic behavior requires the design principle for a *one-pass, state-sampling process step*. Waiting in a tight loop reading a failed sensor is an illustration of a module that ultimately deadlocks the system. A better method of event handling is to sample the environment during each GCCP control cycle. This predictive control methodology can be generalized for all external event handling as well. Event handling can be classified as either demand or periodic [3]. A demand function is triggered by the occurrence of an event every time it is performed. For example, a trip switch may only be activated intermittently, but could be triggered successively at indeterminate intervals. A periodic function is performed repeatedly without being requested each time. For example, a vision system may supply a new image at fixed intervals, regardless of whether the system requested the new image.

One design principle that provides for a more rigorous specification of a control system is to *sample all sensor or real-world inputs at a periodic rate, and remove unpredictable demand functions*. Restructuring demand functions into periodic functions can be done with the use of latches and polling, and repeated sampling every control cycle. Now, sensors are bounded by a fixed time period. Sampling sensors anticipates peak loads and eliminates racing conditions.

The GCCP process step will be considered *time-critical correct* when it displays deterministic execution bounded by a response time. Time critical correctness results from strictly enforcing the event-handling principles: 1) one-pass state evaluation as opposed to conditional-termination loops and 2) periodic sampling of sensors as opposed to intermittent demand handling. A process step that is determinably time-critical correct can then be assured to avoid deadlock.

3.2 Communication Deadlock Avoidance.

Assuming deterministic processing, the communication part of the algorithm must be proved deadlock free. Communication in the algorithm consists of the lock, read/write, unlock operations. The synchronization primitives lock and unlock are assumed correct. Synchronous communication without time-out could cause deadlock in the following manner. A set of communicating processes can become communication-deadlocked if each process in the set is waiting for a message from another process in the set and no process in the set ever sends a message. Constructs such as the rendezvous of ADA, or Hoare's CSP [8] allow only a high-level of analysis of communication and leave open the problem of low-level implementation analysis [2].

As a result, high-level, purely-synchronous communication lacks the level of detail necessary to ascertain the determinism of the control system. Guaranteeing deterministic behavior and avoiding communication deadlock necessitates the principle that the *communication must be non-blocking*. Communication with time-out is one simple method to guarantee cyclic behavior and prove communication deadlock free. The section on algorithm performance discusses other non-blocking communication strategies for better performance based on cyclic frequency of response.

The necessity for asynchronous communication leads to some additional algorithm constraints tied to synchronizing the message exchange. Each process would then be required to either receive a new message arrive each cycle, or have the capability to check for a new message, and if none, use the old message. Because cycle rates among processes may vary, some processes may be running faster than others. A time stamp can correlate the message passing among varying cycle rate processes. The strategy NIST has adopted whenever possible is to use 1 writer to 1 reader message exchange where only the latest message is timely. This implies a one-deep message queue. Messages include a time stamp and are sent every cycle, overwriting the last message. Command and status buffers are paired to handle handshaking acknowledgments. The cyclic, time-stamped protocol simplifies real-time communication.

3.3 Communication Time-Critical Correctness

Communication between processes is required to exchange information among processes. We have presented a scenario where communication will not deadlock. Now, we must study the effect of communication on functional correctness, especially processes exchanging time-sensitive information. The basic interprocessor communication design principle is *efficiency must justify the additional overhead of communication.*

Communication is evaluated using the performance measures of latency and throughput. To characterize a robot control system as real-time implies a guaranteed maximum latency for interprocessor communication. *Latency* is defined as the elapsed time before a message is acknowledged. T*hroughput* is defined as the rate one process can send information to another process, especially important for systems that share large amounts of data. When designing, the distinction between guaranteeing arrival of small amounts of data every 20 milliseconds must be contrasted to transferring large amounts of data efficiently across the transport link. Thus, 20 bytes of information that has to be shipped every 20 milliseconds cannot be handled with a communication protocol that is efficient for moving large amounts of data, but may require 100 milliseconds setting up the physical link handshakes and transport software headers.

