Applying Adaptive Middleware to Manage End-to-End QoS for Next-generation Distributed Applications *

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

Douglas C. Schmidt
schmidt@uci.edu
Department of Electrical and Computer Engineering
University of California, Irvine
Irvine, CA 92697, USA

Joseph P. Loyall and Richard E. Schantz
{jloyall,schantz}@bbn.com
BBN Technologies/GTE Internetworking
Cambridge, MA 02138, USA

This paper has been submitted to the Special Issue of Computer Communications on QoS-Sensitive Network Applications and Systems, edited by Klara Nahrstedt and Tarek Abdelzaher.

Abstract

Delivering end-to-end quality of service (QoS) for diverse classes of distributed applications remains a significant R&D challenge. While individual technologies based on prior research have touched upon these QoS delivery problems for specific domains or usage patterns, these isolated achievements have yielded only a fraction of the potential benefit for the broad domain of QoS-enabled distributed applications. We present our coordinated middleware-based strategy for broadening delivery of, and simplifying from the user's perspective, end-to-end QoS to a wider range of next-generation QoS-enabled distributed applications.

This paper makes the following contributions to research on end-to-end QoS. First, we describe an architecture for integrating and coordinating QoS technologies (1) at all levels of the system, (2) on all time scales of system development, deployment, and operation, and (3) across all system resources. Second, we describe results from several projects implementing particular segments of this overall architecture. We analyze these results and summarize how our work can be applied more broadly to future research on middleware for next-generation QoS-enabled applications.

*This work was supported in part by Boeing, BBN, DARPA contract 9701516, and DARPA Quorum program contract F#0602-98-C-0187 monitored by Rome Air Force Laboratory.

1 Introduction

Motivation: Many domains, such as aerospace, manufacturing, and health care, rely heavily on predictable computing and networking services to perform their respective missions. Increasingly, applications in these domains are needing to perform more demanding functions over highly networked environments, which in turn places more stringent requirements on the underlying computing and networking systems. In particular, next-generation distributed applications are requiring a broad range of features, such as service guarantees and adaptive resource management, to support a widening range of quality-of-service (QoS) aspects, such as predictable performance, secure operation, dependability, and fault tolerance [1, 2].

Limitations with current techniques: Due to deregulation, global competition, and budget constraints, even systems with stringent QoS demands are increasingly required to use commercial-off-the-shelf (COTS) hardware and software components. Although a variety of research and commercial operating systems, networks, and protocols now support some QoS management features, *integrated* end-to-end solutions are not yet available. For instance, research on QoS for ATM networks has focused largely on policies and mechanisms for allocating network bandwidth on a virtual-circuit basis. Similarly, recent research on Internet2 topics has focused on either specific signaling and enforcement mechanisms, such as RSVP [3], or on broadly based global resource sharing techniques, such as Differentiated Services [4]. In addition, research on real-time operating systems [5] 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 some key requirements and end-to-end usage characteristics of mission-critical real-time systems, especially on COTS platforms. In particular, existing approaches have not focused on providing both a *vertically* (*i.e.*, network interface ↔ 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 the results from earlier QoS research on global policies and local enforcement techniques onto a more suitable system architecture is an important open research issue that is crucial to solve the challenges of next-generation QoS-enabled distributed applications.

Solution approach → Adaptive QoS-enabled COTS middleware: To meet these research challenges, we believe it is necessary to devise an architectural framework that (1) preserves and extends the benefits of existing research areas, while (2) simultaneously defining new *middleware* services, protocols, and finterface that. This framework must provide adaptivity encompassing the end-to-end resources needed to address QoS requirements of next-generation applications that involve cooperation of multiple systems.

One promising architectural framework that meets these requirements is our TAO [6, 7] implementation of the Real-time CORBA specification [8]. Real-time CORBA is a COTS middleware standard that supports end-to-end predictability for operations in *fixed-priority*[1] CORBA applications. As shown in Figure 1, the Real-time CORBA specification defines standard APIs and policies that improve an application's ability 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.

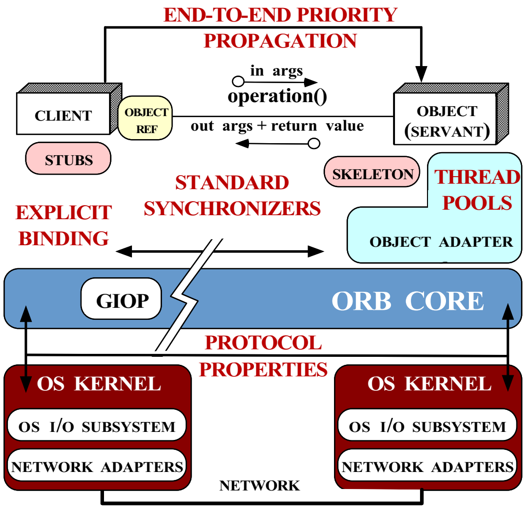

Figure 1: TAO Support for the Real-Time CORBA Specification

TAO is an open-source[2] CORBA-compliant COTS ORB designed to support applications with stringent quality of service (QoS) requirements. The TAO real-time ORB provides a rich set of middleware mechanisms for representing and enforcing real-time requirements in applications. Directly programming TAO's lower-level real-time mechanisms to achieve specific end-to-end quality of service (QoS) goals can be excessively tedious and error-prone, however, particularly for large-scale next-generation QoS-enabled distributed applications. Therefore, higher-level middleware capabilities for end-to-end QoS specification and control are needed.

To meet these needs, we have developed a complementary architectural framework called *Quality Objects (QuO)* [11, 12, 13]. QuO offers the following two capabilities for higher level specification and control of TAO's real-time CORBA middleware mechanisms:

1. QuO provides additional mechanisms for middleware adaptation that complement and improve the application control of lower-level real-time capabilities of ORB middleware, as well as the underlying operating systems and networks.

2. QuO allows developers to specify higher-level aspects of real-time requirements, such as the type of real-time required (*e.g.*, periodic or end-to-end), the relative priority of events, and the tradeoffs between real-time and other QoS requirements. It then maps these higher-level specifications into QuO and TAO mechanisms that implement, measure, and control them.

As shown in Figure 2, QuO defines interfaces that enable

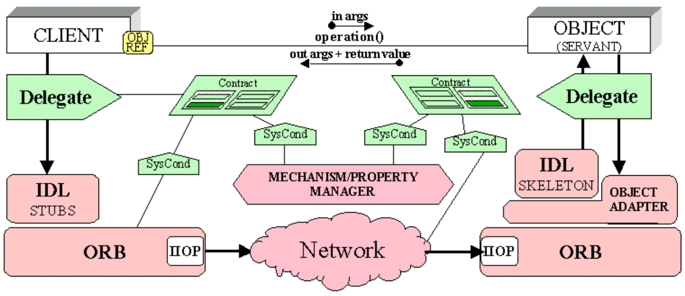

Figure 2: The QuO Distributed Object Computing Model

[1] Subsequent OMG specifications are standardizing dynamic scheduling techniques, such as deadline-based [9] or value-based [10] scheduling.

[2] The source code and documentation for TAO can be downloaded from `www.cs.wustl.edu/~schmidt/TAO.html`.

CORBA applications to *specify* QoS aspects of concern, *control* resources and mechanisms that provide QoS, *measure* the QoS provided by the system, and *adapt* to changing levels of QoS in the system. To do this, we introduce the middleware abstractions of *contracts* to organize the intended behavior into operating regions, *system condition* objects to effect measurement, and *delegates* to coordinate changing behavior underneath the client/server interactions.

The adaptive specification, control, and measurement capabilities of QuO are further enhanced when integrated with TAO's capabilities for resource configuration and management. QuO's higher level QoS *policies* are *enforced* using TAO's lower level mechanisms. By combining these complementary middleware layer frameworks, as shown in Figure 3, we are taking a major step forward to aligning (1) adaptively controlled behavior with (2) a more predictable operating environment that is oriented toward the needs of next-generation QoS-enabled systems.

