Applying Adaptive Real-time Middleware to Address Grand Challenges of COTS-based Mission-Critical Real-Time Systems

Christopher D. Gill, Fred Kuhns, David L. Levine, and Douglas C. Schmidt
{cdgill,fredk,levine,schmidt}@cs.wustl.edu
Department of Computer Science, Washington University
St. Louis, MO 63130, USA*

Bryan S. Doerr
Bryan.S.Doerr@boeing.com
The Boeing Company
St. Louis, MO 63130, USA

Richard E. Schantz and Alia K. Atlas
{schantz,akatlas}@bbn.com
BBN Technologies/GTE Internetworking
Cambridge, MA 02138, USA

Presented at the 1st International Workshop on Real-Time Mission-Critical Systems: Grand Challenge Problems, November 30, 1999; Phoenix, Arizona USA.

Abstract

Commercial-off-the-shelf (COTS) middleware addresses many design forces for developing mission-critical distributed systems, including reducing development cost and cycle-time. However, meeting additional requirements for real-time quality of service (QoS) in these systems is currently beyond the state-of-the-art in available COTS middleware solutions. In this paper, we discuss key research challenges associated with determining the policies, mechanisms, and patterns required to create a new generation of QoS-enabled COTS middleware for real-time mission-critical systems.

1 Introduction

Limitations with current practice: Due to constraints on footprint, performance, and weight/power consumption, development of mission-critical real-time systems has historically lagged far behind mainstream software development methodologies. As a result, real-time software systems are extremely expensive and time-consuming to develop, validate, optimize, deploy, maintain, and upgrade. Moreover, they are often so specialized and tightly coupled to their current configuration and operating environment that they cannot adapt readily to new market opportunities, technology innovations, or changes in run-time situational environments.

In addition to the development methodology and system lifecycle constraints mentioned above, designers of mission-critical real-time systems also have historically used relatively static methods when allocating scarce or shared resources to system components. For instance, flight-qualified avionics mission computing systems [1] establish the priorities for all resource allocation and scheduling decisions very early in the system lifecycle, *i.e.*, *well* before run-time. Static strategies have traditionally been used for mission-critical real-time applications because (1) system resources were insufficient for more computationally-intensive dynamic on-line approaches and (2) simplifying analysis and validation was essential to remain on budget and on schedule, particularly when systems were designed from scratch using low-level, proprietary tools.

Emerging trends and solutions: The next generation of mission-critical real-time systems requires an increasingly wide range of features to support various quality of service (QoS) aspects, such as bandwidth, latency, jitter, and dependability. These systems include avionics mission computing systems, manufacturing process control systems, and tactical command and control systems. In addition to requiring support for stringent QoS requirements, these systems have become *enabling technologies* for companies competing in markets where deregulation and global competition motivate the need for increased software productivity, quality, and cost-effectiveness.

For instance, next-generation avionics mission computing systems [2], such as the sensor-driven example shown in Figure 1, must collaborate with remote command and control systems, provide on-demand browsing capabilities for a human operator, and respond flexibly to unanticipated situational factors that arise in the run-time environment [3]. Moreover, these systems must perform unobtrusively, shielding hu-

*This work was supported in part by Boeing, NSF grant NCR-9628218, DARPA contract 9701516, and Nortel.

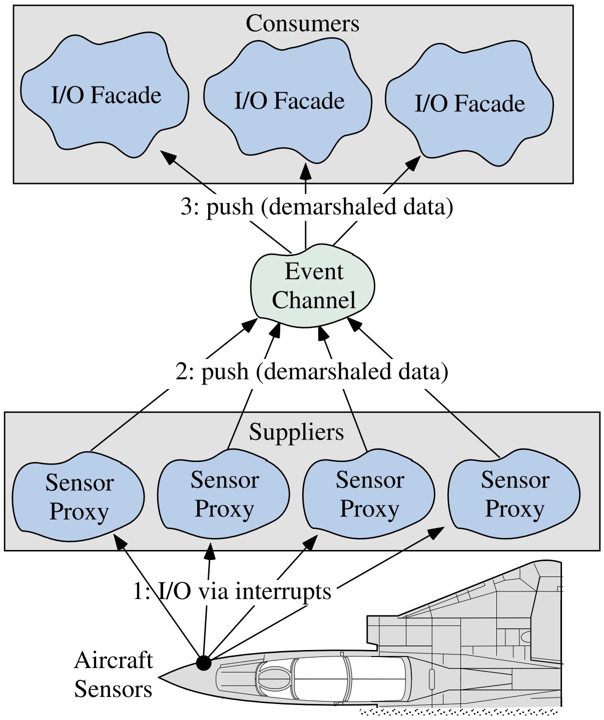

Figure 1: Sensor-driven Avionics Mission Computing Example

man operators from unnecessary details, while simultaneously communicating and responding to mission-critical information at an accelerated operational tempo.

The characteristics of next-generation real-time systems outlined above present resource requirements that can vary significantly at run-time. In turn, this increases the demands on end-to-end system resource management, thereby making it hard to simultaneously (1) create effective resource managers using traditional statically constrained allocators and schedulers and (2) achieve reasonable resource utilization. In addition, the mission-critical aspects of these systems require that they respond adequately to changing situational features in their run-time environment.

