Controlling Quality-of-Service in Distributed Real-time and Embedded Systems via Adaptive Middleware

Richard E. Schantz, Joseph P. Loyall, Craig Rodrigues
BBN Technologies
Cambridge, MA, USA
{schantz, jloyall, crodrigu}@bbn.com

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

Abstract

Computing systems are increasingly distributed, real-time, and embedded (DRE) and must operate under highly unpredictable and changeable conditions. An important and challenging problem for DRE systems is therefore adaptation of behavior and reconfiguration of resources to maintain the best possible application performance in the face of changes in system load and available resources. To provide predictable mission-critical quality of service (QoS) end-to-end, QoS-enabled middleware services and mechanisms have begun to emerge. The current generation of commercial-off-the-shelf (COTS) middleware, however, lacks adequate support for applications with stringent QoS requirements in changing, dynamic environments.

This paper presents two contributions to research on adaptive and reconfigurable DRE systems. First, we describe the structure and functionality of an advanced middleware platform for developing applications that apply various techniques to adapt themselves to changes in resource availability to meet real-time quality of service (QoS) requirements. Second, we present the results of a case study of a multimedia application for Unmanned Aerial Vehicle (UAV) video distribution we developed using this middleware platform in conjunction with QoS-enabled operating systems and networking protocols. We describe the design of the multimedia application using our middleware platform and report empirical results showing how adaptive behavior and end-to-end resource management techniques are used to reconfigure the system dynamically to meet timeliness requirements, even in the face of processing power and network bandwidth restrictions that are characteristic of many types of DRE systems. Our results show that our adaptive middleware can effectively coordinate the reconfiguration of resources end-to-end and adapt application behavior to continue to meet QoS requirements over changing environments.

Keywords: Adaptive middleware, reconfigurable DRE systems, aspect-oriented design, and multimedia applications.

1 Introduction

1.1 Emerging Trends and Challenges

Some of the most challenging problems facing software developers are those associated with producing software for *real-time and embedded systems* in which computer processors control physical, chemical, or biological processes or devices. Examples of such systems include airplanes, automobiles, CD players, cellular phones, nuclear reactors, oil refineries, and patient monitors. In most of these real-time and embedded systems, *the right answer delivered too late becomes the wrong answer*, i.e., achieving real-time performance end-to-end is essential.

Real-time and embedded systems have historically been relatively small-scale, but the trend is toward much greater functionality and complexity. In particular, real-time and embedded systems are increasingly being connected via wireless and wireline networks – including the Internet – to create large-scale *distributed real-time and embedded (DRE) systems*, such as hot rolling mills, tele-immersion environments, fly-by-wire aircraft, and total ship computing environments. These DRE systems include many interdependent levels, such as network/bus interconnects, many coordinated local and remote endsystems, and multiple layers of software, that together yield the following challenges.

- As *distributed systems*, DRE systems require capabilities to manage connections and message transfer between separate machines.
- As *real-time systems*, DRE systems require predictable and efficient control over end-to-end system resources, such as memory, CPU, and network bandwidth.
- As *embedded systems*, DRE systems have weight, cost, and power constraints that limit their computing and memory resources. For example, embedded systems often cannot use conventional virtual memory, since software must fit on low-capacity storage media, such as EEPROM or NVRAM.

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 are most suitable for *closed* DRE systems, where the set of application tasks that will run in the system and the loads they will place on system resources change infrequently and are known in advance. They are not well-suited, however, for the next-generation of *open* DRE systems, which evolve more rapidly and must collaborate with multiple remote sensors, provide on-demand browsing and actuation capabilities for human operators, and respond flexi-

bly to unanticipated situational factors that arise at run-time [9].

In open DRE systems, end-to-end control and adaptation of various application quality of service (QoS) properties (such as latency, jitter, and throughput) are essential to maintain the best possible performance in the face of changes in available computing and networking resources and changes in mission requirements. The computing and networking infrastructure must therefore be flexible enough to support varying workloads at different times during an application's lifecycle, while also maintaining highly predictable and dependable behavior. Key challenges for DRE systems are therefore control and adaptation of resources to maintain the best possible real-time application performance in the face of competing demands and changes in load and available resources.

1.2 Towards Adaptive Middleware for DRE Systems

R&D activities on QoS for DRE systems over the past decade [2, 3, 4, 6, 7, 8, 10, 12, 13] have yielded a number of improvements to commonly available computing and networking infrastructures that can recognize and react to environmental changes. At the heart of these infrastructures is *middleware*. Middleware is systems software that resides between the applications and the underlying operating systems and networks to provide reusable services that can be composed, configured, and deployed to create DRE applications rapidly and robustly [5].

As computing and networking performance continues to increase, so too does application demand for more control over computing and networking resources through middleware interfaces. Meeting this growing demand motivates the need for adaptive middleware-centric QoS management abstractions [6,8,10] and aspect-oriented software techniques [20,22]. Supporting this adaptive middleware QoS management architecture efficiently, predictably, and scalably requires new dynamic and adaptive resource management techniques that can (1) integrate control and measurement of resources end-to-end, (2) mediate the resource requirements of multiple (often competing) applications, and (3) dynamically adjust resource allocation in response to changing requirements and conditions. Our earlier R&D efforts on these topics have focused on *The ACE ORB* (TAO) [3] and the *Quality Objects* (QuO) framework [2], which leverage Real-time CORBA [12] to provide efficient and predictable middleware structures and services, and adaptive QoS management policies, respectively, in supporting DRE system QoS requirements. TAO (deuce.doc.wustl.edu/Download.html) is an open-source distribution middleware layer platform targeted (for DRE applications with both deterministic and statistical QoS requirements, as well as best-effort requirements. QuO (quo.bbn.com/quorelease.html) is an open-source QoS adaptive middleware layer framework designed to run on distribution middleware (such as CORBA and Java RMI) using aspect-oriented languages [19] and techniques [21] to support applications that can specify (1) their QoS requirements via rule-based contracts, (2) the system elements that must be monitored and controlled to measure and provide QoS, and (3) the structure and behavior for adapting to QoS variations that occur at run-time.