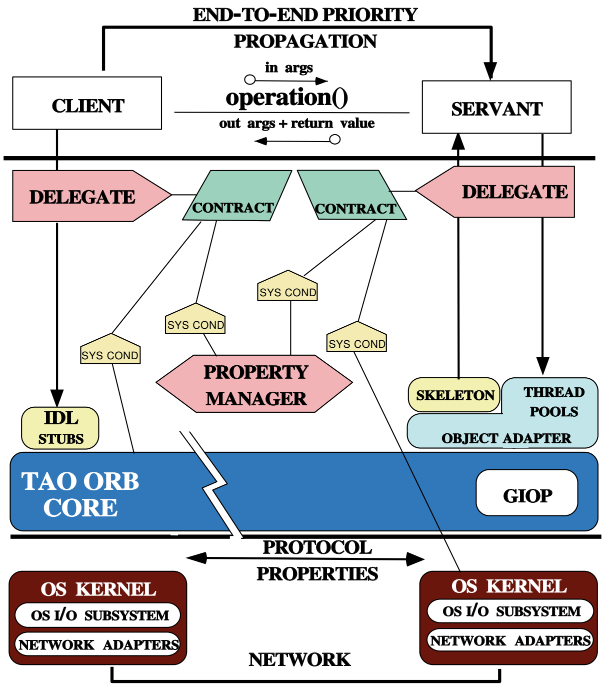

Figure 3: Integrated TAO+QuO Middleware Framework

From the bottom up, we are developing and using mechanisms to enhance execution predictability and control resource management decisions across system boundaries to meet end-to-end requirements. From the top down, we are providing advanced application-oriented QoS interfaces that adapt to changing conditions and affect resource management decisions at lower levels of middleware, OS, and network infrastructure. In the current phase of our joint DARPA Quorum integration project [2], we are focusing on controlling the real time behavior aspect of delivered QoS. Work is simultaneously ongoing to control and integrate other QoS aspects, such as dependability and security, as well as advanced software engineering concepts and tools for controlling the intended behavior of next-generation QoS-enabled applications.

Paper organization: The remainder of this paper is structured as follows: Section 2 describes properties of next-generation distributed applications that illustrate and motivate the key research challenges and design forces addressed by our QoS research; Section 3 describes our integrated TAO and QuO middleware strategy for delivering end-to-end QoS adaptively and presents quantitative and qualitative results gleaned from applying TAO and QuO to several mission-critical real-time distributed applications; Section 4 compares our efforts to related work on end-to-end QoS; and Section 5 presents concluding remarks and summarizes our directions for research on middleware for next-generation QoS-enabled applications.

2 Synopsis of Key Research Challenges and Design Forces

Development methodologies for many types of distributed applications, particularly those with stringent real-time requirements, have historically lagged behind the state of the art due to the constraints on footprint, performance, and weight/power consumption. As a result, such systems are 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 real-time applications have historically used relatively static methods to allocate scarce or shared resources to system components. For instance, flight-qualified avionics mission computing systems [14] 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.

Unfortunately, the static methodologies and techniques outlined above are too inflexible to support the requirements of next-generation QoS-enabled distributed applications. The remainder of this section describes requirements of several representative next-generation QoS-enabled applications and distills the key research challenges and design forces that are being addressed by our middleware-based QoS research to support these requirements.

2.1 Key Features of Next-generation Applications

One of the most demanding next-generation QoS-enabled distributed applications is *tele-immersion* [15], which combines tele-conferencing, tele-presence, and virtual reality. Tele-immersion places stringent demands at all levels along the end-to-end path for distributed applications. It requires real-time, predictable behavior from *endsystems* in order to (1) interact with the physical world within specific delay bounds and (2) present images or other stimuli in real-time to users [15]. Likewise, users may be distributed across intranets or the Internet thus requiring predictable performance from the *network* to provide low-latency and high-bandwidth to applications end-to-end [16].

Applying tele-immersion to health care: Intensive care medicine is a domain where tele-immersion can provide significant benefits. For instance, teams of medical personnel must make critical decisions, often at an accelerated tempo, based on information emerging at a range of time scales and from a variety of sources. Consultations with remote experts, modeling of physiological processes, and integration of both existing and emerging information often must be performed while in close proximity to the patient, as illustrated in Figure 4. In this context, it is essential that the computing and networking technologies perform and adapt in real-time to the changing situational requirements, while still maintaining QoS guarantees.

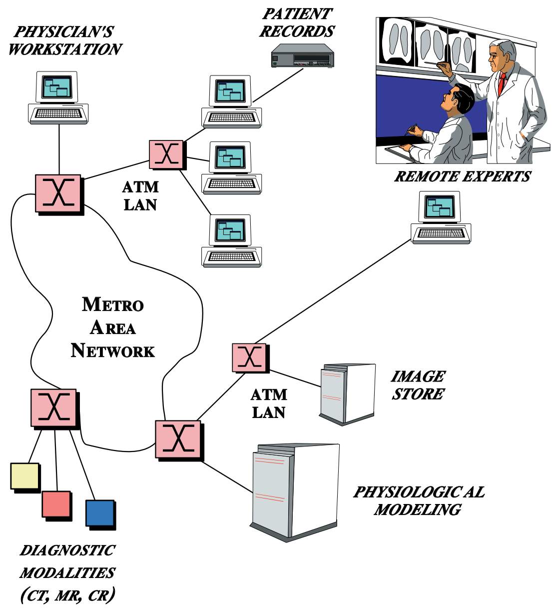

Figure 4: Real-Time Medical Informatics Example

Applying tele-immersion to aerospace: The aerospace domain is tele-immersion applications. In the battle zone of the future, a distributed web of sensors, weapons, and decision-makers must interact rapidly in real-time to gain and preserve military advantage. The battle environment will be changing constantly, requiring the system to adapt both globally and locally. For instance, multiple unmanned combat air vehicles (UCAVs) can provide surveillance, weapons delivery, and battle damage assessment capabilities both on tactical and strategic scales.

With tele-immersion, immediate remote interaction with the physical environment can help maximize effectiveness at all levels of the system. For example, a group of UCAVs can share sensor data, post-process data products, and remote operator requests. Next-generation avionics mission computing systems [17], such as the sensor-driven example shown in Figure 5, 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 [18]. Moreover, these systems must perform unobtrusively, shielding human operators from unnecessary details, while simultaneously communicating, highlighting, and responding to mission-critical information in real-time.

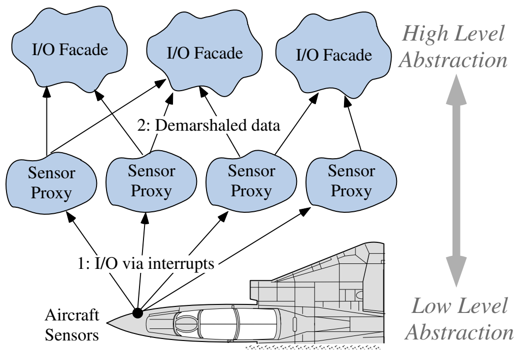

Figure 5: Sensor-driven Avionics Mission Computing Example

The next-generation applications outlined above will require a range of QoS support from middleware, endsystems, and networks. The end-to-end QoS received by the applications will translate directly into users' perceived worth of the new applications and related services. For example, if a medical video conference application routinely delivers packets late, it will have a relatively low value to its users. Thus, by providing real-time access to emerging information and real-time actuation of responses, QoS-enabled systems can provide (1) improved situational awareness, (2) reduced decision-action times, and (3) greater overall responsiveness to emerging situations.

2.2 Synopsis of QoS Requirements for Next-generation Applications

The characteristics of the next-generation systems outlined in Section 2.1 present QoS requirements that can vary significantly at run-time. In turn, this increases the demands on end-to-end system resource management, which makes 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 adaptively to changing situational features in their run-time environment.

