Experiences Applying Adaptive QoS Management in Middleware

Christopher Gill
cdgill@cse.wustl.edu
Washington University
St. Louis, MO, USA

Jeanna Gossett and David Corman
{jeanna.m.gossett,david.e.corman}@boeing.com
The Boeing Company
St. Louis, MO, USA

Douglas Schmidt
schmidt@dre.vanderbilt.edu
Vanderbilt University
Nashville, TN, USA

Joe Loyall, Rick Schantz, and Michael Atighetchi
{jloyall,schantz,matighet}@bbn.com
BBN Technologies
Cambridge, MA, USA

Abstract

Dynamic distributed real-time and embedded (DRE) systems in which application requirements and environmental conditions may not be known a priori, or which may vary at run-time, can benefit from self-adaptive and reconfigurable approaches to quality of service (QoS) management. Adaptive QoS management is particularly useful to address key constraints (such as solution quality and end-to-end timeliness) that must be traded off in dynamic DRE systems. To meet overall system constraints in complex DRE systems, moreover, adaptive management of multiple QoS aspects must be coordinated across multiple layers of applications and their supporting middleware.

This paper makes three contributions to the study of applying adaptive QoS management techniques to DRE systems in practice: (1) it presents an architectural resource management model for integrating inter-related QoS management aspects across multiple layers of common-off-the-shelf (COTS) middleware, (2) it describes our experience integrating multiple middleware QoS management technologies to manage quality and timeliness of imagery adaptively within a representative DRE avionics system case study, and (3) is analyzes empirical results that quantify the impact of our approach on key trade-offs between timeliness and solution quality, both for the avionics system case study and for the broader practice of developing adaptively managed DRE systems.

Keywords: Adaptive Middleware, Empirical Case Studies, Distributed Real-Time and Embedded (DRE) Systems

1 Introduction

Over the past decade, various technologies have been devised to alleviate many complexities associated with developing software for distributed real-time and embedded (DRE) systems, such as tele-immersion environments, fly-by-wire aircraft, industrial process automation, and total ship computing environments. Some of the most successful emerging technologies have centered on *middleware*, which is systems software that

- Functionally bridges the gap between application programs and the lower-level hardware and software infrastructure to coordinate how parts of applications are connected and how they interoperate
- Enables and simplifies the integration of components developed by multiple technology suppliers and
- Provides common services that were formerly placed directly in applications, but in actuality are application-independent and need not be developed separately for each new application.

As middleware has matured and been applied to a variety of use cases, there has been a natural growth in extensions, features, and services to support these use cases. For example, the Minimum CORBA [1] and Real-time CORBA [2] specifications, as well as the Real-Time Specification for Java (RTSJ) [3], are examples of standards that have emerged from research and experience supporting the quality of service (QoS) needs of DRE systems.

DRE systems have historically been developed and validated using relatively static development and analysis techniques (such as function-oriented design and rate monotonic analysis) to implement, allocate, schedule, and manage their resources. These static approaches have proven to be acceptable for closed DRE systems, where the set of ap-plication tasks that will run in the system and the loads they will place on system resources change infrequently and are known in advance. Static approaches are *not* well-suited, however, for *open* DRE systems (such as collaborative mission re-planning, adaptive audio-video streaming, or robotics applications designed for close interaction with their environments), which evolve more rapidly and must collaborate with multiple remote sensors, provide on-demand browsing and actuation capabilities for human operators, and respond flexibly to unanticipated situational factors that arise at run-time. Due to the dynamic contexts in which these open DRE systems oper-

ate, self-adaptation and reconfigurability are essential to maintain QoS assurances (1) under widely varying environmental conditions, (2) across heterogeneous "systems of systems," (3) and at time scales that would consume an inappropriate amount of attention from human operators if managed manually.

Previous research [4, 5] has shown the benefits of integrating multiple QoS management techniques in standards-based middleware and applying single-layer adaptive resource management techniques real-world DRE systems [6, 7, 8]. Only limited practical experience is available, however, with integrating resource management techniques across multiple layers of standards-based DRE systems. To help fill this gap, this paper presents an empirical case study of the vertical integration of three layers of middleware QoS management technologies within Boeing's Bold Stroke [9, 10], which is a standards-based DRE avionics platform that is representative of a broader class of applications (including mission-critical distributed audio/video processing [11] and real-time robotic systems [12]) that require both static and dynamic support for QoS. In this paper, we describe the integration of the following three layered adaptive QoS management technologies, show empirical results of their use in the Bold Stroke avionics system, and analyze each technology's contribution to adaptive QoS management:

Quality Objects (QuO). QuO [13] is an adaptive middleware framework designed by BBN Technologies to support the development of QoS behavior of a system separate from - but complementary to - the development of its functional behavior. To enhance end-to-end adaptive behavior in our Bold Stroke testbed we used three QuO capabilities: (1) *contracts*, which specify desired and available QoS, along with the policies for controlling QoS and adapting to changes, (2) *delegates*, which are remote object proxies, with well-defined points to insert adaptive behaviors into end-to-end paths, and (3) *system condition objects*, which provide interfaces to parts of the system that must be measured or controlled by contracts.

Real-Time Adaptive Resource Manager (RT-ARM). RT-ARM [14] is a resource adaptation service developed by Honeywell Technologies to evaluate one or more QoS metrics repeatedly during system operation and adaptively adjust resource allocations according to specified constraints on acceptable QoS. RT-ARM is designed to monitor a wide variety of QoS metrics and enforce a wide variety of QoS constraints. In this paper we only consider how the RT-ARM can monitor one QoS metric (*application progress*) and one constraint (*timeliness with respect to a specified deadline*).

TAO reconfigurable scheduler. The TAO Reconfigurable Scheduler [8] is a CORBA-based scheduling service designed for flexible support of hybrid static/dynamic scheduling developed at Washington University, St. Louis. It selects a feasible set of rates of operation invocation and assigns priorities to the operations according to the scheduling strategy with which it was configured. When the RT-ARM modifies the ranges of invocation rates, the TAO Reconfigurable Scheduler first provides criticality assurance for the hard real-time operations by ensuring each operation is scheduled at a rate in its available range and that all critical operations can be feasibly scheduled at those rates. TAO's Reconfigurable Scheduler then adds non-critical processing (*e.g.*, re-planning operations) and optimizes processor utilization for the image processing operations by maximizing their rates subject to schedule feasibility.

This paper is organized as follows: Section 2 presents the example avionics system application that provides the context in which we applied our three adaptive QoS management middleware technologies outlined above; Section 3 explains how these adaptive technologies were integrated within the avionics system and examines the issues and optimizations resulting from the integration; Section 4 discusses the methodology and overall design of our experiments with the integrated system; Section 5 reports our results and analyzes trade-offs between timeliness and solution quality under different adaptation approaches; Section 6 summarizes the lessons learned from our experiences integrating and evaluating middleware technologies for adaptive QoS management; Section 7 compares our efforts with related work; and Section 8 presents concluding remarks.

2 An Avionics Case Study

This presents an overview of the example avionics system application that provides the context in which we applied our three adaptive QoS management middleware technologies outlined in Section 1.

2.1 Overview of the Application Testbed

The case study and experiments on DRE middleware adaptive QoS management presented in this paper were conducted using an open experimentation platform (OEP)[1] developed by Boeing.

OEP application goals. As shown in Figure 1, the OEP consisted of two aircraft (one playing the role of server and the other playing the role of client) that interoperated in real-time over a very low-bandwidth radio data link to enable operators to (1) download imagery from the server to the client and (2) collaboratively annotate the downloaded images on both the client and server aircraft to enhance *collaboration* between

[1] An OEP is a hardware/software laboratory environment incorporating COTS infrastructure and representative applications operating in it, which can be modified and augmented with technology and application innovations, toward evaluating their contribution to technical challenges in that context.

personnel on the two remote aircraft. These capabilities increase the ability of the aircraft to respond dynamically both to rapidly changing environmental conditions and to new information that arrives during the re-planning collaboration. These capabilities offer potential improvements in the effectiveness of a wide range of applications, from airborne search and rescue, to aerial suppression of forest fires, to disaster assessment and response. Collaborative re-planning enables operators to

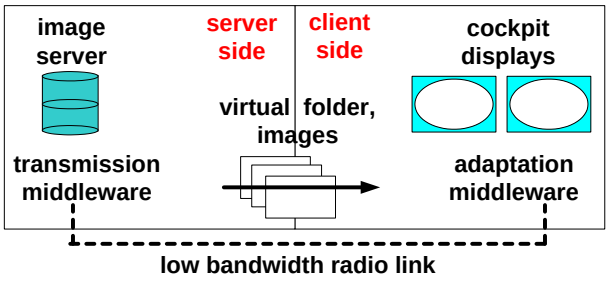

Figure 1: **Collaboration Scenario**

respond more rapidly to changing situations in-flight, *e.g.*, the server sends links to downloadable imagery to the client aircraft, which it then uses for re-planning. In our example scenario used to evaluate the OEP, an off-board sensor detects time-sensitive information that initiates re-planning and provides this information to the server node. The server node has authority to initiate re-planning with the client node and sends an alert to the client node, along with a "virtual folder" that contains thumbnails of relevant images and the associated links to the complete images. Operators on the client and server nodes collaborate to develop a new plan, which the client then performs.