Meeting the increasing demands of next-generation real-time systems motivates the need for adaptive techniques, such as the dynamic scheduling, reconfiguration, and layered resource management techniques being explored in the context of the DARPA Quorum program [4], among others. Some of these techniques, such as the dynamic CPU scheduling strategies in TAO [1] [5], are intended to defer many of the resource allocation and scheduling decisions until run-time, while still providing QoS guarantees for critical operations [6].

Other Quorum technologies, such as DeSiDeRaTa [2] [7], the *Quality Objects*[3] (QuO) distributed object middleware [8], and the RT ARM [4] adaptive resource manager [9], focus on monitoring and adaptive management of run-time QoS. Likewise, techniques for multi-dimensional QoS management in operating systems have been developed outside the Quorum program, including RT-Mach [10], RED-Linux [11], and Scout [12].

Synopsis of a grand challenge: We believe that QoS-focused research activities outlined above are necessary. However, they are not sufficient by themselves to address a key "grand challenge" facing researchers and developers: *determining the policies, mechanisms, and patterns necessary to create commercial-off-the-shelf (COTS) middleware that can meet the QoS requirements of next-generation mission-critical real-time systems*. COTS middleware resides between applications and the underlying operating systems, protocol stacks, and hardware in complex mission-critical systems [13]. Common examples of COTS *middleware*, such as the Object Management Group's (OMG) CORBA [14] and Real-time CORBA [15], Sun's Jini [16], Java RMI [17], and EJB [18] frameworks, Microsoft's DCOM [19], and IBM's MQSeries message-oriented middleware (MOM) [20].

The goal of COTS middleware is to decrease the cycle-time and effort required to develop high-quality systems by composing applications out of flexible and modular reusable software components and services, rather than building them entirely from scratch. While it is possible *in theory* to develop these complex systems from scratch, *i.e.*, without using COTS middleware, contemporary economic and organizational constraints are making it implausible to do so *in practice*. Thus, COTS middleware plays an increasingly strategic role in software intensive, real-time mission-critical systems.

Paper organization: The remainder of this paper is structured as follows: Section 2 describes key research challenges and design forces that must be addressed to build next-generation mission-critical adaptive real-time systems; Section 3 outlines the key patterns, policies, and mechanisms necessary to develop COTS middleware that possesses effective adaptive and dynamic resource management capabilities, and shows how this approach helps resolve the challenges described in Section 2; Section 4 summarizes our current progress in (1) developing adaptive and dynamic resource management techniques for COTS middleware and (2) applying them to real-time mission-critical systems; and Section 5 presents concluding remarks.

[1] TAO was developed at Washington University.

[2] DeSiDeRaTa was developed at the University of Texas, Arlington and Ohio University.

[3] QuO was developed at BBN Technologies.

[4] The RT ARM was developed jointly by the Honeywell Technology Center, Texas A&M University, and the Georgia Institute of Technology.

2 Synopsis of Key Research Challenges and Design Forces

The following design forces characterize the key research challenges we have identified based on our work developing mission-critical adaptive real-time systems [1, 8, 2, 6, 3, 21]. These forces must be addressed by researchers to ensure system correctness, adaptability, and adequate resource utilization.

Diverse inputs: Mission-critical systems must simultaneously use diverse sources of information, such as raw sensor data, command and control directives, and operator inputs, while sustaining real-time timing behavior.

Diverse outputs: Mission-critical systems often must concurrently produce diverse outputs, such as filtered sensor data, mechanical device commands, and imagery, whose solution quality and timeliness is crucial to other systems with which they interact.

Shared resources: Mission-critical and/or time-critical operations must effectively share resources with operations that possess less stringent timing or criticality constraints.

Critical operations: Systems with hard timing constraints for mission-critical operations must insulate mission-critical operations from the resource demands of non-critical operations.

High availability: Systems must react to hardware failures and network topology changes, and return to correct real-time operation within a bounded interval after such a failure or change.

Diverse resource management goals: Systems must balance different and sometimes competing resource management goals involving different kinds of resources, *e.g.*, maximizing utilization of the CPU or sharing link bandwidth fairly between threads at the same priority.

End-to-end requirements: Many mission-critical real-time systems operate in heterogeneous environments, and must manage distributed resources to enforce QoS requirements end-to-end. For example, such systems may need to manage resource allocations consisting of several end-system CPUs and network links along a request-response path between client and server endsystems.

Programming models: Developers of real-time systems must be able to trade off the relative complexity of different programming models, which can increase or decrease system development time and effort, with the real-time system performance benefits that each programming model provides.

System configuration: Developers of real-time systems must be able to control the internal concurrency, resource management, and resource utilization configurations throughout middleware and applications, to provide the necessary level of end-to-end quality of service (QoS) to applications.

System adaptation: The system must be able to (1) reflect on situational factors as they arise dynamically in the run-time environment and (2) adapt to these factors while preserving the integrity of key mission-critical activities. Operators must be insulated from the programming model for resource management, *e.g.*, via a set of suitable abstractions for communicating operator QoS requirements and monitoring/controlling the received QoS.

Development time and cost management: The time and effort expended to develop, validate, optimize, deploy, maintain, and upgrade mission-critical systems must be managed very carefully across an entire product line [21]. Thus, to achieve an effective economy of scale, a significant portion of the cost required to develop software for a particular application or system should be amortized across the development lifecycles of other applications and systems in a product line.

