CHIMERA: A Real-time Programming Environment

For Manipulator Control

Donald Schmitz
The Robotics Institute

Pradeep Khosla
Department of Electrical and Computer Engineering and The Robotics Institute

Regis Hoffman
Department of Computer Science

Takeo Kanade
Department of Computer Science and The Robotics Institute

Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213 U.S.A.

Abstract

CHIMERA is a real-time computing environment for the CMU Reconfigurable Modular Manipulator System (RMMS) project. CHIMERA is both a hardware and software environment which allows rapid development and implementation of real-time control programs. It provides a C/UNIX flavored concurrent programming environment for a Motorola 68020 multiprocessor hardware configuration connected to a SUN workstation. CHIMERA has been implemented using commercial hardware in conjunction with a sophisticated, locally developed software package, making for a reliable, reasonably priced, and easily duplicated system. We are currently porting CHIMERA for real-time control of the CMU Direct-Drive Arm II. This paper describes the implementation and capabilities of the CHIMERA environment, and illustrates how these features are used in robot control applications.

1. Introduction

The CMU Reconfigurable Modular Manipulator System (RMMS) project is a research effort aimed at developing a generalized robot manipulator and controller. The goal of the project is to allow a user to rapidly design and assemble a manipulator, and immediately put it into operation. Physically, the RMMS consists of a number of standard actuator (joint) and structural (link) modules. These modules can be assembled via quick connect mechanical and electrical couplings into a manipulator with an arbitrary, user specified configuration and range of capabilities. One component of this research addresses the theoretical issues in mapping the kinematic and dynamic characteristics of a task into a manipulator configuration. Another component is the development and implementation of configuration independent algorithms for manipulator path planning, kinematics, dynamics, and position control.

From the outset, the RMMS project was envisioned as extremely programming intensive. Our prior experience with the CMU Direct Drive Arm II project was that making even a minor change in a control program required a considerable development effort, yielding a precious few minutes of actual experimentation. Within the RMMS project, we intended to implement and exhaustively test numerous control algorithms. To make this programming requirement more manageable, a computing environment was envisioned which would extend the powerful program development features and unified software environment of computer workstations to the real-time computing arena.

The resulting computing environment (christened CHIMERA) consists of one or more Ironics 68020 CPU boards, acting as real-time computing engines, residing on the VME bus of a SUN 3 workstation. The SUN provides a powerful, high-level computing environment (editors, compilers, *etc.*) in which to develop real-time control programs, which are then executed on the Ironics real-time processors. CHIMERA also includes a multi-tasking real-time kernel, an inter-processor communication package, and a software library that emulates common SUN/UNIX utilities on the real-time processors, and provides a high-level interface to the system hardware.

This paper discusses the CHIMERA design, implementation, and capabilities in detail. The paper is organized as follows: In the next section, we survey other efforts in this area and present the CHIMERA design philosophy. In Section 3, we describe the hardware architecture and evaluate its real-time performance. The CHIMERA kernel is described in Section 4, and inter-processor communication and the CHIMERA extended file system in Section 5. Finally, in Section 6 we demonstrate the utility of the system through several examples.

2. CHIMERA Design Influences

Robot computer systems have traditionally used a dual processor family computing architecture: a general purpose, multi-tasking machine supporting a high-level software development environment that is closely coupled to one or more high performance numeric processors with real-time capabilities. This architecture is dictated by the following software requirements of an experimental robot controller:

1. High-level computing, performed on a general purpose processor, for real-time program development and offline storage and analysis of experimental results.
2. High performance numerical computing, with well defined execution timing criteria, executing on dedicated real-time processor(s) to implement the control law, and sample and buffer data for later storage on the general purpose system.

While such an architecture is efficient in terms of hardware utilization, the accompanying software environment is often a clumsy patchwork of special purpose compilers and interface code. Fortunately, advances in micro-processor technology now allow general purpose processors to achieve the computing performance required of robot controllers, while remaining compatible with a large base of existing software. By replacing the special purpose numeric processors in previous architectures with high performance general purpose processors, a unified, high level software environment can be supported for the entire processing system.