This paper describes how TAO and QuO have been integrated with multimedia middleware services (such as the CORBA Audio/Video Streaming Service [11]), real-time operating systems (such as Real-time Linux [14]), and QoS-enabled networking protocols (such as IntServ [16] and DiffServ [15]). The goal of this integration is to simplify the development and operation of robust DRE systems that can adapt to changes in resource availability toward meeting their applications' QoS requirements. We present our approach in the context of a multimedia application for Unmanned Air Vehicles (UAV) video distribution we have developed using adaptive middleware. In this application a video flow from a UAV source adapts to meet its mission QoS requirements (such as timeliness and video quality) in the face of restrictions in processing power and network bandwidth.

The ideas underlying this paper represent a long term and continuing research initiative in this area. The paper is based on, incorporates, and extends earlier work, some of which has been reported earlier in fragmentary fashion. A key contribution of this paper is to pull together more detailed aspects of the technical design and applied application context with new and more extensive empirical evaluation results. In particular, we discuss distinct behaviors and techniques that can be used to adapt to limitations and restrictions in processing power and network bandwidth, e.g., reduction of the video flow volume by selectively dropping frames and managing the resources associated with the end-to-end paths. We also present and analyze empirical results gathered to evaluate this application in the context of an open experimentation platform (OEP) developed to evaluate these technologies in operational contexts.

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. We are currently using the Emulab facility at the University of Utah (www.emulab.net) to host the multimedia application OEP environment. The results from our OEP experiments in Section 5 show how the adaptation techniques presented in Section 4 can be controlled effectively by applying the integrated resource management framework described in Section 3 and by superimposing application-level policies managed via middleware to regulate performance problems caused by processor and/or network load. Our multimedia application case study in Section 2 provides insight into emerging aspect-oriented engineering practices where applications are composed from existing software component building blocks, and highlights some of the difficulties encountered and solution paths

taken to meet end-to-end QoS constraints within this development paradigm.

2 Applying Managed QoS to DRE Systems: the Multimedia Application Case Study

This section presents our case study of a multimedia application for UAV video distribution. In this application, multi-layer resource management mechanisms are coordinated via adaptive middleware to ensure video flows can meet their mission QoS requirements (such as timeliness, jitter, and image resolution) by adapting to restrictions in available processing power and network bandwidth. As shown in Figure 1, the resulting architecture adaptively controls video transmission captured from cameras via a distribution process to viewers on various computer displays using the following three stage pipeline:

1. ***Sensor sources***, (endsystems 1-3) including processes with live camera feeds (and those that simply replay from a pre-recorded file to simulate airborne sensors), which send video images to
2. ***Distributor processes***, (endsystem 4) which are responsible for distributing the video to one or more
3. ***Receivers***, (endsystems 5-7) including human-oriented video displays and CPU-intensive image processing software.

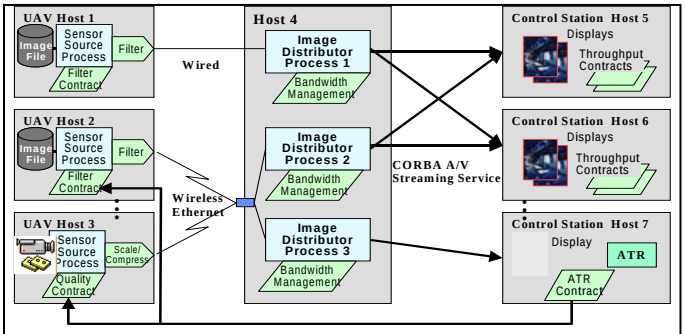

Fig. 1. Architecture of the Multimedia Application Suite

The management of end-to-end QoS in the UAV video dissemination application *crosscuts* the core functional decomposition of the application. It therefore represents a *separate concern*, one representing *how* the work is done, rather than *what* is done and requires coordinating the QoS management from end-to-end in a video stream and across video streams that are sharing resources. Too often, these types of applications have been developed by intertwining the QoS management code within the core functionality code.

Our UAV multimedia application uses the QuO and TAO middleware outlined in Section 1.3 to separate the QoS concerns and manage them by engaging application adaptive behavior, such as dropping frames, requesting resource reservations, indicating prioritization among data streams, and ensuring transparent fault recovery in a bounded amount of time. QuO includes a QoS encapsulation model [21] and a set of aspect-oriented languages [23] we used to develop encapsulated QoS behaviors as separate aspects and weave the QoS management code into the places where QoS can be measured, controlled, and managed, as illustrated in Figure 2. In the QuO QoS encapsulation model the individual QoS behaviors are known as *qoskets*, and this term is used throughout the paper to denote specific, narrowly focused aspects of an overall QoS management approach.

This application exhibits a wide variety of characteristics that are representative of many multimedia applications. These characteristics include varying (1) data formats, such as MPEG and PPM, with different data sizes and compression characteristics, (2) network transports, such as wireless, LAN, and WAN, with variable and constrained bandwidth over both noisy and private channels, (3) image processing algorithms, such as image display and image recognition processes, with different CPU usage patterns, (4) granularities of real-time deadlines, ranging from microseconds to milliseconds and seconds, and (5) resource constraints. Thus, while this paper demonstrates our results on a particular application suite, the characteristics of that suite are representative of a broad class of time-sensitive, mobile, and dispersed applications, especially in the domains of pervasive computing, remote sensing, hazardous operating environments, and automated process control.

