Responsive systems: An introduction

R. F. Berry
P. E. McKenney
F. N. Parr

This paper introduces responsive systems: systems that are real-time, event-based, or time-dependent. There are a number of trends that are accelerating the adoption of responsive systems: timeliness requirements for business information systems are becoming more prevalent, embedded systems are increasingly integrated into soft real-time command-and-control systems, improved message-oriented middleware is facilitating growth in event-processing applications, and advances in service-oriented and component-based techniques are lowering the costs of developing and deploying responsive applications. The use of responsive systems is illustrated here in two application areas: the defense industry and online gaming. The papers in this special issue of the *IBM Systems Journal* are then introduced. The paper concludes with a discussion of the key remaining challenges in this area and ideas for further work.

INTRODUCTION

This paper provides an introduction to responsive systems, a term in which we include real-time, event-based, and time-dependent systems. A real-time system is a computing system subject to operational deadlines from the occurrence of an external event until the system response. A real-time constraint can be *hard*, in which case failure to complete the required computation before the deadline amounts to a system failure, or it can be *soft*, in which case lateness in the system response is tolerated and amounts to decreased quality of service (QoS).

An event-based system is a computing system whose components operate in response to events. These events can be either external to the system or generated by another component. A time-dependent system has time constraints on its response. One example of a non-event-based time-dependent system would be a real-time industrial control system in which there are no events, but instead small changes in a control output based on sensing of continuously varying inputs.

The capabilities of a responsive system often must include requirements such as detection of significant information, even when obscured by other data (noise); timeliness as specified by QoS requirements; and an open architecture to enable flexible integration into existing information technology (IT) installations.

It is important to note that the sheer size of modern IT installations mandates that responsive systems span data centers, rather than being confined to a single computer system, as has traditionally been the case for the more-familiar embedded real-time systems. This situation prompts the addition of timeliness QoS to the existing service-oriented architecture (SOA)[1] instead of creating a parallel reference architecture for responsive systems.

This paper is organized as follows. The next section discusses current trends driving demand for responsive systems. The following section presents two examples of responsive systems: an application from the defense industry and another from online gaming. The penultimate section introduces the papers included in this issue. The papers are grouped in three categories: responsive system platforms, middleware for responsive systems, and development support and tooling for responsive systems. The final section focuses on key challenges in this area still to be overcome and discusses ideas for future standardization and development work.

TRENDS IN THE ADOPTION OF RESPONSIVE SYSTEMS

A number of trends are contributing to significant growth in the adoption rate of responsive systems. Business information processing systems are increasingly expected to meet stringent latency requirements in handling requests for service. Embedded system applications are being integrated more and more into soft real-time command-and-control systems. Improved message-oriented middleware facilitates growth in event-processing applications. In addition, the expanded use of service-oriented and component-based techniques is reducing the costs of developing and deploying responsive applications. We expand on each of these in the following subsections.

Responsiveness of business information systems

The earliest electronic computer systems were costly, physically large, permitted only one user at a given time, and often required setup for each processing job. Individual users were required to sign up for machine time, often days or even weeks in advance. The advent of batch-processing systems provided automated isolation between successive jobs, so that users could submit decks of punch cards and collect their output, usually within hours. Later, interactive timesharing systems often provided sub-minute response times, which improved in the decades that followed. By 1989, the Transaction Processing Performance Council's TPC-A online transaction processing (OLTP) benchmark featured a response-time constraint: benchmark runs provided a throughput rating in terms of transactions executed per second, but any runs that did not offer transaction response times of 2 seconds or less for at least 90 percent of the transactions were deemed invalid.[2] More recently, studies have established that users of commercial Web sites judge response times of a few seconds to be adequate, and response times in the 100-millisecond range to be ideal.[3] Real-time constraints are thus expanding beyond their traditional embedded niche into mainstream applications, and they are becoming orders of magnitude more stringent. Such a sharp increase in requirements indicates that new methodologies and technologies must be used.

The following question arises: at what point does the law of diminishing returns cause further decreases in latency to be uninteresting, that is, to have only marginal benefits? This question can be answered by considering the time value of information, which in many cases decays rapidly. In some industries, this rate of decay can be extremely rapid indeed. In investment banking, it is conceivable that for each thousandth of a second that the trading software of a bank can act faster than the software of its competitors, the bank would gain $100 million per year in additional revenue.

Although program-driven trading in financial markets is not generally considered to be a safety-critical system, we can expect real-time scheduling deadlines to be taken quite seriously in this domain. Other areas in which real-time QoS requirements are emerging include battle-scenario decision making, virtual worlds and multiplayer online games, Web-based retail infrastructure, and real-time credit-card fraud detection. These mainstream workloads will therefore need to run on platforms that provide real-time QoS.

Although currently it appears that real-time requirements grow more stringent over time, the question arises whether there is a natural limit to this process. After all, millisecond-scale and smaller latencies might seem unnecessary for global-scope mainstream workloads, given that the speed-of-light round-trip latency around the globe comes to more

than 130 milliseconds. However, many transactions are local, and can thus be completed much more quickly. For example, a 10-mile-diameter financial district has a speed-of-light latency more than three orders of magnitude faster, at about 100 microseconds, and a number of financial districts are smaller yet. Furthermore, speed-of-light delays can be reduced to essentially zero by collocating the computing infrastructure, a possibility that becomes increasingly feasible as servers continue to shrink in physical size. Therefore, speed-of-light delays need not be a limiting factor for real-time QoS in the near term.

Another traditional limitation on the need for low latencies has been the relatively slow speed of human decision making and reflexes. However, this limitation disappears when machine-to-machine operations are considered. The New York Stock Exchange has reported that more one-third of all U.S. stock trades involve programmed trading, which must respond to market data in milliseconds rather than in the hundreds-of-milliseconds time frame of the human reflex, let alone the many seconds or minutes required for a human being to make a reasoned decision in such a complex domain.

We therefore have ample reason to believe that there is considerable room for further increases in the stringency of real-time QoS requirements, even for large-scale applications.

Broadened scope of embedded systems

Embedded and safety-critical system design has traditionally focused on meeting timeliness and associated reliability requirements. The current trend suggests these applications should be better integrated with other real-time applications, with higher-level command-and-control logic, and with business systems. For example, in a military context, weapon control and tracking systems may need to be integrated and combined with higher-level command-and-control logic, which in turn may be receiving additional policy information about threats from neighboring regions. In an electronic or chemical manufacturing system, time-critical embedded controls and sensed measurements on individual processing steps need to be reported and evaluated by command-and-control logic that understands the current production orders and manufacturing schedules currently in process. In a city government context, we now expect that 911 emergency-response systems are fully integrated with video and audio monitoring of key locations and with traffic-management systems.

Design techniques for embedded systems have traditionally been based on exhaustive analysis of all possible states of the system and proof that enough computational power is available to meet critical time deadlines. This is feasible when the state space of the application is small. When a real-time application is integrated with higher-level command-and-control logic, as in the examples above, the complexity and number of states in the integrated system greatly exceeds that in any of the parts. Communication between components introduces additional timing and state complexity. Some assurance of response time of critical paths must be provided, but in the form of soft real-time analysis and program correctness rather than by exhaustive state and computational path-length analysis.

Event processing and message-oriented middleware

Message-oriented middleware (MOM)[4] was introduced in the 1980s to facilitate integration of independent applications. Each application preserves its own scheduling, transaction and data model while exchanging information with other applications asynchronously via reliably delivered queued messages. Scalability and responsiveness of MOM systems was subsequently enhanced through the introduction of message brokers and push-based publish-subscribe messaging modes.[5] An individual application registers interest in specific topics, or message contents; the MOM takes responsibility for delivering relevant messages to the application and driving appropriate application processing when it arrives. This technique increases the responsiveness of applications because processing is driven as soon as an event of interest is detected, rather than waiting for the application to poll for messages at prescheduled intervals. The inclusion of Java** Message Service and message-driven beans in the Java Platform, Enterprise Edition (J2EE**), regularizes this basic event-driven processing.

Complex event processing (CEP) further refines this methodology by allowing each application to specify sets of rules or sequences of messages identifying situations of interest. The MOM continuously scans streams of arriving information to detect these

situations and, when such a situation occurs, triggers application processing. Various stream extensions to Structured Query Language (SQL) have also been proposed as situation flags. Another more recent refinement is the introduction of event-processing networks in which situation detection may be both distributed and layered.[6] In this scheme, low-level event detectors use rules or stream-processing specifications to extract higher-level events, which are then processed.

These messaging and event-processing technologies improve the responsiveness of applications by having middleware take on the responsibility of analyzing input data and detecting critical situations to which they must respond.

Advances in service-oriented and component-based techniques

Use of real-time and event-based solutions will be limited as long as each deployment requires the involvement of professionals with expertise in configuring the system and verifying that responsiveness goals are met. The use of a standardized enterprise Java programming model and more recently the use of service-oriented component models have been a key to reducing life-cycle costs for transactional enterprise applications.

For such applications Java is recognized as a widely used aid to improve quality and reduce application development costs. However, most embedded and real-time systems are currently developed in lower-level languages such as C or C++, sometimes with distributed real-time services provided through real-time Common Object Request Broker Architecture (CORBA), because of the unpredictability of pauses in Java processing for garbage collection, dynamic loading of Java class libraries, and so on. One approach to enabling Java for real-time applications is the JSR-1 real-time Java standard, which provides a modified programming model and options for storage managed without garbage collection. Recent advances in Java virtual machines have removed the garbage collection unpredictability by factoring garbage collection and other time-consuming operations into fine-grained steps.[7]

APPLICATIONS OF RESPONSIVE SYSTEMS

In this section two example applications of responsive systems are presented: one from the defense industry, the other from online gaming.

Missile defense in naval warfare

The U.S. Navy DDG 1000 destroyer is tailored for land attack and littoral dominance (i.e., dominance on the sea and the shore), and must counter threats anticipated over the next decade.[8] To that end, this 600-foot-long vessel contains multiple sensor systems, including multimode dual-band radar, multiple sonar arrays, and vertical takeoff and landing tactical unmanned aerial vehicles, which are used in conjunction with Tomahawk missiles, 155-mm advanced gun systems, and 57-mm close-in guns. Coordination of these systems in the time frames imposed by modern warfare requires significant real-time computational power that is highly predictable with extremely low latencies (microseconds to milliseconds). This capability is provided by the Total Ship Computing Environment infrastructure (TSCEi) supplied by IBM, the U.S. Navy, and its major system integrator.[9] TSCEi includes IBM BladeCenter* Systems,[10] IBM WebSphere* Real Time (WRT), an implementation of the Object Management Group's Data Distribution Service for Real-Time Systems specification, an in-memory database, and various mission applications.

WRT includes real-time implementations of Java and of Linux**.[11] The use of the real-time Java implementation, including real-time garbage collection, is due to the higher productivity and availability of Java programmers compared to C, C++, and Ada programmers. This motivation for moving to Java from C, C++, and Ada has been recognized within the defense community for some years.[12]

A possible missile-engagement situation that the DDG 1000 destroyer might encounter on a mission involves a missile attack. This scenario includes a number of phases. Data supplied by sensors indicate the approach of an object (detection); at this time it is not known whether the object represents a hostile action. Data that originate from multiple sensors (such as phased-array radar) and that relate to the same object are combined (correlation). The system determines the type of the object in question: missile, fighter jet, and so on (classification). Sensor data is further collected and analyzed in order to track the course and speed of the object (track). Data collected is used to confirm the result of the classification phase (validation). The system determines whether the object is a threat (assess). A check by a human agent is

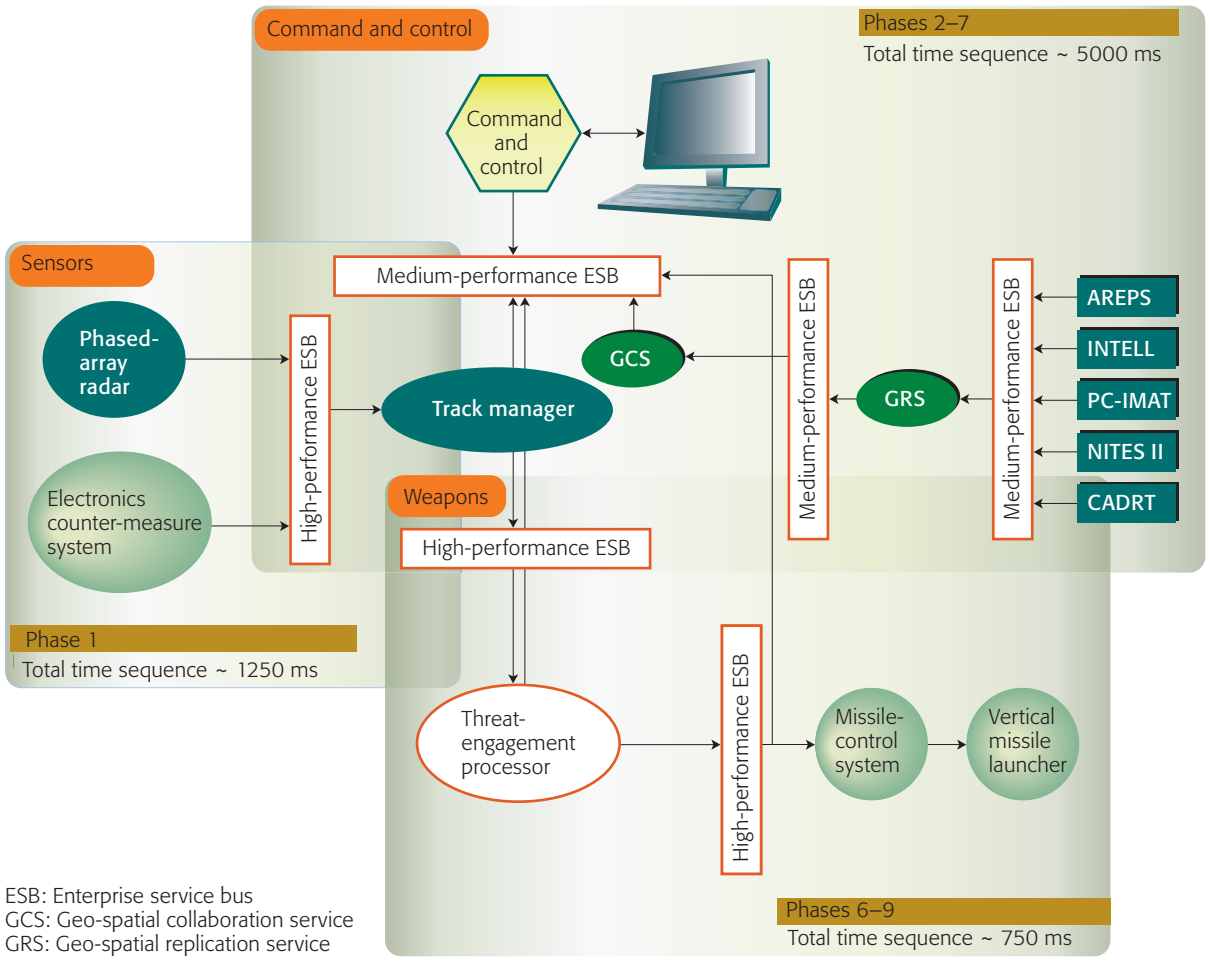

Figure 1
Process flow for the missile-engagement scenario

performed (confirm). The system determines which of several available weapons is to be used to defend against the threat (task). The system performs the defense action (execute) that is detected by the phased-array radar. The process flow corresponding to the missile encounter is shown in ***Figure 1***. Sensor data inputs, shown on the left, are processed by the command-and-control components, shown in the upper right, which then actuate weapons control, shown at the bottom. The command-and-control processing involves considerable analysis, including the Advanced Refractive Effects Prediction System (AREPS), input from intelligence sources (INTELL), the Personal-Computer Interactive Multisensor Analysis Training (PC-IMAT), the Naval Integrated Tactical Environmental System (NITES II), and the Computer-Aided Dead Reckoning Tracer (CADRT).

The process of detection and correlation may take a few hundred milliseconds, classification and tracking may consume another 500 to 1000 milliseconds. Command and control involves human intervention; thus, validation ("Is it really aimed at us?"), assess, and confirm ("Have electronic countermeasures been effective?") may consume another 5000 milliseconds of processing budget across the TSCEi infrastructure. Finally, task and execute may consume the remaining 750 milliseconds of the processing budget, for a total of 7000 milliseconds from initial detection to defensive-weapon firing. This leaves 3000 milliseconds for the defensive weapons to actuate and reach the target, so that the incoming missile is destroyed within 10 seconds.

This scenario illustrates that the extremely short time frames encountered in modern naval warfare

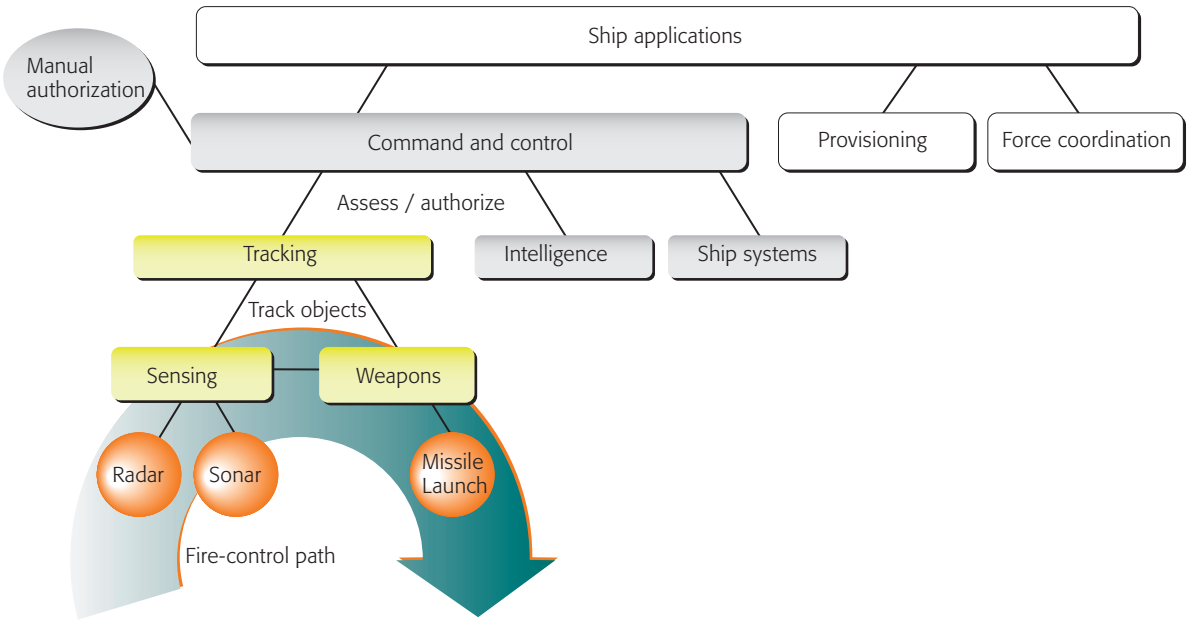

Figure 2
Naval warfare application with multiple QoS domains

prohibit manual sensing and control: predictable and extremely low-latency automation is required. The IBM TSCEi hard real-time environment was specifically designed to achieve worst-case intra-node latencies down to 1 millisecond for standard Java (with garbage collector) and down to 70 *micro*seconds for Real-Time Specification for Java (RTSJ) (which avoids the delays associated with the use of garbage collection), in both cases running on real-time Linux. In this environment, an x86 blade has been configured to support up to 2000 concurrent real-time Java threads across the four processors of the blade. Such blades may communicate via the Data-Distribution-Service-based (DDS-based) real-time middleware, guaranteeing bounded inter-node latencies. In this environment, real-time Java (as well as C/C++ and Ada legacy applications) can communicate in real time via the DDS APIs. In addition, an in-memory database is used as a real-time data repository to support the correlation and classification tasks within the required time frames: the high latencies associated with mass-storage access disqualify the use of traditional relational databases for these processing tasks.

It is important to note that many TSCEi activities do not require extreme levels of response, as depicted in a very-high-level architectural diagram shown in *Figure 2*. The individual hardware and software leaf components (in the orange circles) must provide sub-10-millisecond hard real-time response, the tracking, sensing, and weapons systems (in yellow) must provide 10- to 100-millisecond hard real-time response, the command-and-control systems (in gray) must provide 100- to 1000-millisecond soft real-time response, and the ship applications (in white) must provide enterprise-like transactional response. Although it would be possible to construct the entire application using a hard real-time style of implementation, it is often much cheaper and easier to restrict the real-time implementation style to those portions of the application requiring real-time response. This situation requires multi-domain real-time QoS, as does the online-gaming example discussed in the following section.

Online gaming

At present, the more aggressive the real-time QoS, the more complex and expensive it is to develop and maintain the responsive system. Although the papers in this issue describe advanced tools that are expected to greatly reduce the cost of developing and maintaining responsive systems, cost considerations will nevertheless guide the design toward combining a small component with aggressively

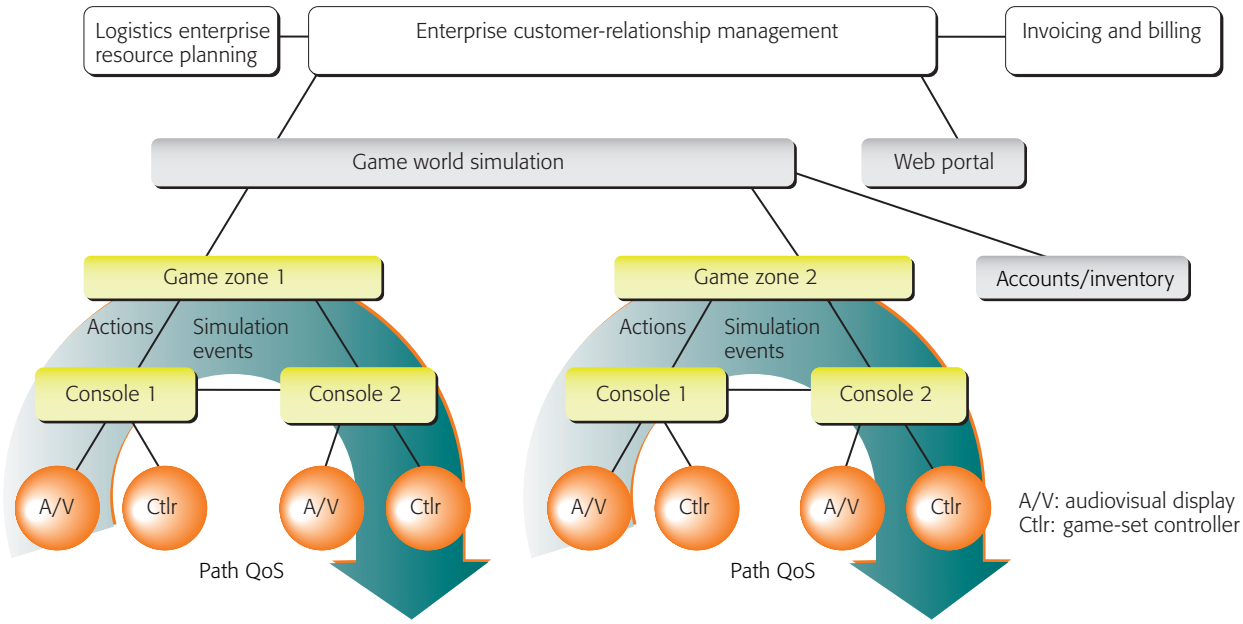

Figure 3
Online gaming architecture with multiple QoS domains

real-time performance requirements with a much larger non-real-time component, and at the same time requiring the addition of real-time performance functions to many of the existing SOA components.

Figure 3 shows a simplified online gaming architecture with multiple QoS domains. The white rectangles at the top denote traditional enterprise-like QoS, the gray rectangles underneath denote a soft real-time domain with latency requirements in the 300-ms to 1-s range, the yellow rectangles denote a harder real-time domain with latency requirements in the 100- to 300-ms range, and the orange circles represent hard real-time domains with sub-human-reflex latency requirements, which will typically be game-set controllers (Ctlr) and audiovisual display (A/V). The large arrows represent an end-to-end path QoS, again with 100- to 300-ms latency requirements.

One option would be to implement the entire responsive system shown in Figure 3 to meet the most aggressive QoS, but such an approach would be unnecessarily expensive. A much better approach would be to implement this system with multiple QoS domains, using middleware and tooling that support multiple QoS domains. In particular, such middleware must support communications between different QoS domains while still permitting each domain to meet its QoS requirements. In addition, there will be a need for tooling and middleware that support end-to-end path QoS requirements.

THIS ISSUE OF THE IBM SYSTEMS JOURNAL

This issue features papers on topics related to platforms, middleware, and development support/tooling for responsive systems.

Platforms for responsive systems

Responsive systems are only as good as their underlying platforms, and so these papers describe improvements to the Linux kernel and Java Virtual Machines (JVMs). In "Real-time Linux in real time,"[13] Hart, Stultz, and Ts'o describe challenges inherent in use of open-source software early in its development cycle and how those challenges were overcome. This paper also provides an overview of many of the changes that were made to the Linux kernel in order to attain real-time response. The advent of commodity multi-core/multi-threaded computer systems means that real-time response must be provided on parallel hardware. In "The read-copy-update mechanism for supporting real-time applications on shared-memory multiprocessor systems with Linux,"[14] Guniguntala et al. present a new synchronization mechanism that provides

deterministic access to readers while permitting full preemption and thus avoiding writer starvation. This synchronization mechanism has been used in production for well over a decade, but has only recently been adapted to real-time use. Recent Java offerings from IBM have featured a real-time garbage collector,[7] greatly expanding the real-time capabilities of the Java language, applicable to embedded systems.[15] Further work on this Java environment led to the concepts of *eventrons* and *exotasks*. Eventrons are tasks that can preempt the garbage collector, providing sub-millisecond response times while still permitting access to the garbage-collected heap.[16] Exotasks can also help meet aggressive scheduling deadlines through the use of private heaps, and this is illustrated by a real-time application involving a helicopter that is piloted by a Java application.[17]

Middleware for responsive systems

Whereas middleware with real-time support has been available for some time, implementing real-time support in SOA deployments requires enhanced capabilities, performance, and scalability. To this end, Bauer et al., in "Toward scalable real-time messaging,"[18] provide a perspective on implementing high-message-rate real-time messaging middleware for multi-core processors, while Magid et al., in "Generating real-time complex event-processing applications,"[19] present a framework that evaluates worst-case execution time for user-written rules for complex-event processing, and that also compiles these rules into a general-purpose language. Distributed real-time systems require a means to control end-to-end latency across a set of network-connected systems, a topic taken up by Astley et al. in "Pulsar: A resource-control architecture for time-critical service-oriented applications."[20] Dube et al., in "Harmony: Holistic messaging middleware for event-driven systems," address the topic of end-to-end latency in the presence of geographic distribution and nodal failure.[21]

Development support for responsive systems

Responsive systems are widely viewed as being especially difficult to design and implement, and development tools and environments are needed to ease this task. To this end, Chen et al., in "DRIVE: A tool for developing, deploying, and managing distributed sensor and actuator applications,"[22] describe an Eclipse-based tool for developing, deploying, and managing sensor/actuator applications used in the automation of business processes. To illustrate the benefits of the tool, they present an example application and show the ways in which the tool supports the modeling and development of the application logic and the multiphase deployment of the resulting application in a production environment.

Responsive applications place timeliness requirements on the underlying software, firmware, and hardware platform. In "Accounting for platform effects in the design of real-time software using model-based methods,"[23] Selic presents a conceptual framework that helps account for timeliness properties of the platform. Portability is a key concern in any software undertaking, and Sharon and Etzion, in "Event-processing network model and implementation,"[24] describe an implementation-independent notation for event-processing networks and describe alternative implementations driven by this notation. Finally, Parr and Yusuf, in "Patterns for real-world aware and real-time solutions,"[25] introduce a pattern language for real-time solutions to enable developers to better design and implement responsive applications.

THE FUTURE OF RESPONSIVE SYSTEMS

Although the past few years have seen great progress in the area of responsive systems, a considerable number of challenges remain. We decompose these challenges into three major areas: (1) real-time programming model; (2) runtime support for real-time QoS delivery, both platform and middleware; and (3) tooling support for real-time QoS specification, delivery, and management.

A programming model describes how developers interact with a system, including descriptions of the requisite APIs, protocols, and formats. To this end, we augment the IBM programming model for SOA[26] with event-processing and real-time-QoS capabilities. Although some workloads will continue to be well-supported by the traditional practice of hard-coding QoS requirements, portability needs and new classes of applications will increasingly require policy-oriented real-time QoS specifications. The Object Management Group (OMG) standards body's work on the real-time QoS specification[27] represents an important step forward in this area, and next steps include gaining experience with implementations of this specification. In addition, composition of QoS requirements is required in order to support the

flexible composition provided by SOA platforms. The event-processing domain is still relatively new, and so working groups have only recently begun forming around the definition of programming semantics.[28]

The key challenge underlying runtime support for real-time QoS delivery is the fact that any scheduling delays inserted by a lower level in the stack (e.g., firmware or operating systems [OS] kernel) cannot in general be compensated for by upper levels (e.g., JVM or Java application). This means that real-time response must be built into the system from the bottom up. For example, the JVM cannot meet a given scheduling-latency constraint unless the corresponding latencies from the OS kernel, firmware, and hardware are well within the desired JVM latency. Until such time as real-time constraints become commonplace across the bulk of the marketplace, the construction of real-time systems will require great attention to detail.

Although real-time requirements are not yet pervasive, they are becoming increasingly commonplace. Fortunately, technologies to support mainstream development and use of real-time systems are starting to emerge. Although much more work will be required before construction of real-time systems can be said to be routine, we look forward to continued progress toward that goal.

ACKNOWLEDGMENTS

We are indebted to Greg Porpora and Paul Giangarra for many invaluable discussions, including sessions on real-time architectures in the defense industry and online gaming, respectively.

CITED REFERENCES

1. P. Bruni, M. H. S. Caurim, A. Koerner, C. Law, M. Liberman, W. Schuh, E. Van Aerschot, J. Wang, and P. Wansch, "Powering SOA with IBM Data Servers," *IBM Redbook* SG24-7259-00, IBM Corporation (2006), http://publib-b.boulder.ibm.com/abstracts/sg247259.html?Open.
2. "TPC Benchmark A," Transaction Processing Performance Council (1989), http://www.tpc.org/tpca/default.asp.
3. J. Nielsen, *Designing Web Usability*, New Riders Publishing (Macmillan), Indianapolis (1999).
4. D. Wackerow, "MQSeries Primer," *IBM Redpaper*, REDP-0021-00, IBM Corporation (1999), http://www.redbooks.ibm.com/abstracts/redp0021.html?Open.
5. S. Davies, L. Cowen, C. Giddings, and H. Parker, "WebSphere Message Broker Basics," *IBM Redbook* SG24-7137-00, IBM Corporation (2005), http://www.redbooks.ibm.com/abstracts/sg247137.html?Open.
6. L. Perrochon, W. Mann, S. Kasriel, and D. C. Luckham, "Event Mining with Event Processing Networks," *Proceedings of the Third Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD-99)*, Beijing, Lecture Notes in Computer Science 1574, Springer, New York (1999), pp. 474–478.
7. D. F. Bacon, P. Cheng, and V. T. Rajan, "A Real-Time Garbage Collector with Low-Overhead and Consistent Utilization," *Proceedings of the Thirtieth ACM Symposium on Principles of Programming Languages*, New Orleans, ACM, New York (2003), pp. 285–298.
8. "Navy Designates Next-Generation Zumwalt Destroyer," U.S. Navy Program Executive Office Ships, Washington, DC (2007), http://peoships.crane.navy.mil/DDG1000/default.htm.
9. IBM Corporation (Feb. 7, 2007), *IBM and Raytheon Deliver Technology Solution for DDG 1000 Next Generation Navy Destroyers*, Press release, http://www-03.ibm.com/press/us/en/pressrelease/21033.wss.
10. D. M. Desai, T. M. Bradicich, D. Champion, W. G. Holland, and B. M. Kreuz, "BladeCenter System Overview," *IBM Journal of Research and Development* **49**, No. 6, 809–821 (2005).
11. I. Molnar, Realtime-Preempt Patch Set, Red Hat, Inc., Raleigh, NC, (2007), http://www.kernel.org/pub/linux/kernel/projects/rt/.
12. J. Child, "Java Proving Itself Worthy for Defense Apps," *COTS Journal*, July (2003), pp. 25–28, http://www.cotsjournalonline.com/pdfs/2003/07/COTS07_softside.pdf.
13. D. Hart, J. Stultz, and T. Ts'o, "Real-Time Linux in Real Time," *IBM Systems Journal* **47**, No. 2, 207–220 (2008, this issue).
14. D. Guniguntala, P. E. McKenney, J. Triplett, and J. Walpole, "The Read-Copy-Update Mechanism for Supporting Real-Time Applications on Shared-Memory Multiprocessor Systems with Linux," *IBM Systems Journal* **47**, No. 2, 221–236 (2008, this issue).
15. D. F. Bacon, P. Cheng, and V. T. Rajan, "Garbage Collection for Embedded Systems," *Proceedings of Fourth ACM International Conference On Embedded Software (EMSOFT)*, Pisa, Italy, ACM, New York (2004), pp. 125–136.
16. D. Spoonhower, J. Auerbach, D. F. Bacon, P. Cheng, and D. Grove, "Eventrons: a Safe Programming Construct for High-Frequency Hard Real-Time Applications," *Proceedings of ACM SIGPLAN 2006 Conference on Programming Languages Design and Implementation*, Ottawa, Canada, ACM, New York (2006), pp. 283–294.
17. D. F. Bacon, D. Iercan, C. Kirsch, V. T. Rajan, H. Roeck, and R. Trummer, "Java Takes Flight: Time-portable Real-time Programming with Exotasks," *Proceedings of the 2007 ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES'07)*, San Diego, CA, ACM, New York (2007), pp. 51–62.
18. D. Bauer, L. Garcés-Erice, S. Rooney, and P. Scotton, "Toward Scalable Real-Time Messaging," *IBM Systems Journal* **47**, No. 2, 237–250 (2008, this issue).

19. Y. Magid, D. Oren, D. Botzer, A. Adi, B. Shulman, E. Rabinovich, and M. Barnea, "Generating Real-Time Complex Event-Processing Applications," *IBM Systems Journal* **47**, No. 2, 251–263 (2008, this issue).
20. M. Astley, S. Bhola, M. J. Ward, K. Shagin, H. Paz, and G. Gershinsky, "Pulsar: A Resource-Control Architecture for Time-Critical Service-Oriented Applications," *IBM Systems Journal* **47**, No. 2, 265–280 (2008, this issue).
21. P. Dube, N. Halim, K. Karenos, M. Kim, Z. Liu, S. Parthasarathy, D. Pendarakis, and H. Yang, "Harmony: Holistic Messaging Middleware for Event-Driven Systems," *IBM Systems Journal* **47**, No. 2, 281–287 (2008, this issue).
22. H. Chen, P. B. Chou, N. H. Cohen, S. S. Duri, and C. W. Jung, "DRIVE: A Tool for Developing, Deploying, and Managing Distributed Sensor and Actuator Applications," *IBM Systems Journal* **47**, No. 2, 289–307 (2008, this issue).
23. B. Selic, "Accounting for Platform Effects in the Design of Real-Time Software Using Model-Based Methods," *IBM Systems Journal* **47**, No. 2, 309–320 (2008, this issue).
24. G. Sharon and O. Etzion, "Event-Processing Network Model and Implementation," *IBM Systems Journal* **47**, No. 2, 321–334 (2008, this issue).
25. F. N. Parr and L. Yusuf, "Patterns for Real-world-aware and Real-time Solutions," *IBM Systems Journal* **47**, No. 2, 335–350 (2008, this issue).
26. D. F. Ferguson and M. L. Stockton, "Service-Oriented Architecture: Programming Model and Product Architecture," *IBM Systems Journal* **44**, No. 4, 753–780 (2005).
27. Object Management Group (OMG), "A UML Profile for Modeling and Analysis of Real-Time and Embedded Systems (MARTE)," *Beta 1 Specification*, August (2007), http://www.omg.org/docs/ptc/07-08-04.pdf.
28. M. Chandy, O. Etzion, and R. Von Ammon, Editors, *Proceedings of the Dagstuhl Seminar 07191 on Event Processing*, Schloss Dagstuhl GmbH (2007), http://www.dagstuhl.de/de/programm/kalender/semhp/?semnr=2007191.

Accepted for publication December 17, 2007.
Published online May 6, 2008.

Robert F. Berry
IBM Software Group, Hursley Park, Hursley SO21 2JN, UK (brobert@uk.ibm.com). Dr. Berry joined IBM in 1987 at the Thomas J. Watson Research Center, New York, to work in the area of systems performance. In that role, he has contributed to the performance measurement and improvement of several IBM operating systems, including MVS™, VM, AIX®, and OS/2®. In 1995 his focus changed to Java and to the performance of several Java Virtual Machine implementations. He held this role until 2004, when he was named CTO for Messaging Technology. He presently leads an effort within the IBM Software Group exploring the opportunities and challenges for introducing new timeliness qualities of service into middleware. He earned his Ph.D. in computer sciences from the University of Texas at Austin in 1983. In 1999 he was elected to the IBM Academy of Technology, and was appointed an IBM Distinguished Engineer in 2000. He was elected Vice President for the IBM Academy of Technology in 2005. He is a member of the Software Group Architecture Board steering committee. He serves as the IBM Partnership Executive for Imperial College, London, where he is also a Visiting Professor and a member of several Advisory Boards.

Paul E. McKenney
IBM Linux Technology Center, 15350 SW Koll Parkway, Beaverton, OR 97006 (paulmck@linux.vnet.ibm.com). Dr. McKenney is an IBM Distinguished Engineer in the Open Systems Development department and a member of the IBM Academy of Technology. In this position he has worked on SMP, NUMA, and RCU algorithms for Linux and AIX. He is currently working on synchronization primitives within the Linux kernel for real-time multi-core/multi-threaded systems. Before joining IBM he worked on similar algorithms for DYNIX/ptx at Sequent Computer Systems, on packet-radio and Internet protocols at SRI International, and on real-time systems and business applications as a self-employed contract programmer. He received B.S. degrees in computer science and in mechanical engineering in 1981 from Oregon State University, an M.S. degree in computer science in 1988, also from Oregon State University, and a Ph.D. degree in computer science and engineering in 2004 from Oregon Health and Sciences University. He holds more than 20 patents and has published more than 50 papers.

Francis N. Parr
IBM Research Division, Thomas J. Watson Research Center, 19 Skyline Drive, Hawthorne, NY 10598 (fnparr@us.ibm.com). Dr. Parr is a Research Staff Member working on messaging, event, and real-time integration middleware and tooling. He is the recipient of IBM Research awards for architecture contributions to the built-in Java Message Server in IBM WebSphere and for work on IBM Parallel DB2®. He is also involved in scalable algorithms for financial services applications. He received a Ph.D. from Harvard University and lectured at Imperial College of Science and Technology, London University, before joining the IBM Research Division. ■