3 Proposed Solution: Adaptive Real-time COTS Middleware

3.1 Overview of Real-time COTS Middleware

Although some operating systems, networks, and protocols now support real-time scheduling, they do not provide *integrated* end-to-end solutions. For instance, research on QoS for ATM networks has focused largely on policies and mechanisms for allocating network bandwidth on a virtual-circuit basis. Likewise, recent research on Internet2 topics has focused on either specific signaling and enforcement mechanisms (such as RSVP [22]) or on broadly based and global resource sharing techniques (such as Differentiated Services [23]). In addition, research on real-time operating systems [24] has focused largely on avoiding priority inversions and non-determinism in synchronization and scheduling mechanisms for multi-threaded applications.

In general, QoS research on networks and operating systems has not addressed key requirements and end-to-end usage characteristics of mission-critical real-time systems developed using COTS middleware. In particular, existing approaches have not focused on providing both a *vertically* (*i.e.*, network interface $\leftrightarrow$ application layer) and *horizontally* (*i.e.*, end-to-end) integrated solution that provides a higher-level service model, or global policy framework, to developers and end-users. Determining how to map (1) the results from earlier

QoS research on global policies and local enforcement techniques onto (2) adaptive real-time COTS middleware is an important open research issue that is crucial to solve the grand challenges of mission-critical real-time systems.

To meet these research challenges, and to resolve the key design forces described in Section 2, we believe it is necessary to devise an architectural framework that preserves the benefits of existing research areas, while simultaneously defining new protocols and interfaces that encompass the end-to-end network and host resources needed to adequately characterize operations that require the cooperation of multiple systems. One promising architectural framework that meets these requirements is based on the Real-time CORBA specification [15], which is a COTS middleware standard that supports end-to-end predictability for operations in *fixed-priority*[5] CORBA applications.

As shown in Figure 2, the Real-time CORBA specification defines standard APIs and policies that allow applications to configure and control (1) *processor resources* via thread pools, priority mechanisms, intra-process mutexes, and a global scheduling service, (2) *communication resources* via protocol properties and explicit bindings, and (3) *memory resources* via request queues and bounded thread pools.

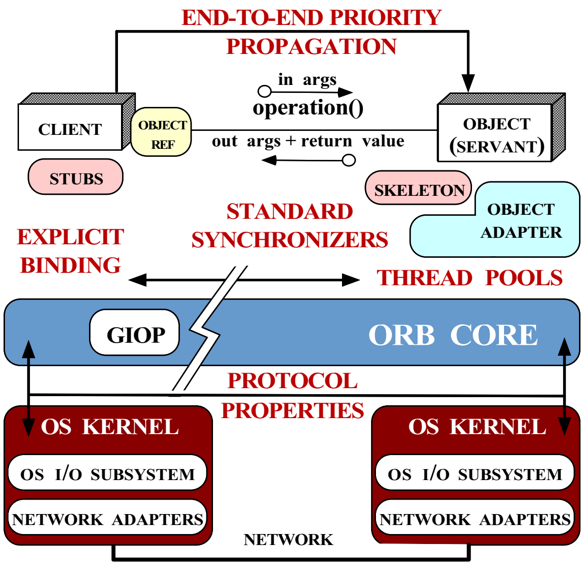

Figure 2: Features in the Real-Time CORBA Specification

The remainder of this section is structured as follows: Section 3.2 articulates steps that are necessary to resolve the key design forces described in Section 2; Section 3.3 describes steps that are beneficial, but not strictly required; Section 3.4 identifies risky or detrimental steps that should be avoided to achieve the full potential of COTS middleware for mission-critical real-time systems; finally, Section 3.5 relates the adaptive and dynamic middleware techniques presented in this paper (1) to static approaches that have been used historically to develop real-time mission-critical systems and (2) to the dynamic and adaptive techniques developed by the operating system and networking research communities.

[5] Subsequent OMG specifications are standardizing dynamic scheduling techniques, such as deadline-based [11] or value-based [25] scheduling.

3.2 Necessary Requirements

We believe that a major step toward achieving vertical and horizontal integration of QoS management capabilities into middleware for mission-critical real-time systems will occur when the level of abstraction used by developers is raised above the OS-level and network-level APIs provided by current COTS tools and run-time software. One of the key leverage points for emerging COTS middleware-based abstractions is that it is the first protocol/interface layer to encompass both the network and host resources needed to adequately characterize operations that require the cooperation of multiple systems using standard APIs and components.

The following requirements must be met to enable developers of mission-critical adaptive real-time middleware and applications to meet the key research challenges and design forces described in Section 2.

Achieve systematic reuse through COTS middleware frameworks: Given sufficient time and effort, it is possible to achieve the specific requirements of mission-critical real-time systems in an *ad hoc* manner. In practice, however, the competitive business environment in which these systems are developed places increasingly stringent constraints on time and budgets for software development. Furthermore, the increasing scarcity of qualified software professionals exacerbates the risk of failing to complete mission-critical projects, unless the scope of software development required for each project can be tightly constrained.

For these reasons, it is necessary that mission-critical real-time systems be built as much as possible from *reusable COTS middleware components*. Figure 3 illustrates the following two layers of middleware that can reside between the (1) underlying OS and protocol stacks and (2) applications and services.