In the context of our multimedia application, managing real-time end-to-end QoS requires supporting and coordinating the following measures of operational effectiveness:

- ***Minimal frame rate.*** Full motion video is typically 30 frames per second (fps), but smooth video is still acceptable above 20 fps. Lower frame rates are visibly less smooth, but are usable as long as other qualities (such as data fidelity and jitter) are controlled. Our multimedia application uses variable frame rates

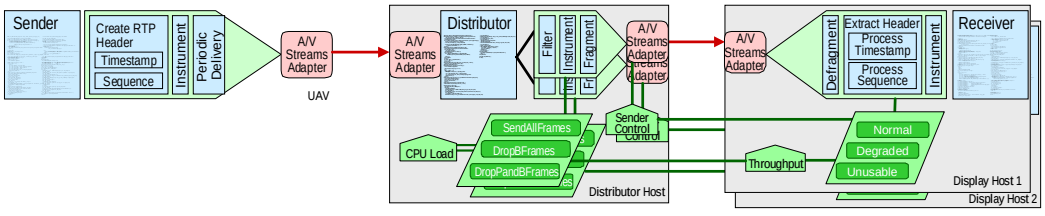

Fig. 2. QoS Aspects are Separately Programmed and Woven into the UAV Video Dissemination Application to Manage End-to-End QoS

as low as 2 fps for human viewing and lower for image processing.

- ***Minimal latency.*** Some uses of sensor information (such as remote piloting) require remote end viewers to see an accurate and timely view of the sensor data, which implies a minimal latency requirement. Studies have indicated that humans can perceive a delay of more than 100-200 ms, which provides a lower bound timeliness requirement in cases where the video is meant for human viewing and precision action. In cases where an image is processed automatically, the latency should be low enough such that the information being processed is never stale relative to what else is already available.
- ***Minimal jitter.*** Controlling the smoothness of the video can have greater impact on the perceived quality than the frame rate. Minimizing jitter requires control throughout the end-to-end path since it can be affected by changes to video transmission rates, delivery latency, and display rates. Common strategies for reducing jitter (such as buffering) are not as useful in real-time video because of the timeliness constraints.
- ***Image quality.*** The image must be of high enough quality (i.e., have the requisite image size, pixel depth, etc.) for the purpose it is being used. For human viewing, the video must be large enough and clear enough to discern details that humans need. For automated processing, it means the image must contain whatever important features the processing is intended to detect.
- ***Coordination of multiple activities.*** The middleware, in conjunction with OS, network, and application directives, must control and coordinate the necessary allocations and tradeoffs that are made to ensure that the highest priority streams and the most important characteristics (e.g., frame rate, latency, and jitter) are favored, even while other, less important characteristics may be minimized or neglected.

Satisfying the measures of operational effectiveness outlined above requires managing resources (particularly CPU and network bandwidth) along the entire path from video source to sink. It also involves trading off one property (e.g., timeliness) against another property (e.g., fidelity) based on the particular requirements of end-users at that moment. For example, our multimedia application cannot suspend the display during a period of network congestion and resume the display from the same point in the video flow when bandwidth is restored because that can violate the timeliness constraint of the delivered images. It is likewise unacceptable to drop arbitrary frames or retransmit lost frames continuously.

All remote operation calls in our multimedia application are made via the TAO real-time ORB [3]. The TAO implementation of the CORBA Audio/Video (A/V) Streaming Service [4] is used to establish the video streams and to transport the data. We encode QoS measurement, control, and adaptation directives and policies via QuO contracts [2] that are distributed throughout the multimedia application. These contracts are responsible for managing the resource and application/data adaptation necessary to achieve an appropriate end-to-end QoS matched to the circumstances relevant at that time. Often, a single QoS aspect will require contracts at many places to implement a particular QoS aspect. For example, controlling image latency might include prioritizing the network traffic (e.g., setting a Diffserv codepoint), shaping the data to the amount of bandwidth available (e.g., compressing or scaling), and monitoring the latency by timestamping. These operations affect code all along the video path, since compressed images will need to be uncompressed and timestamps will need to be inserted, removed, and processed.

Our multimedia application illustrates a number of challenges and design decisions that must be made, such as centralized, policy-driven QoS management versus localized, application-driven QoS management. In this type of distributed system with remote UAVs connected via narrow-pipe tactical links, a centralized QoS management is only practical if it is based on the dissemination of policy at discrete epochs, such as theater-wide mission mode changes. Another design is decentralized, cooperative control, in which all the UAV senders are aware of their relative place in the mission and cooperate to divide resources and achieve the mission. We have experimented with both designs and favor an approach in which local qosket behaviors provide cooperative control and management, directed by policy (such as the relative importance and available resources) pushed by a central QoS management authority residing coincident with the theater command and control authority.

Another design challenge is the correct level at which to represent QoS behaviors. Even when the application is decomposed into components and the QoS behaviors are organized into qosket components, the two crosscut one another. A high-level QoS behavior (such as end-to-end latency) might consist of a single design-time qosket, but requires multiple run-time qosket components in order to implement it. We have therefore combined aspect-oriented and object-oriented designs, weaving the contracts and other QuO mechanisms where they are needed [34].

3 Resource Management for DRE Multimedia Applications

This section describes the priority- and reservation-based OS and network resource management mechanisms we integrated and evaluated within our QoS management framework for multimedia applications based on QuO and TAO. These OS and network mechanisms are *necessary* conditions for establishing end-to-end QoS, but they are not *sufficient* by themselves. To achieve end-to-end QoS, therefore, we use a middleware-mediated QoS management framework to control and coordinate these individual resource management mechanisms, augmented with

additional adaptation mechanisms for making dynamic adjustments and modulating the application's footprint for using resources as discussed in this section.

3.1 Mechanisms for Prioritized and Reserved Management of Computing and Networking Resources

Achieving end-to-end QoS for multimedia applications requires management and control of the processing resources on endsystems in a distributed system and the network resources that connect them. A number of mechanisms for managing these individual resources are emerging, including the mechanisms described below that (1) prioritize competing network traffic using standard Internet technologies and (2) reserve pre-specified amounts of processor time on endsystem computers. In addition to outlining these mechanisms, we describe how we have experimented with – and augmented with complementary mechanisms – various combinations to find the most effective solutions to end-to-end management in the context of our UAV video distribution application described in Section 2.

Priority-based OS resource management. The management of CPU resources in most operating systems has traditionally been handled by assigning priorities to tasks in the system (usually threads or processes) and applying scheduling algorithms to assign each task a share of CPU time. CORBA (as well as other standards-based COTS middleware) historically lacked features that leverage these priority-based OS resource management capabilities, which made it hard to ensure and coordinate predictable platform processing behavior via middleware. To remedy this omission, the Real-time CORBA 1.0 specification [12] defines standard features that support end-to-end predictability for operations in fixed-priority CORBA applications, thereby enabling fine granularity allocation, scheduling, and control of key endsystem OS resources.

