Integrated Adaptive QoS Management in Middleware: A Case Study[1]

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

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

Joseph P. Loyall, Richard E. Schantz, and Michael Atighetchi
BBN Technologies, Cambridge, MA
{jloyall,schantz,matighet}@bbn.com

Douglas C. Schmidt

Vanderbilt University, Nashville, TN
schmidt@dre.vanderbilt.edu

Abstract

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 an adaptive approach to management of quality-of-service (QoS) to meet key constraints, such as end-to-end timeliness. Moreover, coordinated management of multiple QoS capabilities across multiple layers of applications and their supporting middleware can help to achieve necessary assurances of meeting these constraints.

This paper offers two contributions to the study of adaptive DRE computing systems: (1) a case study of our integration of multiple middleware QoS management technologies to manage quality and timeliness of imagery adaptively within a representative DRE avionics system and (2) empirical results and analysis of the impact of that integration on key trade-offs between timeliness and image quality in that system.

Index terms – *Empirical Case Studies, Distributed Real-Time and Embedded (DRE) Systems, Adaptive Middleware*

1. Introduction

Distributed Object Computing (DOC) middleware has become a widely accepted paradigm for developing numerous applications in a wide variety of environments, including distributed real-time and embedded (DRE) systems and applications. As DOC 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 applications.

Although previous research has shown the benefits of integrating multiple QoS management techniques in standards-based middleware [4] and applying single-layer adaptive resource management techniques real-world DRE systems [5], only limited practical experience is available, however, with integrating resource management techniques *across multiple layers of standards-based DRE systems*. As a step towards filling this gap, this paper presents a case study of the vertical integration of three layers of middleware QoS management technologies [6] within Boeing's Bold Stroke framework, which is a standards-based DRE avionics platform. Bold Stroke is representative of a broader class of DRE applications (including, *e.g.*, mission critical distributed audio/video processing [7] and real-time robotic systems [8]) that require both static and dynamic support for QoS. In this paper, we describe the integration of our three layered QoS management technologies, show results of their use in the Bold Stroke avionics mission computing system, and analyze each technology's contribution to adaptive QoS management.

This paper is organized as follows: Section 2 describes the Bold Stroke avionics system's application context; Section 3 describes each of the three QoS management technologies and examines the issues and optimizations we discovered while integrating them within the avionics system; Section 4 describes architectural modifications to the interaction between the adaptive resource management and scheduling layers, to improve inter-layer adaptation performance; Section

[1] This work was supported in part by AFRL contract F33615-97-D-1155/0005 (WSOA), NSF ITR CCR-0312859, Siemens, and DARPA/AFRL contracts F33615-03-C-4112, F30602-98-C-0187 and F33615-00-C-1694. Approved for public release, distribution unlimited.

5 presents the methodology and overall design of our experiments; Section 6 reports our results, and analyzes trade-offs under different adaptation approaches; Section 7 summarizes the lessons learned from our empirical studies; Section 8 describes work related to our research on middleware QoS management techniques; and Section 9 presents concluding remarks.

2. Application Overview

We conducted our experiments using the Weapons Systems Open Architecture (WSOA) Open Experimentation Platform (OEP) shown in Figure 1. The WSOA OEP consisted of two airborne server and client nodes (a command and control aircraft and an F-15 fighter aircraft respectively) that collaborated over a very low-bandwidth radio data link to re-plan the client's mission parameters in real-time.

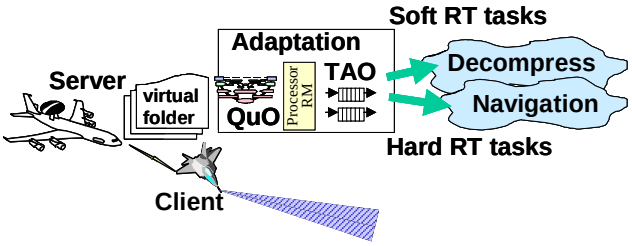

Figure 1: Collaborative Re-planning in WSOA

Collaborative re-planning enables responding more rapidly to situational changes in-flight, *e.g.*, the server (C2 node) sends links to downloadable imagery to the client (F-15 node), which it then uses for re-planning. In the example scenario we used to evaluate the WSOA 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. Personnel on the client and server nodes collaborate to develop a new plan, which the client then performs.

The research described in this paper applies multi-layer adaptive middleware techniques to alleviate key limitations that impede successful mission re-planning:

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

A key goal of the WSOA OEP evaluation system illustrated in Figure 1 is to use *adaptation* to provide the client 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 mission re-planning alerts. Therefore, in addition to providing the client up-to-date information detected by remote sensors (*e.g.*, fresh images of the new destination) and about the environment it will encounter 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 is to implement QoS-managed browser-like collaboration capabilities to (1) enable the client and server nodes to view the same displays and information and (2) ensure image quality and transmission latency stay within acceptable bounds, in a manner that is as independent as possible of the available resources (obviously there is a minimum, below which nothing useful can be accomplished). This common browser view also allows server-side personnel 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 personnel at 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.

2.1. Design and Implementation Overview

In the WSOA OEP application, 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. The collaboration client application (on the fighter aircraft) contains a virtual folder manager component, which 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. Personnel on the client node can navigate the virtual folder 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 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 personnel can then draw annotations and move commonly viewed individual cursors during the collaboration. To avoid problems with having both the server and client 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 personnel 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

Our DOC middleware approach provides an open systems "bridge" between legacy on-board embedded avionics systems and off-board information sources and systems. The foundation of this bridge is a Real-time CORBA Object Request Broker (ORB) [2] using a pluggable protocol to communicate over a very low bandwidth (approximately 2,400 baud in each direction) Link-16 tactical data network. Link-16 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.

