1-1-1992

The Chimera II Real-Time Operating System for Advanced Sensor-Based Control Applications

David B. Stewart

Donald E. Schmitz

Pradeep Khosla
Carnegie Mellon University, pkhosla@cmu.edu

This paper appeared is a draft copy of the article of the same name that appeared in IEEE Transactions on Systems, Man, and Cybernetics, Vol. 22, No. 6, pp. 1282-1295, Nov/Dec 1992. Unfortunately the published article, as it appeared in the journal, is not available on-line.

The Chimera II Real-Time Operating System for Advanced Sensor-Based Control Applications

David B. Stewart, Donald E. Schmitz, and Pradeep K. Khosla

Department of Electrical and Computer Engineering

The Robotics Institute

Carnegie Mellon University

Pittsburgh, PA 15312-3890

***Abstract**: This paper describes the Chimera II Real-time Operating System, which has been developed for advanced sensor-based control applications. It has been designed as a local operating system, to be used in conjunction with a global operating system. It executes on one or more single board computers in a VMEbus-based system. Advanced sensor-based control systems are both statically and dynamically reconfigurable. As a result, they require many special features, which are currently not found in commercial real-time operating systems. In this paper, we present several design issues for such systems, and we also present the features we have developed and implemented as part of Chimera II. These features include a real-time kernel with dynamic scheduling, global error handling, user signals, and two levels of device drivers; an enhanced collection of interprocessor communication mechanisms, including global shared memory, spin-locks, remote semaphores, priority message passing, global state variable tables, multiprocessor servo task control, and host workstation integration; and several support utilities, including a UNIX C and math libraries, a matrix library, a command interpreter library, and a configuration file library. Chimera II is currently being used with a variety of systems, including the CMU Direct Drive Arm II, the CMU Reconfigurable Modular Manipulator System, the Troikabot System for Rapid Assembly, and the Self-Mobile Space Manipulator.*

I. INTRODUCTION

Advanced sensor-based control applications, such as robotics, process control, and intelligent manufacturing systems have several different hierarchical levels of control, which typically fall into three broad categories: *servo levels, supervisory levels,* and *planning levels*. The *servo levels* involve reading data from sensors, analyzing the data, and controlling electromechanical devices, such as robots and machines. The timing of these levels is critical, and often involves periodic processes ranging from 1 Hz to 1000 Hz. The *supervisory levels* are higher level actions, such as specifying a task, issuing commands like *turn on motor 3* or *move to position B*, and selecting different modes of control based on data received from sensors at the servo level. Time at these levels is a factor, but not as critical as for the servo levels. In the *planning levels* time is usually not a critical factor. Examples of processes at this level include generating accounting or performance logs of the real-time system, simulating a task, and programming new tasks for the system to take on. In order to develop sensor-based control applications, a multitasking, multiprocessing, and flexible *real-time operating system (RTOS)* is needed.

An RTOS can be subdivided into several parts, including the real-time kernel, the multiprocessor support, the file system, and the programming environment. The *real-time kernel* provides local task management, scheduling, timing primitives, memory management, local communication, interrupt handling, error handling, and an interface to hardware devices. The *multiprocessor support* includes interprocessor communication and synchronization, remote interrupts, access to special purpose processors, and distributed task management. The *file system* provides access to secondary storage, such as disks and tapes, and to local-area-networks. The *programming environment* provides the tools for building applications; it includes the editor, compiler, loader, debugger, windowing environment, graphic interface, and command interpreter (also called a *shell*). The level of support provided for each part of the operating system (OS) varies greatly among RTOS.

In this paper, we present the Chimera II Real-Time Operating System, which has been designed especially to support advanced sensor-based control applications. Chimera II is designed as a *local OS* within a global/local OS framework, as shown in Figure 1. In such a framework, the global OS provides the programming environment and file system, while the local OS provides the real-time kernel, multiprocessor support, and an interface to the global OS. For many applications the global OS may be non-real-time, such as UNIX or Mach. However, the use of a real-time global OS such as Alpha OS [7] and RT-Mach [30] can add real-time predictability to file accesses, networking, and graphical user interfaces.

Most commercial RTOS, including iRMX II [5], OS-9 [13], and pSOS+ [24], do not use the global/local OS framework, and hence they provide their own custom programming environment and file system. The environments, including the editors, compilers, file system, and graphics facilities are generally inferior to their counterparts in UNIX-based OS. In addition, since much development effort for these RTOS goes into the programming environment, they have inferior real-time kernels as compared to other RTOS. Some commercial RTOS, such as VRTX [20] and VxWorks [32], do use the global/local OS framework. However, as compared to Chimera II, they provide very little multiprocessor support, and their communications interface to the global OS is limited to networking protocols, thus making the communication slow and inflexible. The commercial RTOS only provide basic kernel features, such as static priority scheduling and very limited exception handling capabilities, and multiprocessor support is minimal or non-existent. Previous research efforts in developing an RTOS for sensor-based control systems include Condor [18], the Spring Kernel [25], Sage [21], Chimera [23], and Harmony [3]. They have generally only concentrated on selected features for the real-time kernel, or were designed for a specific target application. Chimera II differs from these systems in that it not only provides the basic necessities of an RTOS, but also provides the advanced features required for implementation of advanced sensor-based control systems, which may be both dynamically and statically reconfigurable.