The research presented in this paper applies multi-layer adaptive middleware techniques to alleviate the following limitations that impede successful re-planning in existing systems:

1. Limits on radio data link bandwidth that constrain the operational utility of existing systems to collaboratively replan the activities of airborne nodes.
2. Static resource management schemes that often rely on over-allocation strategies and reduce and sometimes eliminate the amount of processor and network resources available for re-planning and rehearsal.

A key goal of the OEP testbed illustrated in Figure 1 is to use adaptation and reconfiguration to provide operators the same level of confidence in the re-directed plan as in the original pre-planned version, even in the face of dynamic environmental factors, such as variations in network bandwidth and unannounced re-planning alerts. Thus, in addition to providing client-side operators up-to-date information detected by remote sensors (*e.g.*, fresh images of the new destination) about the environment encountered en-route to and from the new destination, the OEP must manage key trade-offs between transmission quality and latency for that information. Our solution involves implementing QoS-managed browser-based collaboration capabilities to (1) enable client and server nodes to view the same displays and information and (2) ensure image quality and transmission latency stay within acceptable bounds as independent as possible given the available resources (obviously there is a minimum, below which nothing useful can be accomplished).

This common browser view also allows server-side operators to decorate imagery with annotations that will be visible on the client node rapidly, *i.e.*, within one second. The advantage of this approach is that features can be located on an image via an icon placed at a precise location relative to an easily identified reference point. This capability in turn allows operators on the client and server nodes to establish a common frame of reference of the plan update and the new destination environment while the client is en-route to that destination, which is far better than the voice-only radio communications previously available in conventional re-planning systems.

Our solution is readily extensible to scenarios encompassing multiple client and server nodes, as well as other applications (such as coordination within teams of autonomous agents in rapidly changing environments or circumventing cascades of failures in distributed critical infrastructure) that require adaptive run-time support for collaborative re-planning.

OEP client and server interactions. In our OEP application testbed, a server-side operator first uses a user interface to send an alert to the client along with a virtual target folder containing a set of thumbnail images to the client. The collaboration client application on the client aircraft) also contains a virtual folder manager component that provides it access to, and storage of, virtual folders and their images. If sufficient memory is available, the virtual folder manager can hold more than one virtual folder, though only a single virtual folder was downloaded for our OEP evaluation.

The client node determines which page of the virtual folder is displayed. Operators on the client node can navigate the virtual folder both forward and backward using "next" and "previous" buttons on their cockpit display. The virtual folder can also be reset to a home page by touching another button. A thumbnail page in the virtual folder allows the operator to select images to download and view without the overhead of downloading each complete image. A bar next to each thumbnail indicates whether its corresponding image has been downloaded: the bar is green if so and if not is red.

Server and client node operators can draw annotations and move commonly viewed individual cursors during the collaboration. To avoid problems with having both server and client operators manipulate the image simultaneously, the client is given control of image download and manipulation during the collaboration, including panning side-to-side, rotation, and

zooming. Server and client node operators can move their respective cursors to indicate a specific location on the image. They are also able to draw circle, line, rectangle, and triangle annotations to designate larger regions on the image.

Update messages are sent between the collaboration server and client to update cursor positions and annotations. The server to client update message contains server cursor movements and annotations drawn on the server. The client to server update message contains image manipulation information in addition to client cursor movements and client-drawn annotations. Update messages are only sent as needed and only contain updates since the last such message. Displays on both client and server are updated with the update information to maintain a common synchronized view of the virtual folder.

2.2 Improvements in the State of the Art

As discussed in Section 1, we implemented a prototype of the OEP application described above using three layered adaptive QoS management technologies, *i.e.*, QuO, RT-ARM, and TAO's Reconfigurable Scheduler. This layered middleware design provides an open systems "bridge" between legacy on-board embedded avionics systems and off-board information sources and systems. The technical foundation of this bridge is a Real-time CORBA Object Request Broker (ORB) [2] called TAO [15]. TAO is a widely-used, open-source (deuce.doc.wustl.edu/Download.html) middleware platform targeted for DRE applications with deterministic and statistical QoS requirements, as well as best-effort requirements.

In our OEP testbed, TAO uses a pluggable protocol to communicate over a very low bandwidth (approximately 2,400 baud in each direction) radio network. Network time slots were allocated asymmetrically in the OEP so that the image tiles were downloaded at close to 4,800 baud with a small fraction of the bandwidth allocated to carry tile requests and update messages from the client to the server. Middleware technologies were applied at several architectural layers to manage key resources and ensure the timely exchange and processing of mission-critical information. In combination, these techniques support Internet-like connectivity between server and client nodes, with the added assurance of real-time performance in a highly resource-constrained environment.

The OEP testbed leverages existing open systems client and server platforms. On the client side, we used an Operational Flight Program (OFP) system architecture based upon commercial hardware, software, standards, and practices [16] that supports re-use of application components across multiple client platforms. The OFP architecture includes the Bold Stroke avionics domain-specific middleware layer [9] built upon TAO. The Bold Stroke middleware isolates avionics applications from the underlying hardware and operating system (OS), enabling hardware or OS advances from the commercial marketplace to be integrated more easily with the avionics application. This architecture uses the adaptive middleware technologies outlined in Section 1 (and described in more detail in Section 3) to address the limitations with existing time-sensitive re-planning noted in Section 2.1.

2.3 System Resource Management Model

The layered adaptive resource management model for the OEP testbed is illustrated in Figure 2. Figure 2 shows both (1) end-to-end request and response paths for image transmission in the OEP in detail and (2) QoS adaptation interactions between the different middleware technologies described in Section 3. Sections 4 and 5 also reference the resource management model shown in Figure 2. When client operators request

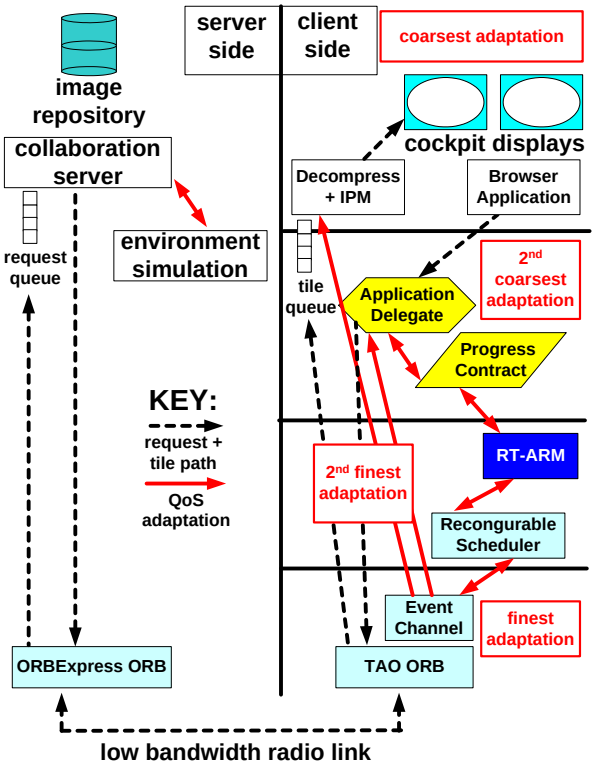

Figure 2: **Resource Management Model for OEP Testbed**

an image, that request is sent from the browser application to a QuO application delegate [16], which then sends a series of requests for individual tiles via TAO over a low-bandwidth radio connection to the server. The delegate initially sends a burst of requests to fill the server request queue; after that it sends a new request each time a tile is received. For each request, the delegate sends the tile's desired compression ratio, which is determined by the progress of the overall image download when the request is made.