We have applied middleware technologies 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 WSOA OEP evaluation system 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 [9] that supports re-use of application components across multiple client platforms. The OFP architecture includes the Bold Stroke avionics domain-specific middleware layer [10] built upon The ACE ORB (TAO) [11], a widely-used C++ Real-time CORBA implementation available from deuce.doc.wustl.edu/Download.html.

This middleware isolates 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 described in Section 3 to address the limitations with time-sensitive mission re-planning noted at the beginning of this section.

2.3. System Resource Management Model

The resource management model for the WSOA OEP evaluation system is illustrated in Figure 2. When client personnel request an image, that request is sent from the *browser application* to a QuO *application delegate* [9], which then sends a series of requests for individual tiles via TAO over a low-bandwidth Link-16 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, determined by the progress of the overall image download when the request is made.

On the server, the *ORBExpress* Ada ORB [12] receives each request from the Link-16 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 Link-16 to the client. Server-side environmental simulation services emulate additional workloads that would be seen on the command and control (C2) server under realistic operating conditions.

Back on the client, each compressed tile is received from Link-16 by TAO and delivered to a servant that

places the tile in a queue where it waits to be decompressed. The tile is removed from the queue, 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 [13] at rates selected in concert by the RT-ARM [14] and the TAO Reconfigurable Scheduler [5][15], as described in Sections 3.2 and 3.3, respectively.

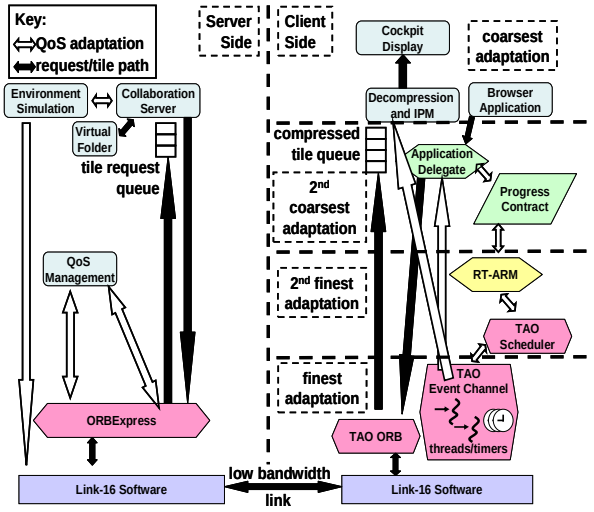

Figure 2: Resource Management Model

3. Overview of Adaptive Middleware

To address the challenges described in Section 2, we have designed, implemented, and flight-tested an integrated multi-layered QoS enforcement architecture based on the Real-time CORBA standard. 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 and hardware). To enhance performance, our architecture tries to handle adaptation at the lowest layer possible, moving up to higher layers only if QoS requirements cannot be met via adaptation in the current layer.

Figure 2 illustrates the resource adaptation architecture of the WSOA OEP evaluation platforms and middleware. The *finest* granularity of adaptation in the WSOA system architecture is the lowest priority dynamic scheduling of non-critical operations [5] by the dispatcher of the TAO Real-Time Event Channel, which we developed in previous research [13]. The *second finest* level of adaptation granularity is achieved by a Real Time Adaptive Resource Manager (RT-ARM) [14] and the TAO Reconfigurable Scheduler [5][15], which re-schedule 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 [9], which monitors progress downloading and processing image tiles toward the desired deadline for the entire image.

While 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 *coarsest grain* requirements for image quality and timeliness. The remainder of this section describes each middleware layer outlined above in detail, ranging from the coarsest to the finest granularity of adaptation.

3.1. QuO: 2nd Coarsest Grain Adaptation

QuO is an aspect-oriented middleware framework created by BBN Technologies to support the development of QoS behavior of a system separate from – but in conjunction with – the development of its functional behavior.

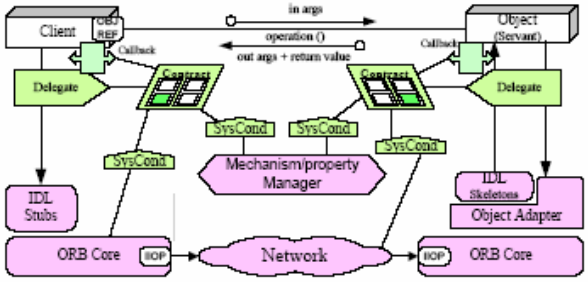

Figure 3: QuO Architecture Overview

The following QuO components are shown in Figure 3 and used in the WSOA OEP test-bed:

1. *Contracts* specify desired and available QoS, along with the policies for controlling QoS and adapting to changes.
2. *Delegates* are remote object proxies, with well-defined points to insert adaptive behaviors into end-to-end paths.
3. *System condition* objects provide interfaces to parts of the system that must be measured or controlled by contracts.

Since QuO is general-purpose framework that can support a variety of adaptation strategies, we developed a *reactive* QoS adaptation policy [16] for the OEP evaluation system that manages the overall trade-offs of timeliness versus image quality. When the client node requests an image from the server node, a QuO dele-

gate 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. 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, with the early tiles containing the most important data, so that decreased quality of later tiles will have minimal impact on the overall mission re-planning capabilities.

In the OEP evaluation system, 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 delegate first 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 evaluation system the choices were limited to 1, 16, or 64 tiles. Our experiments (described in Section 5) 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 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 most likely for the application to download image tiles at compression ratios greater than or equal to that of the region of interest, which is the model we adopted for our experiments described in Section 5.