- **Low-level middleware:** This layer encapsulates core OS communication and concurrency services to eliminate many tedious, error-prone, and non-portable aspects of developing and maintaining distributed applications using low-level network programming mechanisms, such as sockets. A widely-used example of low-level middleware for real-time systems is the ADAPTIVE Communication Environment (ACE) [26].

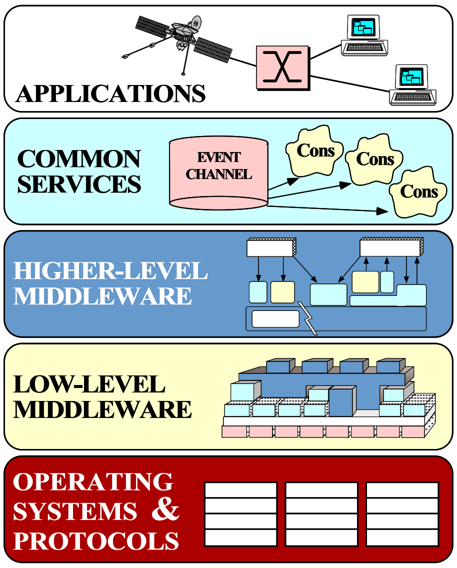

Figure 3: Layers of Middleware

- **Higher-level middleware:** This layer builds upon the lower-level middleware to automate common network programming tasks, such as parameter marshaling/demarshaling, socket and request demultiplexing, and fault detection/recovery. At the heart of higher-level real-time middleware is Real-time CORBA [14, 15].

In general, employing low-level and higher-level COTS middleware has the following benefits: (1) it shields software developers from low-level, tedious, and error-prone details, such as socket level programming [27], (2) it provides a consistent set of higher-level abstractions [6, 8] for developing adaptive mission-critical real-time systems, (3) it leverages previous design and development expertise by capturing implementations of key design patterns [28] in reusable frameworks, and (4) it amortizes software lifecycle costs across many development efforts.

Componentized services: Figure 3 also illustrates how a layer of standard service components can be supported atop the COTS middleware. These components [29] provide domain-independent capabilities that can be reused by various applications. Common services [30] include persistence [31], security [32], transactions [33], fault tolerance [34], and concurrency [35]. For example, it is essential to provide higher-level services that allow the system to effectively manage system resources. These distributed services manage key system aspects, such as global scheduling parameters, I/O bandwidth, memory allocation, or work loads.

More specifically, typical mechanisms managed by global resource services include end-to-end global thread priorities, pluggable mappings of global thread priorities into native end-system thread priorities, thread pools, and synchronization primitives, such as mutexes and semaphores. These mechanisms are important building blocks and must be both accessible through and configurable by the adaptive real-time middleware.

Embed configurable policies: Providing predictable end-to-end performance requires the collaboration of different policies and mechanisms at different locations and architectural levels in the system. This mandates a framework that supports configurable dynamic and/or adaptive resource management policies at different points along a system request-response or request-execution path to meet different and changing conditions.

For example, to minimize the resource demands of non-critical, time-bounded operations, each of several identified decision points along a request-response path can check whether a request for such an operation is still "on schedule" to meet its deadline. If an operation cannot meet its deadline, the request may be dropped and/or an exception propagated back to the originating client.

Flexibly specify QoS attributes: Middleware must provide flexible interfaces that allow applications to specify which QoS attributes are (1) considered by resource allocation policies, (2) propagated with requests, or (3) specified by an application. For example, applications may want to specify general QoS attributes, such as execution deadlines, rates, or throughput reservations. Likewise, applications should be able to use middleware APIs to specify application QoS, *e.g.*, to define deadlines for individual end-to-end requests or transactions. The middleware is also responsible for mapping these higher-level specifications onto lower-level OS mechanisms, which deal with I/O bandwidth, local thread scheduling parameters, and other lower-level QoS attributes and enforcement mechanisms.

Standardization: Standards for dynamic and adaptive resource management in COTS middleware should be based on operational systems used *in practice* or on prototypes that reflect key aspects of real-world applications. This helps ensure that the results of standardization efforts will reflect, rather than invent, realistic dynamic and adaptive resource management strategies. If a standard attempts to invent solutions to dynamic and adaptive resource management for COTS middleware prior to sufficient practical experience being gained with such systems, it increases the risk of excluding key algorithms and implementation techniques that are crucial for important use-cases.

For example, in Section 3.3, we describe notification of failure as a beneficial feature for many real-time systems, but one that is not feasible in some use-cases. Nuances such as this abound in active research areas, such as QoS management for mission-critical real-time systems. Thus, attempts to mandate specific features prior to a reasonably thorough exploration of the problem domain in practice can have unintended and undesirable consequences.

3.3 Desired Capabilities

We believe the following features are useful to developers of mission-critical adaptive real-time systems, but are perhaps not sufficiently general to be cast as necessary requirements for all mission-critical real-time systems.