On the server node, the ORBExpress Ada ORB receives

each request from the radio connection, and from there each tile goes into a queue of pending tile requests. A collaboration server pulls each request from that queue, fetches the tile from the server's virtual target folder containing the image, and compresses the tile at the ratio specified in the request. The collaboration server then sends the compressed tile back through ORBExpress and across the low-bandwidth radio link to the client. Server-side environmental simulation services emulate additional workloads that would be seen on the imagery server under realistic operating conditions.

After a compressed tile is received by the client from the radio network, TAO delivers it to a servant that places the tile in a queue, where it waits to be decompressed. When the tile is removed from the queue it is decompressed and then delivered by client-side operations to Image Presentation Module (IPM) hardware, which renders the tile on the cockpit display. The decompression and IPM delivery operations are dispatched by the TAO Event Channel [17] at rates selected in concert by the RT-ARM [14] and the TAO Reconfigurable Scheduler [18, 8], as described in Sections 3.2 and 3.3, respectively.

3 Adaptive Middleware Applied

To address the challenges with in-flight re-planning described in Section 2, we have designed, implemented, and flight-tested an integrated QoS management architecture based on Real-time CORBA [2]. A key theme in this architecture is that coarser-grain adaptation is performed by higher layers of the architecture (*i.e.*, closer to the application), with finer grained adaptation at each lower layer (*i.e.*, closer to the OS, network, and hardware). To enhance performance, our multi-layered architecture (shown in Figure 2) tries to handle adaptation at the lowest layer possible, with adaptation moving up to higher layers only if QoS requirements cannot be met via adaptation in the current layer, as follows:

- The finest granularity of adaptation in the OEP system architecture is the lowest priority dynamic scheduling of non-critical operations [8] by the dispatcher of the TAO Real-Time Event Channel, which we developed in previous work [17].
- The second finest level of adaptation granularity is achieved by a Real Time Adaptive Resource Manager (RT-ARM) [14] and the TAO Reconfigurable Scheduler [8, 18], which reschedule rates of invocation of application components, while maintaining deadline-feasible scheduling of critical operations.
- The second coarsest level of adaptation is performed by the Quality Objects (QuO) framework [11], which monitors progress downloading and processing image tiles toward the desired deadline for the entire image.
- Although QuO represents the highest middleware layer in the OEP system architecture, the highest layer at which adaptation can be performed is the application layer, where the client personnel can specify the coarsest grain requirements for image quality and timeliness.

The remainder of this section describes the adaptive middleware in our multi-layered architecture, ranging from the coarsest to the finest granularity of adaptation.

3.1 QuO: Second Coarsest Grain Adaptation

Figure 3 illustrates the overall architecture of QuO. QuO is a general-purpose framework that supports a variety of adaptation strategies. We therefore developed a reactive QoS adaptation policy [19] for the OEP testbed that manages the overall trade-offs of timeliness versus image quality. As Figure 2 in Section 2.3 illustrates, we integrated QuO at the second coarsest level of adaptation in the OEP system, just below the application level. When the client node requests an image from the server node, a QuO delegate breaks the image request up into a sequence of separate tile requests-each tile is a smaller-sized piece of the entire image for which a separate compression ratio can be assigned.

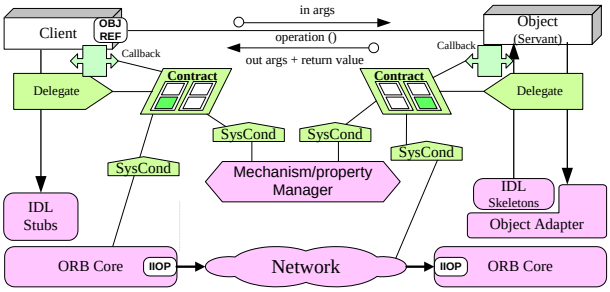

Figure 3: **QuO Architecture Overview**

The number of tiles requested by the delegate is based upon the image size, while the compression level of an individual tile can be adjusted dynamically based upon the deadline for receiving the full image and the expected download time for the tile. The image is tiled from the point of interest first. The early tiles contain the most important data, so that decreased quality of later tiles will have minimal impact on the overall mission re-planning capabilities.

In our OEP testbed, a QuO delegate adapts the compression level of the next tile requested. A QuO contract monitors progress of the image download through system condition objects and influences the compression level of subsequent tiles based upon whether the image is behind schedule, on schedule, or ahead of schedule. If the processing of the image tiles falls behind schedule, the contract prompts the RT-ARM (described in Section 3.2) to attempt to adjust invocation rates to allocate more CPU cycles to tile decompression.

The QuO delegate initially determines the number of tiles into which the image will be broken. Due to constraints on both the server tiling software and the client display software, in the OEP testbed the choices were limited to 1, 16, or 64 tiles. Our experiments (described in Section 4) revealed that breaking a 512 x 512 pixel image into 64 tiles introduced too much overhead, which increased the download time dramatically. We therefore always requested either 16 tiles or the entire image.

The QuO delegate also determines the initial compression ratio for the image. We used the lowest compression ratio available for the initial tiles, because tiles are requested starting from the region of interest first and subsequent tiles are not as valuable. It therefore is more important for the application to download image tiles at compression ratios greater than or equal to that of the region of interest. After the number and initial compression ratio of tiles have been set, the delegate makes several calls to the server to request the first set of tiles.

The number of tiles requested initially is determined by the size of a tile request queue that holds outstanding tiles requested from the server, but not yet received by the client. This queue enables the QuO encoded policy to delay requesting tiles until necessary to provide the maximum impact of compression ratio adaptation, while ensuring that there is always a tile request ready for the server to process. Finally, the delegate initiates periodic callbacks to its methods, so that it can perform contract evaluation, adjust compression ratios, and request subsequent tiles as needed to fill the tile request queue. As tiles are received from the server node, QuO system conditions count tiles received, processed, and displayed.

There are four operating regions specified by the QuO contract: *inactive*, *early*, *on time*, and *late*. The inactive operating region is entered when the entire image has been downloaded. The on time operating region indicates that the image is on pace to complete before - but close to - its deadline. Similarly, the early region indicates that the image is on pace to finish well before its deadline and the late operating region indicates that the image will finish after the deadline at the current rate of progress.

There is no change in the compression ratio if the current operating region is on time. If the current region is early, then the compression ratio is lowered to the initial compression ratio, so that the remaining tiles can have the same quality as the initial tiles. If the current operating region is late, the compression ratio is increased in increments of 25:1 in the range [50:1, 75:1, 100:1]. After checking progress - and if necessary setting a new compression ratio and notifying RT-ARM of any changes in the operating region - QuO checks the request queue's depth and requests additional tiles until the tile request queue is full or the last tile has been requested. QuO can be downloaded in open-source format from quo.bbn.com.

3.2 RT-ARM: Second Finest Grain Adaptation

As Figure 2 in Section 2.3 illustrates, we integrated the Real-Time Adaptive Resource Manager (RT-ARM) at the second finest level of adaptation in the OEP system, just below QuO. Figure 4 shows the structure of the RT-ARM service manager we integrated within the OEP testbed. RT-ARM is used in the OEP testbed to manage the progress of the thread(s) for decompressing received tiles and delivering them to the application by the client of the OEP. When triggered to react, RT-ARM manipulates the CPU usage of key operations on the request/tile path, such as tile decompression and delivery of tiles to the IPM processor in the cockpit. RT-ARM does this by manipulating subsets of task invocation event rates from application-specified available rate sets. If image tile processing falls behind schedule, the QuO contract prompts RT-ARM to adjust ranges of invocation rates to reallocate more CPU cycles to decompressing remaining tiles.