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 the 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*, and the compression ratio is not already at the highest possible compression of 100:1, the compression ratio is increased by an increment of 25:1 from its current position in the range [50:1, 75:1, 100:1]. After checking progress – and if necessary setting a new compression ratio and notifying the 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: 2nd Finest Grain Adaptation

The RT-ARM is a reactive resource adaptation service developed by Honeywell Technologies and used in the WSOA OEP 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, the 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. The RT-ARM does this by manipulating subsets of task invocation event rates from application-specified available rate sets, as Figure 4 illustrates.

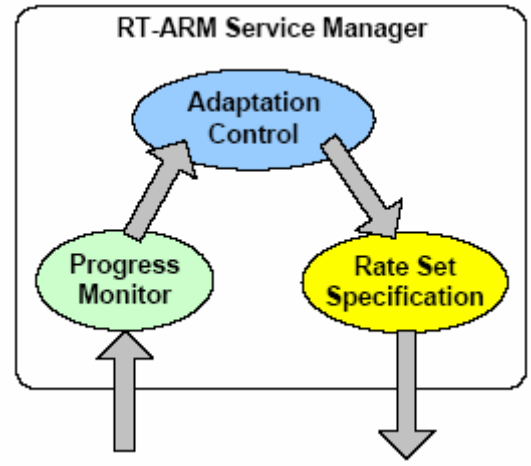

Figure 4: RT-ARM Service

If image tile *processing* falls behind schedule, the QuO contract prompts the RT-ARM to adjust ranges of invocation rates to re-allocate more CPU cycles to decompressing remaining tiles. In response to changing environmental conditions, the RT-ARM can trigger such adaptation in two ways: (1) *reactively* when the QuO contract notifies the 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 the 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.

The 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 the RT-ARM operating region and the compression level of the next tile.

3.3. TAO Reconfigurable Scheduler: 2nd Finest Grain Adaptation

The TAO Reconfigurable Scheduler is a CORBA scheduling service implementation designed for flexible support of hybrid static/dynamic scheduling [5], developed by Washington University, St. Louis. The TAO Reconfigurable Scheduler 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.

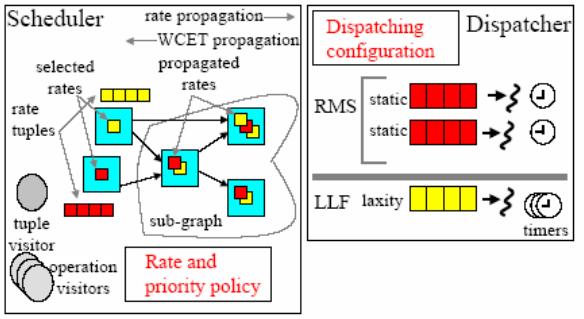

Figure 5: Reconfigurable Scheduler and Event Channel Dispatcher Interoperation in TAO

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. The TAO Reconfigurable Scheduler then adds non-critical processing and optimizes processor utilization for the image processing operations by maximizing their rates subject to schedule feasibility. In this application, operations associated with re-planning are non-critical.

In the earlier Adaptive Software Test Demonstration (ASTD) program [17], we tried a simple integration of the TAO Reconfigurable Scheduler with the RT-ARM, in which the 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. Those results, however, pointed to the solution pursued in this work: closer integration of adaptation mechanisms. We evolved the TAO Reconfigurable Scheduler so that the rate selection *mechanism* was pushed down into it, while the *policy* for rate selection was supplied by the RT-ARM. Specifically, the RT-ARM provided a specific rate selection strategy to the TAO Reconfigurable Scheduler at system initialization time based upon operation criticality and available rates.

We describe the design and implementation of these architectural improvements in detail, in Section 4. These revisions are released in TAO's Reconfigurable Scheduler, which can be downloaded as open-source at

deuce.doc.wustl.edu/Download.html, along with the rest of the TAO middleware.

4. Architectural Improvements to Optimize RT-ARM and TAO Scheduler Interaction

The first revision we made to the TAO Reconfigurable Scheduler for the WSOA OEP case study was to refactor its implementation for greater re-configurability, extending similar efforts started during the ASTD program. The original implementation of hybrid static/dynamic scheduling in TAO used a single recursive algorithm to traverse the graph of operation dependencies. Although this worked well for simple dependency relationships between operations, it was difficult (1) to integrate new actions such as rate and criticality propagation across dependencies, or (2) to select which actions were relevant to – and so should be applied with – different scheduling policies. We therefore refactored the monolithic algorithm to apply different actions as *visitors*, as illustrated in Figure 5.

The use of visitors for different actions greatly simplified implementation of our second revision to the TAO Reconfigurable Scheduler. In the second revision we 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 in turn allows one scheduler to be used for efficient rate selection and priority assignment, all adaptively at run-time. Figure 6 illustrates the four optimizations made to the TAO Reconfigurable Scheduler to support efficient adaptive rescheduling of both operation rates and operation priorities under a range of scheduling and rate selection policies.

A. De-normalized operation descriptors: We de-normalize the available rate set and fixed characteristics for each operation into a sequence of flat tuples of characteristics (containing *e.g.*, the operation handle, a particular rate, the execution time at that rate). We then derive information that facilitates sorting and utilization bounds checking. For example, we specify the index of a tuple within an operation's ordered set of rates, and the utilization difference for an operation between each pair of its consecutively indexed tuples. This optimization can help meet our goal to trade performance of individual elements (*i.e.*, rate of execution) for overall performance objectives (*i.e.*, maximizing the number of feasible operations).

