Applications that call for real-time systems are particularly susceptible to failures. Our challenge is to design, analyze, and build such systems to prevent failures or (at least) to mitigate their effect on operations. The theme articles in this issue, summarized in the box on p. 14, explore different ways to meet this challenge.

What differentiates the development of real-time systems software from other applications? How do reliability and safety figure into their development?

A real-time system must respond to externally generated stimuli within a finite, specifiable time delay. Real-time systems are typically embedded systems that interface directly to the physical equipment they operate, monitor, and control. Examples of such systems are flight- and weapons-control systems, air-traffic-control systems, telecommunications-switching and network equipment, manufacturing process-control, and even speech-recognition systems.
ARTICLE SUMMARIES: SAFETY AND RELIABILITY

- Software Testability: The New Verification, pp. 17-28
  Jeffrey M. Voss and Keith W. Miller

As software begins to replace human decisionmakers, a fundamental concern is whether a machine will be able to perform the tasks with the same level of precision as a skilled person. The reliability of an automated system must be high enough to avoid a catastrophe. But how do you determine that critical automated systems are acceptably safe and reliable? In this article, we present a new view of verification and offer techniques that will help developers make this assessment. Our view, which we label software testability, looks at dynamic behavior, not just syntax. This differs from traditional verification and testability views.

Our research suggests that software testability clarifies a characteristic of programs that has largely been ignored. We think that testability offers significant insights that are useful during design, testing, and reliability assessment. In conjunction with existing testing and formal verification methods, testability holds promise for quantitative improvement in statistically verified software quality. The methods described in this article are applicable to any software system, not just real-time systems.

- Reliability Through Consistency, pp. 29-41
  Kenneth P. Birman and Bradford B Glade

Distributed computing systems are increasingly common in critical settings, in which unavailability can result in loss of revenue or incapacitate information-dependent infrastructure components. In many distributed computing environments, failures are reported in a way that violates even the simplest notions of consistency. We think such practices are at the root of reliability problems.

Failure notification is key to effective failure management. When processes are not aware that another process has failed, they may not operate reliably. Achieving such awareness requires some mechanism for consistent failure reporting among processes.

Unfortunately, many modern computing systems treat failure reporting as an application-specific problem, which puts the burden of failure recovery on the application developer. Thus, failure-reporting mechanisms will vary from system to system, even though the reliability requirements of applications span these systems.

Standards bodies have also overlooked this issue: no communications standard today requires consistency in failure reporting. Indeed, no standard even provides for the addition of consistency-preserving mechanisms.

We believe inconsistent failure reporting is one of the major barriers to progress in developing highly reliable, self-managed, distributed software systems and applications. Such inconsistency is dismayingly obvious, even if it seems to be unnecessary: implementing consistent reporting is both particularly costly and overwhelmingly difficult. If more developers came to appreciate its value, we believe that a major barrier to distributed application development could be eliminated.

- Analyzing Safety Requirements for Process-Control Systems, pp. 42-53
  Rogério de Lemos, Aimer Saeed, and Tom Anderson

Demands for dependability are rising faster than what can currently be achieved, especially for complex systems. As the trend continues, researchers are looking more closely at the requirement phase as the potential solution for managing errors. Experience shows that mistakes made during this phase can easily introduce faults that lead to accidents, so preventing faults during this phase should produce more dependable systems.

However, in safety-critical systems, you cannot assess the adequacy of safety requirements except with respect to overall system risk. That means you must also conduct a safety analysis of the resulting safety specifications to ensure that the software's contribution to system risk is acceptable.

We have developed an approach to identify and analyze the safety requirements of safety critical process control systems. Our approach increases the visibility of the requirements process by partitioning the analysis into distinct phases that are based on the domains of analysis established from the system structure. Freedom to tailor a technique to a specific domain of analysis and perspective ensures that the most appropriate techniques are applied. Formally recording the safety specifications helps in building and comparing safety strategies. Finally, using both qualitative and quantitative techniques gives a precise picture of the specification's contribution to overall system risk.

- Scheduling in Hard Real-Time Applications, pp. 54-64
  Jiang Zhu and Ted G. Lewis

A major problem with hard real-time systems is how to confirm that they really work. In these systems, the computer periodically gets information from the environment through sensors, updates its internal systems' states based on those inputs and its current internal states, and generates control commands to change the environment through actuators.

The success of such a system depends not only on its logical correctness but also on its timing correctness. A timing-dependent life-critical system is called a hard real-time system. It must make correct responses to environmental changes within specified time intervals or deadlines.

Our work involves proving theorems that guarantee deadlines in a uniprocessor environment will be met and developing new graphical languages for the design of hard real-time systems. Our research has shown that we can effectively combine graphical design languages and deadline-scheduling algorithms. Our graphical design language integrates dataflow and control flow into one diagram, which easily reveals the entire picture of a hard, real-time application. This picture is much more difficult to grasp when the dataflow and control flow diagrams are viewed separately.