Key features of these next-generation systems, such as interaction with the real world, produce stringent requirements that serve to distill the key research challenges and design forces that must be addressed by QoS research to support these applications. The following design forces characterize the key research challenges we have identified based on our R&D efforts [14, 1, 17, 19, 18, 20] developing next-generation avionics mission computing systems. These forces must be addressed by researchers to ensure system correctness, performance, adaptability, and adequate resource utilization.

Diverse inputs: Many next-generation distributed applications 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: Next-generation distributed applications often must concurrently produce diverse outputs, such as filtered sensor data, mechanical device commands, and imagery, whose resolution quality and timeliness is crucial to other systems with which they interact.

Critical operations: QoS management for next-generation distributed applications with hard timing constraints for application-critical operations must insulate critical operations from the resource demands of non-critical operations.

End-to-end requirements: Many next-generation distributed applications may 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 reservations and allocations involving several end-system CPUs and network links along a request-response path between client and server endsystems.

System configuration: Developers and managers of next-generation distributed applications must be able to control the internal concurrency, resource management, and resource utilization configurations throughout networks, endsystems, middleware and applications, to provide the necessary level of end-to-end QoS to applications.

System adaptation: Next-generation distributed infrastructure frameworks and applications 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.

The distilled requirements of next-generation QoS-enabled distributed applications outlined above motivate solutions that (1) offer deterministic real-time performance end-to-end, (2) protect resources needed by application-critical operations, (3) promote adaptation to a rapidly evolving environment, and (4) offer flexible configuration and control of key mechanisms for resource management. In Section 3, we present our approach to addressing these requirements, based on adaptive QoS-enabled middleware.

3 Solution Approach: Adaptive QoS-enabled Middleware

This section presents our approach to integrating the individual capabilities of existing QoS technologies to create a unified adaptive middleware solution. Our approach leverages properties of deterministic end-to-end performance, combined with configurable and adaptive QoS management capabilities, to meet the requirements of next-generation QoS-enabled distributed applications described in Section 2.

Our work focuses on supplying additional coordination and control capabilities across diverse lower-level QoS mechanisms to provide end-to-end QoS to a broad range of advanced QoS-enabled distributed applications. 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 has focused on *adaptive QoS-enabled middleware architectures*, which we describe below in Section 3.1. Section 3.2 then presents quantitative and qualitative results derived from applying our adaptive middleware to several mission-critical real-time distributed applications.

3.1 Adaptive System Architectures

During our earlier efforts to integrate adaptation capabilities from different low-level system layers and components manually, it became evident that a higher-level, highly automated integration capability was desirable for the following reasons:

Simplified programming model: Providing a higher-level description of the various adaptive capabilities in different system layers helps to simplify and reify the programming model for adaptive real-time mission-critical systems.

Application-independence: Providing a higher-level description of system operating regions decouples the adaptive architecture from the particulars of any specific application, thereby 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.

To provide these capabilities, we have developed an architectural framework that (1) preserves and extends the benefits of individual QoS research contributions while (2) simultaneously defining new middleware services, protocols, and interfaces that provide adaptivity encompassing end-to-end resources needed to address QoS requirements of next-generation applications involving cooperation of multiple systems. This architectural framework is based on *Quality Objects (QuO)* and *The ACE ORB* (TAO) [7] technologies developed under the DARPA Quorum object integration [2] program. Below, we summarize how QuO and TAO help provide an adaptive architecture for QoS-enabled applications.

3.1.1 Overview of QuO

QuO is a middleware framework designed to develop distributed applications that can specify (1) their QoS requirements, (2) the system elements that must be monitored and controlled to measure and provide QoS, and (3) the behavior for adapting to QoS variations that occur at run-time. By providing these features, QuO opens up distributed object implementations [21] to control an application's functional aspects and implementation strategies that are encapsulated within its functional interfaces.

The functional path of QuO illustrated in Figure 2 is a superset of the functional path of CORBA. The components provided by QuO to support the above operations are defined below.

Contracts: The operating regions and service requirements of the application are encoded in *contracts*, which describe the possible states the system might be in, as well as the actions to perform when the state changes.

Delegates: QuO inserts *delegates* into the CORBA functional path. Delegates project the same interfaces as the stub (client-side delegate) and the skeleton (server-side delegate), but support adaptive behavior upon method call and return. When a method call or return is made, the delegate checks the system state, as recorded by a set of contracts, and selects a behavior based upon it.

Contracts and delegates support two means for triggering manager-level, middleware-level, and application-level adaptation. The delegate triggers *in-band* adaptation by making choices upon method calls and returns. The contract triggers *out-of-band* adaptation when region transitions occur which can be caused by changes in observed system condition objects.

System Condition Objects: These objects provide uniform interfaces to multiple levels of system resources, mechanisms, and managers to translate between application-level concepts, such as operating modes, to resource and mechanism-level concepts, such as scheduling methods and real-time attributes. System condition objects are used to measure the states of system resources, mechanisms, and managers that are relevant to contracts in the overall system. In addition, they can pass information to interfaces that control the levels of desired services.

Higher-level system condition objects can interface to other, lower-level system condition objects, forming a tree of system condition objects that translate mechanism data into application data. System condition objects can be either *observed* or *non-observed*. Changes in the values measured by observed system conditions trigger contract evaluation, possibly resulting in region transitions and triggering adaptive behavior.

Observed system condition objects are suitable for measuring conditions that either change infrequently or for whom a measured change can indicate an event of notice to the application or system. Non-observed system condition objects represent the current value of whatever condition they are measuring, but do not trigger an event whenever the value changes. Instead, they provide the value upon demand, whenever the contract needs it, *i.e.*, whenever the contract is evaluated due to a method call or return or due to an event from an observed system condition object.

Instrumentation Probes: QuO provides a library of *instrumentation probes* that can be inserted throughout the remote method invocation path. These probes can be used by the QuO infrastructure to gather performance statistics and validation information unobtrusively. To accomplish this, the QuO delegate adds a data structure to each method call and return. This structure can be populated or read by any or all the instrumentation probes along the method call/return path.

Quality Description Languages (QDLs) and Code Generators: QuO provides a suite of QDLs, which are similar to CORBA's Interface Description Language (IDL), and *code generators*, which are similar to the stub and skeleton generators of CORBA IDL compilers. QDLs and code generators describe and automatically output, respectively, the components of QuO applications [11, 12, 13]. QuO currently provides a contract description language (CDL); a structure description language (SDL) to specify adaptive behavior and adaptation strategies; and a connector setup language (CSL) to specify the components of a QuO application and how they are instantiated, connected, and initialized.

QuO Runtime Kernel and GUI Monitor: QuO provides a *runtime kernel* that coordinates contract evaluation and provides other runtime QuO services [22]. These services include initializing contracts and system conditions, binding them to each other and to delegates, triggering contract evaluation, and triggering adaptive behavior. In addition, the QuO kernel provides a graphical user interface (GUI) that enables monitoring applications to observe the QuO middleware in action. The GUI displays contracts and regions and indicates the current active region and the previously active regions. It also displays the system condition objects in the system and their values, indicating when region transitions occur and the adaptive behavior triggered by the transition. Finally, it displays statistics showing how much time applications have spent in each contract region.

QuO Gateway: QuO provides a general object gateway component, illustrated in Figure 6, which allows low-level communication mechanisms and special-purpose to be *plugged into* an application [23]. The QuO gateway resides between the client and server ORBs. It is a mediator [24] that intercepts IIOP messages sent from the client-side ORB and delivers IIOP messages to the server-side ORB (on the message return the roles are reversed). On the way, the gateway translates the IIOP messages into a custom transport protocol, such as group multicast in a replicated, dependable system. The QuO gateway is implemented using TAO's pluggable protocol feature [25].