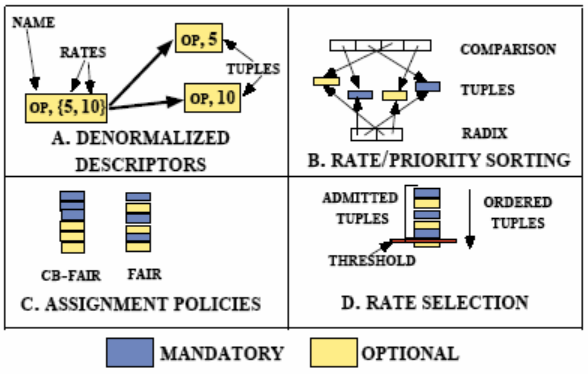

Figure 6: Scheduler Adaptation Optimizations

B. Rate and priority sorting: We recast rate and priority assignment as a sorting problem over operation characteristics, with at worst an $O(n \bullet \log(n))$ bound on worst-case performance, and an $O(n)$ bound on worst-case performance in certain special instances of the more general problem. Since our scheduling approach applies to arbitrary collections of operation characteristics, for some combinations of operations and scheduling strategies an $O(n \bullet \log(n))$ comparison sort may be needed. For our target avionics application, however, all operations are known in advance and the value spaces of the characteristics of interest (*e.g.*, whether an operation is mandatory, and its available periods) are small, so the more efficient $O(n)$ radix sorts are applicable in many cases. This optimization can help meet our system goal to perform adaptive resource reallocations within firmly bounded time-scales.

C. Rate assignment policies: We encapsulate specific sort ordering strategies as policies for rate assignment, much as we have done previously for scheduling policies [15]. To illustrate the range of possible strategies for ordering tuples during rate selection, we present two canonical strategies, based on two different views of fairness:

- **FAIR strategy:** In the first strategy, called *Fair Assignment by Indexed Rate* (FAIR), we emphasize fairness across all operations, ordering tuples by ascending rate index, then descending criticality, then mean rate, and finally (to ensure a total ordering of tuples) by descriptor handle. This strategy selects the lowest rate for each operation, first for mandatory operations and then for optional operations, then the next rate for each mandatory operation and then for each optional operation, and so forth.
- **CB-FAIR strategy:** In the second strategy, called *Criticality-Biased FAIR* (CB-FAIR), we emphasize criticality partitioning, and order tuples first by descending criticality, then by ascending rate index index, then by mean rate, and finally (again to ensure a

total ordering of tuples) by descriptor handle. This optimization adds flexibility to meet our goal to improve real-time performance across heterogeneous criteria, *i.e.*, both rate and criticality.

D. Rate Selection: Once the tuples are sorted, we perform a single O(n) traversal of the tuples to select the rate of each operation and determine expected utilization values based on the rates selected and the advertised execution times. As we iterate through the sorted tuples, we maintain variables for (1) the total utilization by mandatory operations, and (2) the total utilization by all operations, based on the tuples selected so far. A tuple is selected if and only if the additional utilization, compared to the utilization for the previously admitted tuple for that operation, will still fit within the utilization threshold associated with that tuple. The highest rate of any tuple selected for an operation becomes the assigned rate for that operation. This optimization can also help meet our goals to trade performance of individual elements for overall real-time objectives, and to perform adaptive resource reallocations within firmly bounded time-scales.

5. Methodology for Empirical Studies

This section introduces the objectives and approach to a set of adaptive middleware experiments completed during post-flight ground tests of the WSOA OEP in January 2003, which followed the actual flight tests conducted in December 2002. The four primary goals of our experiments were (1) to quantify the ability of multiple layered QoS management mechanisms within the Bold Stroke middleware framework to maximize image fidelity while meeting download deadlines, (2) to offer preliminary assessment of the relative contributions of the different QoS management mechanisms outlined above, (3) to profile the temporal performance of those mechanisms, and (4) to quantify the relative benefits of this approach compared to the same application running without adaptation.

We note that perceivable image quality decreases monotonically as image compression increases over the range from 50:1 to 100:1. Moreover, our assessment of the compression quality achieved for a given image is weighted by whether or not it met its deadline. These experiments also measure trade-offs between timeliness and image quality in a relatively sanitary system environment, to remove all influences outside the scope of the metrics considered here. In doing so, we established a baseline against which realistic parameters (*e.g.*, network latency jitter, traffic loads, or other factors) can be varied in a managed way and *their* contributions to system behavior also quantified.

Section 5.1 first introduces the metrics we used to evaluate the OEP architecture. Section 5.2 then describes the design of the experiments themselves, grouped into the following four distinct studies of adaptive QoS management: (1) the OEP system with no adaptation (which serves as an experimental baseline), (2) the QoS management approach described in Section 3, with reactive adaptation of both image compression levels and scheduling (rates and priorities) of image tile processing operations, (3) the same approach but with scheduling adaptation turned off, and (4) a simple control-based approach to image compression adaptation that explored the system's response to this kind of control. Finally, Section 5.3 describes the platform on which the experiments were run. The results of these experiments are presented in Section 6.

5.1. Evaluation Metrics

The key metrics assessed by our experiments were:

1. *Timeliness of image download*, *i.e.*, 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* in terms of the compression ratios of the image tiles, compared to the uncompressed version of each tile, and
3. *Scalability of the resource management approach*, 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, while 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.

5.2. Experiment Design

Our experiments were conducted using the server and client software systems developed for the WSOA OEP evaluations, including a representative Operational Flight Program (OFP) on the F-15 fighter airplane client and a representative imagery server on the command and control (C2) airplane. Resource management

was conducted primarily on the client side, which is where we have focused the bulk of our analysis.

The experiments were run on realistic hardware in the Avionics Integration Center (AIC) laboratory at Boeing, St. Louis. 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 thumbnails 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) give 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 WSOA adaptation architecture we confine our attention to the images themselves, though for completeness we also measured thumbnail download latencies and present them in Section 6.

To assess the viability of the individual QoS adaptation technologies and the overall WSOA architecture, we ran the four experiment trials described below. 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 a tile was received by the client, it was queued pending processing by an operation which decompressed the tile then delivered it via an image transfer operation to the IPM for display on the client.

