Strategic Directions in Real-Time and Embedded Systems

JOHN A. STANKOVIC ET AL.[1]

Department of Computer Science, University of Virginia, Charlottesville, VA 22903
⟨stankovic@cs.virginia.edu⟩

1. INTRODUCTION

Real-time computing is an enabling technology for many important application areas, including process control, nuclear power plants, agile manufacturing, intelligent vehicle highway systems, avionics, air-traffic control, telecommunications (the information superhighway), multimedia, real-time simulation, virtual reality, medical applications (e.g., telemedicine and intensive-care monitoring), and defense applications (e.g., command, control and communications). In particular, almost all safety-critical systems and many embedded computer systems are real-time systems. Further, real-time technology is becoming increasingly important and pervasive, e.g., more and more infrastructure of the world depends on it.

Strategic directions for research in real-time computing involve addressing new types of real-time systems including open real-time systems, globally distributed real-time, and multimedia systems. For each of these, research is required in the areas of system evolution, composibility, software engineering, the science of performance guarantees, reliability and formal verification, general system issues, programming languages, and education. Economic and safety considerations, as well as the special problems that timing constraints cause, must be taken into account in the solutions.

In Section 2, several examples of real-time systems, their corresponding importance, and several examples of research success are presented. In Section 3 key future challenges and research related to strategic directions are highlighted. A vision of the field for the next ten years is presented in Section 4. Section 5 summarizes the paper.

2. EXAMPLES OF REAL-TIME SYSTEMS AND RESEARCH SUCCESSES

A *real-time system* is one in which the correctness of the system depends not only on the logical results, but also on the time at which the results are produced. Many real-time systems are embedded systems, i.e., they are components of a larger system. If incorrect operation of a system can lead to loss of life or other catastrophes, it is called a safety-critical system—air-traffic control, for instance. An air-traffic control system must continuously manage massive amounts of data. Unlike some large data-management systems, such as airline reservations, air-traffic control data is constantly changing and has extremely high value (related to public safety) for very short amounts of time (response-time requirements vary from a few milliseconds for radar data to several seconds for flight control information). At completion, the new U.S.

[1] Contributors to this article include Alan Burns, Kevin Jeffay, Mike Jones, Gary Koob, Insup Lee, John Lehoczky, Jane Liu, Al Mok, Krithi Ramamritham, Jim Ready, Lui Sha, and Andre van Tilborg.

air-traffic control system is estimated to cost over five billion dollars. However, the system is so large and complex (the new system will have between 1 and 2 million lines of code and thousands of consoles) that new real-time research is needed to improve safety even further, lower the cost of the system and its maintenance, and provide for its continual evolution as the system grows in size and complexity. Together with the need for safety, there is a need for a more scientific basis for the coherent treatment of time and time-based functionality, concurrency, and dependability.

Real-time and embedded computing also plays a key role in many industrial sectors. Consider the automobile industry. Auto manufacturers can remain competitive only if they incorporate state-of-the-art real-time computing systems into their cars. In the future, distributed real-time control systems will replace and enhance many of the conventional control systems of the car, making cars more efficient and adding to public safety. Before distributed real-time control systems can be used, several significant research challenges must be addressed: precise real-time response to the microsecond in a distributed system, fault tolerance under strict timing requirements, maintainability, and testability under competitive pricing pressures.

Research in real-time computing has been very effective. For example, advances in the science of temporal quality-of-service (QoS) guarantees have led to many schedulability conditions and efficient, robust, and accurate validation algorithms for real-time applications such as digital control and constant-bit-rate video and audio. The validation technology built on these theoretical advances [Klein et al. 1993] in the design and development of real-life systems has now been used successfully. (The software system onboard the satellites in the NAVSTAR Global Positioning System (GPS), for example). Together with specialized hardware, this system maintains an accurate timing signal for navigation users, monitors the integrity of the navigation information, estimates the satellite orbital parameters, and maintains the synchronization of the GPS constellation. The timely completion of many tasks in this system must be guaranteed because many terrestrial applications are dependent on GPS information and erroneous GPS information could have serious consequences.