We have developed a CASE tool that implements the design diagram and our schedulability checking methods. The tool automatically transforms a graphical design into a task set and then schedules it. The tool can be used by control-application engineers to graphically design an application and automatically analyze the design for schedulability on a given processor. If the design is feasible, it can then be automatically transformed into Ada.
systems, which are beginning to appear in telecommunications networks and personal computers.

GROWING PREVALENCE

Not a day passes that we don’t come into contact with a real-time system. And now they are becoming more prevalent in critical applications. A failure in a critical application such as a telecommunications system may result in great financial loss; in a flight-control system it may result in loss of life.

More effort must be expended to analyze the reliability and safety of such systems. Analysis of hardware components in critical applications has matured over the years and commonly followed techniques have emerged. However, methods and techniques for analyzing the reliability and safety of the software part of critical applications are relatively new and still maturing. Yet the vulnerability of the system to software failures is on the rise and may (in some cases does) exceed hardware failures.

Software is not only becoming more prevalent in real-time systems, it is becoming a larger part of real-time systems. By larger, we mean the amount of effort expended in designing and implementing the software over total expended effort.

DIFFERENTIATING FACTORS

What differentiates real-time software development from other software?

+ First, its design is resource-constrained. The primary resource that is constrained is time. Depending on processor speed, this time constraint equates to the number of processing cycles required to complete the task. However, time is not the only resource that can be constrained. The use of main memory may also be constrained. In fact, designers may trade off reducing processing cycles at the expense of using more memory.

+ Second, real-time software is compact yet complex. Even though the entire system may have millions of lines of code, the time-critical part is but a small fraction of the total software. Yet these time-critical parts are highly complex, with much of the complexity introduced to conserve constrained resources.

+ Third, unlike other software systems, with real-time systems there may not be a human around to help the software recover from failure. Such software must detect when failure occurs, continue to operate despite the failure, contain the damage to surrounding data and processes, and recover quickly so as to minimize operating problems.

In developing real-time software, more time is spent in design, especially analyzing ways to improve performance and enhance reliability and safety. Development of most other software focuses on how to handle a normal situation, but real-time, critical-application development also focuses on how to handle the abnormal situation. Also, more time is spent in testing real-time systems.

Testing not only removes faults (the underlying source of software failure) and hence improves its reliability, it also demonstrates reliability and
The world's best data security is now available in these distinctive packages

*All these products, and many others, use RSA technology. To see more, call us for your free RSA Security Solutions Catalog.

You don't have to be one of the biggest names in the software business to have unbeatable RSA security features in your product. Right now, for as little as $290, you can start developing with the same RSA software developer's toolkits used by companies like Apple®, Lotus®, Novell™, and WordPerfect® in some of the world's best-selling software. It's easy, too. Because when you let the most trusted name in the crypto business handle your encryption and authentication features, you can concentrate on what you know best—writing your application.

BSAFE 2.1 is the latest release of the world's most popular, general purpose cryptography toolkit. And the new TIPER 1.1 is specifically designed for standards-based secure e-mail and messaging development. Both are built for fast, easy integration of today's most advanced crypto technology into your existing applications. So make sure that when your customers look for security, they can pick the most distinctive package: Yours. Call RSA today at 415/595-8782, or e-mail us at info@rsa.com.

RSA. Because some things are better left unread.

William W. Everett is a distinguished member of the technical staff in the Software Technology Center at AT&T Bell Laboratories. He is a former associate editor-in-chief of IEEE Software and served as guest editor for the "Software Beyond 2001: A Global Vision" special issue (November 1994). His research interests include software-reliability engineering and software-process engineering. Everett received an engineer's degree from the Colorado School of Mines and a PhD in applied mathematics from the California Institute of Technology.

Shinichi Honiden is the manager of Toshiba Corp.'s Systems and Software Engineering Laboratory and chief researcher for the Laboratory for New Software Architectures at Japan's Information Technology Promotion Agency. His research interests include object-oriented methodologies and agent-oriented models. Honiden received a BE, an ME, and a DEng., all in electrical engineering, from Waseda University. He is a member of the IEEE, the Information Processing Society of Japan, and the Japan Society for Software Science and Technology.

Address questions about this issue to Everett at AT&T Bell Labs, Room 3D-416, PO Box 638, Murray Hill, NJ 07974-0636; w.w.everett@att.com or to Honiden at Systems and Software Engineering Laboratory, Toshiba Corp., 70 Yanagi-cho, Saiwai-ku, Kawasaki 210, Japan; honiden@ssel.toshiba.co.jp.