2.1. A Case Study of an Early System

An example demonstrating the problems associated with past robot controller designs is the system developed for the CMU Direct Drive Arm II [1, 5]. In this system, a major emphasis was placed on obtaining high floating point computing performance. The system implemented an inverse dynamics based non-linear control scheme for a six axis manipulator, at a control update rate of 500 Hz. This required scalar floating point performance on the order 1.0 MFlop. The CMU Direct Drive Arm II control architecture consisted of several special purpose real-time processors operating in parallel under the supervision of another general purpose processor. A Marinco APB-3204 floating point processor performed most of the floating point computation, while six Texas Instruments TMS32010 DSP optimized processors were dedicated to the low level I/O and sensor signal processing of each manipulator axis. These two sets of processors were coordinated by a Motorola 68010 based, Omnibyte single board CPU, with 3 Mbytes of local memory for data recording. This real-time system was coupled to a general purpose VAX/UNIX based development system via an Ethernet connection, which supported both program downloading and data transmission between the two families of processors.

While the computing performance of the system was more than adequate, the programming environment imposed by this hardware was barely usable. The Marinco FPU and the TMS32010 DSP processors each required program development in their own (somewhat unconventional) assembly languages. The Marinco FPU utilized a different floating point format than the M68010 and VAX, requiring a number of format conversion operations when transferring data between processors. Although a C compiler was available for the M68010, many speed critical sections of code were re-written in assembly language. High Ethernet network traffic often caused severe communication problems between the host and M68010 system, forcing data recording to be offline (the attainable transmission rate to the VAX was much slower than the rate at which data was generated).

An analysis of the system hardware and software revealed several problems:

- The execution speed of the control calculation was critical, and a great deal of effort was spent in optimizing this code. This was made extremely difficult by the need to write entirely in assembly language.
- Surprisingly, an equal amount of effort was required to implement the remainder of the code, comprising the user interface and data recording utilities. Although in general much less demanding in terms of efficiency, this code is much larger than that of the control calculation, and requires extensive error detection and special case handling capabilities.
- The principle source of programming errors involved the interface to hardware devices, such as I/O ports, timers, etc. Documentation beyond the data sheets for the devices was often non-existent - the only way to debug such code was with an oscilloscope and a great deal of patience.

Our experience shows that attempting to develop hardware and software at the same time should be avoided. Every element in such a system is an unknown - it is very difficult to decide if a system failure is a hardware or software failure. Also, building and using special purpose hardware can introduce maintenance nightmares. The larger and more diverse a system becomes, the more likely it is to fail, the more difficult it is to diagnose, and the longer it takes to repair or replace.

2.2. Trends in Controller Architecture

Advances in commercial micro-processor performance have had two important effects on the design of robot control systems. First, they have made the computer workstation a reality, allowing an entire general purpose software development system to be (affordably) dedicated to controlling a manipulator. The high-level and real-time processors can be coupled on the same system bus, allowing for very reliable, high bandwidth communication and coordination between the two. Second, the computing performance now available from a general purpose micro-processor rivals that of special purpose numeric processors available several years ago. For many control applications, this allows a general purpose CPU to replace a number of special purpose processors, reducing both the hardware and software complexity of the total system.

Several examples of such close coupled, workstation/real-time processor systems have been reported in recent literature:

The IBM SPARTA project [6] has built a multiple DSP processor system residing in an IBM PC that also serves as a software development system. Coupled with an extensive software package, PC programs can interact with the DSP processors via shared memory and memory mapped control registers. A high level language is provided to write programs for the DSP processors.