Start with well-known policies: Reference implementations of well-known resource allocation policies, such as the RMS [36], EDF [36], MLF [37], and MUF [37] CPU scheduling strategies, serve as starting points for development and prototyping activities. To support these well-known policies, the middleware framework should support sufficient generality and/or flexibility in the mechanisms for resource allocation so that it is both feasible and efficient to substitute one such policy for another. For example, the request dispatching mechanism shown in Figure 4 can be configured to support

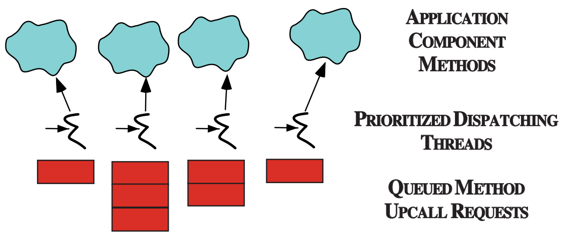

Figure 4: Request Dispatching Mechanism

RMS, EDF, MLF, or MUF simply by specifying the number and types of dispatching queues, and the priorities of the corresponding dispatching threads [6].

Identified decision points: Real-time COTS middleware must support identifiers for resource management decision points along each request-and-response path. Supporting such identifiers allows real-time aspects, such as deadlines and reservations, to be specified with respect to these individual scheduling decision points. For example, reserving CPU cycles and network bandwidth along a distributed request-response path requires that specific reservations be made on *identified* endsystems and network links.

Notification of failure: It is often desirable to support exception propagation back along the request-response path when real-time semantics fail, *e.g.*, a deadline is missed. For many real-time systems, failure notification is mandatory, particularly for critical operations. As discussed in Section 3.2, however, a system may drop a request for a non-critical time-constrained operation that will miss its deadline and thus give no value for completion. Dropping such non-critical operations conserves resources for other more productive activities, and it is reasonable to allow that the overhead for failure notification be avoided in such cases, as well. Therefore, we cast exception propagation as a desirable capability, rather than a mandatory one.

3.4 Necessary Exclusions

Based on our previous experience [1, 8, 2, 6, 3, 21] developing a wide-range of real-time applications on QoS-enabled middleware, we believe it is necessary to make the following exclusions to preserve the flexibility of developers to build correct implementations of diverse features for mission-critical adaptive real-time systems:

Attribute restrictions: We believe there should be no restrictions on which attributes, such as execution time or criticality, are (1) considered by a resource allocation policy, (2) forwarded along a request-response path, or (3) supplied by the application. While the set of attributes shown in Figure 5 has been sufficient for previous work on QoS-enabled middle-

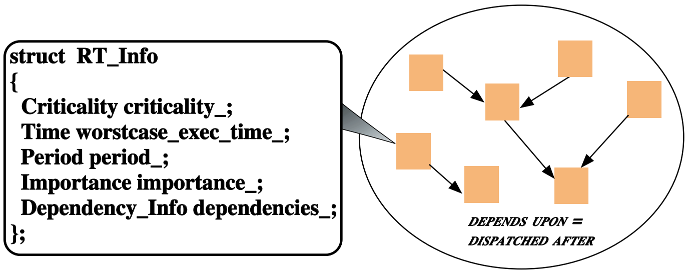

Figure 5: Operation Characteristics

ware, we anticipate that there will be use-cases where additional attributes are needed. Therefore, we do not support the notion that any one set of attributes is *necessarily* complete for all systems.

Feature deprecation: No removal of features from COTS middleware standards, such as the Real-time CORBA specification [15], unless a fundamental contradiction with mandatory dynamic or adaptive resource scheduling features is discovered.

Restrictions on non-real-time issues: It is important that there be no restrictions on the ability of resource allocation policies to address non-real-time issues such as throughput and fairness, as well as real-time issues, such as as priority preservation and deterministic timing bounds.

3.5 Relationship to Existing Techniques and Research Communities

We view the techniques proposed in this paper, such as dynamic scheduling [6], multi-resource scheduling [9], and adaptive reconfiguration [8], as necessary and appropriate extensions to the static resource allocation techniques that have been used historically. By preserving the best attributes of these approaches and extending their capabilities as efficiently as possible, we believe a new generation of mission-critical adaptive real-time systems can be realized. For example, sensor-driven systems with hard real-time processing requirements can benefit greatly from dynamic scheduling capabilities, particularly to make effective use of over-provisioned resources during non-peak loads.

Another valuable feature used in many real-time systems is statically allocated priority banding [6], which can be enforced by preemptive thread priorities. Priority banding is essential because higher priority operations can be shielded from the resource demands of lower priority operations. Hybrid static-dynamic scheduling techniques [37] offer a way to preserve the off-line scheduling guarantees for critical operations, while increasing overall system utilization.

As more real-time systems are interconnected, both with each other and with non-real-time systems, the need to support flexible and configurable scheduling capabilities [6] becomes increasingly important. We also believe that emerging standards for dynamic and adaptive resource management in real-time mission-critical systems should extend corresponding standards for static resource management. For example, standards for dynamic CPU scheduling in real-time middleware should extend the existing static CPU scheduling mechanisms of current real-time middleware specifications, so that the existing static mechanisms will interoperate with additional capabilities for dynamic scheduling.

Finally, important insights can be gleaned from the operating system and networking research communities. These communities have developed a plethora of QoS policies and mechanisms that address enforcement, allocation, and adaptation. These research activities have addressed specific issues, such as hierarchical scheduling [38], fair resource allocation [39], distributed signaling protocols [40], and admission control policies [41].