This leads to the design principle that *latency or guaranteed performance, not efficiency or throughput, is necessary in a time-critical correct environment.* To prove a generic communicating process is time-critical correct, one must establish a bounds on the communication latency. For many communication schemes, this is not an easy task. Having proved that the internal communication with time-out will not deadlock a process, the adherence to guaranteed latency between communicating processes establishes the correctness of the GCCP communication steps.

4. GCCP ALGORITHM EVALUATION

Although we have proven GCCP as time-critical correct, this correctness does not imply that the algorithm is desirous. The algorithm must be efficient enough to allow even the most time-critical processes to function while flexible enough to allow easy maintenance over the life of the software. This section will study the efficacy of the algorithm. The study of the algorithm will lead to further principles, that although do not effect the correctness, will improve the GCCP algorithm.

4.1 Algorithm Performance.

Estimating GCCP performance is based on establishing an upper bound on cycle time. The performance bound can be derived from summing the worst-case processing step time interval with the time required for communication. Several judicious programming measures can be incorporated into the algorithm to streamline performance. The avoidance of deadlock in communication could be done with a time-out on each communication, but this alternative is less attractive since it requires each communication exchange to estimate its time-out parameter. A more expedient approach to optimizing performance and minimizing complexity with no loss of correctness is to choose a maximum time bound per cycle and use no-wait communication.

The previous algorithm is now extended to reflect this cyclic behavior with the addition of two synchronizing steps embodied in the algorithm by a synchronizing *wait_until* and *update_cycle* construct. This updated algorithm containing synchronization is illustrated in figure 3. The *wait_until* must be of appropriate time length and be an accurate estimate of time, not a general metric as in the delay construct in ADA. An exact interval per cycle yields more efficient performance

```
procedure LEVEL(N, INCREMENT)
N: level_number;
NEXT_CYCLE: time;
INCREMENT: time;                                  "computation period"
begin
  initialization_procedure;
  repeat
    if CUR_TIME > NEXT_CYCLE                      "check for bounded cycle"
      then error_routine;                         "determinism forsaken"
    end if;
    wait_until( NEXT_CYCLE);                      "timing sync primitive"
    update_cycle(NEXT_CYCLE, INCREMENT );         "update next cycle count"

    lock, read_command(N+1), unlock;              "read command"
    lock, read_status(N-1) unlock ;               "read status"
    process;                                      "execute level"
    lock, write_command(N-1), unlock;             "write command"
    lock, write_status(N+1), unlock;              "write status"
  until termination_condition;
  termination_procedure;
end LEVEL;
```

Figure 3. GCCP Algorithm with Cyclic Synchronization

since many processing algorithms can then use time differentiation between cycles in dynamic calculations. Further, a predetermined cycle time can offer an easier estimate of overall system performance that is guaranteed by some upper bound. This leads to the principle *that all communicating concurrent processes must adhere to an à priori cycle period.* As a safety feature, we have included the step to check the current time against the present cycle time and make sure that it has not been exceeded. This reaffirms process obligation to meet deterministic cycle time and preserves system integrity.

4.2 Analysis of Cyclic Timing.

There exits a trade-off between as small as possible cycle time and a larger cycle time. Too large a cycle time implies a loss of timeliness. Too small a cycle time implies high message querying traffic and restrictive processing bounds. Highly variable process duration implies a need to overlap processing from one cycle to the next to optimize performance. This is acceptable if new commands are generated to dependent processes within an appropriate response time, but implies a loss of determinism. Processes ranging outside the à priori cycle time introduces a high degree of system variability. Variable duration implies multi-cycle processes scattered throughout the system which weakens determinism. Especially troubling is that system response time can now not be accurately estimated. This implies the need to minimize variability by adhering to a one cycle algorithm by containing the worst-case duration to one cycle.

The single-cycle timing restriction may be insufficient for many processes. This constraint leads to the motivation to concurrently partition time-critical response from non-time-critical response with separate GCCP. Each GCCP process maintains cyclic behavior. In NASREM, robot control is divided into planning and executing. The *planner* is responsible for generating a plan consisting of a series of actions while an *executor* enables the real-time state transition of plans.