The gateway also provides an API that allows adaptive behavior or processing control to be configured below the ORB layer. For example, the gateway can select between alternate transport mechanisms based on low-level message filtering or shaping, as well as the overall system's state and condition objects. Likewise, the gateway can be used to integrate security measures, such as authenticating the sender and verifying access rights to the destination object.

Potential applications of this integrated adaptive architecture include end-to-end control of distinct QoS aspects in a distributed real-time environment with high variability of situational factors.

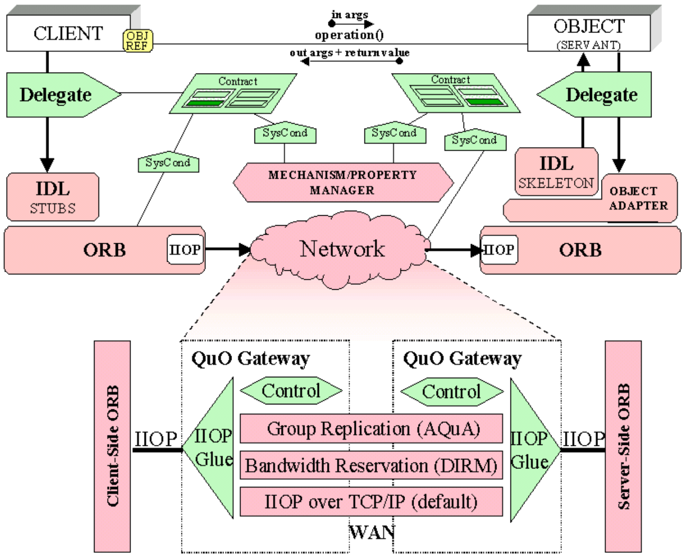

Figure 6: The QuO gateway

3.1.2 Overview of TAO

TAO is a high-performance, real-time ORB endsystem targeted for applications with deterministic and statistical QoS requirements, as well as best-effort requirements. The TAO ORB endsystem contains the network interface, OS, communication protocol, and CORBA-compliant middleware components and services shown in Figure 7.

TAO supports the standard OMG CORBA reference model [26] and Real-time CORBA specification [8], with enhancements designed to ensure efficient, predictable, and scalable QoS behavior for high-performance and real-time applications. Below, we outline the features of TAO's components shown in Figure 7.

Optimized IDL Stubs and Skeletons: IDL stubs and skeletons perform marshaling and demarshaling of application operation parameters, respectively. TAO's IDL compiler generates stubs/skeletons that can selectively use highly optimized compiled and/or interpretive (de)marshaling [27]. This flexibility allows application developers to selectively trade off time and space, which is crucial for high-performance, real-time, and/or embedded distributed systems.

Real-time Object Adapter: An Object Adapter associates servants with the ORB and demultiplexes incoming requests to servants. TAO's real-time Object Adapter [28] uses perfect hashing [29] and active demultiplexing [28] optimizations to dispatch servant operations in constant $O(1)$ time, regardless

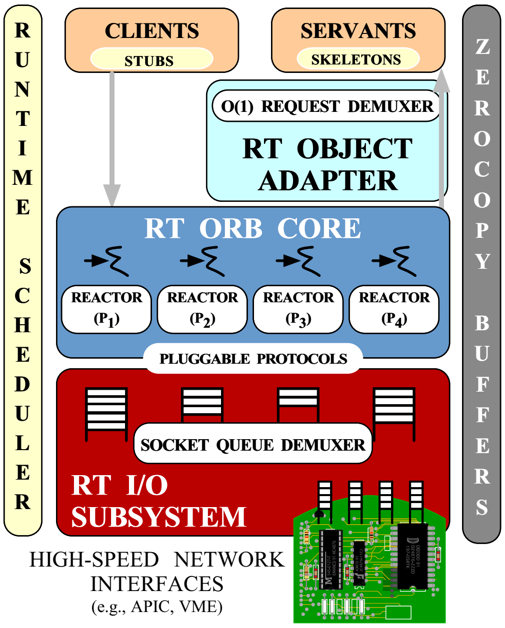

Figure 7: Components in the TAO Real-time ORB Endsystem

of the number of active connections, servants, and operations defined in IDL interfaces.

Run-time Scheduler: TAO's run-time scheduler [8] maps application QoS requirements, such as bounding end-to-end latency and meeting periodic scheduling deadlines, to ORB endsystem/network resources, such as CPU, memory, network connections, and storage devices. TAO's run-time scheduler supports both static [7] and dynamic [19] real-time scheduling strategies.

Real-time ORB Core: An ORB Core delivers client requests to the Object Adapter and returns responses (if any) to clients. TAO's real-time ORB Core [30] uses a multi-threaded, preemptive, priority-based connection and concurrency architecture [27] to provide an efficient and predictable CORBA protocol engine. TAO's ORB Core allows customized protocols to be plugged into the ORB without affecting the standard CORBA application programming model.

Real-time I/O subsystem: TAO's real-time I/O (RIO) subsystem [31] extends support for CORBA into the OS. RIO assigns priorities to real-time I/O threads so that the schedulability of application components and ORB endsystem resources can be enforced. When integrated with advanced hardware, such as the high-speed network interfaces described below, RIO can (1) perform early demultiplexing of I/O events onto prioritized kernel threads to avoid thread-based priority inversion and (2) maintain distinct priority streams to avoid packet-based priority inversion. TAO also runs efficiently and relatively predictably on conventional I/O subsystems that lack advanced QoS features.

High-speed network interface: At the core of TAO's I/O subsystem is a "daisy-chained" network interface consisting of one or more ATM Port Interconnect Controller (APIC) chips [32]. The APIC is designed to sustain an aggregate bi-directional data rate of 2.4 Gbps using zero-copy buffering optimization to avoid data copying across endsystem layers. In addition, TAO runs on conventional real-time interconnects, such as VME backplanes and multi-processor shared memory environments, as well as Internet protocols like TCP/IP.

TAO internals: TAO is developed using lower-level middleware called ACE [33], which implements core concurrency and distribution patterns [34] for communication software. ACE provides reusable C++ wrapper facades and framework components that support the QoS requirements of high-performance, real-time applications and higher-level middleware like TAO. ACE and TAO run on a wide range of OS platforms, including Win32, most versions of UNIX, and real-time operating systems like Sun/Chorus ClassiX, LynxOS, and VxWorks.

3.2 Adaptive System Architecture Implementation and Performance

Our recent research has focused on two principal activities. First, we have quantified the performance of adaptation on small time scales via dynamic scheduling in the TAO Real-Time Event Service when integrated with an adaptive [18] avionics mission computing application, under varying conditions of CPU load. Second, we have demonstrated the ability of the QuO middleware to guide adaptation to changes in system conditions, by adjusting both the rate of event generation and the priorities of events. Below, we summarize the quantitative and qualitative results gleaned from both these research activities.

3.2.1 Avionics Mission Computing Application Integration

Benchmark overview: The focus of the benchmarks described below is to quantify the benefits and costs of scheduling systems using hybrid static/dynamic approaches, when compared to statically scheduled systems. Our hypothesis is that hybrid approaches, though they can incur additional run-time overhead, will prove to be more flexible, both in terms of application development ease and overall computational throughput.

Ease of application development is facilitated by two adaptive properties of hybrid static/dynamic scheduling: (1) when load exceeds the schedulable bound, non-critical operations are dropped, whereas critical operations are scheduled, and (2) dynamic scheduling supports selectively dropping non-critical operations that will miss deadlines, while preserving non-critical operations that might be schedulable later. Encapsulating fine-grain adaptive control over operation dispatching in the middleware layers relieves developers of tedious, error-prone, and often redundant tasks related to developing this aspect of their applications.