4 Progress to Date

Our progress to date in identifying key patterns and developing techniques for adaptive and dynamic resource management and applying them to real-time mission-critical systems can be classified into three main areas: (1) *adaptive resource management*, (2) *adaptive architectures*, and (3) *frameworks for monitoring and visualization*, which we describe below.

4.1 Adaptive Resource Management

Responsiveness to changing situational factors is a key requirement of many real-time mission-critical systems. To respond effectively to the new combination of situational factors, a real-time mission-critical system must often modify its operating characteristics. For instance, it may require different strategies for allocating scarce system resources, such as CPU cycles and network bandwidth, to respond effectively to the new combination of situation factors.

During the past three years, we have explored several related topics in the context of real-time embedded information systems. These topics include developing a strategized framework for static, dynamic, and hybrid static/dynamic scheduling [6] illustrated in Figure 6. In addition, we have applied this

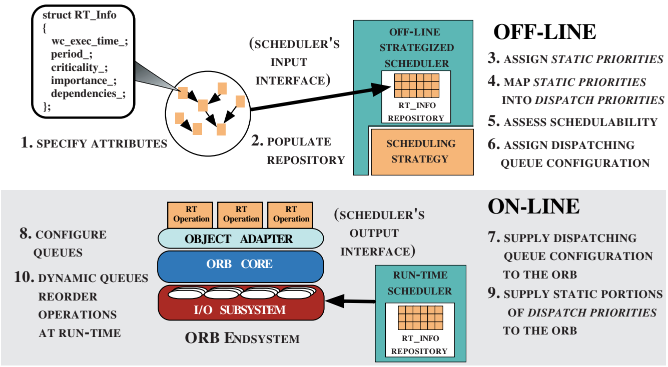

Figure 6: Strategized Scheduling Framework

framework to integrate adaptation capabilities in the application, resource manager, and operation scheduling/dispatching layers [3] of real-time, mission-critical avionics systems [2, 1].

4.2 Adaptive System Architectures

During our earlier efforts to manually integrate adaptation capabilities from different system layers, it became evident that a *meta-level* integration capability was desirable for the following reasons:

Simplified programming model: Providing a meta-level description of the various adaptive capabilities in different system layers simplifies and reifies the programming model for adaptive real-time mission-critical systems.

Application-independence: Providing a meta-level description of the system operating regions decouples the adaptive architecture from the particulars of any specific application, increasing the relevance of the adaptive system architecture across real-time mission-critical system domains.

Automated language and tool support: Providing language and tool support for these descriptions helps to automate and decouple system aspects, such as functionality, timing behavior, and fault tolerance, so that (1) new aspects can be integrated when new system requirements arise and (2) interactions between the various aspects can be managed effectively.

The work described in Section 4.1 involved integrating key portions of the RT ARM [9] and TAO [5] technologies developed under the DARPA Quorum [4] program, with a sample application from the avionics domain. We are in the process of evolving our adaptive architecture to integrate key aspect-language and resource management capabilities from the QuO [8] technologies, which were also developed under Quorum. The QuO adaptive architecture is shown in Figure 7. We envision significant applicability for this integrated adaptive architecture, including end-to-end control of distinct QoS aspects in a distributed real-time environment with high variability of situational factors.

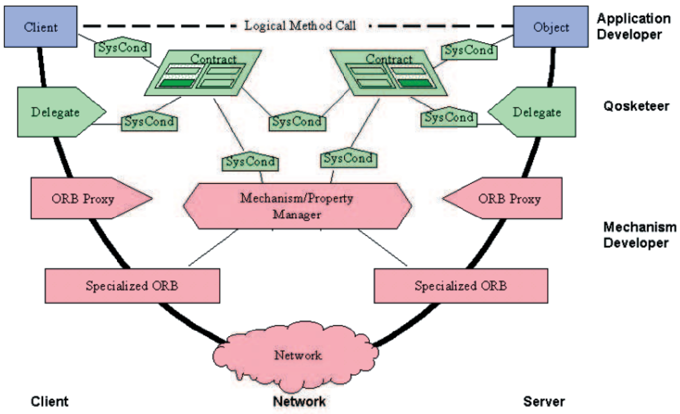

Figure 7: QuO Adaptive System Architecture

4.3 Frameworks for Monitoring and Visualization

To integrate and demonstrate the cooperation of different adaptation mechanisms in the work described in Section 4.1, it was necessary to develop mechanisms for monitoring and reporting real-time system behavior. These mechanisms were used by the resource manager layer to identify the system's current operating region, so that it could make adaptation decisions based on that region. These mechanisms were also used to verify correct real-time behavior of the adaptive system as a whole, and to assess scalability by measuring behavior under different operating conditions.