The MIT CONDOR project [4] has developed a multiple Motorola 68020 processor system built around a SUN 3 workstation. The SUN 3 C compiler is used to generate single process thread programs for each of the real-time processors. A runtime library and communication package are also provided to simplify programming the real-time processors and cooperating SUN programs. Although developed independently (and concurrently), CONDOR is very similar to the CHIMERA environment in both concept and implementation - the primary difference between the two are the programming constructs supported by the two packages, reflecting a difference in applications programming philosophy between the two projects.

3. The CHIMERA System Design

Based on our past experience and available hardware, we established a number of requirements for the CHIMERA computing environment:

- The software environment would appear to the user as a real-time extension of a typical UNIX development system:
  - The C programming language would be used for *all* levels of the control program.
  - Real-time programs would be designed as multiple, *concurrent processes*, running under a real-time executive or kernel. This kernel would support access to hardware devices via a library of high level routines, hiding the hardware details from the applications programmer.
  - Standard UNIX utility libraries would be ported or emulated, allowing ready portability of existing UNIX programs.
- Only one family of CPU would be used in the entire system. This would be a well supported, general purpose CPU chosen for overall performance and software portability.
- The system would be expandable through the addition of one or more CPUs operating in parallel to achieve the desired level of computing performance.
- The hardware, aside from application specific I/O devices, would be commercially available items, and be easily integrated into the locally supported SUN workstation environment.

Based on the above requirements, we developed the CHIMERA computing environment. CHIMERA is built around the SUN 3 workstation and Ironics 68020 CPU boards. The SUN's extensive utilities, *e.g.* window managers, editors, and debuggers are used for program development. Because the Ironics and SUN CPUs share the same 68020 based architecture, no additional cross-compilers/linkers are required. CHIMERA code is compiled and linked with the standard SUN C compiler and linker (although a separate run-time library is loaded with CHIMERA code, not the usual SUN library). This ensures that code developed on the SUN for the real-time processors will behave exactly the same when executed on the real-time processor.

The CHIMERA architecture is shown schematically in Figure 3-1. The major component of the CHIMERA hardware is a SUN 3 workstation, based on the VME bus and the Motorola 68020/68881 CPU. The SUN is a VAX class computer, running the UNIX operating system, supporting a window based software development environment. Residing on the SUN VME bus are one or more Ironics IV-320X family single board computers.

As expected, the Ironics 3204 is faster than a SUN 3/160 by roughly the ratio of their clock speeds on the Dhrystone benchmark, which is a measure of data transfer and integer arithmetic performance. The combination of a faster FPU clock speed and the revised M68882 architecture yielded an even larger increase in floating point intensive operations, as indicated by the Whetstone and matrix arithmetic results. Interestingly, although the scalar floating point speed (speed of a single arithmetic operation) of the Ironics (and SUN) is much slower than either the VAX or the Marinco, operations such as matrix multiplication do not reflect this disparity. This

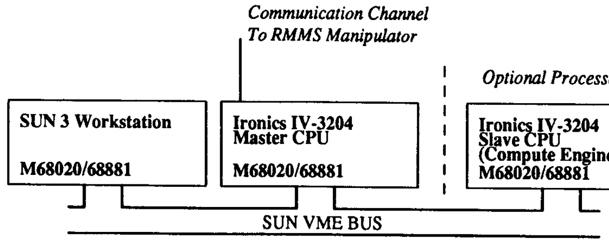

Figure 3-1: Block Diagram of CHIMERA Hardware Architecture

suggests that typical floating point code actually performs a good deal of integer arithmetic to support loop constructs and address generation. These results indicate that a well balanced system such as the M68020/68882, with both good data manipulation, integer and floating point performance can compete with less general, floating point optimized processors in many real applications.

The Ironics CPU is also M68020/68881 based, with up to 4 megabytes of local RAM. The Ironics dual port memory can be accessed by both the Ironics and SUN CPUs. This allows programs to be easily downloaded from the SUN to Ironics. This also allows simple mailbox communication between the two CPUs. The local VMX bus connection of the Ironics provides the interface to the manipulator ArmBus, discussed in a following section.