II. DESIGN ISSUES

Advanced sensor-based control systems should be *dynamically reconfigurable*, even for the simplest of applications. Consider the example of a robotic manipulator which is required to move an object whose location is known. This task can be broken up into three separate phases: pick up object; move to new position; put down object. When the manipulator is picking up the object, it must use force control for contacting the object, and gripper control to properly pick up the object. To move the object a different controller is required for the free motion phase. Possibly vision processing is also required to track the object's target location. To put down the object, force control and gripper control is again needed. The set of modules executing and the sensors required during each phase is different. Some of the modules and sensors are shared by the different phases, while others must be dynamically changed. As applications become more complex, the number of possible configurations also increases.

Advanced sensor-based control systems should be implemented on open-architecture hardware, so that new sensors and additional processing may be incrementally added to the system to increase the system's intelligence. In addition, the hardware and set of software configurations may have to be changed to support a different class of applications. Thus advanced sensor-based control systems must also be *statically reconfigurable*.

Several design issues, discussed in detail below, were considered in developing Chimera II, an RTOS for advanced sensor-based control systems that are both statically and dynamically reconfigurable.

A. Programming Environment and File System

The basic functionality found in all RTOS includes the programming environment, the file system, and the real-time kernel. The programming environment is required to quickly develop, debug, and maintain code. The basic requirements for the environment are an editor, a high-level language compiler, a linker, and a loader. The file system is required to store code and data on secondary storage, and to electronically transfer information to other systems. In order to provide all of the advantages of the full-featured programming environments and file systems of today's UNIX workstations, we adopted the global/local OS framework, and developed Chimera II as a local OS. Chimera II then requires a global OS to operate as the *host*, which makes all of the global OS's programming environment and file system features available to Chimera II. Given such a framework, we have developed a powerful interface between the host workstation and real-time environment, as described in Section III.B.

B. Open Architecture Real-Time Hardware

A typical hardware configuration for advanced sensor-based control applications may consist of multiple general purpose processors, possibly on multiple buses. The system may contain special processing units, such as floating point accelerators, digital signal processors, and image processing systems. The system will also include interfaces to several sensory devices, such as force sensors, cameras, tactile sensors, and range finders to gather data about the environment, and a variety of control devices such as actuators, switches, and amplifiers to control electromechanical equipment. We have implemented Chimera II around the VMEbus [16], since it is the most popular bus standard. However, much of the design of Chimera II is independent of the target hardware architecture, and hence can be used with other architectures.

Figure 2 shows a typical hardware configuration. General purpose processing is provided by commercially-available single board computers, which we call *real-time processing units (RTPUs)*. We have chosen to support the Motorola MC680x0 family of general purpose processors because of their popularity and their one-to-one mapping of hardware signals with the VMEbus signals [15]. The design of Chimera II allows other processor-based RTPUs, such as the Intel 80x86 and SPARC families, to also be used; however, we currently have not ported any software to those platforms. Any VMEbus-based I/O device or special purpose processor can be incorporated into Chimera II, by using the two-level device driver support offered by our OS.

C. Real-Time Kernel

The real-time kernel must include all the basic features found in any RTOS. These include task management, memory management, local shared memory, local semaphores, timer access, and hardware independence. *Task management* includes actions such as creating, destroying, blocking, waking up, setting priorities, and scheduling of concurrent tasks. *Memory management* is the allocation and deallocation of physical memory. We do not consider virtual memory, because we are not aware of any method to do paging in real-time. *Local shared memory* allows tasks on the same RTPU to share memory. *Local semaphores* provide basic synchronization for tasks on the same RTPU. *Timer access* allows the execution of tasks to be controlled by time. *Hardware independence* is a virtual machine layer, which allows user software to use the hardware without having to program hardware specific code. The level of hardware independence provided by an OS varies. We have developed an expanded real-time kernel suitable for reconfigurable systems. In addition to the basic kernel functions, our kernel also provides dynamic scheduling, two levels of device drivers, built-in control of timers, and global error handling. Details of the Chimera II kernel are given in Section III.A.