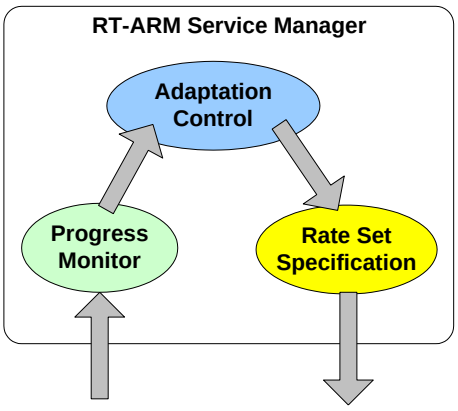

Figure 4: **RT-ARM Service Manager**

In response to changing environmental conditions, RT-ARM can trigger such adaptation in two ways: (1) reactively when the QuO contract notifies RT-ARM that the operating region boundary has changed or (2) proactively when it periodically checks the status of the system and notices a current or impending violation of the operating region limits. We distinguish the case where RT-ARM simply evaluates its operating status and takes no action from the case where that evaluation triggers a change in rate ranges and a corresponding recomputation of rates and priorities by the TAO Reconfigurable Scheduler described in Section 3.3.

RT-ARM attempts to keep operations within the on time QoS region by shrinking or expanding their respective ranges of selectable rates. This strategy was implemented by computing the average number of dispatches required by an operation at a given time, then discarding the rates that would cause the operation to complete too early or too late. As a result, rates of image processing operations that begin to veer towards the "early" and "late" regions are forced to adapt. If this level of adaptation is insufficient to keep the overall image download

on time, QuO steps in and adjusts both RT-ARM operating region and the compression level of the next tile.

3.3 TAO Reconfigurable Scheduler: Second Finest Grain Adaptation

We also integrated the TAO Reconfigurable Scheduler within the second finest level of adaptation shown in Figure 2 in Section 2.3, to manipulate the rates and priorities of operations dispatched at the finest level of adaptation within the TAO ORB core and Event Channel. Figure 5 illustrates the relationship between the TAO Reconfigurable Scheduler and TAO's Event Channel Dispatcher.

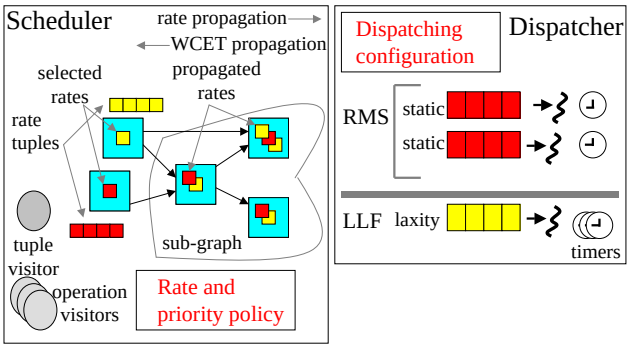

Figure 5: **TAO Reconfigurable Scheduler and TAO Event Channel Dispatcher**

In earlier work [4], we tried a simple integration of the TAO Reconfigurable Scheduler with RT-ARM, in which RT-ARM would propose a set of rates for operations and TAO's Reconfigurable Scheduler would generate a schedule and then evaluate that schedule's feasibility. Unfortunately, that approach proved computationally inefficient since RT-ARM and TAO's scheduler operated too independently. Our initial results, however, pointed to the solution pursued in this work: *closer integration of adaptation mechanisms*. We therefore evolved TAO's Reconfigurable Scheduler so that the rate selection mechanism was pushed down into it, while the policy for rate selection was supplied by RT-ARM. Specifically, RT-ARM provided a specific rate selection strategy to TAO's Reconfigurable Scheduler at system initialization time based upon operation criticality and available rates.

The first revision we made to TAO's Reconfigurable Scheduler for our OEP case study was to refactor its implementation for greater re-configurability, extending similar efforts started during earlier work. Our second revision incorporated rate selection into the schedule generation and feasibility analysis steps to determine an ordering of key operation characteristics used by a particular scheduling heuristic, assign both rates and priorities through different forms of sorting, and apply the most efficient sorting algorithm for each case. This strategy allows one scheduler to be used for efficient rate selection and priority assignment, all adaptively at run-time. These revisions are released in TAO's Reconfigurable Scheduler, which can be downloaded at deuce.doc.wustl.edu/Download.html in open-source format along with the rest of the TAO middleware, documentation, and examples.

4 Methodology for Experiments

This section introduces the objectives and approach to a set of adaptive middleware experiments completed during post-flight ground tests of the OEP in January 2003, which followed the actual flight tests conducted in December 2002. The four primary goals of these experiments were to

- Quantify the ability of the multi-layered QoS management mechanisms (Section 3) running within the Bold Stroke OEP (Section 2) to maximize image fidelity while meeting download deadlines;
- Offer preliminary assessment of the relative contributions of our different QoS management mechanisms within the OEP resource management model (Figure 2 in Section 2.3);
- Profile the temporal performance of those mechanisms; and
- Quantify the relative benefits of this approach compared to the same application running without adaptation.

These experiments also measure trade-offs between timeliness and image quality in a controlled system environment designed to remove influences outside the scope of the metrics considered here. This approach enabled us to establish a baseline against which realistic parameters (*e.g.*, network latency jitter, traffic loads, or other factors) can be varied in a controlled way and their contributions to system behavior also quantified.

The remainder of this section describes the platform on which our experiments were run, introduces the metrics we used to evaluate the OEP architecture, explains the design of the experiments. The results of these experiments are presented in Section 5.

4.1 OEP Platform Hardware and Software Characteristics

Our experiments were run on realistic hardware in the Avionics Integration Center (AIC) laboratory at Boeing, St. Louis. The client platform was a 400 MHz Dy-4 PPC 750 processor with 128 MB of memory, running the VxWorks real-time OS, version 5.3.1, with TAO version 1.0.7. The server was hosted on a flight-ready chassis incorporating multiple Alpha processors running the DEC Unix OS, with ORBExpress/RT version

. A Boeing-owned operator console with dual Digital Alpha 480 MHz single board computers was used by the server-side operator. The majority of server functionality was inherited from a legacy Boeing project, whose software was tested on Digital Alpha and Sun Solaris variants of the UNIX OS.

At the time of system design, only the Alpha platform was available in a ruggedized, flight-worthy package. Alpha UNIX represents a high-performance, soft real-time OS. System components were distributed across both computers, using a simulated low-bandwidth (approximately 4,800 baud from the server to the client) radio network over 100Base-T Ethernet cabling.

4.2 Evaluation Metrics

The key metrics assessed by our experiments were:

1. Timeliness of image download, which is measured in terms of whether the entire image was downloaded and displayed before an advertised deadline relative to the time of the image request from the application.
2. Quality of the downloaded image, which is measured in terms of the compression ratios of the image tiles, compared to the uncompressed version of each tile.
3. Scalability of the resource management approach, which is measured in terms of the overheads of specific mechanisms in the critical path of the resource management services, *i.e.*, the QuO infrastructure, the RT-ARM service, and the TAO Reconfigurable Scheduler.

The first two metrics assess the ability of the OEP to manage multiple QoS properties simultaneously, as perceived by the collaborative mission re-planning application. The third metric assesses the underlying middleware infrastructure itself.

In addition to studying our overall resource management approach, we also sought to examine the relative contributions of the individual mechanisms. In particular, we sought to isolate the impacts of mechanisms for (1) end-to-end reactive image compression management and (2) client-side reactive rescheduling of tile processing operation rates.

4.3 Experiment Design

Our experiments were conducted using the server and client software systems developed for the OEP evaluations, including a representative Operational Flight Program (OFP) on the client aircraft and a remote imagery server. Resource management was conducted primarily on the client side, which is where we focused the bulk of our analysis. We ran each experiment using the client and server system terminals in that laboratory and ran each set of trials over a range of download deadlines. Each experiment consisted of requesting a virtual folder containing compressed thumbnail renditions of the actual images being downloaded from the server. When the virtual folder arrived at the client, it then immediately requested four images in succession from the server.

Within each experiment, the same trial was then repeated with different deadlines, except for the case of experiments without adaptation where instead we set the compression ratio explicitly, and measured the download time at each of 3 fixed image compression ratios, *i.e.*, 50:1, 75:1, and 100:1. Compression ratios of 50:1 and 100:1 were selected by Boeing system engineers as upper and lower boundaries of image quality for the experiment. There was no noticeable degradation in image quality below 50:1 compression (thus making it a baseline calibration point for adaptation), while degradation was significant at 100:1. Due to time and cost constraints, we did not seek to examine the effects of different characteristics of the images themselves, but instead experimented with an assortment of images so that we could (1) quantify performance of the adaptation techniques over a range of image effects and (2) determine preliminary indications of sensitivity to image makeup for future study.