Increased computational throughput is achieved through greater processor utilization compared to static systems, which generally require under-utilization of the CPU to be schedulable. Here too, hybrid static/dynamic scheduling provides fine-grain adaptive control over operation dispatching so that more operations can be scheduled to increase CPU utilization. Moreover, dropping operation dispatch requests that will not meet their QoS requirements can improve the amount of useful computation that is performed.

Below, we report the results of benchmarks that quantify key aspects of our hypothesis outlined above. As shown below, computational overhead is a primary metric because scheduling operations are run frequently with respect to application execution frequency. Thus, overly burdensome algorithms or algorithm implementations that scale poorly as application size grows will be undesirable in most real-time applications.

Benchmark configuration: Our experiment used a complete real-time embedded information systems application, with roughly 70 distinct operations. The application ran using the TAO ORB [7], the TAO Scheduling Service [19], and the TAO Real-Time Event Service [14], configured for various scheduling strategies. We conducted measurements on four key areas of resource control overhead: *dispatching overhead*, *operation execution times*, *operation cancellation*, and *protecting critical operations*. The analysis below features a comparison of two publically available scheduling algorithms, Maximum Urgency First (MUF) [35] and Rate Monotonic Scheduling (RMS) [36]. Measurements were conducted on 200 MHz Power PC Single Board Computers running the VxWorks 5.3 operating system.

Benchmark Results:

- **Dispatching overhead:** We measured the time spent within the mechanisms that actually assign the processor to application functions. The dispatching mechanism is made up of multiple dispatching queues, each serviced by a thread at a different priority level. For dynamic scheduling, the queues must be reordered according to laxity or time to deadline as requests age.

Figure 8 shows a graph of the measured enqueue overhead, collected at the same time as the dequeue measurements. Dynamic queues may perform re-ordering before trying to

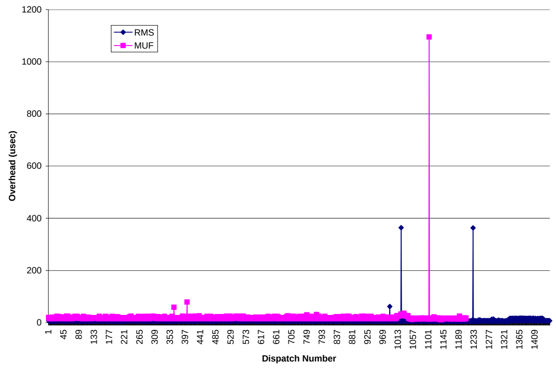

Figure 8: Measured RMS and MUF Enqueue Dispatching Overhead

wait on a *not empty* or *not full* condition variable and then enqueue or dequeue the operation after acquiring the appropriate lock. Therefore, it was necessary to exclude the time spent waiting for locks from the measurement, so that only the CPU time actually consumed by the dynamic queue was measured. This was achieved by extending the time probe class provided by ACE [33] framework to log suspend and resume time probe events around the call to acquire the lock, and to assess total overhead accordingly.

Figure 9 shows a graph of the measured dequeue dispatching overhead using both the MUF and RMS scheduling strategies. As Figure 8 and Figure 9 show, several anomalous data

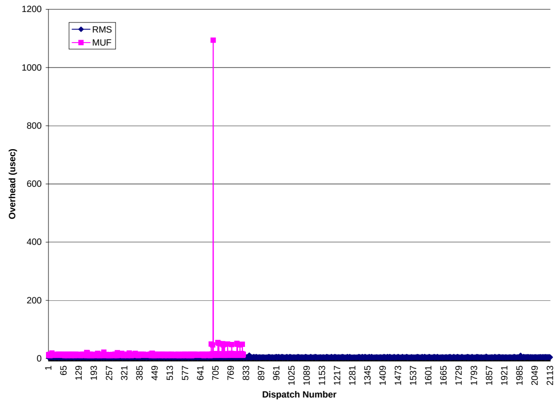

Figure 9: Measured RMS and MUF Dequeue Dispatching Overhead

points were observed in the measured enqueue and dequeue overheads. We attribute these to non-determinism in our experimental setup, possibly due to network interrupt handling in the VxWorks `tNetTask`, rather than to the behavior of the queues themselves. Excluding these outlying data points, the observed enqueue and dequeue overheads were approximately the same for the static RMS scheduling strategy and the hybrid static/dynamic MUF scheduling strategy, with a slightly higher overhead observed for the dynamic queues used for MUF.

These results indicate that (1) the amount of dynamic reordering was low in this experiment, and (2) the fundamental overhead for dynamic and static queue management is comparable when there is little dynamic reordering. For future investigation, we plan to conduct similar experiments across a wider range of real-time embedded applications and application features. In particular, we hope to determine whether increased heterogeneity of application features would induce greater levels of reordering for dynamic scheduling, and if so at what resulting cost.

• **Operation execution times:** We compared the execution times of both critical and non-critical operations, which comprise a representative subset of all operations in the system. All operations were scheduled using MUF, which reorders operations dynamically by laxity in each priority level. Figure 10 illustrates this comparison. The operation execution times showed several anomalous spikes, similar in value and prevalence to those observed in the dispatching overhead measurements. We again interpret these as the result of nondeterminism in our experimental configuration rather than in dispatching the operations themselves. Otherwise, the operation execution times were reasonably deterministic, even with all operations dispatched from dynamically managed queues.

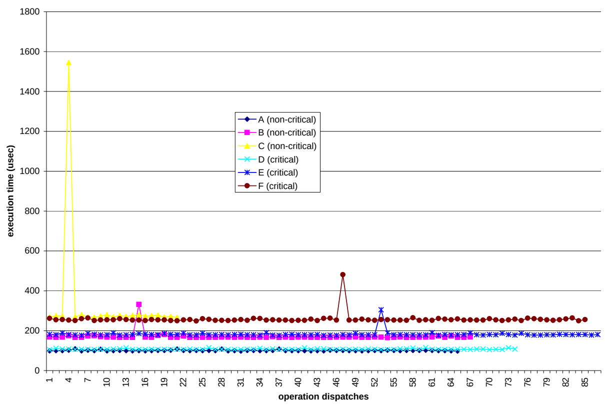

Figure 10: Execution Times of Critical and Non-Critical Operations

• **Operation cancellation:** Figure 11 shows the effects of operation cancellation for non-critical operations in dynamic scheduling strategies. As described above, the MUF scheduling strategy can use operation cancellation to reduce the amount of wasted work performed in operations that miss their deadlines. Assuming there is no residual value of an operation that completes past its deadline, this time increases the amount of unusable overhead. Note that while the MUF strategy with operation cancellation was more effective in limiting the number of operations that were dispatched and then missed their deadlines, the number of operations that made their deadlines in each case was comparable. We attribute this to the short execution times of several of the non-critical operations. In fact, the variation with cancellation had slightly lower numbers of non-critical operations that were successfully dispatched, as operation cancellation is necessarily pessimistic.