D. Interprocessor Communication and Synchronization

Most RTOS give very few mechanisms for interprocessor communication (IPC) and synchronization. The mechanisms available generally include support for an atomic read-modify-write instruction for limiting access to critical sections, and some form of message passing for higher-level communication. The VMEbus by default offers shared memory; however, most RTOS do not make any provisions for allocating that memory, or for automatically performing the address calculations required to access different parts of memory on the VMEbus. As a result, programs which use any of these mechanisms become bound to a single processor. The same code cannot execute on a different processor, without modification of VMEbus addresses, or in the case of message passing, modifying the names of the source and destination of the messages. In a reconfigurable system, modules must be designed such that they are independent of the target RTPU; hence these methods are not satisfactory. Second, these mechanisms do not provide all the tools desirable for quickly developing multiprocessor applications. In Section III.B. we describe the enhanced IPC and synchronization mechanisms developed in Chimera II, including global shared memory, spin-locks, remote semaphores, prioritized message passing, global state variables, and multiprocessor task control.

E. Predictability

The primary concern in a real-time system is not that it is fast, but rather that it is *predictable*. A fast system with unpredictable behavior can cause serious damage. In theory, the rate monotonic algorithm [10] is used to ensure predictable execution. However, this static scheduling algorithm is not suitable for dynamically reconfigurable systems and does not provide the CPU utilization achievable with dynamic scheduling algorithms [11]. We have developed the *maximum-urgency-first* algorithm to provide a predictable dynamic scheduling algorithm [26]. This algorithm has been implemented as the default scheduler for Chimera II. Support for the rate monotonic algorithm, as is provided in most RTOS, is also available with the default scheduler.

Most real-time scheduling theory concentrates on ensuring that tasks always meet their deadlines. However, nothing is said about what happens to tasks that fail to meet deadlines. In addition, even though a task can meet all deadlines in theory, abnormalities in the implementation may cause the task to miss its deadline in practice. In Chimera II we have addressed this issue by designing a deadline failure handling mechanism, which allows an exception handler to be automatically called when a task fails to meet its deadline. Possible error handling includes aborting the task and preparing it to restart the next period; sending a message to some other part of the system to handler the error; performing emergency handling, such as a graceful shutdown of the system or sounding an alarm; maintaining statistics on failure frequency to aid in tuning the system; or in the case of iterative algorithms, returning the current approximate value regardless of precision. Details of the Chimera II scheduler and deadline failure handling are included in Section III.A.

Deadline failures account for only one of the many types of errors which may occur in a system. Deadline failures are unique in that the errors are a function of time. Other errors that may occur in a system include hardware failures, software bugs, invalid data, invalid states, and processor exceptions. These errors may occur at any time, and are often detected within the user's code through the use of consistency *if-then* statements. The most typical method of dealing with these errors is to have the routine detecting the error to either handle it itself, or to return an error value, such as –1. Unfortunately, in a hierarchical software architecture, this method has two major problems. First, the exception handling code is embedded within the application code, making the code inflexible and difficult to maintain. Second, if an error occurs at the lowest level of the architecture, and it is only to be handled by a higher level module, then the error must be continually passed up through the intermediate software levels. This method is both cumbersome to code, and is also very inefficient, in that the time to enter the exception handling code requires the additional overhead of propagating the error through multiple hierarchical levels. In Chimera II we have developed a *global error handling mechanism*, which allows exception handling code and main program code to be coded independently. Details are given in Section III.A.

F. Modular and Reusable Software

In order to save on development time and make software more maintainable, it is generally accepted that applications should be designed in a modular fashion, and the software modules should be reusable. Chimera II extends this ideology to practice, by providing the tools which make it possible to quickly develop modular and reusable software. First, all communication mechanisms are processor transparent, and hence modules can be developed without knowledge of the final target hardware architecture. Second, the two-level device drivers supported by Chimera II provide a standard interface not only to I/O devices, as offered by other RTOS, but also to sensors and actuators. Third, the servo task control mechanism forces the programmer to develop code as reconfigurable, hence resulting in reusable modules. Each module can then execute on any RTPU at any frequency, and all modules can be controlled by a single task. The modules form a library of modules, any of which can be used in later systems without the need for recompiling any parts of the code. The intercommunication between the modules is handled automatically using a high-performance global state variable table mechanism. The servo task control mechanism and global state variable table are described in Section III.B. Details on developing a library of reusable modules can be found in [27].

G. Libraries

The usefulness of an operating system does not only lie in the features given, but also on the supporting libraries, which save on application development time. Like most other RTOS, Chimera II provides the standard UNIX C and math libraries. It also provides a concurrent standard I/O library (stdio), which is suitable for using the stdio facilities in a multiprocessor environment. In addition, several other libraries are provided, which are generally not found in other OS. These include a matrix math library, a command line interpreter library, and a configuration file support library. These libraries provide utilities which are often required by advanced sensor-based control applications. More details on these libraries are in Section III.C.

III. SOFTWARE ARCHITECTURE

The Chimera II software is divided into two distinct parts: 1) Code that runs on the RTPUs, and 2) Code that runs on the host workstation.

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.