A real-time computer must produce a correct result within a specified time. A real-time system is an integrated system (with embedded computers) that must respond to real-world requests or events in a timely fashion.

In these systems, an action performed too early or too late, even if it is functionally correct, is useless or even counterproductive. In safety-critical applications, it may be catastrophic.

A real-time system's timing requirements are usually defined by the application, not by the computer. So real-time computers must try to keep up with their host environments, which range from microcontrollers to large complex military systems.

Typical real-time applications include systems for aerospace (flight controllers and navigation systems), business (automated tellers and program traders), industry (process control and automated factories), instrumentation (automatic sensors and intensive-care units), military (missile controllers and myriad defense systems), and telecommunications (switching systems and multimedia).

The realm of real-time applications is expanding rapidly. Many new products now provide processor chips suited for powerful real-time systems; hardware and software interfaces for advanced real-time systems; and design and integration tools for building efficient real-time systems.
Coming to Grips with

REAL-TIME REALITIES

KWEI-JAY LIN, University of Illinois at Urbana-Champaign
EMMETT J. BURKE, Northrop Aircraft Division

- In the future, most consumer products will rely on embedded real-time computers. To produce more robust applications, development techniques must keep pace with ever-changing requirements.
Many people think that most future consumer products, computer-related or not, will have some real-time computers embedded in them.

Although progress in hardware technology has made high-performance processors available, high-speed execution may not solve all the problems real-time systems face. In fact, it is often better to have predictable performance over fast but unpredictable performance so that the system and the human operator know what to expect. It's better to drive a car that always brakes in 0.5 seconds than one that usually responds in 0.2 seconds but sometimes takes as long as 2 seconds.

Therefore, the primary objective in building a real-time system is to guarantee a consistent response that satisfies the system's timing requirements. The secondary objective is to ensure an acceptable result that meets functional requirements. The third objective is reasonable fast performance.

PERFORMANCE

Because real-time systems are complex, they are difficult to build. Many real-time systems now use hundreds of processors and thousands of processes to control activities worth millions of dollars. We need new software technology to guarantee the performance of real-time tasks in such systems. Many questions remain:

- How do we define the system's performance requirements?
- How do we predict each component's performance?
- How do we analyze the effect of component interactions on performance?
- How do we measure or monitor the performance of each module?
- How do we guarantee that performance will not be degraded by known factors like concurrency or unknown factors like interrupts?

In the past, performance has been considered only after all modules are implemented and individually tested, when performance engineers then conduct performance tests and, if the performance is not acceptable, apply tuning techniques like code optimization, load balancing, memory management, and scheduling policy. Often the result is a system that is so delicate and fragile and so tuned to the specific configuration and workload that any change in input parameters or system configuration causes a disaster. This is why real-time systems are very difficult to debug and modify.

We can no longer use this development process. Some military systems must handle workload from equipment that is not even completely defined. Even in a more static working environment, like an automated factory, there are always demands for new functions and configurations. Real-time systems must therefore be easy to change and reconfigure. We need new models, design methods, system structures, and programming tools to meet the demands of the next-generation real-time systems.

PAST COVERAGE

Past issues of *IEEE Software* have carried some important articles on real-time systems. Last year, an article by John Stankovic and Krithi Ramamritham ("The Spring Kernel: A New Paradigm for Real-Time Systems,' May 1991, pp. 62-72) examined the Spring kernel. The kernel implementation is part of the Spring project, an important real-time research project.

The goal of the Spring project is to support time-critical systems that must be flexible, adaptive, and reliable, as well as fast and predictable. The project takes a synergistic approach and includes the development of programming and runtime models, scheduling algorithms, operating systems, a distributed architecture, and development tools.

Also last year, an article by Kevin Kenny and Kwei-Jay Lin ("Measuring and Analyzing Real-Time Performance," Sept. 91, pp. 41-49) presented an empirical approach for determining the timing behavior of real-time programs. The article describes a timing tool that measures the actual execution time and uses the measurement results to determine the parameters of a programmer-supplied timing model. The timing model defines the programmer's understanding of the program's timing behavior. This empirical approach complements timing tools that use an analytical approach (such as the one described on p. 35 in this issue).

Finally, an article earlier this year by Luqi ("Computer-Aided Prototyping for a Command-and-Control System Using CAPS," Jan. 92, pp. 36-67) shows the feasibility of using a computer-aided prototyping system to validate a system's requirements. CAPS generates Ada code from a prototype's specification automatically, supports system management, and helps control system evolution. The project shows that prototyping lets users try designs quickly, without cutting corners.