The TAO implementation supports the Real-time CORBA interfaces and QoS policies. As a result, applications that use TAO have standard ways to configure (1) *processor resources* via end-to-end priority preservation mechanisms, thread pools, intra-process mutexes, and a global scheduling service, (2) *networking resources* via protocol properties and explicit bindings, and (3) *memory resources* by bounding request buffering and thread pool size. Our earlier work (www.dist-systems.bbn.com/papers) describes how these priority-based OS resource management mechanisms have been applied to avionics mission computing systems via TAO.

Reservation-based OS resource management. An alternative to priority-based OS resource management is to reserve sufficient resources *a priori* for estimated application needs. TimeSys has applied this approach to resource management by implementing a CPU reservation feature for their TimeSys Linux real-time OS. An application – or a middleware proxy for the application – running on top of the TimeSys OS can specify its QoS requirements for timeliness, and their underlying resource kernel [17] will manage the OS resources so that these requirements can be met. For CPU resources, TimeSys Linux allows applications to specify their timeliness requirements by specifying parameters for *compute time* and *period*. If the resource kernel can allocate resources that meet these requirements, it grants an application a *reserve*, which guarantees that for every period, the application will have the requested amount of CPU compute time and will not be preempted.

Although TimeSys Linux provides mechanisms for reserving OS CPU resources, the QuO and TAO middleware are ultimately responsible for determining who gets the reserved capacity, how much, and for how long. These policy decisions are performed by the middleware since it retains the end-to-end perspective to set the OS resources appropriately. We have worked with the University of Utah to develop a CORBA-based CPU reservation manager [7] that (1) is the local agent for setting up reservations on an endsystem and (2) translates various representations of reservation specification into the style supported by TimeSys Linux. Section 5.2, especially measurement 2, reports the results of applying reservation-based OS resource management within our multimedia application context described in Section 2.

Priority-based network resource management. The Internet Engineering Task Force (IETF) Differentiated Services (DiffServ) architecture provides different types or levels of service for IP network traffic. Individual traffic flows can be made more resistant to packet dropping (and hence get preferential delivery) by setting the value of each IP packet's DiffServ field appropriately. An IP header has an eight bit DiffServ field that encodes router-level QoS into (1) six bits of DiffServ Codepoint (DSCP), which enables 64 service categories of per-hop behavior, and (2) two bits of explicit congestion notification. The middleware is responsible for adding the appropriate QoS management DSCP encoding to the data packet headers to specify the appropriate type of service within the multi-application environment. DiffServ-enabled routers then use the DSCP to distinguish among varieties of network traffic.

We have enhanced TAO and QuO to leverage DiffServ capabilities. First, TAO provides an efficient and flexible way of setting the DSCP by extending its Real-time CORBA protocol properties on the GIOP request and response packets so that priority can propagate to requests as they transit the network and OS resources. Based on various factors (such as resource availability, application conditions, and operational requirements), the QuO middleware can change these priorities dynamically by marking application streams with appropriate DSCPs to ensure appropriate priority handling against lower priority competing traffic. Second, TAO provides a mechanism to map Real-time CORBA priorities to DiffServ network priorities. The TAO ORB provides a priority-mapping manager that QuO uses to install a custom mapping to override the default mapping. Section 5.2, especially measurement 1, reports on empirical evaluation of the results of applying

priority-based network resource management (in combination with reserved CPU management) to our multimedia application described in Section 2.

Reservation-based network resource management. Setting DSCPs as discussed above makes traffic flows less likely to be dropped due to network congestion in routers. There is no way in the DiffServ model, however, to *guarantee* a level of service to a traffic flow unless it is the single highest priority traffic at each intermediate step. As with the OS-level resource reservations discussed earlier, it is also desirable to request resources from the network to help guarantee properties (such as latency or bandwidth of network traffic) across some competing flows by reserving appropriate capacity in advance.

To address these issues, the IETF developed the Resource Reservation Protocol (RSVP), also commonly referred to as IntServ (for Integrated Services), which is a new reserved capacity mechanism to augment IP. Whereas the DiffServ mechanisms outlined earlier merely classify and prioritize packets for different service levels, IntServ reservations allocate and coordinate router behavior along a communication path flow to ensure the reserved end-to-end bandwidth. Earlier experiments [8] measured and evaluated the effects of reservation-based network resource management mechanisms applied to multimedia applications via the CORBA A/V Streaming Service [4] provided with TAO.

3.2 A QoS Management Framework for Multimedia Applications

The OS and network resource management mechanisms described in Section 3.1 can be used in various combinations that reflect tradeoffs of integrated methodology, current practice, widespread availability, or maximum performance/cost advantage. Although it may be desirable in some circumstances to have a single methodology (i.e., priority-based *or* reservation-based) apply throughout, other combinations can be useful in practice. Likewise, managing an individual resource (e.g., CPU or network connection) will not enable predictable multimedia application performance if the other complementary DRE system resources along an end-to-end path are constrained, unmanaged, or even managed in an uncoordinated manner. Instead, these resources must be managed in combination to achieve appropriate end-to-end and aggregate results.

To enable more effective coordination and control of individual and aggregate end-to-end resources, we have created elements of a QoS management framework for multimedia applications by integrating the TAO and QuO middleware described in Section 1.2 with the mechanisms described in Section 3.1 that manage lower level OS and network resources. The primary focus of the resource management control strategies described in Section 3.1 is to ensure that more important application tasks get the resources they need to complete their actions at the expense of – or isolated from – other less important tasks. In many cases, however, this is not sufficient to achieve managed QoS objectives, either because there may still be insufficient resources available or because it may be more appropriate to share resources using gradations of service levels that could operate simultaneously, each with diminished resources.