4.3 Algorithm Diagnostics

Without assistance, monitoring concurrent processes is difficult. Synchronized stepping of concurrent processes offers some degree of comprehensibility. Centralized synchronization could step a set of GCCP. Each GCCP process awaits the centrally generated cyclic pulse. Although centralized synchronization does simplify testing and analysis, it may not be required for the final system.

5. DISCUSSION

The impact of real-time and concurrency demands additional design principles for a better designed robot control system. This paper studied the use of a generic computation and communication (GCCP) algorithm as the basic concurrent and real-time software component in a robot control system. Adhering to a set of design principles, the GCCP algorithm was shown to be time-critical correct, that is, provides a system response within a bounded time period and will not deadlock.

First, the most important design principle for real-time control is reliable and deterministic performance. Second, the designer must account for operating-system side-effects. The designer must provide priority-based and deterministic scheduling. Third, the GCCP process step must account for timing constraints, guaranteed performance, and exception handling. The principle of periodic and deterministic event-handling prevents system deadlock. Fourth, the principle to use non-blocking communication with guaranteed latency provides communication that is both time-critical and deadlock-free. Finally, the principle to synchronize and bound process cycle-time was presented as a means to improve the efficacy of the software model.

Experience with GCCP has shown that these software principles do in fact simplify design. The generalized GCCP approach to concurrency allows flexible reconfiguration since GCCP processes have the identical interfaces for either uniprocessor or multiprocessor communication. Wheatley et al [12] describes an automated approach to configuring GCCPs and evaluating the system architecture.

References

1. Albus J,S., McCain H.G., Lumia R. NASA/NBS Standard Reference Model Telerobot Control System Architecture (NASREM), NBS Technical Note 1235, National Bureau of Standards, Gaithersburg, Md., June 1987.
2. Anger, F.D. "On Lamport's Interprocessor Communication Model," ACM Transactions on Programming Languages and Systems, Vol. 11, No. 3, July 1989, pp. 404-417.
3. Britton, K.H. "Specifying Software Requirements for Complex Systems: New Techniques and Their Application," Proceedings of the IEEE Specifications of Reliable Software Conference, 1979.
4. Dertouzos, M.L., Mok, A. "Multiprocessor On-Line Scheduling of Hard-Real-Time Tasks," IEEE Transactions on Software Engineering, Vol 15, No. 12, December 1989, pp. 1497-1506
5. Fiala, J.C. "Note on NASREM Implementation" NIST Internal Report 89-4215, National Institute of Standards and Technology, Gaithersburg, Md., December 1989.
6. Fitzgerald, M.L., Barbera, A.J., Albus, J.S. "Real-Time Control Systems for Robots," SPI National Plastics Exposition Conference, 1985.
7. Fitzgerald, M.L., Barbera, A.J. "A Low-Level Control Interface for Robot Manipulators," NBS-Navy NAV/SIM Workshop of Robots Standards, June 6-7, 1985.
8. Hoare, C.A.R. "Communicating Sequential Processes," ACM Vol. 21, No. 8, August, 1978, pp. 666-677.
9. Michaloski, J.L., Wheatley, T.E., Lumia, R. "Analysis of Computational Parallelism with a Concurrent Hierarchical Robot Control System," NISTIR 90-4251, National Institute of Standards and Technology, Gaithersburg, Md., 1990.
10. Schwan, K., Bihari,T., Weide, B., Taulbee, G. "High-Performance Operating System Primitives for Robotics and Real-Time Control Systems," ACM Transactions on Computer Systems, Vol. 5, No. 3, August 1987, pp. 189-231.
11. Sha, L., Goodenough, J.B. "Real-Time Scheduling Theory and Ada," Computer, April 1990, pp. 53-62.
12. Wheatley, T.E., Michaloski, J.L. "Configuration and Performance Evaluation of a Real-Time Robot Control System: The Skeleton Approach," to be presented at the IEEE International Conference on Systems Engineering, Aug. 1990.