The real-time software industry is another example of the success of real-time technology. This industry has closely followed the progress of the semiconductor industry in general and the microprocessor industry in particular. Current estimates are that over 2 billion dollars are spent annually on tools, application software, and embedded operating systems. This market is also growing at approximately 25% per year. The commercial real-time operating system market, which started around 1981, is currently over $100 million per year with a 30–35% annual growth rate. Commercial real-time operating systems are used in a wide variety of embedded systems including avionics, medical, communication, consumer, and instrumentation applications. Many of these systems are "mission-critical" and have been certified by government agencies, including the U.S. FAA, and its foreign equivalents. For example, the McDonnell-Douglas MD-11, the Boeing 757, 767, and 747-400 all fly with commercial off-the-shelf real-time operating systems.

Future success is also expected in many other areas. For example, real-time commerce on the Internet is transforming business. Since for every PC there are thousands of embedded computers, a thriving embedded computing industry can be expected. There will be small embedded processors in millions of products, making them more intelligent. The potential for the future use of real-time technology is unlimited.

3. FUTURE CHALLENGES AND RESEARCH

Real-time systems will undoubtedly become pervasive. They will support an increasingly broader spectrum of applications. Many have widely varying time and resource demands and must deliver dependable and adaptable services with guaranteed temporal qualities. Moreover, both technical and economic considerations make it necessary to build these systems using commodity computer networks and commonly used operating systems and communication protocols.

While all the challenges and important research areas (see Lee and Krishna [1993]; Son [1995]; Stankovic [1988]; Stankovic and Ramamritham [1988]) in real-time computing cannot be covered here, several key areas are stressed. First, several high-level challenges are discussed: system evolution, open real-time systems, composibility, and software engineering. Then, key enabling basic research is discussed, including the science of performance guarantees, reliability and formal verification, general systems issues, real-time multimedia, and programming languages. Finally, the need for education in real-time computing is presented. Since many of these topics are strongly related, some degree of overlap is unavoidable and serves to highlight the strong relationships between these topics.

3.1 System Evolution

Computers have revolutionized the production of goods and delivery of services. Nevertheless, existing real-time computing infrastructure often introduces formidable barriers to continuous process improvement, equipment upgrades, and agility in responding to changing markets and increased global competition. Consider the following anecdotal scenarios from industry.

Process improvement: A research department developed a process modification that significantly improved product yield and quality. With a relatively minor modification of the processing sequence and new set-points for key process variables, the improvements were demonstrated on a pilot plant. Nevertheless, the improvements were never implemented in the plant because the line manager persuaded management that it couldn't be accomplished cost-effectively. Although the required software modifications are simple logic modifications, the process sequence is controlled by a set of networked PLCs (programmable logic controllers) coordinating several valves, sensors, and PID loops with several hundred lines of ladder logic code. The effects of the modifications on the timing requirements are unknown. The technician who wrote the PLC programs left the company, and the last time a modification was attempted on the code it took the process down completely, costing thousands of dollars in downtime. The line manager wants no part of installing the so-called process improvements developed by the research department.

Equipment upgrades: Over the years, aging equipment has been replaced with new production technology so that the factory now has a hodgepodge of old and new equipment with controllers from five different vendors, each with its own programming interface, data structures, data communication protocol, and timing characteristics. One of the older process machines failed recently and the best replacement was from yet another vendor. It had a considerably shorter cycle time, improved reliability, and a more sophisticated control computer. As with previous upgrades, however, installing the new equipment required yet another development and integration effort with yet another proprietary computing environment. It was particularly costly to create the custom interfaces to communicate with the other equipment in the system and there was no way to predict the timing effects. Consequently, the only way the equipment could be installed safely was to perform extensive tests during factory downtime. In the end, integration cost was several times the capital cost of the new equipment, and worse yet, it took several times longer to install the equipment than originally estimated.

These problems and others are widespread in the real-time and embedded systems industry. For example, in response to such problems, Chrysler, Ford, and GM have worked together and stated their requirements for next-generation real-time computing systems in a white paper, "Requirements of Open, Modular Architecture Controllers for Applications in the Automotive Industry" (Aug. 15, 1994). In this paper an important requirement states: "Controllers of these systems [auto manufacturing systems] must support changes with minimal delay and yet maintain performance requirements. The controllers should also allow users to easily add or upgrade controller functionality without relying on technology vendors and controller suppliers."

A paradigm shift is needed for real-time computing, from a focus on technologies for completely new installations to one designed to mitigate the risk and cost of bringing new technology into functioning industrial systems. Industry needs a computing infrastructure in which upgrades are safe and predictable, with negligible down-time. Specifically, the important features of this new real-time software architecture technology should include the following:

—The extensive use of open-standard-based components whenever possible, from backplane buses to operating systems and networks and communication protocols.
—A coherent set of interfaces for the integration of process control, plant-wide scheduling, and the plant management information system.
—A convenient and safe environment for customization, optimization, and reconfiguration of plant operations; on-line development, testing, and integration of promising new technologies and products; and trouble-free replacement of obsolete subsystems.

The prescription of off-the-shelf components for building real-time applications implies two research challenges:

(1) Developing scheduling and resource management schemes for (sub)systems with demonstrable predictability properties. The predictability properties must cover functionality, timeliness, and fault-tolerance.
(2) Given subsystems with known predictability properties, developing schemes to compose them into larger systems such that the predictability properties are also composible. This is essential to make real-time system components reusable. Composing systems to satisfy given functionality requirements is by itself a difficult problem. The addition of fault-tolerance and timeliness makes it a qualitatively formidable problem, but one with enormous payoffs.

System evolution is facilitated by open systems and composibility, which are presented in the next two sections.

3.2 Open Real-Time Systems

Today, most real-time systems are built to achieve a specific set of goals. The set of tasks to be performed is well understood at system design time.

In contrast, one challenge facing the real-time systems community is how to build and deliver general-purpose, open real-time systems and applications that permit a dynamic mix of multiple, independently developed real-time applications to coexist on the same machine or set of machines. Such a real-time architecture would allow consumers to purchase and run multiple applications of their choice, including applications with real-time requirements, on their general-purpose home and business computers, just as they do with nonreal-time applications today.

Some of the difficulties of an effective real-time architecture supporting open real-time computing include:

—Hardware characteristics are unknown until run time (processor speeds, caches, memory, buses, and I/O devices vary from machine to machine).
—The mix of applications and their aggregate resource and timing requirements are unknown until run time. These together lead to the following fundamental difficulty.
—Perfect *a priori* schedulability analysis is effectively impossible. Somewhat different and more flexible approaches will likely be needed than are now typically used for building fixed-purpose real-time systems.

Another possible difference is that perfect execution may not be the most important criterion for open, consumer real-time systems. Rather, for non-safety-critical systems, the right criterion is the most cost-effective execution, as perceived by the consumer. For instance, the consumer might choose to purchase a $50 video player application that happens to drop single frames under rare circumstances rather than a $400 application verified and certified never to drop frames. For these systems, both economic and correctness criteria will be applied.

3.3 Composibility

Many real-time systems are highly dynamic, e.g., defense systems such as early warning aircraft, command and control, autonomous vehicles, missile (control) systems and complex ship systems. These systems operate for long periods in fault-inducing and nondeterministic environments under rigid time constraints. They need to be robust while delivering high real-time performance. They need to evolve and use legacy components. Composition has long been recognized as a key issue for these systems. However, composition has largely focused on functional composition. A current research objective is to develop the notion of composition across three interacting domains: function, time, and fault tolerance. Both off-line and on-line solutions are required. The results should permit verification. Thus, the results will lead to adaptive high-performance fault-tolerant embedded systems that *dynamically* address real-time constraints and provide both *a priori* acceptable system-level performance guarantees and graceful degradation in the presence of failures and time constraints. Any on-line composition is itself subject to time and fault-tolerance requirements as well as having to produce functional, timing, and fault-tolerant components that create the system's actions.

To keep the cost reasonable (in both the short and long term), dynamic real-time systems must utilize vendor-neutral, portable programming and operating environments and adaptive fault-tolerance techniques. The programming environment must contain tools strong on analysis with respect to meeting fault-tolerance and real-time constraints. The operating environment must support dynamic, flexible, and adaptive behavior under time constraints, easy interoperability with commercially available products, and easy porting to other platforms. The adaptive fault tolerance must be user-specifiable and tailored to each application and function.

3.4 Software Engineering

Although the discipline of software engineering has always been motivated by large-scale complex systems, most of which involve substantial real-time requirements, the bulk of the research and products in this field addresses only functional issues. Incorporation of capabilities to express and manage timing, dependability, and other nonfunctional constraints, if done at all, has typically been left to specialized versions that fail to make it into mainstream releases (witness the large number of working groups currently laboring to extend CORBA to accommodate real-world systems). Since these retrofitted tools are never quite satisfactory, real-time system engineers resort to developing their own tools specialized for the current project. Both communities rely on a static approach in which a fixed set of requirements is mapped onto a known platform, with no inherent accommodation for evolvability in either the underlying hardware or the system requirements.

In spite of its limited success to date in addressing the more constrained real-time system problem, software engineering will need to undergo a radical shift in perspective and approach if it is to remain relevant in this new environment:

—Time, dependability, and other QoS constraints must become first-class concerns, coherently integrated with functionality at all levels from requirements specification through architecture, design, implementation, and execution;
—Evolvability must be ensured by separating platform-dependent concerns from application concerns (the same software should run unchanged whether the underlying platform is a dedicated LAN or a shared wide-area environment);
—Software must be structured into composible modules in which interfaces capture not only functionality but assumptions about the environment and conditional guarantees about service relative to the assumptions;
—Software must be designed to be adaptive and configurable, enabling application-dependent tradeoffs for timeliness, precision, and accuracy to be negotiated in response to changes in the environment;
—Timing constraints, in particular on individual components, must be dynamically derived and imposed on the basis of end-to-end requirements.

The technologies required to realize this vision draw on capabilities previously considered in relative isolation. Aside from the language and metaprotocols addressed elsewhere, several technologies currently targeting static design environments such as formal methods, resource management algorithms, compilers, and schedulability analyzers must be integrated and repackaged as efficient run-time services in order to dynamically assure QoS requirements.

This research will result in a capability to rapidly develop, deploy, and evolve complex real-time software systems on arbitrary platforms, with automatic adaptation to the characteristics of those platforms and the operating environment.

3.5 The Science of Performance Guarantees

Classical real-time systems have provided off-line deterministic guarantees in meeting safety and real-time requirements subject to failure and environmental assumptions [Klein et al. 1993]. The collection of algorithms and analysis that exists for these static, deterministic guarantees can be referred to as a science of performance guarantees. As systems become larger, more dynamic, and deployed in nondeterministic and less safety-critical environments, an expanded science of performance guarantees is required to support the development of these new systems.

For example, for dynamic real-time systems, on-line admission control and dynamic guarantees have been utilized for some time. However, for the most part, analysis of these systems has relied on simulation and testing. What is required is a science of performance guarantees that can provide a more formal analysis of dynamic real-time systems when the environment is not fully predictable or controllable and when failures occur.

In addition, as real-time system technology is being applied to applications such as stock market trading and multimedia, there is a need for probabilistic guarantees that are applied to the general notion of QoS requirements. For example, it is neither necessary nor cost-effective to guarantee the delivery of every packet of a multimedia video stream. A science of performance guarantees needs to be developed that permits accurate analysis of meeting probabilistic requirements of various timing and delay specifications. Part of the expanded science of performance guarantees may include new extensions of queueing theory that emphasize meeting deadlines.

Timing validation is a central and key area of research for the science of performance guarantees, and for real-time and embedded computing in general. The merits of different validation algorithms are measured in terms of their complexity, robustness, and degree of success. For example, some existing validation algorithms run in constant time, or $O(n)$ time, where n is the number of tasks in the system. They are well suited for on-line timing validation. More complex algorithms run in pseudo-polynomial time, but have better performance in other dimensions such as minimizing total schedule length. They can be used off-line.

Every schedulability condition and validation algorithm is based on some workload model. When applied to a system, the conclusion of the algorithm is correct if all the assumptions of the model are valid for the system. A validation algorithm is robust if its conclusions remain correct, even when some assumptions of its underlying workload model are not completely accurate. The use of a robust validation algorithm significantly reduces the need for an accurate characterization of the applications and the run-time environment, as well as the efforts in analysis and measurement of the individual applications for validating the workload model. For example, the existing validation algorithms based on the periodic task model are robust. Although the model assumes that jobs in each task are released periodically and execute for an equal amount of time, such a validation algorithm remains correct in the presence of deviations from periodic behavior.

Efficiency and robustness can be achieved easily if the degree of success of the validation test is not a concern. A validation algorithm has a low degree of success when it is overly pessimistic and declares tasks unschedulable except when system resources are unduly underutilized. A scheduler using a validation algorithm with a low degree of success may reject too many new tasks that are in fact schedulable and acceptable.

New research in timing validation technology is required because various limitations in the state of the art make it inadequate for many modern and future real-time systems:

(1) Existing schedulability conditions and validation algorithms produce unacceptably low degrees of success for applications with sporadic processing requirements, i.e., applications in which tasks have widely varying release times, execution times, and resource requirements. Similarly, they cannot take into account, in a sufficiently accurate manner, the unpredictable behavior of the hardware platform and variable amounts of time and resources consumed by the system software and application interfaces. As a consequence, they are overly pessimistic when applied to sporadic applications, especially in large, open run-time environments built on commodity computers, networks, and system software.

(2) Many existing validation algorithms are based on deterministic workload and resource models and work for deterministic timing constraints. Some of these algorithms are not applicable and others may not be robust when used to validate probabilistic timing constraints. The severe shortage of existing real-time

benchmark applications makes accurate calibration and validation of the probability distributions assumed by the underlying workload and resource models nearly impossible now and in the near future. Therefore, probabilistic validation algorithms must be sufficiently robust and remain correct even when applied to systems for which some assumptions of the models are not valid.

(3) Most existing validation algorithms are for statically configured systems (i.e., systems in which applications are partitioned and processors and resources are statically allocated to the partitions).

Recent efforts in validation are directed toward the development of new approaches, theories, and algorithms that do not have these limitations.

3.6 Reliability and Formal Verification

As computers become essential components of complex systems, improvements in reliability become increasingly important. There are many techniques developed and used in practice for improving reliability, ranging from static analysis based on formal methods and scheduling theory to dynamic analysis based on testing and run-time monitoring and checking. Although these techniques are quite effective when applied independently to small-scale systems, it is important to have a common framework on which these different techniques can be uniformly applied to large-scale systems. With a common framework, one model of the system is specified and then used for both static and dynamic analyses. This would save effort in developing and specifying models, since each technique requires its own model of the system. Furthermore, it would make it unnecessary to check consistency between models used by different techniques.

Such a common framework can be developed by extending an existing real-time formalism. There are four different widely used classes of real-time formalisms: logics, automata and state machines, Petri nets, and process algebras. Each of these formalisms can probably be integrated with scheduling, testing, and run-time monitoring and checking techniques to provide a common framework. There have been some promising preliminary efforts in integrating process algebra with scheduling and logics with scheduling theory, in creating test generation from specifications and monitoring based on specifications. For example, real-time processes and logics have been extended with schedulability analysis so that the same specification can be subjected to the analysis of both schedulability and functional correctness. Furthermore, some preliminary work has been done on testing of real-time properties based on formal specifications. In particular, test suites have been automatically generated from specifications, and specifications have been used as oracles during the testing process. Much work is needed to determine their effectiveness in practice.

In general, the amount of detail present in implementations is much greater than that captured in specifications. This makes it imperative to retain traceability of real-time requirements from application description to low-level implementation. A multilevel specification technology is needed whose applicability is not limited to abstractions in terms of state machines, Petri nets, or other formalisms. Instead of performing a mapping from a high-level abstraction of a system to its detailed implementation, what is needed is a multilevel specification mechanism to establish the preservation of real-time requirements in terms of conditions on the low-level implementation. Such a technology should ensure the correct functioning of the system through monitoring and checking at run time. It should be possible to use multilevel specifications to derive what conditions to check at run time. Better understanding of how to check correctness of real-time systems at run time is essential in achieving the reliability of complex real-time systems. In the same vein, a multilevel specification technology can be used at design time to catch design errors through simulation and other techniques. Recent work has indicated that it is possible to combine simulation with model checking. The advantage of this combination is that the part of the state space that needs to be considered by the model checker can be restricted by the path formula representing the simulation trace under question. Since it is unlikely that model checking alone can ascertain the correctness of a low-level implementation, techniques such as multilevel specification, coupled with an appropriate design methodology, for the imposition of real-time requirements are important.

3.7 General System Issues

A vast number of important research issues exist in the architecture, communications, operating systems, and database areas of real-time systems. First and foremost is the specialized support from these areas in solving the key research issues presented in other parts of this paper, i.e., the science of performance guarantees, system evolution, reliability, and so on. Second, each of these areas has its own research problems.

For brevity, a few open questions are listed for each area:

—What architecture changes are necessary to better support the calculation of worst-case execution times?
—How can the impact of various types of caches be included in the computation of worst-case execution time?
—How can guarantee-based time-constrained communication of various types be implemented and analyzed?
—How can real-time multicasting be efficiently implemented?
—How can a predictable real-time thread package be implemented?
—What empirical studies should be undertaken to help support the creation of real-time models?
—What are good resource and workload characterization models for real-time systems?
—How can hardware-software co-design be done for real-time systems?
—What support is needed for virtual environments and multimedia, especially when significant pattern recognition or other types of processing must occur in real time?
—How can real-time transactions provide guarantees?
—What are good architectures and protocols for real-time databases so that the temporal validity of data is maintained?

3.8 Real-Time Multimedia

Multimedia computing and communication has the potential to revolutionize the way humans interact with computers, collaborate with remote colleagues, entertain themselves, and learn about the world. Beyond today's applications such as desktop video conferencing, the future real-time transmission and processing of continuous media will enable the routine use of distributed virtual environments in diverse applications such as telemedicine and remote surgery, monitoring and control of automated factories, and interactive personal advertisements.

Some challenging research issues include:

—Precise specification of the predictability requirements. Most are likely to be probabilistic in nature. One of the challenges lies in separating end-to-end, i.e. application-level predictability requirements from internal (sub)system-level predictability requirements.
—Developing schemes for mapping predictability requirements into mechanisms that can demonstrably meet the requirements. Where algorithms

have been developed for providing absolute guarantees for individual dynamic requests, assuming that worst-case needs are known, the challenge lies in understanding the macro- or system-level predictability properties of such algorithms and in providing probabilistic guarantees when resource needs of requests are also known probabilistically. Accomplishing this in a layered distributed and heterogeneous system demands new approaches for composing and integrating QoS guarantees.

One possible approach to the new analysis requirements is to use queueing theory. Queueing theory is a well-known body of techniques and methods designed to describe the resource-sharing behavior of systems and applications with a substantial stochastic element. The difficulty with queueing theory is that it largely focuses on equilibrium behavior and aggregate QoS measures (such as average response time). It does not allow us to determine if individual applications meet their own QoS specifications.

An important future challenge will be the development of an integrated set of analysis tools that combines the focus on application QoS satisfaction associated with real-time scheduling theory with the ability to handle a wide range of stochastic application and system behavior associated with queueing theory. Furthermore, this methodology must provide predictable resource sharing by tasks with a range of QoS requirements. The resulting theory might be called real-time queueing theory for guaranteed systems.

3.9 Programming Languages

Given the broad range of real-time requirements (and other dependability requirements), many different aspects of guarantees and the numerous scheduling schemes, it is unfortunate but not surprising that programming languages are, in general, very weak in supporting these needs [Burns and Wellings 1989]. They lack the expressive power to deal with the generality of temporal requirements and implementation strategies. The inability to program such properties is a serious impediment to their transfer into industrial practice.

As it is unlikely that a single static language will be able to accommodate all requirements (including emerging ones), it is necessary to look for more effective structuring mechanisms for programming languages. Developments in the OOP paradigm have drawn a distinction between functional and non-functional behaviors. The functional behavior is expressed within a computatinal model, but the computational model itself can be altered (enhanced, modified, etc.) by programming at the metalevel. This process, known as reflection, will enable behavioral aspects to be addressed, and has the potential for significantly improving the effectiveness of programming real-time systems. The ability to manage a wide range of requirements, timing guarantees, and scheduling approaches via metalevel programming is a key challenge.

3.10 Education

Fundamentally, teaching real-time systems is teaching the science of performance guarantees. Ideally, this science would be treated in much the same manner, and indeed in parallel with the treatment of the science of formal verification of logical correctness, in an undergraduate computer science and engineering program. Thus enabling students to manipulate programs and systems both as mathematical objects and as components within larger systems; in particular within systems that interact with processes in the real world.

There are at least two distinct aspects of the teaching of real-time systems. The first concerns the understanding of the execution of a (sequential) program in time. Here the issues are those of management of time as a basic resource, the consideration of explicit time constraints, and the development of appropriate abstractions of execution time for use in performance-critical programs. Just as a course in data structures studies various fundamental abstractions of physical memory and develops insights into how problems can be decomposed into primitive operations on these abstractions, a real-time component in an introductory computer science sequence would emphasize how execution time can be organized to realize a real-time constraint. For example, we might present program-structuring techniques (including data structures) that allow execution time to be manipulated abstractly. Examples of such structures include sieves and refinements from the imprecise computing literature. This portion of the curriculum is largely algorithmic, and indeed builds naturally on existing analysis of algorithms. The key differentiating aspect, however, is the fact that traditional measures of execution time such as asymptotic complexity do not provide enough leverage for solving real-time problems. Alternate techniques such as interval arithmetic and the calculus of execution time intervals are required.

The second aspect concerns the management of logical and physical concurrency in time. This material emphasizes the highly concurrent nature of most real-time systems. Two subthemes are the traditional study of cooperating sequential processes and the more specialized study of the effects of competition for shared resources among processes. The former concerns the distribution of function among, and the synchronization and communication between, multiple, logically parallel processes. Abstractly, the latter is the analysis of how sequential execution time is dilated in the face of competition for resources and how the dilation process can be managed to ensure performance guarantees.

The challenge is to integrate real-time system concepts into existing computer science curricula so as to create an awareness of the basic issues in stating, manipulating, and realizing performance guarantees. For example, the real-time systems community should consider the development of a series of course companion monographs along the lines of the performance modeling Performance Supplement series commissioned by the Computer Measurement Group (CMG) and the ACM Special Interest Group on Measurement and Evaluation (SIGMETRICS). It would also be beneficial to integrate course material with practical training, possibly via industry projects.

4. TEN-YEAR VISION

Ten years from now almost all products and engineering processes will contain real-time features and embedded processors. There will be a greater demand for safe, dependable, and certifiable real-time systems. The demand will increase if there are major financial disasters or loss of life. Eventually, we should see the rise of a vital industry of third-party components that can be composed with respect to functionality, timing, and dependability.

For example, over the next ten years there will be an enormous increase in the prevalence of open systems. A significant fraction of the global population will interact with some form of open real-time system. Such systems will be highly distributed, offer a wide range of services including the control of commonplace artifacts, deal with many forms of data, be accessed over very wide areas, and be constantly changing and evolving. Moreover, they will begin to play an important role in the economies of developed and developing nations. While many operational aspects of these open systems are performance-related rather than real-time, there are at least three areas in which real-time systems are likely to play a key role.

(1) High-integrity real-time services that need to be delivered in well-defined time intervals. For example, there is a need to synchronize the

release of important financial data to a number of different international centers, with very tight jitter requirements.

(2) QoS attributes that equate to real-time parameters. For example, various bandwidth-allocation algorithms require a precise definition of the temporal needs of video and audio streams in order to undertake the necessary run-time reservations. QoS will become a key issue once services demand payment at the point of delivery.

(3) The ubiquitous nature of the pervasive open systems of the future may well require high-level temporal controls so that the sheer complexity of interacting with these systems can be managed. Hence, simple commands such as "deliver A to X 5 minutes before B is delivered to Y" may become a natural way of expressing commands.

These three areas indicate that open system protocols (in particular the interfaces to services, controls, and data objects) must explicitly deal with a range of nonfunctional issues such as dependability and availability of resources, QoS guarantees, and real-time requirements. Failure to address these issues will lead to increased dissatisfaction and eventually economic damage. Past lessons have taught us that nonfunctional properties cannot be added as an afterthought; they must be at the core of the architecture and protocol designs. Success in integrating real-time methods into open systems thus has the potential for significant social impact.

As real-time technology becomes increasingly used in everyday computing, it is our hope that widely agreed-upon interfaces and methods supporting open real-time systems will evolve. Such a development will produce a vital industry providing dependable third-party real-time components that can be composed, assembled, and used when building real-time systems—just as independently developed software components from multiple vendors can be used for building nonreal-time consumer applications today. This is both desirable and achievable.

The eventual result should be the proliferation of many products and systems that are safer, cheaper, and more available. If real-time technology is truly successful, the technology will be invisible to the public.

If accurate, our predictions of the growing importance of real-time systems technology over the next decade imply the need for a concerted effort to integrate real-time systems concepts into computer science curricula. Many of the basic concepts are well developed and well understood within the research community; others are still evolving and under investigation. A first round of real-time systems textbooks is due out during academic year 1996–1997, and will serve to formalize the training of graduate students and specialized undergraduates in real-time systems.

5. SUMMARY

Real-time computing is an *enabling technology* for many current and future applications that affect public safety, competitiveness, the economy, and lifestyle. Many results have been developed, but difficult research and transfer of technology issues remain [Stankovic 1988]. For example, real-time research has yet to grapple with three major realities concerning real-time applications:

—real-world real-time systems are expected to survive and continue to operate even when not all timing constraints are met or when components fail;

—due to economic and portability considerations, the tendency towards the use of off-the-shelf hardware and software components to build real-time systems is increasing;

—the prohibitive cost of modernizing an industrial real-time computing system often results from the down time and risks associated with inserting time-dependent new technologies into a functioning industrial system.

One proposed strategic decision is to develop a major funding and international research initiative in real-time computing to capitalize on current results, establish generic technology for the future, and thereby pay large dividends for safety and the economy. As an example, results that make possible an evolvable real-time computing *open infrastructure* would aid the safe and cost-effective insertion of hardware, software, and domain technologies into functioning industrial systems, creating a direct linkage between the ability to innovate and superiority in product quality and process agility.

REFERENCES

Burns, A., and Wellings, A. 1989. *Real-Time Systems and Their Programming Languages*. Addison-Wesley, Reading, MA.

Klein, M. H., Ralya, T., Pollak, B., Obenza, R., and Harbour, M. G. 1993. *A Practitioner's Handbook for Real-Time Analysis*. Kluwer Academic, Boston, MA.

Lee, Y.-H., and Krishna, C. M. 1993. *Readings in Real-Time Systems*. IEEE Computer Society Press, Los Alamitos, CA.

Son, S. 1995. *Advances in Real-Time Systems*. Prentice Hall, Englewood Cliffs, NJ.

Stankovic, J. 1988. Misconceptions about real-time computing: A serious problem for next generation systems. *IEEE Computer 21*, 10 (Oct.), 10–19.

Stankovic, J., and Ramamritham, K. 1988. *Hard Real-Time Systems*. IEEE Computer Society Press, Los Alamitos, CA.