In the experiments, processing is initiated by transmission of an Alert from the server to the client, followed by a virtual folder with two thumbnail images. Each thumbnail serves as an additional icon to distinguish that image from the others in the virtual folder. For evaluating the performance of the OEP adaptation architecture we confine our attention to the images themselves, though for completeness we also measured thumbnail download latencies and present them in Section 5.

To assess the viability of the individual QoS adaptation technologies and the overall OEP architecture, we ran four experimental trials:

Trial 1: No adaptation of compression or scheduling. We first benchmarked the OEP application performance without adaptation to establish a baseline against which we measure improvement for the three other experiment trials. We measured the download time of each of the 4 images at each of three compression ratios (50:1, 75:1, and 100:1). We note that the available compression ratios used in all the trials defined the range of *perceivable* image quality, which decreases monotonically as image compression increases over the range from 50:1 to 100:1.

Trial 2: Reactive compression and scheduling adaptation. We then measured the OEP system with adaptation of both image compression parameters and operation scheduling parameters. We instrumented the system to record the (1) end-to-end performance of the application, (2) performance of particular segments of the data and computation paths affecting end-to-end performance, and (3) overhead for key adaptation mechanisms in the infrastructure.

Trial 3: Reactive compression adaptation only. To assess the relative contributions of compression vs. scheduling adaptation, we ran the same set of experiments used in the second set of trials, but with scheduling adaptation disabled. The need for this set of experiments was reinforced late in the system development phase when Boeing engineers noticed the contribution of scheduling adaptation to end-to-end performance was not evident in the Boeing Windows NT-based Desktop Test Environment (DTE). As the results in Section 5 reveal, this was solely an artifact of the non-real-time performance of the DTE, *i.e.*, when the VxWorks real-time OS was used in the ground and flight demo environments, the contribution of scheduling adaptation to end-to-end timeliness became clear.

Trial 4: Linear control law experiments. During the experiments in the AIC laboratory, we noticed that the reactive style of compression adaptation used in the system design was resulting in very coarse-grained transitions in the image tile compression ratios, albeit with the resulting performance being suitable to the specific collaboration application. To further explore the potential applicability of our adaptation technologies outside the particular application studied, we conducted a narrowly focused set of experiments to examine the responsiveness of the OEP evaluation system to finer-grained image tile compression management.

In each trial the image was divided into 16 tiles, which were sent from the region of interest outward. For each tile, a message was sent from the client to the server with a request for the tile to be sent at a given compression ratio. The server selected the closest achievable compression ratio to that requested, transmitted the tile to the client, and recorded the ratio actually used. When the tile was received by the client, it was queued pending processing by a decompression operation, which then decompressed the tile and delivered it via an image transfer operation to the IPM for display on the client.

Since imagery tiling was done from the point of interest and radiating outward, the net effect of the reactive adaptation policy was to show the largest possible area around the point of interest at highest quality and then degrade the remaining tiles as a step function to a lower resolution. While this approach is suitable for our avionics application, it is reasonable that other applications (such as opportunistic recognition of features from real-time imagery) might show less bias toward a particular single location in an image, and thus could benefit from maximizing the quality of all tiles. We therefore experimented with replacing the reactive tile compression adaptation strategy encoded in the QuO contract with a simple controller that sought to minimize image tile compression while still meeting the image download deadline. When each tile was received, the controller calculated a new minimum feasible compression ratio based on the image deadline and the download progress to that point.

All of the experiments conducted in this work were designed to provide a first quantitative picture of the strengths and weaknesses of our multi-layered adaptive QoS management approach, while respecting the schedule and budget constraints of the OEP program itself. An even broader set of experiments would be appropriate and in fact are motivated by the results shown in Section 5, as we discuss in Section 8. For these experiments, we found that 38, 42, 46, 50, 54, and 58 seconds represented a covering set of image download deadlines for the trials with both compression and scheduling adaptation, and ran only those deadlines for the two remaining trials with compression adaptation but not scheduling adaptation.

5 Empirical Results

This section presents the results of the four experimental trials described in Section 4.3. We first examine baseline end-to-end image latencies for images compressed at the fixed ratios of 50:1, 75:1, and 100:1. We next present latencies when using the adaptation techniques described in Section 3. After this, we examine image tile compression adaptation response under different strategies and present image tile queueing latencies measured on the client node. We then explore the overhead of the adaptation techniques. Finally, we present overhead results for adaptive rescheduling of operation rates using the integrated RT-ARM and TAO Reconfigurable Scheduler described in Section 3.3.

End-to-end image latency at fixed compression ratios. We first examine the total time from initial request to receive and process each image, as well as the arrival latency of the two initial thumbnail images downloaded in the virtual folder. In the following analysis we focus our attention on the download times for the entire images. For completeness, however, we include fixed-compression download times for the thumbnails. From Trial 1, Figure 6 shows quantitative points of reference for the image download times achievable at the endpoints (50:1 and 100:1) and midpoint (75:1) of the image quality range in the OEP evaluation system. We use these points of reference to compare results achieved with different kinds and degrees of adaptation in the other trials, to assess the effectiveness of adaptation in each case and to establish quantitative bounds on the image quality and download time trade-offs achievable by adaptation in the OEP evaluation system. Over the bandwidth-limited radio data link, images compressed at the highest ratio (lowest image quality) of 100:1 took roughly 40 seconds to download (a lower bound on timeliness), and each factor of 25 reduction in the compression ratio (corresponding to improved image quality) cost another 6 to 7 seconds to download the image, thus establishing a baseline for the trade-off between timeliness and compression. We also note latency variations between the images themselves, which

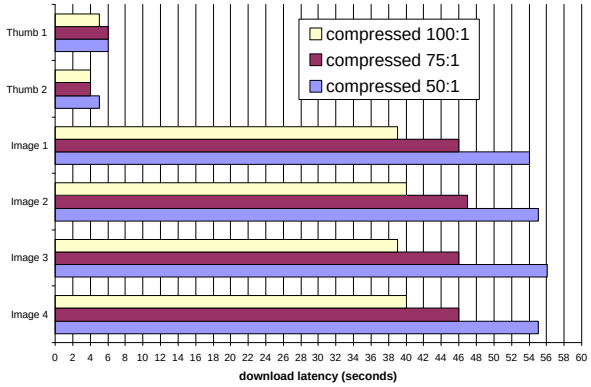

Figure 6: **Image Latency without Adaptation**

are also seen in the other trials.

Image latency with adaptation to specific deadlines. We next compare end-to-end image download times to respective deadlines in the covering set - the respective deadline is shown by the rightmost column in each group. We note that the 38 second deadline is lower than the measured latency for any image downloaded without adaptation at the highest compression ratio of 100:1, so that meeting it is infeasible. Similarly, a deadline of 58 seconds exceeds the maximum latency of any image at the lowest compression ratio of 50:1, and thus does not require any adaptation.

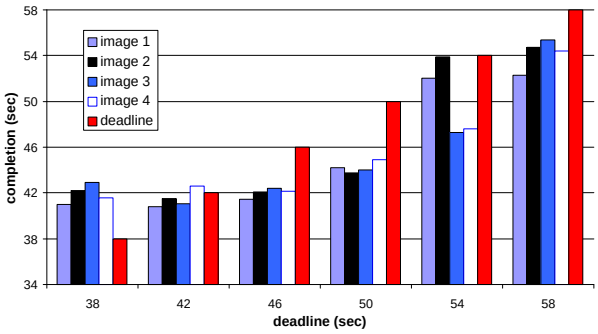

Figure 7: **Compression and Scheduling Adaptation**

From Trials 2 and 3 respectively, Figure 7 and Figure 8 show the end-to-end image download latencies for deadlines of 38, 42, 46, 50, 54, and 58 seconds, both with reactive adaptation of compression ratios; in Trial 2, adaptation of operation invocation rates was also performed, while in Trial 3 it was not. These results demonstrate that compression adaptation alone is insufficient to ensure key deadlines are met. Even with adaptation of both image tile compression and operation invocation rates, however, the additional overhead of adaptation can make tight deadlines (*e.g.*, 42 seconds) infeasible even though without adaptation they are (barely) achievable. Interestingly, the benefit of adding adaptation of operation invocation rates outweighs its cost even with tight deadlines, *e.g.*, Figure 7 and Figure 8 show that more images made the 42 second deadline with adaptation of operation invocation rates than without such rate adaptation.