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. We therefore ran only those deadlines for the two remaining trials with compression adaptation but not scheduling adaptation.

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).

Trial 2: Reactive Compression + 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 turned off. 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 6 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 environments, the contribution of scheduling adaptation to end-to-end timeliness became clear.

Trial 4: Linear Control Law Experiments. We noticed that the reactive style of compression adaptation used in the system design resulted 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 applicability of our approach 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.

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, 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.

5.3. Experimental Platform

In the WSOA experiments, 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 with multiple Alpha processors running the DEC Unix OS and ORB-express/RT Ada version 2.0.2. A Boeing-owned console with dual Digital Alpha 480 MHz single board computers was used by the server-side operator.

System components were distributed across both computers, using a simulated Link-16 network over 100Base-T Ethernet cabling. 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 is also representative of a broader class of high-performance, soft real-time operating systems.

6. Empirical Results

This section presents the results of the experiments described in Section 5. We first examine baseline end-to-end download latencies for images compressed at the fixed ratios of 50:1, 75:1, and 100:1 and then present latencies when using the adaptation techniques described in Section 3. We next examine image tile compression adaptation response under different strategies and present image tile queueing latencies measured on the client node. We finally explore the overhead of the adaptation techniques and characterize the interactions between the integrated RT-ARM and TAO Reconfigurable Scheduler described in Section 4.

End-to-End Image Latency at Fixed Compression Ratios. We first measure the total time from initial request to receive and process each image. We use this baseline information to compare results of the other trials to assess the effectiveness of adaptation and establish quantitative bounds on the image quality and download time trade-offs achievable by adaptation in the OEP evaluation system. Figure 7 summarizes those results.

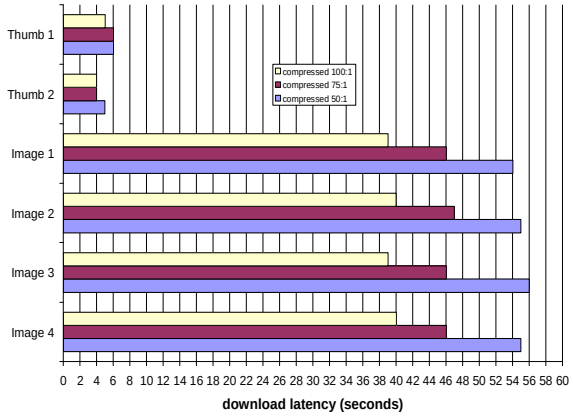

Figure 7: Image Latency without Adaptation

In Trial 1, 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 appeared in all the trials.

Image Latency with Adaptation to End-to-end Deadlines. We next compare end-to-end image download times to respective deadlines. From Trials 2 and 3 respectively, we measured end-to-end image download latencies for deadlines of 38, 42, 46, 50, 54, and 58 seconds. In Trial 2, adaptation of operation invocation rates was also performed, while in Trial 3 it was not. We note that from Trial 1 the 38 second deadline is infeasible even at the highest compression ratio of 100:1, and the 58 second deadline can be met at the lowest compression ratio of 50:1, and thus does not require any adaptation. For the rest of this paper we therefore confine our attention to the 42, 46, 50, and 54 second deadlines.

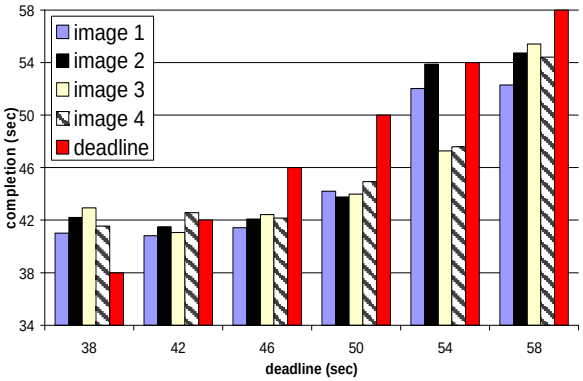

Figure 8: Adaptation of both Compression and Scheduling

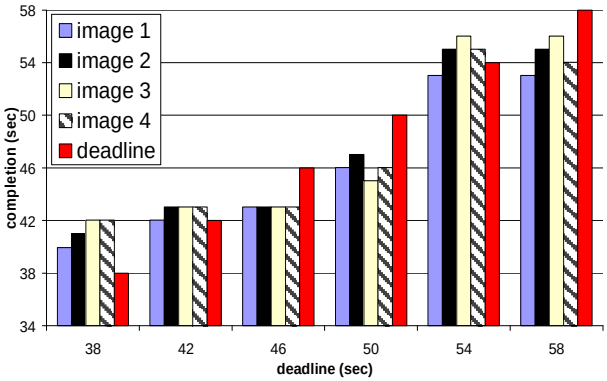

Figure 9: Compression Adaptation Only

The observed results, seen in Figures 8 and 9, showed that compression adaptation alone is insufficient to ensure key deadlines are met, with images 2, 3, and 4 missing both the 42 second and 54 second deadlines in Trial 3, but only image 4 missing the 42 second deadline in Trial 2. 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 adaptation of operation invocation rates outweighs its cost even with tight deadlines, *e.g.*, more images made the 42 second deadline with adaptation of operation invocation rates than without rate adaptation.

Image Compression Adaptation Response. We now consider the recorded image tile compression levels in each trial. In the cases where the sequence of compression ratios was the same for more than one deadline in a given tile, we consider only the *latest* deadline of each such equivalent set. In Trial 3, we confined our attention to image tile compression only.