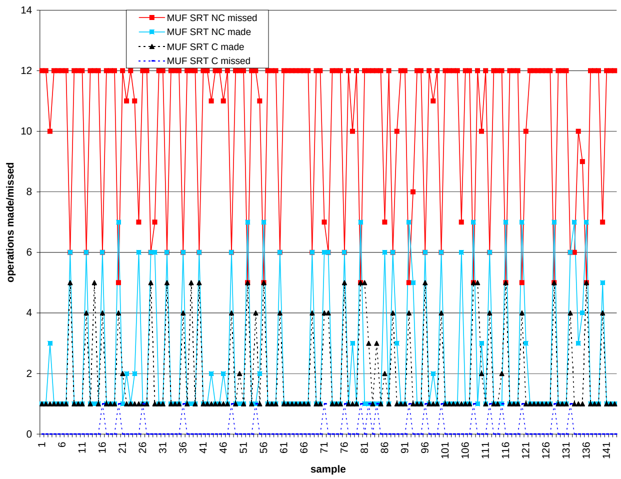

Figure 11: Effects of Non-Critical Operation Cancellation

• **Protecting critical operations:** We examined the relative effects of CPU overload on critical and non-critical operations, in the hybrid static/dynamic MUF scheduling strategy and the static RMS strategy. Figure 12 shows the number of deadlines made and missed for each strategy. With no operation cancellation, MUF meets all of its deadlines, while RMS misses between 2 and 6 critical operations per sample. Furthermore, MUF successfully dispatches additional non-critical operations. We investigated whether adding operation can-

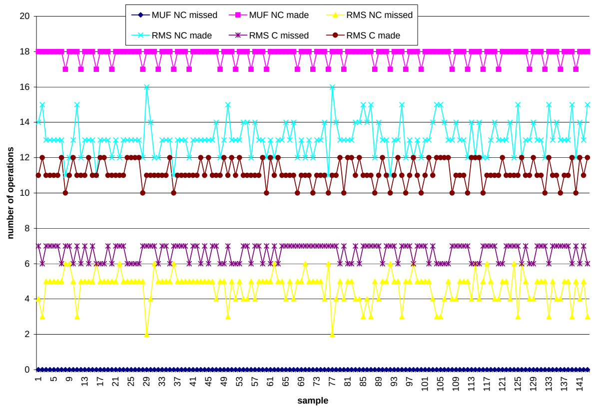

Figure 12: Effects of CPU Overload under RMS and MUF

cellation might have reduced the number of missed deadlines for critical operations with RMS, by reducing the amount of wasted work. However, it appears that the overhead of operation cancellation in fact makes matters worse, missing between 6 and 7 operations per sample. We interpret this to mean that there were few opportunities for effective non-critical operation cancellation in RMS under the experimental conditions.

3.2.2 Adaptive Middleware Layer Integration

Integration overview: The QuO and TAO QoS policies and mechanisms described in Sections 3.1.1 and 3.1.2 provide an adaptive framework for meeting the application requirements listed in Section 2.2. To illustrate how we have integrated TAO and QuO framework to meet the QoS requirements of mission-critical real-time applications, we describe an example sensor-actuator application, representative of those found in event-driven avionics systems [14].

Synopsis of sensor-actuator applications: As illustrated in Figure 13, sensor-actuator applications contain many subsystems operating in concert, responding to sensor data events, and managing functions of the aircraft. These subsystems include functionality, such as the heads-up display and navigation subsystems. Sensor data can come from a number of sensors on the aircraft, such as a global positioning satellite receivers, or various radar sensors.

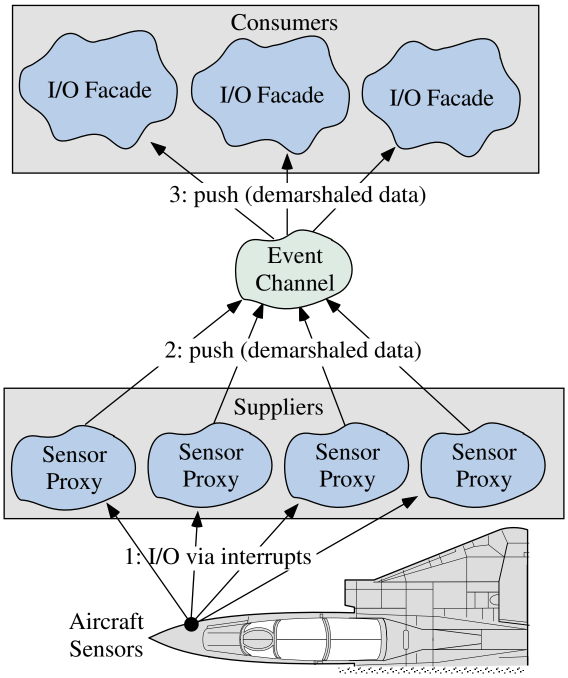

Figure 13: A Real-time Event-Driven Avionics System

In general, sensor-actuator applications have crucial QoS requirements, such as real-time response, dependability, and resource utilization. Moreover, the set of QoS requirements that must be satisfied can be highly variable, differing (1) between families of aircraft and between specific products within a family of aircraft, (2) between subsystems within a single aircraft, and (3) even between missions and between operating modes, within a single aircraft subsystem.

Currently fielded avionics systems are designed to be configured between missions, so that pilots can manually switch between mission computer operating modes [20]. However, for the most part current avionics software systems are configured statically. Therefore, changes occur in the form of software upgrade cycles and mission reprogramming. These legacy sensor-actuator systems are inflexible because the sensors are tightly coupled to the actuators, and the software is often tightly coupled to special-purpose hardware.

To overcome these limitations, it is necessary to apply new engineering methods to the process of developing these systems. In particular, improving the reliability and flexibility of distributed real-time systems requires advanced techniques, such as leveraging COTS hardware and software, increasing software reuse through middleware, and applying design patterns and adaptive object-oriented programming techniques. Moreover, these techniques serve to manage the monetary and time costs of the overall system development lifecycle.

Supporting sensor-actuator applications with QuO and TAO: As part of the TAO and QuO integration, we have developed a prototypical sensor-actuator application test-bed that uses the QuO adaptation engine to adjust the rate of event generation and the priority of generated events in response to system conditions. As illustrated in Figure 14, this test-bed can be configured with multiple suppliers that generate events at similar priorities. Other suppliers can flood the TAO real-time event channel in response to an external stimulus. A QuO system condition object recognizes that events are not being delivered on time and, in response, the QuO delegate of the non-critical supplier reduces the rate at which it is generating events. Similarly, the delegate of the non-critical supplier can

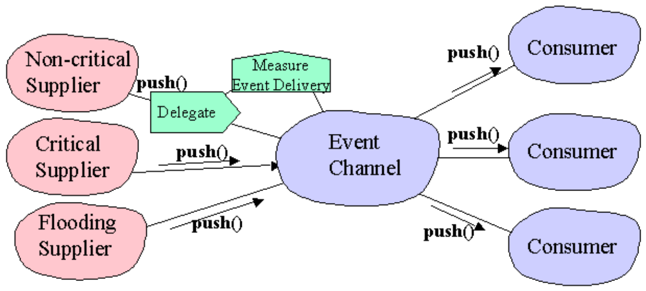

Figure 14: QuO Control of TAO Real-time Event Channel

reduce the priority of the events that it is generating. Conversely, a delegate of the critical supplier can increase the priorities of its events.

Our results to date indicate that adaptive QoS-enabled middleware frameworks, such as QuO and TAO, implement the necessary patterns, strategies, and infrastructure needed to build modern, more flexible avionics systems. In the example illustrated in Figure 13, sensors and actuators are decoupled and largely hidden from one another through sensor proxies and event channels. This allows sensors and actuators to be independently reconfigured, upgraded, or replaced dynamically without affecting the other subsystems. Furthermore, the avionics software can automatically adapt to changing missions and operational conditions by making tradeoffs between QoS dimensions, and dynamically reallocating resources. For example, an avionics system may temporarily sacrifice progress of non-critical operations for increased performance of critical operations.

Integration benefits: This adaptive TAO+QuO architecture provides the following combined assets:

- **Decoupling and enforcement:** The integrated middleware can decouple sensors and actuators while offering real-time enforcement, such as that provided by the TAO real-time ORB.

- **Flexible integration:** The architecture readily supports integrating other layers and components, such as dynamic resource managers and mechanisms, such as RT-ARM [37] or Darwin [38].

- **Application control:** Adaptable middleware, such as the QuO system, can provide application-level control and adaptation based upon changing mission goals, operational modes, environmental conditions, and changing QoS tradeoffs.

These capabilities are complementary. The TAO ORB enables the decoupling of sensor and actuator functionality while guaranteeing real-time delivery of sensor events. Dynamic resource managers enable access to and reallocation of resources in response to changing system conditions and mission needs, while the QuO middleware enables the application- and subsystem-level control to allocate the resources and functionality to the proper mission or operating mode.