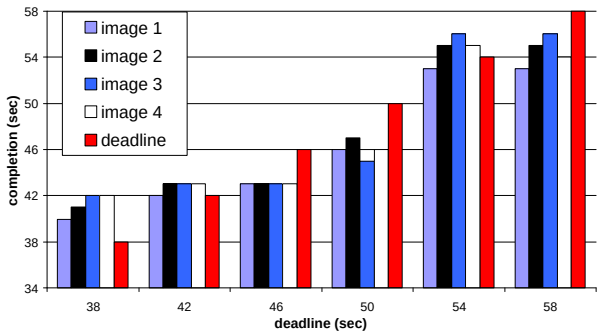

Figure 8: **Compression Adaptation Only**

Image compression adaptation response. We now consider the recorded image tile compression levels in each of the trials. In the cases where the sequence of compression ratios was the same for more than one deadline in a given tile, we plot only the latest deadline of each such equivalent set.

In Trial 3, we confined our attention to control of image tile compression only. It is therefore most appropriate to compare the experiments with simple compression control in Trial 4 to those in Trial 3. Since the scheduling adaptation mechanisms in the RT-ARM were deactivated in both experiments, the effects of scheduling adaptation are suppressed, letting us focus on the effects of compression control in isolation.

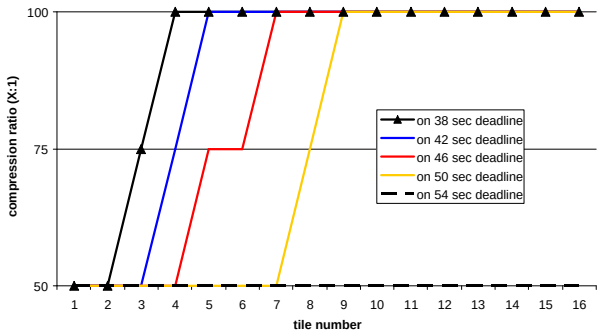

Figure 9: **Reactive Compression Adaptation**

From Trial 3, Figure 9 shows the compression level for each of the 16 tiles of image 1 for deadlines of 38, 42, 46, 50, and 54 seconds, with reactive adaptation of both compression ratios and scheduling. Similar results were observed for other images with this strategy and for all images with the strategy with reactive adaptation of both compression ratios and operation invocation rates. From Trial 4, Figure 10 shows the compression for each tile of image 1 for deadlines of 38, 42, 46, 50, 54, and 58 seconds, with simple control of compression, but no scheduling adaptation.

These results show that although it is possible to adapt image download times effectively at coarse-granularity in

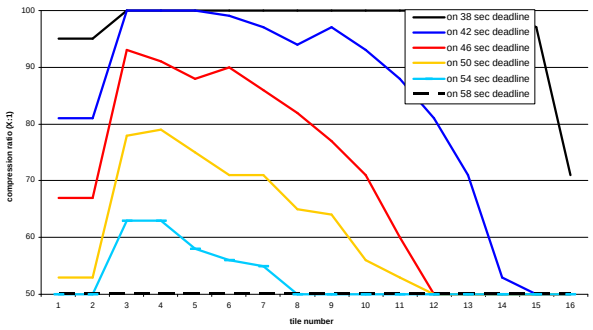

Figure 10: **Compression with Simple Control**

the compression ratios (100:1, 75:1, and 50:1), the OEP is amenable to much finer-grained compression adaptation management. This is a particularly important result when we consider the excess laxity observed at the 46 and 50 second deadlines in Figure 7. In particular, some of the time by which each image arrived early could potentially be traded back for finer-grained improvements in image quality in practice.

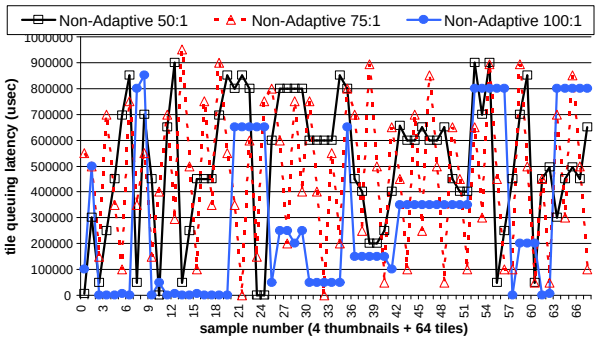

Figure 11: **Tile Queuing Latency w/o Adaptation**

Client-side image tile queueing latency. Upon receipt from the network, each tile sent by the server is stored in a queue on the client until it is retrieved from the queue by the tile decompression operation. The rate at which the decompression operation is invoked, and thus at which tiles are retrieved from the queue was fixed at 1 Hz in Trials 1, 3, and 4, and managed adaptively in Trial 2. From Trial 1, Figure 11 shows tile queueing latencies without adaptation of either compression ratios or operation invocation rates. Similar queueing latencies were seen in Trials 3 and 4, where no adaptation of invocation rates was done. From Trial 2, Figure 12 shows the tile queueing latencies measured on the client with reactive adaptation of both compression and operation invocation rates.

These results identify the client-side tile receive queue as a crucial stage of the end-to-end QoS performance model for the OEP. They also highlight the importance of adaptively managing client-side tile processing operations. Adjusting the rates at which those operations are run significantly decreases the time image tiles spend idly in the queue.

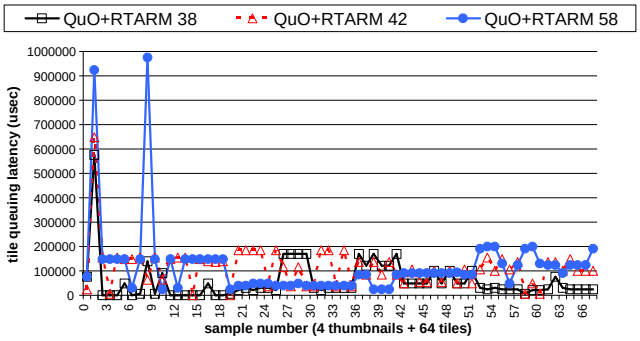

Figure 12: **Tile Queuing Latency with Adaptation**

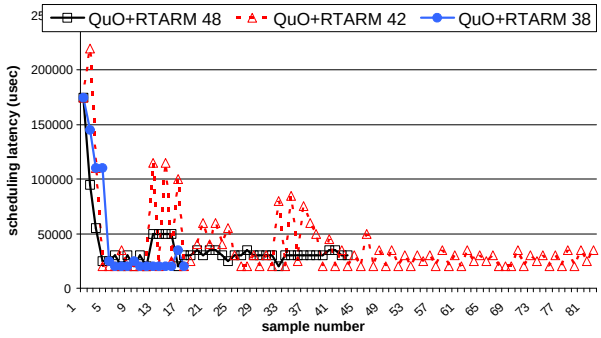

Figure 13: **Adaptive Schedule Computation Latency**

Scheduler re-computation latency under RT-ARM management. Our next area of study was the measurement of schedule re-computation overhead resulting from narrowing of rate ranges by the RT-ARM and priority and rate reassignment by the TAO Reconfigurable Scheduler. From Trial 2, Figure 13 plots schedule re-computations with adaptation of both compression and scheduling, at deadlines of 48, 42, and 38 seconds. The key insight from these results is that the number and duration of re-scheduling computations is both (1) reduced overall compared to our earlier results in the ASTD program [4] and (2) proportional to the degree of rate adaptation that is useful and necessary for each deadline. All trials showed an initial schedule computation time identical to the initial schedule computation times without rate adaptation.

Overhead of QoS management mechanisms. In addition to examining the performance of the application as a whole, we quantify overhead of the individual adaptation services, for preliminary evaluation of scalability and possible optimization, and to guide further expansion of our resource management approach to both systems with constraints at smaller time scales and larger-scale systems of systems. Table 1 summarizes these results. These results suggest scalability of our approach will be reasonably good overall. It is important to note that the timing capabilities of the VxWorks OS where these experiments ran was only accurate to within 5 ms, which is relevant to the overhead measurements in Table 1, many of which are in the range of 0 to 10's of ms.

Table 1: QoS Management Latency

Mechanism	Trial 2	Trials 1, 3, 4
QuO Contract	0 - 30 msec	0 - 10 msec
Region Transition	0 - 10 msec	¡ 5 msec
QuO Delegate	0 -20 msec	0 - 5 msec
RT-ARM Triggering	0 - 10 msec	N/A
Initial Schedule	185 msec	N/A

6 Lessons Learned from Empirical Studies

This section summarizes the implications of the empirical results presented in Section 5 and describes the key lessons learned from our experiments with the multi-layered adaptive middleware techniques presented in Section 3.