To complement the resource control strategies, our QoS management framework supports adaptive strategies that seek to change the resource consumption of an individual DRE application dynamically. Our QoS management framework therefore offers a set of aspect languages to program the adaptive strategies separately from the core functionality of the application [23] and an encapsulation model for packaging adaptive behaviors so that they can be instantiated and reused throughout an application and maintained separately across applications.

By intelligently modifying the approach to the application's core functionality (e.g., by using alternative algorithms, changing heuristics, or being more selective about degrees of fidelity for various aspects of a computation), we can change the way an application performs its task (and indirectly shape/reduce the amount and timing resources needed to perform that task) to dynamically adapt to the current load, resource availability, or operating conditions prevalent at the time. Section 4 describes the key adaptive strategy used by our UAV video distribution application, each of which have been implemented as qoskets and QoS aspects provided by QuO.

4 Maintaining Real-time QoS Under Reduced Resource Availability in the Multimedia Application

This section describes how we augmented and applied the QoS management control aspects described in Section 3 with application-level adaptation to complement resource control by shaping the interactions between components so they can continue to meet the QoS requirements under diminished resources available to applications.

4.1 Using Adaptation to Meet Multimedia Application QoS Requirements

A bottleneck may occur in our multimedia application because at some point along the video transport path there are not enough resources to send the entire video to the viewers in real time. For example, the distributor endsystem may not have enough CPU power available to dispatch video frames to all viewers at that rate or failures could cause there to be insufficient bandwidth in the network path to one or more viewers. A bottleneck can also occur when one or more of the competing UAVs has (or gains) priority access to significant fractions of the available resources, while the rest must operate within the diminished resources available. When such a bottleneck is detected, we use adaptation techniques (e.g. rate changing and filtering) to mitigate the damage to our QoS objectives. Depending on user requirements, it is possible to omit some frames of the video entirely, yet still retain an end-user video that displays the motion of the scene in

real time without the total fidelity of continuously displayed motion achieved at frame rates of 24 frames or more per second.

To perform data filtering in the UAV prototype, we employ the technique of reducing the transmitted frame rate, e.g., from the distributor to the viewer or between the video source and the distributor. In one important mode of operation, the frame rate must not be reduced in such a way as to create a "slow motion effect," i.e., a vehicle that crossed the field of view of the video source camera in say, 2.5 seconds, should still cross the viewer in 2.5 seconds. A video source attempts to transmit data at the standard rate of 30 fps, which is received at that rate (when system resources permit), but an adaptive behavior can be interposed that sends out a smaller number of frames representing the action that occurs during each second. The subset to be sent is selected by *dropping* some frames from the video, and also sending out the remaining frames at a reduced rate.

The implementation of data filtering to reduce the volume of video data is dependent on the video encoding format. MPEG-encoded video results in sequences of 15 frames, each of which consist of an independent I-frame, as well as 10 derived B-frames and 4 derived P-frames (see Figure 3, and [1] for a synopsis of MPEG encoding of video). One second of video at the full rate of 30 fps requires two sequences of these frames. The best frame-dropping strategies drop B-frames when only a few frames needed to be dropped. There are 20 B-frames in each second of video, so this technique can bring the sending rate down to a still effective 10 frames per second. To drop more frames, P-frames can then be dropped. I-frames can be dropped only if intervals of 1 second or more between images are acceptable, which in our application it was not.

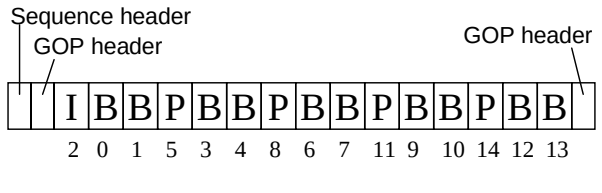

Fig. 3. Sequence of Frames in MPEG File

For practical implementation reasons we chose to drop frames entirely in such a way that the remaining frames were to be displayed at a constant rate. This strategy provided us with three significantly different levels of QoS among which to adapt the application, as determined by the frame rate: (1) 30 fps, which is done by transmitting the video intact to provide the highest level of QoS, (2) 10 fps, which is done by dropping all B-frames from the video and transmitting all the I- and P-frames, which preserves most perception of motion in the video scene, and (3) 2 fps, which is done by dropping all P- and B-frames from the video and transmitting all I-frames, which loses the finer details of motion and some very short-lived actions. We then adaptively switch among these three frame rates by assigning each frame rate to a different region of a QuO contract, and setting the frame-dropping strategy at any given time according to the current region (and indirectly the currently available resources). Below 2 fps, the application would go dormant, until appropriate conditions were restored, because these were below the threshold of operator usability.

4.2 Analysis of Bandwidth Reduction from Frame Filtering

In the video used in our experiments, I-frames averaged approximately 13,800 bytes, P-frames approximately 5,000 bytes, and B-frames approximately 2,900 bytes. The approximate size in bits of two average MPEG encoded sequences is therefore (2(13,800) + 8(5,000) + 20(2,900)) * 8 = 1,004,800, i.e., near the capacity of a 1.5 Mbit link, which is the bandwidth requirement of sending one second of the video at the full rate of 30 fps. If we drop the rate to 10 frames per second by eliminating the B-frames, the bandwidth required, in bits per second, falls to approximately (2(13,800) + 8(5,000)) * 8 = 540,800 and if we drop the rate to 2 fps by eliminating the P-frames as well, the required bandwidth in bits per second falls to approximately 2(13,800) * 8 = 220,800, i.e., reducing the frame rate from 30 to 10 (a 67% reduction) reduces the bit rate by 46%, and reducing the frame rate from 30 to 2 (a 93 % reduction) reduces the bit rate by 78%.

The reductions of bandwidth and other system resource demands outlined above are substantial, so it is not hard to find system conditions under which the full bandwidth is not supportable, but one of the reduced-bandwidth adaptations is. The reduction in bit rate is not proportional to the reduction in frame rate because the frames that are dropped first are precisely those frames that have the greatest dependency on other frames (and the fewest frames depending on them), and consequently the encoded sizes of these dropped frames are relatively smaller. Conversely, reduction in the perceived value of the reduced-frame-rate display to a human viewer also is not proportional to the reduction in frame rate, judging from the reactions of system operators who watched demonstrations of the application adapting.