4 Relationship to Existing Techniques and Research Communities

We view the techniques discussed in this paper, such as dynamic scheduling [19], multi-resource scheduling [39], and adaptive reconfiguration [1], 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 [19], 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 [35] 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 [19] becomes increasingly important. We also believe that emerging standards for dynamic and adaptive resource management in real-time mission-critical systems, *e.g.*, the OMG Dynamic Scheduling RFP [40], 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 [41], fair resource allocation [42], distributed signaling protocols [43], and admission control policies [44].

Core networking technologies: During the past decade, there has been substantial R&D emphasis on *high-speed networking* and *performance optimizations* for network elements [45] and protocols [3]. These efforts have paid off such that networking products are now available off-the-shelf that can support Gbps on every port, *e.g.*, Gigabit Ethernet and ATM switches. Moreover, OC-12 (622 Mbps) ATM connectivity in WAN backbones are becoming standard and OC-48 (2.4 Gbps) is being deployed for advanced networks such as Abilene [46] and Advanced Technology Demonstration Network (ATDnet) [47]. There are already plans to deploy OC-192 (9.6Gbps) within these backbones as it becomes practical.

Advanced architectures for modern high-performance routers and switches are being designed and constructed to support novel approaches for providing QoS. For example, the Active Network Node (ANN) [48] project at Washington University is using the Washington University Gigabit Switch (WUGS) [49] switch with the Smart Port Cards (SPC) [50] to provide a robust environment to support active networking and QoS research and development.

QoS architectures and models: The various real-time applications demand QoS assurance at the endsystem and network resource levels. Providing QoS guarantees at both these levels ensures true end-to-end QoS. There is extensive ongoing research at both these levels. AQUA (Adaptive QUality of service Architecture) [51] is a resource-management architecture, at the endsystem level, in which applications and the OS cooperate to dynamically adapt to variations in resource requirements and availability. AQUA manages the CPU and network-I/O resources in an integrated fashion to provide predictable QoS. At the network resource level the current Internet supports only best-effort service, irrespective of user expectations. Moreover, application heterogeneity dictates that there be service heterogeneity and service differentiation. QoS architectures and models have been proposed to address the end-to-end QoS challenge. For example, the IETF has several ongoing efforts directed to defining an architecture and proposing necessary protocols and infrastructure requirements. These working groups include Differentiated Services (DiffServ) [52], Integrated Services (IntServ) [53] and Integrated Services over Specific Link Layers (ISSLL) [54]. Additionally, the Internet2 QoS working group has proposed a testbed for IP differentiated services (QBone [55]) where commercial equipment is deployed in order to investigate different approaches or implementations supporting the DiffServ model. These all support the allocation of resources to provide different levels of guarantees to applications.

IntServ is defined in RFC 1633 [56] and is intended to provide QoS transport over IP internets. IntServ effort uses RSVP (Resource ReSerVation Protocol) [3] for signaling resource requirements. IntServ requires flow classification and forwarding state for each active flow at each router along each QoS path. ISSLL is intended to provide QoS transport for IP over specific networking technologies.

As an alternative, the *Differentiated Services* (DiffServ) [4] working group was formed to address perceived scalability and implementation issues associated with IntServ. DiffServ aggregates flows into service classes rather than maintaining per flow state. Moreover, QoS requirements are specified out-of-band, removing the necessity for a signaling protocol such as RSVP. Packet classification is based on the setting of a few bits in the IP header.

Providing QoS to applications: Most existing approaches are highly platform/protocol-specific, however, which makes it hard to develop and deploy portable applications. The different R&D focuses outlined above have not, in general, addressed providing middleware with standard QoS models and interfaces. And very little has been done to provide application developers with a standard programming interface that can leverage the underlying advances to provide end-to-end QoS guarantees.

Application developers need a standardized framework and interfaces which allow for QoS specification and to receive guarantees from the underlying network and QoS infrastructure. There have been several attempts [57] at designing and implementing a unified QoS API that leverages the QoS features available in networks and end-systems. Our QoS API (1) provides a simple interface for the users to QoS enable their applications, (2) hides the underlying platform/protocol specific issues of a QoS implementation, and (3) is integrated with middleware like CORBA, so the application not only continues to benefit from the middleware for distribution but also gets QoS guarantees through the standard middleware APIs.

5 Concluding Remarks

Over the past decade, individual QoS technologies, such as Differentiated Services [52] or the Resource ReSerVation Protocol (RSVP) [3], have emerged from previous R&D efforts and been applied successfully to specific application domains, such as audio/video streaming. In isolation, however, these achievements yield only a portion of the potential benefits for the broad domain of next-generation QoS-enabled distributed applications and services. For example, managing network resource reservations, without coordinating these reservations with other resource management mechanisms, such as prioritized thread pools or global middleware resource management, is insufficient to meet the end-to-end QoS requirements of next-generation systems.

During the same time period, commercial-off-the-shelf (COTS) middleware, such as CORBA, Java EJB, and COM+, has emerged from previous R&D efforts and been applied successfully to reduce the development cost and cycle-time associated with developing distributed applications. However, meeting the increasingly demanding QoS requirements of next-generation applications is currently beyond the capabilities of conventional COTS middleware solutions. In particular, meeting the QoS requirements of these 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, network, and application patterns, policies, and mechanisms, that can deliver end-to-end QoS support at multiple levels in distributed systems.

This paper has illustrated how next-generation applications with a variety of QoS requirements can be supported by adaptive middleware, such as QuO and TAO, in order to meet the QoS requirements end-to-end. To make the example concrete, and to document our on-going R&D activities in the DARPA Quorum integration effort [2], we have focused our examples and empirical benchmarks on the avionics mission computing domain. In our future work, however, we are addressing the following research issues to demonstrate the broader applicability of our adaptive multi-level middleware strategy for QoS-enabled distributed applications:

Leveraging existing QoS research: The operating system and networking research communities have produced a wealth of techniques, architectures, and empirical information for QoS management issues in the network and OS kernel layers. These techniques must be used as the basis for developing and evaluating middleware QoS management approaches, and wherever possible built into end-to-end middleware solutions. Some middleware solutions leverage particular point-solutions for QoS management, *e.g.*, TAO leverages preemptive thread scheduling in the OS kernel to enforce static priorities. However, a more comprehensive integration of policies and mechanisms *at the middleware level* is needed.

Identifying general-purpose patterns: To leverage existing QoS research at the OS and networking levels effectively, it is necessary to identify the key general-purpose patterns for *composing* the lower level mechanisms end-to-end. For example, identifying different patterns for co-scheduling network and CPU resources along a request-response path between a client and a server will be relevant to many applications. These client-server resource allocation patterns will in turn guide the creation of flexible middleware that is suited to the common requirements of a wide range of QoS-enabled client-server applications.

Identifying domain-specific patterns: Where effective resolutions of common design forces are captured by general-purpose patterns, each individual application domain also produces design forces that are specific to that domain. QoS requirements such as timing, utilization, or reliability constraints may differ between different application domains, *e.g.*, telecommunications and sensor-actuator systems. Additional research is needed to identify the key design forces for each domain, along with the patterns that can resolve those forces.

Building flexible QoS frameworks: After identifying the general-purpose and domain-specific patterns outlined above, along with the necessary lower-level mechanisms for QoS enforcement, it is possible to reify these patterns in flexible QoS frameworks. Implementing key QoS mechanisms, strategies and policies, and embedding these within middleware frameworks, allows middleware to support (1) the common requirements of a wide range of QoS-enabled applications and (2) the specific requirements of individual domains and applications. Moreover, building these frameworks offers practical insights into additional patterns and techniques for QoS management in adaptive middleware for distributed and embedded systems.