Adaptation of both tile compression and operation rates improves timeliness, but at some overhead cost. As shown in Figure 7, image 4 missed the 42 second deadline by a small margin with adaptation of both compression ratios and operation scheduling. The same image missed that deadline with all of the adaptive strategies, however, even though this deadline is achievable with a fixed compression ratio of 100:1 as shown in Figure 6. Imprecision of the adaptation strategies contributed to missing the deadline, *i.e.*, reactive adaptation always started with the first two tile requests being at the lowest compression ratio of 50:1 and control adaptation started at a lower compression ratio (and finished at a lower compression ratio after the deadline was missed).

We surmise that the overhead of adaptation - though small - contributed to the difficulty in attaining this deadline. It is possible that a variation on the adaptation strategy would exhibit better results in similar situations. For example, while our adaptation policy could degrade all but the initial tiles containing the area of interest, it did not consider dropping any of the later tiles. The tightest feasible deadlines, *i.e.*, 42 seconds, could only be met by compressing the whole image at 100:1 as Figure 6 shows. With looser deadlines, however, it might be preferable to get the first tiles at high quality and drop the last few tiles rather than degrade the whole image.

Choice of adaptation strategy is important. Overall, the strategy without scheduling adaptation sent fewer tiles at the lowest compression ratio of 50:1 before changing to the highest compression ratio of 100:1. This effect reflects an attempt by the strategy to compensate for fixed rates of tile processing operations. This strategy was somewhat (but not entirely) successful per the latency-to-deadline comparison in Figure 8.

The principal feature of interest with the simple control strategy is the more continuous arc of the compression levels shown in Figure 10, in contrast to the coarser-grained transitions shown in Figure 9. In future work the linear interpolation used by the controller itself should be replaced with the more rigorous control-theoretic approach described in Section 8. The experimental application and supporting middleware infrastructure appear to be amenable to fine-grained adaptation, however, as shown by the fairly continuous response of the image tile management infrastructure.

Operation rate adaptation reduces image tile queueing latencies. The main feature of interest in the image tile queueing measurements on the client is the much larger magnitude and jitter of queueing latencies without adaptation seen in Figure 11, compared to Figure 12, which shows tile queueing measurements for the strategy with adaptation of both compression ratios and tile processing operation scheduling parameters. The other two strategies without scheduling adaptation (*i.e.*, with reactive adaptation or simple control of image tile compression only) showed similar results to those without any adaptation at all, which singles out operation scheduling adaptation as a key contributor to end-to-end QoS. It is especially interesting that improvements were seen in both the precision and tightness of the latency bound - operation rate adaptation can therefore give increased confidence in how close to that bound we can come in improving image quality without risking missed deadlines.

Overhead for adaptive QoS management is acceptable. The first feature of interest for the overhead results reported in Table 1 is the relatively low latency of QuO contract evaluation, region transitions, and delegate processing. With scheduling adaptation, contract evaluations had the highest latencies but were bounded by 30 msec, and most of these evaluations took much less time than that. Without scheduling adaptation, the latencies are bounded by 10 msec and the common case is that the latencies are negligible. The version of QuO used for these experiments was designed for predictable low-latency response in DRE systems [11], and our results confirm the efficacy of that design.

The second feature of interest in these results is the difference in contract evaluation latency between these two strategies. Due to the low latencies seen with adaptation of compression only, we suspect that much of the increased latency seen when scheduling adaptation is added arises from preemption by OFP operations. We also observed an increased number of contract evaluations with rate adaptation enabled, however, so further studies are motivated to assess relative scalability in terms of both load and responsiveness.

We also note the relatively low latency of RT-ARM triggering operations, bounded by 10 msec, so that in concert the QuO and RT-ARM adaptation mechanisms imposed suitably low overheads. When computing the initial assignment of priorities and rates to operations, the TAO Reconfigurable Scheduler showed highly predictable timing of 185 msec. With the

same initial set of scheduling parameters when no scheduling adaptation was involved, there was one invocation of the scheduler at system initialization. We note that in comparison to the latency of other adaptation mechanisms, initial schedule computation latency is an order of magnitude greater. However, the optimizations described in Section 3.3 significantly reduce the post-initialization cost of rescheduling, which we consider next.

Scheduler re-computation latency under RT-ARM management is reasonable. The main feature of interest in Figure 13 is the downward settling of schedule computation times toward a similar level of overhead as for QuO contract evaluation, as RT-ARM narrowed ranges of available rates and thus reduced the input set over which the scheduler performs its computation. We also observed an interesting phase transition in the number of re-computations between the infeasible and barely feasible deadlines. If we arrange trials in descending order according to the number of re-computations in each, we get 42, 46, 48, 50, 52, 54, and then 58 seconds, and then finally 38 second and 1 second deadlines showed the same minimal number of computations. The duration of the experiment for the 42 second deadline was comparable to that for other deadlines.

Our results show that RT-ARM was in fact triggering scheduling adaptation more frequently. As Section 5 showed, 42 seconds was the most difficult feasible deadline, with image 4 missing that deadline in all of the adaptive strategies. RT-ARM performed very little adaptation for infeasible deadlines since this is futile. It did perform more adaptation for feasible deadlines, however, but at a level proportional to the difficulty of achieving the deadline.

7 Related Work

This section describes related work on QoS management middleware technologies. We first summarize two projects that are representative of earlier foundational research on QoS management frameworks. We then describe several recent projects related to our work, in which results of earlier work on QoS management have been abstracted into modeling tools, made configurable in QoS-aware component technologies, and woven at finer granularity and across a variety of levels throughout complex DRE systems.

Previous focus: QoS management middleware frameworks. A number of earlier projects developed self-contained QoS frameworks to manage end-to-end QoS in distributed systems. These efforts set the stage for subsequent work on finer-grained integration of QoS management mechanisms and policies. Two major examples of those foundational research efforts are the Realize and ARMADA projects.

• **UCSB Realize.** The Realize project at UCSB [6] supports soft real-time resource management of CORBA distributed systems. Realize integrates distributed real-time scheduling with fault-tolerance, fault-tolerance with totally-ordered multicasting, and totally-ordered multicasting with distributed real-time scheduling, within the context of OO programming and existing standard operating systems. The Realize resource management model can be hosted on top of TAO [6].

• **ARMADA.** The ARMADA project [20, 7] defines a set of communication and middleware services that support fault-tolerant and end-to-end guarantees for real-time distributed applications. ARMADA provides real-time communication services based on the X-kernel and the Open Group's MK microkernel. This infrastructure provides a foundation for constructing higher-level real-time middleware services.

New focus: QoS aspect integration. Recent work on end-to-end QoS management has focused on integrating multiple QoS aspects end-to-end throughout complex DRE systems. Research is being conducted on several related fronts, including integration of systemic QoS aspects and QoS-aware component models. The following projects are representative examples of a larger and rapidly growing field of research.

• **dynamicTAO.** In their dynamicTAO project, Kon and Campbell [21] apply reflective middleware techniques to extend TAO to reconfigure the ORB at runtime by dynamically linking selected modules, according to the features required by the applications. Their work is similar to QuO in that both provide the mechanisms for realizing *dynamic* QoS provisioning at the middleware level. QuO offers a more comprehensive QoS provisioning abstraction, however, whereas Kon and Campbell's work concentrates on configuring *middleware* capabilities.

• **QoS-enabled component middleware.** Middleware can apply the Quality Connector pattern [22] to meta-programming techniques for specifying the QoS behaviors and configuring the supporting mechanisms for these QoS behaviors. The container architecture in component-based middleware frameworks provides the vehicle for applying meta-programming techniques for QoS assurance control in component middleware, as previously identified in [23]. Containers can also help apply aspect-oriented software development [24] techniques to plug in different systemic behaviors [25]. Miguel de Miguel further develops the work on QoS-enabled containers by extending a QoS EJB container interface to support a `QoSContext` interface that allows the exchange of QoS-related information among component instances [26].

8 Concluding Remarks

This paper describes and quantifies the integration of several adaptive middleware technologies, including QuO, RT-ARM, and several layers of The ACE ORB (TAO), including its scheduling and event services capabilities. The paper's R&D contributions involve (1) presenting an architecture for multi-layer adaptive middleware that is applicable to QoS-managed DRE systems, (2) applying that middleware architecture to a real-world avionics DRE application case study, and (3) conducting and analyzing empirical results that quantify the benefits and costs of this architecture for the avionics environment.

Our analysis of the overheads associated with key QoS management mechanisms offers insight into the types of evolution and further experiments needed to apply our approach in other contexts, such as QoS-managed real-time audio/video streaming for disaster response, coordination of multiple autonomous vehicles, and optimization of sensing-decision-actuation paths in industrial control systems. For example, the latencies seen in the face of preemption by other operations establish bounds on responsiveness of adaptation. These bounds, however, to not imply limitations on the ability of the computational and adaptation load to scale with the resources made available. In particular, for system-of-systems approaches (of which the avionics OEP described here is a canonical example), it is reasonable to expect the resources supplied by additional platforms integrated with the system would be commensurate with the new demands they place on the system.

Careful analysis and integration of mechanisms across service boundaries are crucial to achieving necessary performance of adaptive middleware for DRE systems, *e.g.*, the reengineered interaction between the RT-ARM and TAO Reconfigurable Scheduler described in Section 3.3 shows improved performance over that observed previously [27] in a simple integration of those technologies. The settling effect in the adaptation performed by the RT-ARM in conjunction with TAO's Reconfigurable Scheduler (and the questions it raises about the stability of particular schedules with respect to additional adaptation) is intriguing and worthy of further study. These results demonstrate that we have achieved a more efficient mechanism for interactive scheduling adaptation, but now need to focus new work on the policies for, and environmental influences on, that interaction. We have begun investigating implementing various control strategies within our adaptive middleware framework [28, 29].

The main conclusions we draw from the results in this paper are that

- Our integrated QoS management middleware showed successful adaptation of multiple QoS parameters, with a quantitative improvement in managing trade-off between image quality and download times in comparison to the same approach without adaptation.
- Factors in DRE system environments are important and can have a significant impact on the behavior of the system, *e.g.*, our trials revealed interesting latency effects due to interactions (notably preemption) between avionics mission computing operations and the adaptation infrastructure mechanisms.
- Case studies of actual DRE systems provide valuable insights on the maturity and capability of adaptive middleware, *i.e.*, it is an important achievement to have flown and measured the OEP evaluation system in a representative avionics mission-computing context.

Our future work will expand these studies to examine the performance effects of image contrast and size (*e.g.*, to determine why image 3 took longer to download at a compression ratio of 50:1 than any of the other images, and yet took less time to download at a compression ratio of 100:1 than either image 2 or 4), network latency, and traffic loads.

References

[1] Object Management Group, *Minimum CORBA - Joint Revised Submission*, OMG Document orbos/98-08-04 ed., Aug. 1998.

[2] Object Management Group, *Real-Time CORBA Specification*, 1.1 ed., Aug. 2002.

[3] Bollella, Gosling, Brosgol, Dibble, Furr, Hardin, and Turnbull, *The Real-Time Specification for Java*. Addison-Wesley, 2000.

[4] 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.

[5] A. Gokhale, D. C. Schmidt, B. Natarajan, and N. Wang, "Applying Model-Integrated Computing to Component Middleware and Enterprise Applications," *The Communications of the ACM Special Issue on Enterprise Components, Service and Business Rules*, vol. 45, Oct. 2002.

[6] V. Kalogeraki, P. Melliar-Smith, and L. Moser, "Soft Real-Time Resource Management in CORBA Distributed Systems," in *Proceedings of the Workshop on Middleware for Real-Time Systems and Services*, (San Francisco, CA), IEEE, Dec. 1997.

[7] T. Abdelzaher, S. Dawson, W.-C.Feng, F.Jahanian, S. Johnson, A. Mehra, T. Mitton, A. Shaikh, K. Shin, Z. Wang, and H. Zou, "ARMADA Middleware Suite," in *Proceedings of the Workshop on Middleware for Real-Time Systems and Services*, (San Francisco, CA), IEEE, Dec. 1997.

[8] C. Gill, D. C. Schmidt, and R. Cytron, "Multi-Paradigm Scheduling for Distributed Real-Time Embedded Computing," *IEEE Proceedings, Special Issue on Modeling and Design of Embedded Software*, vol. 91, Jan. 2003.

[9] D. C. Sharp, "Reducing Avionics Software Cost Through Component Based Product Line Development," in *Proceedings of the 10th Annual Software Technology Conference*, Apr. 1998.

[10] D. Corman, J. Gossett, and D. Noll, "Experiences in a Distributed Real-Time Avionics Domain," in *Proceedings of the International Symposium on Object-Oriented Real-time Distributed Computing (ISORC)*, (Washington, D.C.), IEEE/IFIP, Apr. 2002.

[11] D. A. Karr, C. Rodrigues, Y. Krishnamurthy, I. Pyarali, and D. C. Schmidt, "Application of the QuO Quality-of-Service Framework to a Distributed Video Application," in *Proceedings of the 3rd International Symposium on Distributed Objects and Applications*, (Rome, Italy), OMG, Sept. 2001.

[12] 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.

[13] 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, pp. 1–20, 1997.

[14] 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.

[15] 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.

[16] J. Loyall, J. Gossett, C. Gill, R. Schantz, J. Zinky, P. Pal, R. Shapiro, C. Rodrigues, M. Atighetchi, and D. Karr, "Comparing and Contrasting Adaptive Middleware Support in Wide-Area and Embedded Distributed Object Applications," in *Proceedings of the 21st International Conference on Distributed Computing Systems (ICDCS-21)*, pp. 625–634, IEEE, Apr. 2001.

[17] 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), pp. 184–199, ACM, Oct. 1997.

[18] C. D. Gill, D. L. Levine, and D. C. Schmidt, "The Design and Performance of a Real-Time CORBA Scheduling Service," *Real-Time Systems, The International Journal of Time-Critical Computing Systems, special issue on Real-Time Middleware*, vol. 20, Mar. 2001.

[19] Joseph K. Cross and Patrick Lardieri, "Proactive and Reactive Resource Reallocation in DoD DRE Systems," in *Proceedings of the OOPSLA 2001 Workshop "Towards Patterns and Pattern Languages for OO Distributed Real-time and Embedded Systems"*, Oct. 2001.

[20] 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.

[21] F. Kon, F. Costa, G. Blair, and R. H. Campbell, "The Case for Reflective Middleware," *Communications ACM*, vol. 45, pp. 33–38, June 2002.

[22] J. K. Cross and D. C. Schmidt, "Applying the Quality Connector Pattern to Optimize Distributed Real-time and Embedded Middleware," in *Patterns and Skeletons for Distributed and Parallel Computing* (F. Rabhi and S. Gorlatch, eds.), Springer Verlag, 2002.

[23] N. Wang, D. C. Schmidt, M. Kircher, and K. Parameswaran, "Towards a Reflective Middleware Framework for QoS-enabled CORBA Component Model Applications," *IEEE Distributed Systems Online*, vol. 2, July 2001.

[24] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J.-M. Loingtier, and J. Irwin, "Aspect-Oriented Programming," in *Proceedings of the 11th European Conference on Object-Oriented Programming*, June 1997.

[25] D. Conan, E. Putrycz, N. Farcet, and M. DeMiguel, "Integration of Non-Functional Properties in Containers," *Proceedings of the Sixth International Workshop on Component-Oriented Programming (WCOP)*, 2001.

[26] M. A. de Miguel, "QoS-Aware Component Frameworks," in *The* 10^{th} *International Workshop on Quality of Service (IWQoS 2002)*, (Miami Beach, Florida), May 2002.

[27] C. D. Gill, R. Cytron, and D. C. Schmidt, "Middleware Scheduling Optimization Techniques for Distributed Real-Time and Embedded Systems," in *Proceedings of the* 7^{th} *Workshop on Object-oriented Real-time Dependable Systems*, (San Diego, CA), IEEE, Jan. 2002.

[28] Abdelwahed, Neema, Loyall, and Shapiro, "Multilevel Online Hybrid Control Design for QoS Management," in *Proceedings of the Real-Time Systems Symposium (RTSS)*, (Cancun, Mexico), IEEE, Dec. 2003.

[29] C. Lu, X. Wang, and C. Gill, "Feedback Control Real-Time Scheduling in ORB Middleware," in *Proceedings of the 9th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, (Washington, DC), IEEE, May 2003.