IN THIS ISSUE

The six articles in this issue explore aspects of the sound, practical development process that we will need to make tomorrow's real-time projects manageable and systems dependable.

The first article, by Swaminathan Natarajan and Wei Zhao, provides a good overview of the issues involved in building dynamic, hard real-time systems, in which resource availability and requirements are not fixed. The authors describe their project, R-Shell, in this context. R-Shell is an object-oriented framework that structures resources in a class hierarchy so that they can be selected or substituted dynamically. This interesting proposal addresses an important problem, but it may be difficult to implement in practice.

The next article, by Stuart Faulk and colleagues, presents the Software Productivity Consortium's formal method to capture requirements for real-time embedded systems. The Core method is object-oriented and integrates graphical and formal methods. One contribution Core makes is that it can describe required behavior as mathematical relations between environmental quantities, so you can check the completeness and consistency of the requirements. On the other hand, it can express only simple timing relations. Whether Core can express large, complex real-time constraints is an open question.
SIGNPOSTS AND LANDMARKS: A REAL-TIME READING LIST


Several conferences and workshops are now devoted partly or completely to real-time systems. In the US, the IEEE Technical Committee on Real-Time Systems sponsors the annual Real-Time Systems Symposium, which this year will be held in Phoenix, Arizona, December 7–9. In Europe in the last few years, a school and symposium on formal techniques in real-time and fault-tolerant systems and a series of Euro micro workshops on real-time systems have been established. The proceedings of these meetings (available from IEEE, New York, and Springer-Verlag, Berlin, respectively) provide coverage of important ongoing work.


While it usually takes considerable expertise to exploit theoretical results to solve specific systems problems, several efforts to encourage the reduction of theory to practice are under way. For example, the Software Engineering Institute of Carnegie Mellon University plans to publish Handbook of Real-Time Systems Analysis Based on the Principles of Rate Monotonic Analysis in early 1993.


Practitioners should also be interested in the real-time-related extensions to major standards such as Posix (P1003.4), IEEE Futurebus+, and Ada 9X.

—Alvin K. Mok, University of Texas at Austin

The next four articles explain systems under development and in use. The first, by Gustav Pospischil and colleagues, presents an integrated programming environment from the Mars project. The environment is built around a tool that analyzes the maximum execution time for each real-time task. It lets users experiment with task-timing combinations by using a text editor and a time editor. However, the environment does not analyze tasks that use shared resources, which could be a serious drawback for building many real-time applications with concurrent tasks.

The second article is a short summary of a compiler project by Prabha Gopinath and colleagues. Their compiler rearranges and partitions codes to improve performance predictability. Although these techniques are very useful, you should apply them cautiously, as they could change the semantics of concurrent tasks.

In the third article, Magnus Morin and colleagues present a framework for intelligent real-time systems that can perform both numeric and symbolic manipulations. Their proposed architecture transforms fluent, which are abstractions of data evolution over time, as they proceed through system layers. Some transformations produce reactions to the environment. The architecture guarantees performance by analyzing the worst-case response times, however, an approach that could result in inefficient implementations.

Finally, Kang Shin and colleagues, describes HARTS, a distributed system designed for fault tolerance around a 19-node hexagonal mesh architecture. All the tools described here—the HARTOS operating system, synthetic workload generator, real-time performance monitor, and fault injector—can be used to evaluate existing software empirically. Although the article answers some questions of performance analysis and measurement, it does not describe how software should be developed on HARTS.

REFERENCES


Kwee-Jay Lin is an associate professor in the computer science department at the University of Illinois at Urbana-Champaign. He is the principal investigator of the Concord Project which studies the software design and scheduling issues for flexible, hard real-time systems. His research interests include real-time systems, scheduling theory, programing languages, and distributed systems.

Emmett J. Burke is manager of software systems development for Northrop’s Aircraft Division. He is interested in methods for training the architects of software systems and building environments that support architectural system analysis.

Kwi-Yew Lin received a BS in electrical engineering from the National Taiwan University, and an MS and a PhD in computer science from the University of Maryland at College Park.

Emmett J. Burke received a BS in mathematics from Loyola-Marymount University and an MS in engineering from Arizona State University.

Address questions about this issue to Lin at Computer Science Dept., University of Illinois, Urbana, IL 61801; Internet kmr@cs.uiuc.edu.