5 Empirical Results of End-to-end Resource Management Experiments

This section presents and analyzes the results of experiments that cover end-to-end management capabilities stemming from the integration of the individual resource management techniques discussed in Section 3.1 within our middleware-mediated QoS management framework described in Section 3.2. These experiments evaluate the ability of multiple resource management technologies coordinated via middleware to effectively and predictably maintain end-to-end QoS as systems scale to include more participants and more competing load. Our earlier work showed the ability of individual technologies to (1) man-

age QoS end-to-end when competing load was concentrated exclusively on either the processing nodes or the network and (2) fail to manage end-to-end QoS when the type of competing load was unconstrained. These results indicated the need to conduct experiments using integrated and coordinated multiple types of resource management (e.g., CPU and network management) provided by our QoS management framework to evaluate its ability to sustain managed QoS in the presence of a more realistic combined resource load.

5.1 Experimental Design and Hardware/Software Testbed

To test the hypothesis that middleware-coordinated CPU and network management working together can maintain end-to-end QoS in systems with constrained and loaded processors and links, we conducted a set of experiments that ran up to 14 simultaneous simulated UAVs sending imagery to the simulated ground control stations (distributors) and control centers (receivers) described in Section 2. The number of image streams was enough to overload the networks transporting the imagery and control information, and to overload the processors executing the image processing systems. We measured the ability of the resource management mechanisms to control resource allocations sufficiently for an image stream designated as most critical (the experimental case) to consistently sustain the resources needed to complete the application requirements (i.e., detecting and reporting identified targets in imagery data), as contrasted with other competing image streams not marked as critical (the control cases).

In this series of experiments, each of the 14 senders transmitted a sequence of images at a constant rate of 2 fps, in accordance with the application architecture depicted in Figure 1 in Section 2. For a single image stream, a sender process sends images to a distributor and the distributor transmits these images to a receiver. The receiver transmits images to an *automated target recognition* (ATR) program. If the ATR identifies a target in the image stream, it sends a notification to a QuO contract monitoring the imaging components, which in turn propagates the alert via the TAO Real-time Event Channel [27] to a consumer. When this consumer receives the alert, it performs a round-trip time calculation designed to measure the overall time that elapsed from (1) when an image with a target in it was sent from the sender to (2) the time when an alert notification reached the ATR Event Channel client. This time represents the desired end-to-end capability for which we are trying to maintain a predictable QoS footprint under heavy load.

In this experiment, there was contention for both network and CPU resources due to the number of processes involved in simultaneously trying to deliver and identify objects in the 14 image streams. Our coordinated network and CPU QoS management framework capability was configured to attempt to sustain the end-to-end performance of a designated image stream (which in these experiments was arbitrarily selected to be the 7th stream, out of 14). This coordinated QoS management capability under test combined DiffServ network prioritization and CPU reservations. For stream 7, we applied DiffServ network prioritization (over other competing, non-prioritized traffic) using QoS management setup to introduce this behavior between the sender and distributor, and between the distributor and the receiver. In addition, we applied the CPU reservation behavior to the ATR for stream number 7 (only), using a middleware-mediated CPU broker developed at the University of Utah [7].

The CORBA object in the ATR that received the frames was encapsulated by a QuO delegate responsible for determining the magnitude of the CPU reservation requested from the CPU broker. The policy used in this experiment adjusted the CPU reservation request to the highest value seen in processing the five previous frames. This adaptive policy works well in general since it can quickly adapt to spikes in usage without overprovisioning for long periods of time. For this experiment, we used a "strict priority" contention policy that favors high-priority processes when making reservations. Under that policy, the designated high priority UAV stream would be granted its reservation request regardless of the requests of the other activities.

Experiments were performed on hardware and software provided by the University of Utah's Emulab testbed. The hardware configuration for each node in our experiments included (a) 850 MHz Intel Pentium III processor, (b) 512MB PC133 ECC SDRAM, (c) 4 Intel EtherExpress Pro 10/100Mbps Ethernet ports (Experimental network), (d) 1 Intel EtherExpress Pro 10/100Mbps Ethernet port (Control network), and (e) 40 GB IBM 60GXP 7200 RPM ATA/100 IDE hard disk The machines' experimental network interfaces are connected to a Cisco 6509 high-end switch and automatically included in "virtual LANs" to simulate the network topology for our experiments (not shown). This network topology was designed to allow multiple UAV sender programs to transmit imagery data to multiple distributor programs, which in turn would transmit this data to receiver programs. The software configuration for our experiments included (a) Red Hat Linux 7.3, (b) TimeSys v3.1 (selected nodes), (c) FreeBSD 4.8 on "router" nodes, modified to support QoS for network traffic using the (ALTQ) extensions, (d) TAO v.1.3.3, (e) QuO v.3.0.11, and (e) CPU Broker v1.

5.2 Managed End-to-end Behavior Observations

We now report the results of the testbed configurations described above, using observed/measured values that indicate how our integrated middleware-mediated QoS management framework can be used effectively to sustain adequately predictable QoS results under heavy competing load using realistic application scenarios.

Measurement 1: Number of frames received at receiver. For this measurement, the number of images received at each of the competing receivers was recorded.

Stream 7 (only) was prioritized for its network traffic using DiffServ and used CPU Reservations to ensure adequate processing resources. Figure 4 shows that UAV#7 received all of its frames (as did unmanaged UAV's #2,4,5), while some of the rest received most of their frames (#8,9,11), and most (#1,3,6,10,12,13,14) received hardly any service at all, as measured by the number of frames that arrived during the experimentation interval. Since frames are received prior to the CPU intensive processing of the ATR, this measure is dominated largely by controlling network behavior.