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

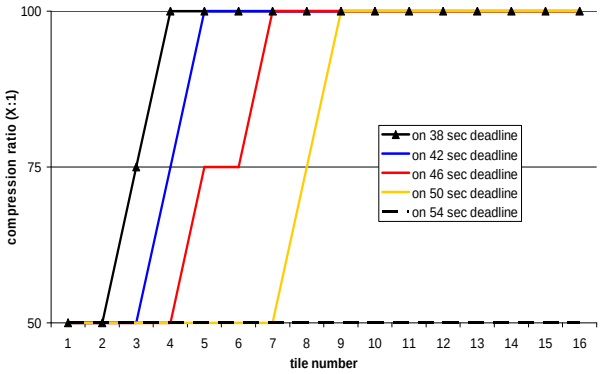

Figure 10: Reactive Compression Adaptation

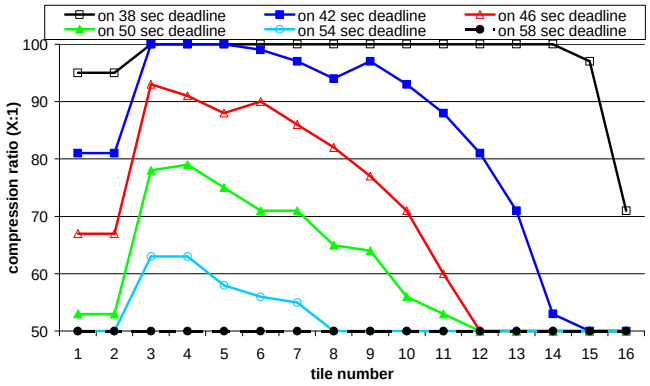

Figure 11: Compression with Simple Control

From Trials 3 and 4, the observed results seen in Figures 10 and 11, show that although it is possible to adapt image download times effectively at coarse-granularity in 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 in light of excess laxity observed at the 46 and 50 second deadlines in Trial 2. *I.e.*, some of the time by which each image arrived early might be traded for image quality in practice.

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.

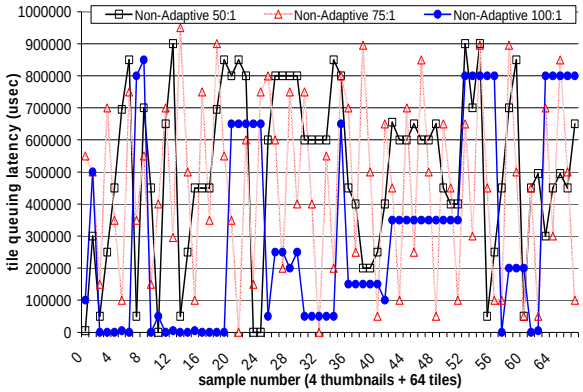

Figure 12: Tile Queuing Latency without Adaptation

The observed results, seen in Figures 12 and 13, showed much lower latencies in Trial 2, and thus identify the client-side tile receive queue as a crucial stage of the end-to-end QoS performance model for the WSOA OEP, and highlight the importance of adaptively managing tile processing operations. Adjusting the rates at which those operations are run significantly decreases the time image tiles spend idly in the queue.

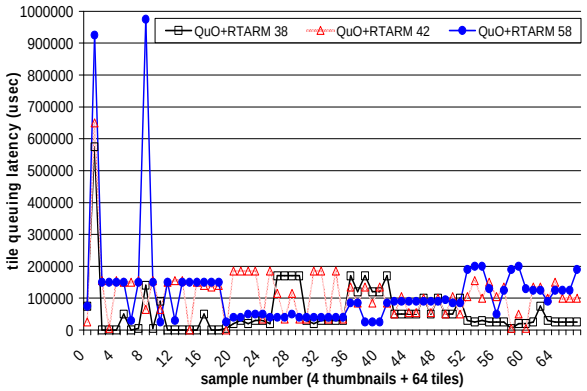

Figure 13: Tile Queuing Latency with Adaptation

Scheduler Re-computation Latency under RT-ARM Management. Our next area of study was the measurement of schedule re-computation overhead resulting from the narrowing of rate ranges by the RT-ARM, and the priority and rate re-assignment by the TAO Reconfigurable Scheduler, described in Section 4. From the results of Trial 2, the key insight is that the number and duration of re-scheduling computations is both (1) reduced overall compared to our earlier results in the ASTD program [17] 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.

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	0 – 10 msec	N/A
Initial Schedule	185 msec	N/A

Table 1. QoS Management Latency

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 10's of ms.

7. 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 8, 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 7. 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 adapta-

tion 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 7 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 9.

The principal feature of interest with the simple control strategy is the more continuous arc of the compression levels shown in Figure 11, in contrast to the coarser-grained transitions shown in Figure 10. The experimental application and supporting middleware infrastructure appear to be amenable to fine-grained (e.g., control-based) adaptation, as shown by the fairly continuous response of the image tile management infrastructure.

Operation rate adaptation reduces image tile queuing latencies. The main feature of interest in the image tile queuing measurements on the client is the much larger magnitude and jitter of queuing latencies without adaptation seen in Figure 12, compared to Figure 13, which shows tile queuing 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 [9], 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 4 significantly reduced the post-initialization cost of rescheduling.

8. 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 other 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.

8.1. 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 [18] 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 [18].

ARMADA. The ARMADA project [19][20] 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.

8.2. 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].

9. Concluding Remarks

This paper has described and quantified the integration of several adaptive middleware technologies, including QuO, RT-ARM, and several layers of The ACE ORB (TAO) (*e.g.*, its Scheduling and Event Services). The paper's contributions involved (1) presenting an architecture for multi-layer adaptive middleware that is applicable to QoS-managed DRE systems and (2) conducting and analyzing empirical results showing the benefits and costs of this architecture for a representative DRE application, *i.e.*, the WSOA OEP mission replanning and real-time avionics mission computing environment.