3.1. Real-Time Computing Performance

We used a general purpose CPU rather than a special purpose FPU (floating point unit) as the CHIMERA real-time compute engine. Special purpose, single board FPUs are now available with performances in the 5 to 10 MFlop range, whereas few general purpose CPUs exceed 0.5 MFlops. However, the relatively low performance of a general purpose machine is offset by also providing high level language support, a simple instruction set, a large memory space, and a simple programming model - features lacking in most special purpose FPUs. In the final analysis, the simplicity of using the SUN 3 C compiler to generate code for the general purpose M68020 was the deciding factor in its choice as the CHIMERA real-time CPU.

Although the real-time processor was not chosen for performance alone, an effort was made to obtain the best performance possible from this general purpose CPU. The Ironics IV-3204 processor was specified with a 20 MHz M68020 CPU; the standard 20 MHz M68881 FPU was replaced with a 20 MHz M68882 second generation FPU. The IV-3204 includes 4 MBytes of 1 wait state DRAM, and the CHIMERA software allows the CPU to run with the on chip CPU instruction cache enabled, without an MMU, for optimum memory performance.

This hardware configuration has been tested using both conventional benchmark programs and a locally written (in C) matrix arithmetic package as computing performance tests. All code was compiled using the standard SUN C compiler, with the -O (optimize) switch. Table 3-1 summarizes the results obtained, including results for a VAX 780, SUN 3/160, Ironics, and the special purpose Marinco FPU used in the Direct Drive Arm II system for comparison.

To give perspective to this performance, consider a numeric reverse kinematics algorithm for a 6 axis robot using the iterative inverse Jacobian algorithm [2]. The dominant computational requirements of this algorithm are six 4x4 matrix multiplications and one 6x6 matrix inversion per iteration. Typically, 3 to 5 iterations are required for the algorithm to converge to an acceptable error. On the Ironics 3204 CPU, each iteration requires approximatelv 8 milliseconds to execute, for a total of 40 milliseconds[6]. This allows an Ironics dedicated to calculating manipulator inverse kinematics for Cartesian space control to run at approximately 25 Hz.

4. The CHIMERA Kernel

The CHIMERA kernel differs from conventional operating systems, such as UNIX, in that constructs are provided to allow precise control of process execution as a function of time (hence its designation as a *real-time kernel*). The CHIMERA kernel is functionally similar to a number of commercial products, such as pSOS. The decision was made to locally develop the kernel to ensure source code availability, considered important for the research environment for which CHIMERA was intended.

The CHIMERA kernel was designed to provide much of the functionality of a true multi-tasking operating system while preserving the response time of a dedicated real-time processor. This was made possible by eliminating many of the high overhead features required of a *multi-user* operating system, in particular inter-process security, a large process space, and virtual memory. In typical real-time control applications, the absence of these features is negligible.

[6]Of course, the algorithm has additional computational requirements, however they are minor compared to the matrix multiplication and inversion. Also, the operands of the matrix multiplication are homogeneous transforms, which allows the matrix multiplication operation to be further optimized if speed is essential.

Benchmark	VAX 780	SUN 3/160[1]	Ironics[2]	Marinco[3]
Dhrystone	1449	3302	4429	N/A[4]
Whetstones	448K	704K	1195K	N/A
Scalar Addition	5.5 μsec	14.6 μsec	8.8 μsec	2.4 μsec
Scalar Multiplication	7.9 μsec	16 μsec	9.9 μsec	2.3 μsec
4x4 Matrix Multiply	1500 μsec	898 μsec	514 μsec	~ 166 μsec
6x6 Matrix Inversion[5]	9900 μsec	7880 μsec	4670 μsec	1164 μsec

Table 3-1: Ironics IV-3204 Performance Benchmark Comparison

The CHIMERA kernel provides the following set of concurrent programming primitives which control process creation and execution:

- *P()* and *V()* - classic semaphore operations.
- *block()* and *wakeup()* - suspend and restart a process.
- *spawn()* and *kill()* - initiate and terminate a process.
- *pause()* - suspend process for some interval.
- *lock()* - lock an executing process in the CPU.