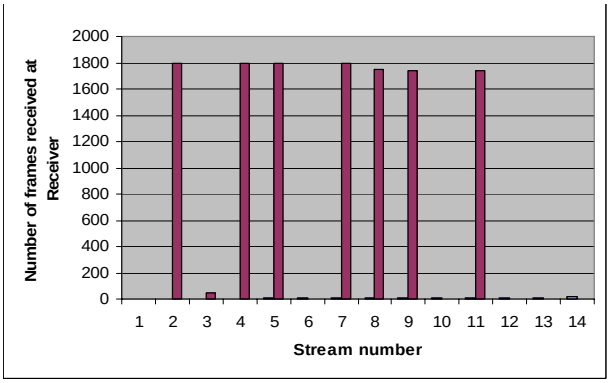

Fig. 4. Number of Frames Received; Stream 7 uses a CPU Reservation and Diffserv Priority

Measurement 2: Number of ATR alert control messages received. For this measurement, the number of ATR Alert control messages, which were received by the ATR Event Channel client program, was recorded. Stream 7 was prioritized with DiffServ and CPU Reservations. These alert messages are sent only after identification of an object of interest by the CPU intensive ATR. Figure 5 shows that only stream #7 successfully identified all of its target objects (as evidenced by receiving all of its alerts). All of the other (unmanaged) streams missed completing the identification cycle (or couldn't get their identifying signal to the collector) at least some of the time, with most (#1,3,5,6,8-14) missing almost all of the identification opportunities. The key factor here is the use of CPU reservation to ensure timely processing of the CPU intensive activity.

Measurement 3: Receiver frame latency. This measurement recorded the time that elapsed when an image was transmitted from the sender to the receiver. Stream 7 was prioritized with DiffServ and CPU Reservations. Figure 6 charts the average latency for frames received (a lower number is better for this chart, in contrast with the previous). This figure shows streams #1,4,6 with average latency per frame delivered lower than for the prioritized stream 7.

Only streams #2,4,5,7,8,9,11, however, had a significant number of successful frame deliveries (from figure 4) so the lower latency for streams #1,6 can be discounted because of the relatively few successful deliveries. Stream #4 had (as yet inexplicably) a lower average latency for delivered frames despite being unmanaged, but stream #7, with controlled resource management working in its favor had a significantly lower standard deviation. This result stems from the more controlled outcome expected by applying course grain resource management strategies to ensure outcome.

5.3 Analysis of End-to-end Resource Management Control Experiments

Out of the 14 image competing streams, half of them did not even come close to receiving and processing even a non-trivial fraction of their intended workload, as shown in Figure 4. The DiffServ prioritized stream processed its intended workload with no observed packet loss. The most significant observations of this experiment were:

- In this and all subsequent runs of the experiment, the prioritized stream (the seventh stream) always reached the receiver endsystem with no observed packet loss. The behavior of non-prioritized streams was not reproducible over multiple runs of the experiment, i.e., sometimes these streams reached the receiver endsystem and sometimes they did not. Which ones did and did not would vary from trial to trial. Non-prioritized streams also had higher rates of

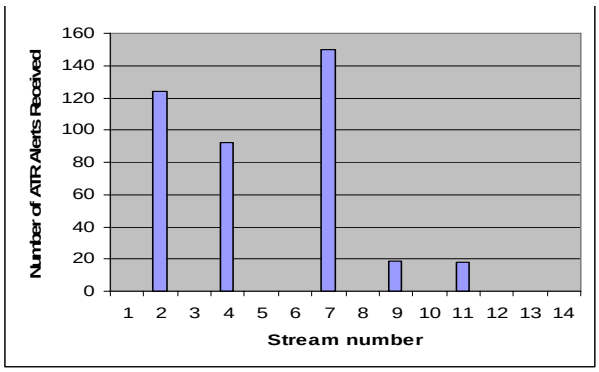

Fig 5. ATR Alerts Successfully Detected; Stream 7 uses a CPU Reservation and Diffserv Priority

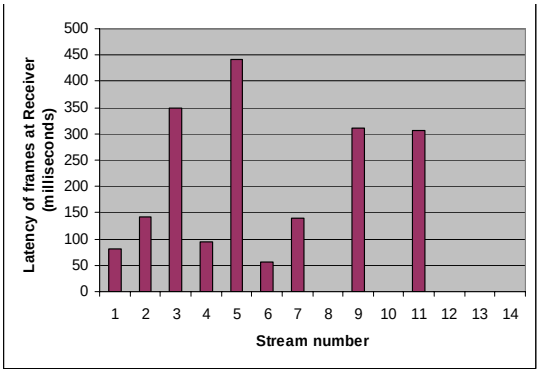

Fig 6. Latency of Images; Stream 7 uses a CPU Reservation and Diffserv Priority

packet loss than the prioritized stream.

- The number of ATR Alert control messages for the prioritized and CPU reserved stream was significantly higher than for any of the other streams. Nine out of the fourteen streams (64%) did not produce ATR Alert control messages indicating successful object identification, and requiring successful and timely upstream delivery and processing. Stream 7 produced 150 alerts, which reached the ATR Event Channel consumer, which was 21% better than the next best stream (Stream 2) that produced 124 alerts. The prioritized stream performed much better than the non-prioritized streams for two reasons: (1) DiffServ prioritizing stream 7 reduced the packet loss compared to other streams, so images with targets in them were more likely to reach the ATR for processing and (2) reserving CPU resources for ATR 7 significantly improved the ability of this ATR to process images and identify targets in a timely fashion despite competing load.

The most significant conclusion drawn from the empirical results described above is that by using a multi-layer middleware-mediated QoS framework that integrates resource management mechanisms (such as DiffServ network priorities and TimeSys Linux CPU reservations), the end-to-end path of a critical, multi-host application exhibits (1) *higher performance* (delivery of all object identification alerts) and (2) *better predictability* (consistently, timely delivery of video images and no observed packet loss) than other less critical applications competing for limited network and CPU resources. The ability to selectively control these end to end behaviors throughout different parts of an overall system and through different areas of technical focus is a giant leap forward in itself. In addition, it represents an important building block in the long-term R&D pursuit of QoS managed adaptive behavior for DRE systems through a common framework, where design time analysis is combined with runtime adaptive mechanisms and policies that manipulate this system level control, while at the same time integrating system-centric adaptation with application-centric adaptation.

6 Related Work

This section reviews related work in enhancing middleware platforms so they can support adaptive DRE QoS properties and application QoS aspects.