The main conclusion we draw from the results in this paper is that our integrated QoS-management middleware infrastructure showed successful adaptation of multiple QoS parameters, with a quantitative improvement in management of the trade-off between image quality and download times in comparison to the same approach without adaptation. Factors in the actual DRE system environment are important, and can have a significant impact on the behavior of the system. It is therefore an important achievement to have flown and measured the WSOA OEP evaluation system in a representative avionics mission-computing context.

Our future work will expand upon the studies reported in this paper to examine the effects of influences such as image contrast and size, network latency, and traffic loads on WSOA OEP performance. For example, we are conducting addition tests 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.

We are also implementing control-theoretic adaptation strategies within the QuO adaptive framework [27] and the ORB itself [28][29] to gain further insights into strategies and tactics for effective adaptive management of QoS properties. The goal of our ongoing work on control-theoretic QoS management in middleware is to apply the rigorous modeling and analysis capabilities offered by control theory, to maintain QoS assurances where possible even in the face of dynamically changing resource availability or demand, due to variations in application modes or environmental conditions.

Acknowledgements

We are grateful to all the program managers involved with the WSOA project, especially K. Littlejohn, M. Mills, J. Luke, Capt J. Lawson, G. Koob, LtCol G. Logan, and LtCol G. Palmer.

References

[1] Object Mgmt. Group. “Minimum CORBA - Joint Revised Submission,” OMG Document orbos/98-08-04.

[2] Object Mgmt. Group. “Real-time CORBA Joint Revised Submission,” OMG Document orbos/99-02-12.

[3] Bollella, *et al.*, *The Real-Time Specification for Java*, Addison Wesley Longman, 2000.

[4] DARPA, "The Quorum Program”, 1999.

[5] Gill, Schmidt, and Cytron, “Multi-Paradigm Scheduling for Distributed Real-Time Embedded Computing”, IEEE Proceedings 91(1), Jan 2003.

[6] D. Corman, J. Gossett, D. Noll, “Experiences in a Distributed, Real-Time Avionics Domain - Weapons System Open Architecture, ISORC, Washington DC, USA, April 2002.

[7] Karr, Rodrigues, Krishnamurthy, Pyarali, and Schmidt, “Application of the QuO Quality-of-Service Framework to a Distributed Video Application,” *3rd International Symposium on Distributed Objects and Applications*, Rome, Italy, September 2001.

[8] 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.

[9] Loyall, Gossett, Gill, Schantz, Zinky, Pal, Shapiro, Rodrigues, Atighetchi and Karr, "Comparing and Contrasting Adaptive Middleware Support in Wide-Area and Embedded Distributed Object Applications", 21st ICDCS, April, 2001.

[10] Sharp, “Reducing Avionics Software Cost Through Component Based Product Line Development”, Software Technology Conference, April 1998.

[11] Schmidt, Levine, and Mungee. “The Design and Performance of the TAO Real-Time Object Request Broker”, *Computer Communications* 21(4), April 1998.

[12] Objective Interface, “*ORBExpress*”, *www.ois.com*

[13] Harrison, Levine, and Schmidt, “The Design and Performance of a Real-time CORBA Event Service,” *OOPSLA '97*, October 1997.

[14] Huang, Jha, Heimerdinger, Muhammad, Lauzac, Kannikeswaran, Schwan, Zhao, and Bettati, “RT-ARM: A Real-Time Adaptive Resource Management System for Distributed Mission-Critical Applications", Workshop on Middleware for Distributed Real-Time Systems, IEEE RTSS, San Francisco, California, 1997.

[15] Gill, Levine, and Schmidt, “The Design and Performance of a Real-Time CORBA Scheduling Service,” *The International Journal of Time-Critical Computing Systems 20(2)*, Kluwer, March 2001.

[16] Cross and Lardieri, “Proactive and Reactive Resource Allocation,” Pattern Lang. of Prog. Conf. (PLoP ‘02), Allerton Park, IL, September 2002

[17] Doerr, Venturella, Jha, Gill, and Schmidt, “Adaptive Scheduling for Real-time, Embedded Information Systems,” *18th IEEE/AIAA DASC*, St. Louis, Oct. 1999.

[18] Kalogeraki, Melliar-Smith, Moser, “Soft Real-Time Resource Management in CORBA Distributed Systems”, IEEE Workshop on Middleware for Real-time Systems and Services, San Francisco, CA, December 1997.

[19] Mehra, Indiresan, and Shin, “Structuring Communication Software for Quality-of-Service Guarantees,” IEEE Transactions on Software Engineering, vol. 23, pp. 616–634, Oct. 1997.

[20] Abdelzaher, Dawson, Feng, Jahanian, Johnson, Mehra, Mitton, Shaikh, Shin, Wang, and Zou, “ARMADA Middleware Suite,” IEEE Workshop on Middleware for Real-Time Systems and Services, San Francisco, CA, December 1997.

[21] Kon, Costa, Blair, and Campbell, “The Case for Reflective Middleware,” Communications ACM, vol. 45, pp. 33–38, June 2002.

[22] Cross and Schmidt, “Applying the Quality Connector Pattern to Optimize Distributed Real-time and Embedded Middleware,” Patterns and Skeletons for Distributed and Parallel Computing (Rabhi and Gorlatch, eds.), Springer Verlag, 2002.

[23] Wang, Schmidt, Kircher, and Parameswaran, “Towards a Reflective Middleware Framework for QoS-enabled CORBA Component Model Applications,” IEEE Distributed Systems Online, vol. 2, July 2001.

[24] Kiczales, Lamping, Mendhekar, Maeda, Lopes, Loingtier, and Irwin, “Aspect-Oriented Programming,” Proceedings of the 11th European Conference on Object-Oriented Programming, June 1997.