We have implemented a generic framework for visualizing distributed object computing systems, called DOVE [42]. We have extended that framework to support visualizing real-time behaviors [42], as shown in Figure 8. The extended framework use monitoring and reporting mechanisms that are similar to the work described in Section 4.1. In addition, we have investigated classes of real-time algorithms suitable for use in adaptive systems, and have implemented a demonstration of a published algorithm that represents one such class, using this visualization framework.[6]

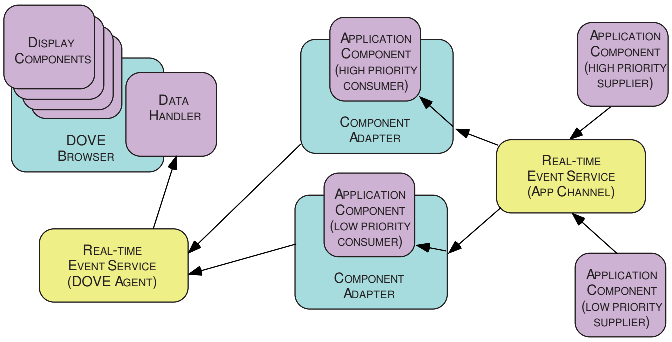

Figure 8: Extended Visualization Framework

5 Concluding Remarks

Adaptive real-time COTS middleware is a promising solution for key grand challenges facing researchers and developers of real-world real-time mission-critical systems. However, meeting the QoS requirements of next-generation systems requires more than higher-level design and programming techniques, such as encapsulation and separation of concerns, associated with conventional COTS middleware. Instead, it requires an integrated architecture, based on adaptive real-time middleware patterns, policies, and mechanisms, that can deliver end-to-end QoS support at multiple levels in distributed real-time and embedded systems.

To support an adaptive COTS middleware architecture effectively requires new dynamic and adaptive resource management techniques that extend existing static resource management techniques. By preserving the key capabilities of the static approaches, and generalizing those capabilities to include dynamic and hybrid static/dynamic capabilities, mission-critical adaptive real-time systems can be built to address the needs of broad application categories. The key is to support a multi-dimensional end-to-end QoS [10] framework that allows middleware and application developers to more easily effect, control, and coordinate the collection of lower-level mechanisms that come into play, using techniques

[6] The visualization framework demonstration is available in the `TAO/examples/Simulator` directory in the TAO ORB at URL `www.cs.wustl.edu/~schmidt/TAO.html`.

that are simple and cost-effective to use, understand, and validate. We believe the research directions outlined in this paper provide the basis for the next-generation of mission-critical, real-time systems.

6 Acknowledgements

We would like to thank Tom Ziomek for providing feedback on this paper. His comments and suggestions were valuable in improving the quality of the writing, and in connecting our ideas to existing technologies for mission-critical systems.

References

[1] T. H. Harrison, D. L. Levine, and D. C. Schmidt, "The Design and Performance of a Real-time CORBA Event Service," in *Proceedings of OOPSLA '97*, (Atlanta, GA), ACM, October 1997.

[2] D. L. Levine, C. D. Gill, and D. C. Schmidt, "Dynamic Scheduling Strategies for Avionics Mission Computing," in *Proceedings of the 17th IEEE/AIAA Digital Avionics Systems Conference (DASC)*, Nov. 1998.

[3] B. S. Doerr, T. Venturella, R. Jha, C. D. Gill, and D. C. Schmidt, "Adaptive Scheduling for Real-time, Embedded Information Systems," in *Proceedings of the 18th IEEE/AIAA Digital Avionics Systems Conference (DASC)*, Oct. 1999.

[4] DARPA, "The Quorum Program." http://www.darpa.mil/ito/research/quorum/index.html, 1999.

[5] D. C. Schmidt, D. L. Levine, and S. Mungee, "The Design and Performance of Real-Time Object Request Brokers," *Computer Communications*, vol. 21, pp. 294–324, Apr. 1998.

[6] C. D. Gill, D. L. Levine, and D. C. Schmidt, "The Design and Performance of a Real-Time CORBA Scheduling Service," *The International Journal of Time-Critical Computing Systems, special issue on Real-Time Middleware*, 2000, to appear.

[7] L. R. Welch, B. A. Shirazi, B. Ravindran, and C. Bruggeman, "DeSiDeRaTa: QoS Management Technology for Dynamic, Scalable, Dependable Real-Time Systems," in *IFACs 15th Symposium on Distributed Computer Control Systems (DCCS98)*, IFAC, 1998.

[8] J. A. Zinky, D. E. Bakken, and R. Schantz, "Architectural Support for Quality of Service for CORBA Objects," *Theory and Practice of Object Systems*, vol. 3, no. 1, 1997.

[9] J. Huang, R. Jha, W. Heimerdinger, M. Muhammad, S. Lauzac, B. Kannikeswaran, K. Schwan, W. Zhao and R. Bettati, "RT-ARM: A real-time adaptive resource management system for distributed mission-critical applications," in *Workshop on Middleware for Distributed Real-Time Systems, RTSS-97*, (San Francisco, California), IEEE, 1997.

[10] R. R. Rajkumar, C. Lee, J. P. Lehoczky, , and D. P. Siewiorek, "Practical Solutions for QoS-based Resource Allocation Problems," in *IEEE Real-Time Systems Symposium*, (Madrid, Spain), IEEE, December 1998.

[11] S. Wang, Y.-C. Wang, and K.-J. Lin, "A General Scheduling Framework for Real-Time Systems," in *IEEE Real-Time Technology and Applications Symposium*, IEEE, June 1999.

[12] D. M. A. Bavier, L. Peterson, "BERT: A Scheduler for Best Effort and Realtime Tasks," Tech. Rep. TR-602-99, Princeton University, 1999.