6 Acknowledgements

We would like to thank Bryan Doerr and Greg Holtmeyer of the Boeing Company for their support of the research described in this paper. Both have contributed to our vision of adaptive end-to-end QoS, and have supported our work toward that vision. We would also like to thank Alia Atlas of BBN Technologies/GTE Internetworking for her contributions to our research on integrating adaptive middleware layers, described in Section 3.2.2.

References

[1] 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.

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

[3] R. Braden et al, "Resource ReSerVation Protocol (RSVP) Version 1 Functional Specification," *Network Working Group RFC 2205*, pp. 1–112, Sep 1997.

[4] 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.

[5] 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.

[6] C. O'Ryan, D. C. Schmidt, F. Kuhns, M. Spivak, J. Parsons, I. Pyarali, and D. Levine, "Evaluating Policies and Mechanisms for Supporting Embedded, Real-Time Applications with CORBA 3.0," in *Proceedings of the 6^{th} IEEE Real-Time Technology and Applications Symposium*, (Washington DC), IEEE, May 2000.

[7] 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.

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

[9] 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.

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

[11] J. P. Loyall, R. E. Schantz, J. A. Zinky, and D. E. Bakken, "Specifying and measuring quality of service in distributed object systems," in *Proceedings of The 1st IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC 98)*, April 1998.

[12] J. P. Loyall, D. E. Bakken, R. E. Schantz, J. A. Zinky, D. Karr, R. Vanegas, and K. R. Anderson, "Qus aspect languages and their runtime integration," *Proceedings of the Fourth Workshop on Languages, Compilers and Runtime Syste,s for Sclable Components*, May 1998.

[13] P. Pal, J. Loyall, R. Schantz, J. Zinky, , R. Shapiro, and J. Megquier, "Using qdl to specify qos aware distributed (quo) application configuration," in *Proceedings of The 3rd IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC 00)*, to appear March 2000.

[14] 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.

[15] A. Network and I. Services, "National Tele-Immersion Initiative." http://www.advanced.org/tele-immersion.

[16] J. Lanier, "Tele-Immersion: The Ultimate QoS-Critical Application," in *First Internet2 Joint Applications/ Engineering QoS Workshop*, May 1998.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] G. Kiczales, "Beyond the black box: Open implementation," *IEEE Software*, 1996.

[22] R. Vanegas, J. A. Zinky, J. P. Loyall, D. Karr, R. E. Schantz, and D. E. Bakken, "Quo's runtime support for quality of service in distributed objects," *Proceedings of Middleware 98, the IFIP International Conference on Distributed Systems Platform and Open Distributed Processing*, September 1998.

[23] R. E. Schantz, J. A. Zinky, D. A. Karr, D. E. Bakken, J. Megquier, and J. P. Loyall, "An object-level gateway supporting integrated-property quality of service," in *Proceedings of The 2nd IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC 99)*, May 1999.

[24] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley, 1995.

[25] C. O'Ryan, F. Kuhns, D. C. Schmidt, O. Othman, and J. Parsons, "The Design and Performance of a Pluggable Protocols Framework for Real-time Distributed Object Computing Middleware," in *Proceedings of the Middleware 2000 Conference*, ACM/IFIP, Apr. 2000.

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

[27] A. Gokhale and D. C. Schmidt, "Optimizing a CORBA IIOP Protocol Engine for Minimal Footprint Multimedia Systems," *Journal on Selected Areas in Communications special issue on Service Enabling Platforms for Networked Multimedia Systems*, vol. 17, Sept. 1999.

[28] I. Pyarali, C. O'Ryan, D. C. Schmidt, N. Wang, V. Kachroo, and A. Gokhale, "Applying Optimization Patterns to the Design of Real-time ORBs," in *Proceedings of the 5^{th} Conference on Object-Oriented Technologies and Systems*, (San Diego, CA), USENIX, May 1999.

[29] D. C. Schmidt, "GPERF: A Perfect Hash Function Generator," in *Proceedings of the 2^{nd} C++ Conference*, (San Francisco, California), pp. 87–102, USENIX, April 1990.

[30] D. C. Schmidt, S. Mungee, S. Flores-Gaitan, and A. Gokhale, "Software Architectures for Reducing Priority Inversion and Non-determinism in Real-time Object Request Brokers," *Journal of Real-time Systems, special issue on Real-time Computing in the Age of the Web and the Internet*, To appear 2000.

[31] F. Kuhns, D. C. Schmidt, C. O'Ryan, and D. Levine, "Supporting High-performance I/O in QoS-enabled ORB Middleware," *Cluster Computing: the Journal on Networks, Software, and Applications*, 2000.

[32] Z. D. Dittia, G. M. Parulkar, and J. R. Cox, Jr., "The APIC Approach to High Performance Network Interface Design: Protected DMA and Other Techniques," in *Proceedings of INFOCOM '97*, (Kobe, Japan), pp. 179–187, IEEE, April 1997.

[33] D. C. Schmidt and T. Suda, "An Object-Oriented Framework for Dynamically Configuring Extensible Distributed Communication Systems," *IEE/BCS Distributed Systems Engineering Journal (Special Issue on Configurable Distributed Systems)*, vol. 2, pp. 280–293, December 1994.

[34] D. C. Schmidt, M. Stal, H. Rohnert, and F. Buschmann, *Pattern-Oriented Software Architecture: Patterns for Concurrency and Distributed Objects, Volume 2*. New York, NY: Wiley & Sons, 2000.

[35] 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.

[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] J. Huang et al., "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.

[38] P. Chandra and et. al, "Darwin: Resource Management for Value-Added Customizable Network Service," in *Sixth IEEE International Conference on Network Protocols (ICNP'98)*, (Austin, TX), IEEE, Oct. 1998.

[39] J. Huang and R. Jha and W. Heimerdinger and M. Muhammad and S. Lauzac and B. Kannikeswaran and K. Schwan and 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.

[40] Object Management Group, *Dynamic Scheduling*, OMG Document orbos/99-03-32 ed., March 1999.

[41] 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.

[42] 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.

[43] 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.

[44] 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.

[45] C. P. et al., "A fifty gigabit per second ip router," *IEEE Journal of Transactions on Networking*, vol. 6, pp. 237–248, June 1998.

[46] U. C. for Advanced Internet Development, "Abilene is an advanced backbone for the Internet2 project." http://www.internet2.edu/abilene/.

[47] ATD, "Advanced Technology Demonstration Network." http://www.atd.net/.

[48] D. Decasper, G. Parulkar, S. Choi, J. DeHart, T. Wolf, and B. Plattner, "A Scalable, High Performance Active Network Node," *IEEE Network Magazine*, vol. 13, January/February 1999.

[49] J. Turner and N. Yamanaka, "Architectural Choices in Large Scale ATM Switches," *ICICE Transactions*, 1998.

[50] W. N. Eatherton and T. Aramaki, "SPC Specification," Applied Research Lab, Working Notes ARL-WN-98-02, Washington University, St. Louis, 1998.

[51] R. F. K. Lakshman, Raj Yavatkar, "Integrated CPU and Network-I/O QoS Management in an Endsystem," in *Proceedings of the IFIP Fifth International Workshop on Quality of Service (IWQoS '97)*, 1997.

[52] IETF, "Differentiated services (diffserv)." http://www.ietf.org/html.charters/diffserv-charter.html, 2000.

[53] IETF, "Integrated services (intserv)." http://www.ietf.org/html.charters/intserv-charter.html, 2000.

[54] I. S. over Specific Link Layers (issll), "IETF." ttp://www.ietf.org/html.charters/issll-charter.html.

[55] I. Q. W. G. Draft, "QBone Architecture (v1.0)," tech. rep., Internet2, August 1999.

[56] B. Braden, D. Clark, and S. Shenker, "Integrated services in the internet architecture," *Network Information Center RFC 1633*, June 1994.

[57] B.Riddle, A. Adamson, "A QoS API Proposal." Pre-Workshop Draft, May 1998. http://www.internet2.edu/qos/may98Workshop/html/apiprop.html.