Adaptive middleware mechanisms. In their dynamicTAO project, Kon and Campbell [15] apply adaptive middleware techniques to extend TAO so it can be reconfigured at runtime by dynamically linking selected modules, according to the features required by the applications. As with our prior efforts on TAO and QuO, Kon and Campbell provide mechanisms to realize QoS provision in the middleware level. The work described in this paper goes further, however, by integrating QoS provisioning mechanisms at the middleware, OS, and network levels.

The Distributed Multimedia Research Group at Lancaster University has developed a prototype of advanced reflective middleware called Adapt [18]. This middleware model concentrates on dynamic composition of objects through open bindings [6], which (1) allows object implementations to be configured dynamically, (2) determines various aspects of object implementations, such as adding or removing methods from an object, and (3) explicitly establishes transport connections between objects that can be used for streaming multimedia data. The Adapt project model also facilitates QoS properties management and monitoring. Compared to the Adapt project, our efforts concentrate on applying QoS provisioning techniques to implement and improve the implementation of an existing middleware standard (CORBA), whereas the Adapt project defines and implements the meta-space of a new middleware framework at a higher level.

Aspect-oriented techniques can be applied to specify middleware QoS behaviors and configure the supporting mechanisms for these QoS behaviors. In particular, the container architecture in component-based middleware, such as Enterprise Javabeans (EJB) and the CORBA Component Model (CCM), provides the vehicle for applying meta-programming techniques that provide QoS assurance control in component middleware. Conan et al [20] use containers together with aspect-oriented software development techniques to plug in different non-functional behaviors. This project is similar to QuO delegates in that mechanisms are provided to inject aspects into applications statically at the middleware level. QuO goes further, however, since it also supports dynamic QoS provisioning via its qosket mechanisms [21].

de Miguel [22] extends other work on QoS-enabled containers by enhancing an EJB container to support a QoSContext interface that allows the exchange of QoS-related information with component instances. To take advantage of the QoS-container, a component must implement QoSBean and QoSNegotiation interfaces. A key difference between de Miguel's approach and ours is the QuO delegates and contracts enable the QoS negotiation protocols to be performed transparently to the component implementations.

Control-theoretic approaches to adaptive middleware. A number of control-theoretic approaches are now being applied to DRE systems to overcome limitations with traditional scheduling approaches that do not handle dynamic changes in resource availability and therefore result in a rigidly scheduled system that adapts poorly to change. A survey of these techniques is presented in [28].

Feedback control scheduling (FCS) [29] is designed to address the challenges of applications with stringent end-to-end QoS executing in open DRE systems. These algorithms provide robust and analytical performance assurances despite uncertainties in resource availability and/or demand. FC-U and FC-M [30] applies this approach to manage processor utilization. CAMRIT [31] applies control-theoretic approaches to ensure transmission deadlines of images over an unpredictable network

link and also presents analytic performance assurance that the transmission deadlines are met.

A hierarchical control scheme that integrates resource reservation mechanisms [32] with application-specific QoS adaptation is proposed in [33]. This control scheme features a two-tier hierarchical structure: (1) a global QoS manager is responsible for allocating computational resources to various applications in the system and (2) application-specific QoS managers/adapters modify application execution to use the allocated resources efficiently and maximize application QoS.

Although these approaches are similar to the work we report in this paper, these algorithms/mechanisms perform resource management of only one type of system resource, *i.e.*, either computing power *or* network bandwidth. In contrast, our QoS management framework performs resource management of both network and computing resources, which is crucial for real-world DRE systems. We have also recently used the two-tier hierarchical management approach similar to that cited above as an extension to our multi-resource QoS management framework, for dynamic mission management applications [35].

7 Concluding Remarks

Developing DRE systems that can maintain the best possible application performance in the face of changes in available resources is an important R&D challenge. This paper describes the design and performance of a QoS management framework that adaptively controls the end-to-end behavior of multimedia applications by applying resource management techniques for processing and communication tasks. This framework integrates QoS-enabled middleware (TAO and QuO), multimedia middleware services (the CORBA Audio/Video Streaming Service), real-time operating systems (Real-time Linux) and QoS-enabled networking protocols (IntServ and DiffServ) to develop robust multimedia applications that can adapt to changes in resource availability to meet their QoS requirements.

Creating a stove-piped one-of-a-kind application would have been an unsatisfying solution, although that has been the state-of-the-practice until recently. We therefore designed our solution based upon advanced software engineering principles, such as separation of concerns and aspect-oriented programming. We have incorporated these concepts as key design principles underlying our middleware framework and have used them to develop representative applications.

Over the past several years we have enhanced, applied, and evaluated these middleware-mediated QoS management technologies in the context of an open experimentation platform (OEP) that embodied complex challenge problems associated with multimedia applications – in particular a UAV video distribution application suite. The relevant QoS management activities associated with this OEP include trading off sensor/image quality and timeliness and coordinating resource usage among competing applications to satisfy changing mission requirements under dynamic (and potentially hostile) environmental conditions. Our experiments used a combination of CPU reservation along with network priority for end-to-end control of resources management policy to effect the controlled QoS behavior reported for our UAV video distribution application. Our empirical results showed how integrating resource management techniques can be effective in sustaining predictable QoS results under heavy competing load.

The work reported here also formed the basis for subsequent exploration and evaluation in the context of a live flight, multiple UAV exercise at White Sands Missile Range in April 2005, which integrated and managed the interactions among widely dispersed air, ground, fixed and mobile assets performing dynamic mission planning for time-sensitive activities [25]. That work combined the middleware-mediated managed resource approach with the adaptation approach used to dynamically shape application behavior, and combined elements of design time analytic approaches with runtime adaptive behavior approaches [26]. Based on experience and experiments with those real applications in their natural operating contexts, we conclude that the approaches and techniques outlined in this paper are both feasible and effective in managing QoS demands in realistic and changing deployment environments.

The UAV video dissemination application illustrates the challenges associated with dimensions of scale. Moving from managing QoS in an application to managing end-to-end QoS in a streaming video application is hard enough. