Research in Real-Time Systems

by
Alan Shaw, Carl Binding,
Wei-laung Hu, and Kevin Jeffay

Department of Computer Science FR-35
University of Washington
Seattle,WA 98195
Technical Report 85-12-05
December 1985

This work was supported in part by the Washington Technology Center
and Boeing Aerospace Corporation.

December, 1985

Research in Real-Time Systems

by
Alan Shaw, Carl Binding, Wei-laung Hu, and Kevin Jeffay
Department of Computer Science FR-35
University of Washington
Seattle, WA 98195

1.0 Introduction

This research project, started in summer 1984, is concerned with methods and tools for generating *real-time software*. A general theme and objective is to include time as a first-class object in programs. Typical applications are air traffic monitoring and control, avionics systems used for guidance and control in air and space vehicles, life support systems in hospitals, and robotics where the software controls robots that perform manufacturing and testing operations.

The systems which we are investigating have a number of features and requirements that distinguish them from more conventional software and applications. The most important is the existence of strict timing constraints; correct systems performance involves not only standard correctness notions for software and hardware (e.g. computing the right answers) but also *timing correctness* such as responding to external events within a specific time interval or meeting given start and completion times for various applications processes. Other characteristics are the need for reliability and fault tolerance, both physical and logical concurrency, stand-alone and applications-specific hardware/software configurations, and difficult but critical pre-testing and certification. Some emphasis has also been given to graphics interfaces for users, both for real-time applications and for program design and generation systems.

As an empirical basis for developing and validating our ideas, we have selected a particular real-time application that has all the features of the general problem area and that can be attacked at a variety of levels. The problem is the design and implementation of an air traffic monitoring system (ATM) that tracks and communicates the position of all aircraft in

a given airspace volume. Real-time inputs and outputs include radar signals and control, communications received when new aircraft enter the volume and sent when aircraft leave the volume, operator input for querying and controlling the system, and one or more screen displays showing aircraft tracks and various textual data.

The following sections describe our current work and approach. Presently, it is concentrated in four related areas: timing specifications in concurrent programs, operating systems and concurrent programming languages, programming in the large, and simulator/prototype implementations.

2.0 Specification of Timing Constraints in Concurrent Programs

The problem is how to deal with time and clocks at different levels of abstraction and within the context of a general specification methodology for software. We have focussed on timing issues in real-time programs and in interactive graphics interfaces.

For specification purposes, it is useful to assume the co-existence of a variety of clocks and timing mechanisms. There may be many *local* (sometimes "virtual") clocks, one or more per process; there may also be *global* clocks that are used collectively by process families or the entire system. Clocks have different *granularities*, covering a broad range from very fine to very coarse. There are the "standard" clocks that generate software accessible ticks and counts at *fixed time intervals*; we also need *variable interval timers* to act as alarm clocks. A clock or timer is conveniently viewed as a separate process, implemented in hardware or software, that produces tick, current "time", and timeout messages and that can receive requests for such tasks as setting or resetting itself. (Clock synchronization problems, for example in distributed systems, have not been a part of this work.)

Flow expressions [Shaw 78], a language-based formalism for expressing concurrent and interleaved behaviors, are being studied as a means for specifying directly the behavior of systems of processes including clocks and timers. The descriptions are essentially sets of interacting regular expressions (or finite state machines) that synchronize and communicate

through a semaphore-like scheme.

A higher-level specification of the ATM software and its timing constraints has been written using flow expressions [Shaw 84] (One example of a timing constraint for ATM is that every aircraft track must be displayed at least once every three seconds). We have also used the notation for describing several instances of user interface elements [Chi 85; Chi and Shaw 85]. Examples are descriptions of a rotating globe, where either the rate of rotation or a sampled angle of rotation are synchronized with a local timer, and clicker software for a mouse input device, that recognizes single and double clicks according to the time intervals between pressing and releasing mouse buttons.

The principal research problems are finding the "right" software abstractions for the various timing constraints and creating a higher-level specification mechanism expressing these abstractions.

3.0 Operating Systems and Concurrent Programming Languages

The state of the practice of real-time design appears to be divided into two main branches. The first uses the generic operating systems tasking model, characterized by high level concurrency and synchronization constructs, such as processes, monitors, rendezvous, and condition variables. Here process scheduling is not explicitly dealt with by the programmer. Instead, scheduling occurs through the synchronization mechanisms and is often based on a priority scheme. The second style of real-time design has lead to models with explicit processor scheduling. Such systems rely mainly on statically-defined scheduling disciplines based on timing constraints.

Tasking models have the advantage that a system's logical correctness can be reasoned about. While they are general in nature, they have often been considered too inefficient for real-time programming. In addition, they provide no basis for discussing a system's timing properties. The explicit scheduling approach, while being a lower level methodology, does provide a framework for controlling time in a system, permitting efficient implementations.

Within each of these classes, we have identified and studied specific instances of real-time design methodologies.

Within the tasking approach, concurrent programming languages have provided useful paradigms for the construction of large systems. Systems developed with languages such as Mesa, Ada (in the future) and Modula fall into this category. (Modula-2 is an intriguing high-level language that does not bind the programmer into a specific model of concurrency; instead it provides lower level primitives which allow the designer to implement the most appropriate form.) An alternate method has been to provide a library of concurrency primitives that can be invoked from an essentially sequential programming language. This "tool-kit" idea is found in many commercial real-time development systems, such as IRMS 86 and VAXElan.

Systems within the explicit process scheduling class may have a static ordering of the real-time tasks or they may be decomposed into schedulable units called *slices, strips* or *chunks*. The cyclic executive approach, popular for example in avionics, often employs static, table-driven schedules for these slices. Part of our work has been to define a practical instance of a strip-based model [Baker and Scallon 85]. Other work has studied the implementation of these kinds of systems using higher level languages such as Ada [Shaw 85]. We have also experimented with a general hierarchic scheduler organization as well as explicit scheduling policies [Hu 85].

One aim of our current research in this area is to combine the flexibility, elegance, and generality of concurrent programming languages with the explicit scheduling and efficiency of the strip-based methodology. (e.g. [Donner 80]). Experiments with an Ada system are being considered to test our ideas. Longer term research will also address the effect of alternate hardware architectures on our solutions.

4.0 Programming in the Large for Real-Time Systems

A long term goal of our project is to provide interactive tools to design, test, and generate

real-time systems. One result may be executable specifications and/or various software generators. The MASCOT programming methodology [MASCOT 80] and the PegaSys system [Moriconi and Hare 85] exhibit some of the desirable features and requirements for these tools.

For this programming environment, we consider a real-time system to be composed from a set of program *component types* and *interconnection types*, such as modules, processes and different interprocess communication mechanisms. Instances of these types, the actual component and interconnection elements, provide specific functional abstraction or dynamic behavior. These elements are assembled together in order to realize a final application system. We started by identifying some reusable operating system components and interconnections, and examined how they could be linked together to produce a variety of OS kernels. This first effort was at the level of semaphores, process management modules, clocks, buffering modules, schedulers, and other kernel elements.

Currently we base the intended programming environment on the more general *entity-relationship* model. Entities represent reusable system components and relationships model the various interconnections between these components. A directed graph underlies the front-end in the preliminary implementation design. Properties are associated with both edges and nodes, including their images or iconic representations for the graphical display. It is still an open research question how to display various module interconnections types in a meaningful graphical way. Traditional methods include flow-charts, module interface dependency graphs, or data flow models, but none of these methods alone seems powerful enough. The user interface tools and model [Binding 85b] are being extended so that they may be used to construct a generalized graph editor which in turn will permit some experimental work in this area.

5.0 Implementations

Two major implementations efforts have been completed. We built a prototype ATM system to understand the problems of real-time design in general and to gain insights into one

specific real-time program organization. We also constructed a simulator to compare various scheduling policies that might be used in real-time systems and to compare various possible organizations for a given application.

5.1 ATM Prototype [Binding 85c]

For the ATM prototype, the C programming language was expanded with a simple concurrency mechanism providing non-preemptable (light-weight) processes [Binding 85a]. We built the prototype using a *process-oriented* model. In essence, one tracker process is associated with each individual aircraft. Tracker processes communicate with various other system components via a shared database. Although our implementation is only a rough prototype, its design is highly modular and easy to extend and maintain; we believe that this is a result of using the generic tasking or process model.

A major part of the ATM prototype is devoted to the user interface realized on a high-resolution bitmap device and using an interactive pointing device. Similar to our programming-in-the-large effort, we have tried to divide the user interface into a set of reusable components that in this case are high level, two dimensional I/O abstractions with specific interactive behavior. The user interface package [Binding 85b] provides various generally useful interface components that were employed to build the ATM operator interface. Although the ATM and other interfaces have been constructed with this package, the underlying user interface model needs extensions in order to coherently incorporate primitive drawing facilities along with the windowing, text, menu, and other existing components.

5.2 Simulator [Hu 85]

The scheduling algorithms which we have integrated in this simulator are the *deadline*

scheduling algorithm (also referred to as *due-dates scheduling*), a *priority preemptive* scheduling algorithm, and a combination of these. The latter uses a hierarchy of three task priority categories: *high, low* and *fill*. In general, high priority tasks are periodic, have a short execution time and a stringent timing requirement, and are non-preemptable. Low priority tasks are either periodic or sporadic (aperiodic) [Mok 83], have some tolerance for delays, and are preemptable only at specific execution states. Fill tasks can be considered as background tasks and have relative priorities.

Tasks of the highest priority category are scheduled at fixed points in time according to an *a priori* established scheduling table. The scheduling of tasks of the other categories is done dynamically: low priority tasks are scheduled according to their deadlines and fill tasks are scheduled based on their relative priorities. Further input to the simulator - beside the schedule table for high priority tasks - models the occurrence of interrupts, describes the execution times for individual tasks and gives the preemption points of low priority tasks.

Deadline scheduling works best when all tasks have the same priority. Some systems, however, contain tasks with differing tolerances for delays and therefore we believe that the hierarchical scheduling policy might yield better results. Using the simulator on our sample ATM problem and other real-time system examples, we hope to demonstrate this. The simulator will also be used to evaluate the high-level, general tasking model against the lower-level, explicit scheduling model of tasks. At this point, the simulator has been validated with a particular ATM organization.

6.0 Summary and Acknowledgements

This research is part of a long term research project on software generation. Our emphasis is on real-time systems and on the paramount role of time as a design and programming object [Shaw 84]. It is a joint industrial and academic project, partially supported by the Washington Technology Center and Boeing Aerospace Company. In addition to the authors, other contributors to the project are Theodore Baker (Florida State University), Gregory Scallon (Boeing), Edwin Stear (Washington Technology Center), and Russell Wilson

(Boeing).

References

[Baker 85] T.P. Baker. *An Ada Runtime System Interface.* Technical Report 85-06-05, Dept. of Computer Science, University of Washington, Seattle, WA., June 1985.

[Baker and Riccardi 85] T.P. Baker and G.A. Riccardi. *Implementing Ada Exceptions.* Technical Report 85-06-01, Dept. of Computer Science, University of Washington, Seattle, WA., June 1985.

[Baker and Scallon 85] T.P. Baker and G. Scallon. *An Architecture for Real-time Software Systems.* Technical Report 85-06-04, Dept. of Computer Science, University of Washington, Seattle, WA., June 1985. (To appear in *IEEE Software*).

[Binding 85a] C. Binding. Cheap Concurrency in C. SIGPLAN NOTICES 20(9): 21-26, September 1985.

[Binding 85b] C. Binding. *User Interface Components based on a Multiple Window Package.* Technical Report 85-08-07, Dept. of Computer Science, University of Washington, Seattle, WA., October 1985.

[Binding 85c] C. Binding. Prototypical Implementation of an Air Traffic Monitoring System. Internal Report, Real-Time Software Generation Project, Dept. of Computer Science, University of Washington, Seattle, WA., December 1985.

[Chi 85] U.H. Chi. *A Model and Notation for Specifying User Interfaces.* Technical Report 85-07-02 (Ph.D. Thesis), Dept. of Computer Science, University of Washington, Seattle, WA., July 1985.

[Chi and Shaw 85] U.H. Chi and A.C. Shaw. *Using Flow Expressions to Specify Timing Constraints in Concurrent Programs.* Technical Report (in preparation), Dept. of Computer Science, University of Washington, Seattle, WA., 1985.

[Donner 85] M.D. Donner. The Design of OWL, a Language for Walking. *Proc. SIGPLAN '83 Symp. on Progr. Lang. Issues in Software Systems.* In SIGPLAN NOTICES 18(6):158-165, June 1983.

[Hu 85] W.L. Hu. The Guaranteed Clocked Schedule and Simulator. Internal Report, Real-Time Software Generation Project, Dept. of Computer Science, University of Washington, Seattle, WA., December 1985.

[MASCOT 80] Mascot Supplier Association. *The Official Handbook of MASCOT*. Computing Standards Section, Royal Signals and Radar Establishment, Worcestershire, UK, 3 December 1980.

[Mok 83] A.K. Mok. *Fundamental Design Problems of Distributed Systems for the Hard Real-Time Environment*. MIT Technical Report, MIT/LCS/TR-297, 1983 (Ph.D. Thesis).

[Moriconi and Hare 85] M. Moriconi and D.F. Hare. PegaSys: A System for Graphical Explanation of Program Designs. In *Proc. ACM SIGPLAN 85 Symp. on Language Issues in Programming Environments*, pages 148-160. SIGPLAN NOTICES, July 1985.

[Shaw 78] A.C. Shaw. Software Descriptions with Flow Expressions. *IEEE Trans. on Software Engineering* SE-4: 242-254, May 1978.

[Shaw 84] A.C. Shaw. *Final Report : Development of Software Tools and Techniques for Automatic Software Program Generation*. Technical Report, Boeing Aerospace Company, Seattle, WA., December 1984.

[Shaw 85] A.C. Shaw. *Concurrent Programming and Slice-Based Scheduling*. Internal Report (in preparation), Dept. of Computer Science, University of Washington, Seattle, WA., December 1985.