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

[26] de Miguel, "QoS-Aware Component Frameworks," International Workshop on Quality of Service (IWQoS), (Miami Beach, Florida), May 2002.

[27] Abdelwahed, Neema, Loyall, and Shapiro. "Multilevel Online Hybrid Control Design for QoS Management," Real-time Systems Symposium (RTSS), Cancun, Mexico, December 2003.

[28] Wang, Lu, and Gill, "Feedback Control Real-Time Scheduling in ORB Middleware", 9th IEEE RTAS, Washington, D.C., May 2003.

[29] Wang, Huang, Subramonian, Lu, Gill, "CAMRIT: Control-based Adaptive Middleware for Real-Time Image Transmission", 10th IEEE RTAS, Toronto, Canada, May 2004.

Dr. Christopher D. Gill is an Assistant Professor in the Department of Computer Science and Engineering at Washington University in St. Louis. He has published over 50 refereed technical articles in leading journals, conferences, workshops, and book series. His research focuses on distributed real-time embedded systems, with particular emphasis on adaptive resource management, scheduling, and software design and implementation for time-and-space constrained systems. Dr. Gill has chaired numerous workshop and conference program committees, and has participated widely in review panels and standards organizations in the distributed and real-time systems areas. The research he has led has produced several freely available open-source software frameworks including the Kokyu scheduling and dispatching framework and the nORB small-footprint real-time object request broker.

Ms. Jeanna Gossett joined The Boeing Company in 1999 as a member of Bold Stroke / Open Systems Architecture team. Jeanna has worked on several CRAD projects including Weapon Systems Open Architecture (WSOA) where she was responsible for incorporating quality of service and resource management software technology into the fighter aircraft real-time embedded system application. Jeanna has since joined the F/A-18 New Product Development Mission Systems team. Prior to joining The Boeing Company in 1999, she worked in the telecommunications industry as an embedded systems developer at Ericsson and Siemens AG. Jeanna received a B.S. in Electrical Engineering from Southern Illinois University, Edwardsville and is a 2005 M.B.A. candidate at Washington University in St. Louis.

Dr. Joseph Loyall is a division scientist at BBN Technologies, where he leads the Distributed Real-time Embedded (DRE) systems research thrust in the Distributed Systems Advanced Middleware Technology group. He is actively involved in developing integrated dynamic resource management capabilities and advanced software engineering using model driven architecture (MDA) approaches, and in applying adaptive behavior to operational embedded systems such as collections of unmanned and manned air vehicles. Dr. Loyall has a Ph.D. and M.S. in computer science from the University of Illinois and a B.S. in computer science from Indiana University. He can be contacted at jloyall@bbn.com.

Dr. Douglas C. Schmidt (d.schmidt@vanderbilt.edu) is a Professor of Electrical Engineering and Computer Science, Associate Chair of the Computer Science and Engineering program, and a Senior Researcher in the Institute for Software Integrated Systems (ISIS) at Vanderbilt University. He has published over 300 technical papers and books that cover a range of research topics, including patterns, optimization techniques, and empirical analyses of software frameworks and domain-specific modeling environments that facilitate the development of distributed real-time and embedded (DRE) middleware and applications running over high-speed networks and embedded system interconnects. Dr. Schmidt has served as a Deputy Office Director and a Program Manager at DARPA, where he led the national R&D effort on middleware for DRE systems. In addition to his academic research and government service, Dr. Schmidt has over fifteen years of experience leading the development of ACE, TAO, CIAO, and CoSMIC, which are widely used, open-source DRE middleware frameworks and model-driven tools that contain a rich set of components and domain-specific languages that implement patterns and product-line architectures for high-performance DRE systems.

Dr. David Corman is a Technical Fellow at the Boeing Company, located in St. Louis, Mo. Dave is the chief scientist for the Network Centric Operations (NCO) thrust in Phantom Works (PW) and is responsible for developing the NCO technology research agenda and investment strategy. He is also the Principle Investigator (PI) for a variety of Air Force and Defense Advanced Research Project Agency (DARPA) programs that are producing technologies for integrating legacy platforms into the emerging Global Information Grid and for autonomous control of unmanned systems. Since joining the former McDonnell-Douglas (now part of the Boeing Company) in 1983, Dave has worked on numerous projects ranging from embedded systems to large C4I and weapon systems. A major focus of Dave's career has been on the development of C4I system simulations and in mission planning system development for aircraft and missiles. He has also served as a consultant to many weapon system and C4I programs in St. Louis, Seattle, and California. Prior to joining McDonnell-Douglas, Dave spent five years at the Johns Hopkins University Applied Physics Laboratory. He was the first recipient of a Naval Research Laboratory Fellowship from the University of Maryland - College Park where he received his PhD in Electrical Engineering in 1983.

Dr. Richard E. Schantz is a principal scientist at BBN Technologies in Cambridge, Mass., where he has been a key contributor to advanced distributed computing R&D for the past 30 years. His research has been instrumental in defining and evolving the concepts underlying middleware since its emergence in the early days of the Internet. He was directly responsible for developing the first operational distributed object computing capability and transitioning it to production use. More recently, he has led research efforts toward developing and demonstrating the effectiveness of middleware support for adaptively managed Quality Of Service control, as principal investigator on a number of key DARPA projects in the areas of adaptive real-time behavior, survivability and advanced software engineering. Schantz received his Ph. D. degree in Computer Science from the State University of New York at Stony Brook, in 1974.

Mr. Michael Atighetchi is a senior scientist at BBN Technologies and a senior member of the Distributed Systems Advanced Middleware Technology group. His interests include use of adaptation in survivable systems, network and operating system security, and distributed coordination. Contact him at matighet@bbn.com