[13] A. Gokhale and D. C. Schmidt, "Techniques for Optimizing CORBA Middleware for Distributed Embedded Systems," in *Proceedings of INFOCOM '99*, Mar. 1999.

[14] Object Management Group, *The Common Object Request Broker: Architecture and Specification*, 2.3 ed., June 1999.

[15] Object Management Group, *Realtime CORBA Joint Revised Submission*, OMG Document orbos/99-02-12 ed., March 1999.

[16] Sun Microsystems, "Jini Connection Technology." http://www.sun.com/jini/index.html, 1999.

[17] A. Wollrath, R. Riggs, and J. Waldo, "A Distributed Object Model for the Java System," *USENIX Computing Systems*, vol. 9, November/December 1996.

[18] Anne Thomas, Patricia Seybold Group, "Enterprise JavaBeans Technology." http://java.sun.com/products/ejb/white_paper.html, Dec. 1998. Prepared for Sun Microsystems, Inc.

[19] D. Box, *Essential COM*. Addison-Wesley, Reading, MA, 1997.

[20] IBM, "MQSeries Family." http://www-4.ibm.com/software/ts/mqseries/, 1999.

[21] B. S. Doerr and D. C. Sharp, "Freeing Product Line Architectures from Execution Dependencies," in *Proceedings of the 11th Annual Software Technology Conference*, Apr. 1999.

[22] R. Braden et al, "Resource ReSerVation Protocol (RSVP) Version 1 Functional Specification." Internet Draft, May 1997. ftp://ietf.org/internet-drafts/draft-ietf-rsvp-spec-15.txt.

[23] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An architecture for differentiated services," *Network Information Center RFC 2475*, December 1998.

[24] R. Rajkumar, L. Sha, and J. P. Lehoczky, "Real-Time Synchronization Protocols for Multiprocessors," in *Proceedings of the Real-Time Systems Symposium*, (Huntsville, Alabama), December 1988.

[25] E. D. Jensen, "Eliminating the Hard/Soft Real-Time Dichotomy," *Embedded Systems Programming*, vol. 7, Oct. 1994.

[26] D. C. Schmidt, "ACE: an Object-Oriented Framework for Developing Distributed Applications," in *Proceedings of the* 6^{th} *USENIX C++ Technical Conference*, (Cambridge, Massachusetts), USENIX Association, April 1994.

[27] D. C. Schmidt, T. H. Harrison, and E. Al-Shaer, "Object-Oriented Components for High-speed Network Programming," in *Proceedings of the* 1^{st} *Conference on Object-Oriented Technologies and Systems*, (Monterey, CA), USENIX, June 1995.

[28] D. C. Schmidt and C. Cleeland, "Applying Patterns to Develop Extensible ORB Middleware," *IEEE Communications Magazine*, vol. 37, April 1999.

[29] BEA Systems, *et al.*, *CORBA Component Model Joint Revised Submission*. Object Management Group, OMG Document orbos/99-07-01 ed., July 1999.

[30] Object Management Group, *CORBAServices: Common Object Services Specification, Revised Edition*, 95-3-31 ed., Mar. 1995.

[31] Object Management Group, *Persistent State Service 2.0 Specification*, OMG Document orbos/99-07-07 ed., July 1999.

[32] Object Management Group, *Security Service Specification*, OMG Document ptc/98-12-03 ed., December 1998.

[33] Object Management Group, *Transaction Services Specification*, OMG Document formal/97-12-17 ed., December 1997.

[34] Object Management Group, *Fault Tolerance CORBA Using Entity Redundancy RFP*, OMG Document orbos/98-04-01 ed., April 1998.

[35] Object Management Group, *Concurrency Services Specification*, OMG Document formal/97-12-14 ed., December 1997.

[36] C. Liu and J. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," *JACM*, vol. 20, pp. 46–61, January 1973.

[37] D. B. Stewart and P. K. Khosla, "Real-Time Scheduling of Sensor-Based Control Systems," in *Real-Time Programming* (W. Halang and K. Ramamritham, eds.), Tarrytown, NY: Pergamon Press, 1992.

[38] Z. Deng and J. W.-S. Liu, "Scheduling Real-Time Applications in an Open Environment," in *Proceedings of the 18th IEEE Real-Time Systems Symposium*, IEEE Computer Society Press, Dec. 1997.

[39] H.-Y. Tyan and J. C. Hou, "A rate-based message scheduling paradigm," in *Fourth International Workshop on Object-Oriented, Real-Time Dependable Systems*, IEEE, January 1999.

[40] P. Newman, W. Edwards, R. Hinden, E. Hoffman, F. Ching Liaw, T. Lyon, and G. Minshall, "Ipsilon's General Switch Management Protocol Specification Version 2.0," Standards Track RFC 2297, Network Working Group, March 1998.

[41] A. Mehra, A. Indiresan, and K. G. Shin, "Structuring Communication Software for Quality-of-Service Guarantees," *IEEE Transactions on Software Engineering*, vol. 23, pp. 616–634, Oct. 1997.

[42] C. D. Gill, D. L. Levine, C. O'Ryan, and D. C. Schmidt, "Distributed Object Visualization for Sensor-Driven Systems," in *Proceedings of the 18th IEEE/AIAA Digital Avionics Systems Conference (DASC)*, Oct. 1999.