The kernel process representation and context switching mechanism is a conventional Motorola 68000 family implementation. Every process is defined by an ***instruction segment***, the ***CPU register set***, a ***user stack***, and a ***process control block***. The ***process control block*** contains the current process state (execution status, scheduling priority, timing constraints), as well as a ***supervisor stack space***. This stack space is normally used to save the ***CPU register set*** when the process is swapped out of the CPU, however it also provides each process with a unique ***supervisor stack*** for exception processing. Process exception handling can thus be multi-tasked, eliminating a potential cause of poor system response.

Process scheduling is performed using two different algorithms depending on the source of the initiating context switch. When a context switch is triggered by the hardware timer (indicating expiration of a time quanta), the highest priority process is scheduled as the next active process. Processes scheduled to execute at a specific time are given higher priority than other inactive processes. A context switch driven by resource contention implements a "round robin" type scheduler, chosing the next active process from a circular list of processes. These two algorithms were chosen to provide accurate control of time critical process execution and fair CPU allocation, with a small scheduling overhead. The scheduler code is written in the C programming language, making for simple modification if necessary.

4.1. Kernel Performance

The duration of a process context switch is considered an approximate measure of a multi-tasking system's overall performance. While many other factors can adversely effect the system performance, context switch time represents a hard limit on system response and CPU utilization which can not be changed by algorithm or process design.

The results shown in Table 4-1 indicate that processor utilization is typically in the 90% range with a time quanta of 1 millisecond. Extrapolating to a 0.5 millisecond time quanta, CHIMERA would yield approximately 80% processor utilization, and at the limit of 0.25 milliseconds 60% processor utilization. Utilization in this range is sufficient for many control applications, while providing extremely fine control over process execution timing. When a process is *locked in* the CPU (at the programmers discretion), the kernel imposes very little overhead, yielding processor utilization of 99.5%. This yields added performance for processes which must run to completion once triggered by timing criteria.

Process Type	Measurement	Minimum	Maximum	Average
Timer Driven Context Switch	Context Switch time	107 μsec	127 μsec	119 μsec
"	Processor Utilization[7]	87.6%	89%	88%
Locked in CPU	Context Switch time	N/A[8]	N/A	5 μsec
"	Processor Utilization	N/A	N/A	99.5%
Exception Driven Context Switch	Context Switch time	73 μsec	94 μsec	85 μsec
"	Processor Utilization	90.6%	92.7%	91.5%

Table 4-1: CHIMERA Context Switch Timing

5. CHIMERA File System

The CHIMERA file system takes a UNIX-like approach and treats all devices as files. A device driver is written for each device in the system, and the driver's entry points are entered into a device table. Examples of CHIMERA devices include pipes (for inter-process communication), a console terminal, and communication channels to other CHIMERA processors. All the standard UNIX I/O functions such as *open()*, *close()*, *read()*, *write()*, and *fprintf()*, operate with any CHIMERA device. This standardized interface eliminates the need for special library functions for each different device. Section 6 presents an example of how an interface to a robot controller is easily incorporated into the CHIMERA file system.

[1]68020 (16 MHz) + 68881 (12 MHz)

[2]68020 (20 MHz) + 68882 (20 MHz)

[3]AMD 29116 (16 Mhz) ALU

[4]Not available

[5]Gauss Jordan Elimination with full pivoting

[7]All CPU utilization ratings based on a nominal 1 millisecond time quanta.

[8]Not applicable

5.1. Inter-Processor Communication

Inter-processor communication (Ironics-Ironics and SUN-Ironics) is supported in CHIMERA by an interrupt driven, UNIX device oriented communication channel. A SUN UNIX driver (and its CHIMERA kernel equivalent) was written to implement a simple byte stream model of communication accessible on both processors by the standard file system utilities. Physically this communication occurs over the VME bus between any pair of CPUs. The driver software establishes several logical communication channels[9] multiplexed over the single physical VME channel. Any SUN process can communicate with any Ironics process by this channel mechanism.

Communication channels serve several important functions. As the Ironics CPU has no physical terminal attached to it, a communication channel is established between a SUN server and the Ironics. The SUN server creates a SUN window to act as a pseudo-terminal. Thus the user can interact directly with programs running on the Ironics CPU. A second application is the extension of the SUN file system to Ironics programs. A communication channel is created between a server process on the SUN and a server process on the Ironics. Requests for file system access by Ironics programs are passed from the Ironics server to the SUN server and serviced (effectively emulating the SUN file system on the Ironics). Effectively, the entire SUN file system is available to Ironics programs.

Communication channels are essential in controlling and monitoring robot performance - without them, it would be impossible to exchange data between the SUN and Ironics processes. High level control programs, running on the SUN under a robot programming language (AML/X) [3], send commands to control programs on the Ironics by opening a communication channel (by a UNIX *open()* call), then read or write data to this port (using any standard UNIX I/O function such as *read()* or *putc()*). An Ironics process waits for commands from the robot programming language and acts upon them. When the commands are completed, data regarding the manipulator motion is stored on disk files, again using the Ironics/SUN communication link.

5.2. File System Performance

The CHIMERA file system performance is inherently limited by that of the SUN file system, as CHIMERA simply requests the SUN perform the file operation for the real-time process. In order to provide a rough measure of this performance, a number of read and write operations of various sizes were performed on a SUN disk file and timed using the CHIMERA system clock. Each transaction was repeated several times to determine the range of expected values.

This experiment demonstrated an inherent shortcoming in the CHIMERA file system implementation: a high, fixed overhead associated with every file transaction independent of the transaction size. By observing the time required to read and write single bytes, an approximation of this overhead was obtained ranging from 7 to 50 milliseconds, with typical values of 15 milliseconds. As the transactions become much larger, the overhead is dominated by the actual data transmission, and transactions times closely track the transaction size. Write transactions of 1000 bytes or more occur at an average rate of 65K bytes per second (500 Kbaud), corresponding read transactions at rates of 190K bytes per second (1.5 Mbaud). These data rates are typical of the SUN Ethernet connection to the SUN file server, suggesting that this is the limiting factor in disk transaction speed.

In practice, the high file overhead, and the large range of variation in this overhead, suggest that time critical file transactions be double buffered. Using the CHIMERA concurrent programming constructs, a process can be dedicated to managing a pool of large buffers on the order of 100K bytes. This manager process monitors the contents of buffers in use, writing almost full buffers to disk, and substituting emptied buffers in their place. This technique allows a real-time program to store data on disk files at nearly the full speed of the SUN file system, while adding very little overhead to the real-time program.

6. Examples

This section illustrates the use of CHIMERA in two solving two common manipulator control problems. The first example describes how CHIMERA is used to isolate the user control programs from the low-level details of special hardware devices. The second shows how a multi-process model coupled with inter-processor communication, can be used to decompose a system into simpler parts.

6.1. Control Hardware Interface

In the RMMS system, a communication channel is needed between the controller computer and the actual manipulator modules. This channel provides both feedback from the manipulator joint sensors, and output commands to the joint actuators. This communication must be done at servo rates, for an arbitrary number of modules. To allow the RMMS modules to be easily assembled, this communication must take place over a fixed, and small number of conductors which are bussed through each module. This communication channel is implemented as a serial data bus, referred to as the *ArmBus* (figure 6-1).

The current ArmBus design is a self clocking, low-overhead, bi-directional serial data bus, operating at a nominal 5 MBit (million bits/second). A data transaction on the ArmBus is initiated by writing an appropriate address and function (read or write), and the data (in the case of a write) to memory mapped control registers on the ArmBus/Ironics interface. Upon completion of the ArmBus operation, an interrupt is generated on the Ironics CPU, indicating that data is ready or has been sent. Conceptually, the ArmBus can be considered a serial port. Data written to it will control the arm - data read from it indicates the state of the arm.

The serial port analogy illustrates the utility of a multi-tasking kernel. User level programs that send commands to the ArmBus are suspended awaiting completion (rather than busy waiting), allowing other compute bound processes to run. The interrupt signaling completion of the ArmBus operation is handled at the system level, transparent from the user level code. Thus the user who is primarily interested in robotic control is insulated from low-level system details (such as how to handle an interrupt). Also, because the ArmBus device is part of the file system, the standard *read()* and *write()* I/O routines are used to access it - no special library routines are needed.

6.2. Task Level Robot Control

A simple example is presented to illustrate the use of CHIMERA in a representative robot control environment. The example consists of generating a high level task description from a robot programming language, sending a series of MOVE commands defining a cartesian trajectory to a robot, interpolating a series of joint angles and supplying these to joint level servo-controllers. During the execution of the trajectory, record the results of the motion on disk files for later analysis of the control scheme.

Without a real-time operating system such as CHIMERA, this common robot programming task becomes formidable. Consider doing this with a single Ironics program with no multi-tasking. The program would be required to:

[9]Currently six

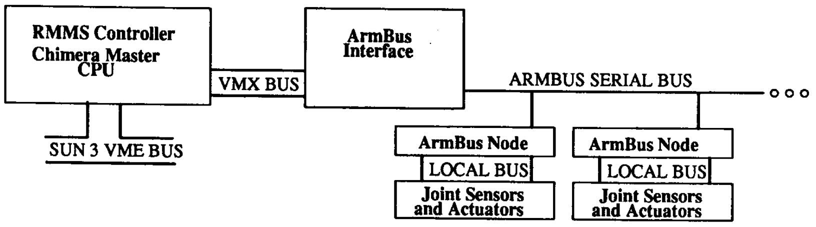

Figure 6-1: ArmBus Architecture

- Poll the SUN for experiment parameters (such as controller gains, sampling time, *etc.*) and trajectory data.
- Every T_m time interval, compute a new series of joint positions by interpolation of MOVE command joint values.
- Every T_c time interval, update the control loop and send these values to the arm.
- Every T_r record joint values (position, velocity). When reading these values from the ArmBus, we must busy wait until the ArmBus interrupts to signal completion of the request.
- Send the recorded data back to the SUN to store on disk files.

Note that in general, $T_c < T_m < T_r$. From a programming standpoint, such time dependent code is difficult to write correctly. Efficiency also suffers, as considerable time is wasted busy waiting for completion of I/O operations.

A multi-tasking operating system makes the design of the system significantly easier, as each logical module of code can be written independently, and can ignore low level system details. For example, one possible division of code into separate Ironics processes:

Trajectory Control

This process reads cartesian MOVE commands from the robot programming language, and converts them into joint space motion. These joint angles are used by the individual joint controller.

```
LOOP:
  Read MOVE commands
  LOOP:
    Compute joint angles
    Send to servo controller
    Pause $T_m$ seconds
  END LOOP
END LOOP
```

Servo Control

This process implements a control law for the manipulator joints. Gets reference angles from trajectory control, and measured angles from arm.

```
LOOP:
  Get reference angles
  Read joint angles
  Compute control law
  Update controller
  Pause $T_c$ seconds
END LOOP
```

Data Logging

Records the measured data from the arm for later analysis.

```
LOOP:
  Buffer arm data
  IF buffer full
    write data to disk
  Pause $T_r$ seconds
END LOOP
```

In the CHIMERA system, this experiment is mediated by several processes, distributed between the SUN and Ironics:

Motion Control	Generates a series of position controller inputs that move the manipulator through a desired path. This is the Ironics trajectory control process.
Position Control	Implement a control calculation at regular intervals to compute the manipulator actuator commands required for the specified motion. This implements the servo control process.
Data Logging	Records the state of the manipulator as a function of time, for later analysis.
ArmLab	An interactive process that simulates a control panel, allowing the operator to set controller gains and monitor the state of the manipulator. This is the Ironics master process.
AML/X	A robot programming language developed by IBM [3]. Used in high level task planning.
File Server	System process that acts as interface to SUN disk file system.

These processes are depicted pictorially in figure 6-2.

The main ArmLab process creates the data logging, motion control, and position control processes on the Ironics. The motion control process generates straight line trajectories and continuously updates the inputs to the position controller. The position controller interfaces to the manipulator via the ArmBus, and generates actuator commands to drive the manipulator through the desired motion. The motion control process receives commands from the robot programming language AML/X running on the SUN. Continuous data logging of the manipulator state requires periodic storing of this data to SUN disk files.

The above examples demonstrate the following important capabilities of CHIMERA:

- Multi-tasking allows use of a process model.
- Reliable high bandwidth communication between separate CPUs allows distributed computing.

Because of CHIMERA's multi-tasking, separate manipulator software modules (such as motion control, data logging, and error detection) are written as stand-alone units and run concurrently. The high level robot programming language (running on the SUN) communicating with an Ironics processor is an example of distributed computing in CHIMERA. Other processors could be added to perform other compute bound functions.

7. Summary

The CHIMERA programming environment provides a high level, UNIX like development system for real-time critical programs. Through careful design of hardware and software, relatively high computing performance is achieved while using standard commercial processors and programming languages. Furthermore, this performance is achieved while supporting high level concurrent programming constructs, allowing time and event driven programs to be written independent of the system hardware.

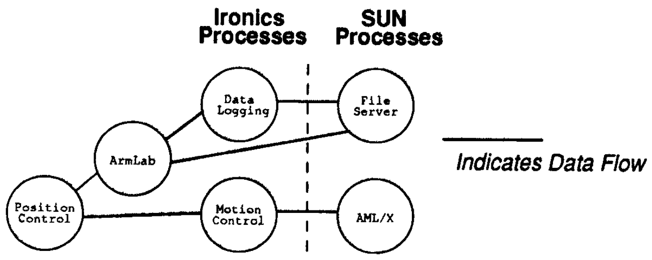

Figure 6-2: Processes to Control and Monitor Arm

8. Acknowledgements

The research reported in this paper was supported in part by NASA under contract NAG5-1091 and by the US Army Armament Research, Development and Engineering Center and DARPA under contract DAAA-21-89C-0001.

References

[1] Takeo Kanade, Pradeep Khosla and Nobuhiko Tanaka.
Real-Time Control of CMU Direct-Drive Arm II Using Customized Inverse Dynamics.
In *23rd IEEE Conference on Decision and Control*. December, 1984.

[2] L. Kelmar and P. Khosla.
Automatic Generation of Kinematics for a Reconfigurable Modular Manipulator System.
In *IEEE Conference on Robotics and Automation*. IEEE, April, 1988.

[3] L. Nackman, M. Lavin, R. Taylor, W. Dietrich Jr., and D. Grossman.
AML/X: A Programming Language for Design and Manufacturing.
In *Proceedings of the Fall Joint Computer Conference*. IEEE, 1986.

[4] S. Narasimhar, D. Siegel, and J. Hollerbach.
Condor: A Revised Architecture for Controlling the Utah-MIT Hand.
In *Proceedings of the IEEE Conference on Robotics and Automation*. IEEE, 1988.

[5] D. Schmitz, P. Khosla and T. Kanade.
Development of CMU Direct-Drive Arm II.
In *Proceedings of the 15th International Symposium on Industrial Robots*, pages 471-8. 1985.

[6] Jeduda Ish-Shalom and Peter Kazanzides.
SPARTA: Multiple Signal Processors for High-Performance Robotic Control.
In *Proceedings of the IEEE Conference on Robotics and Automation*. IEEE, 1988.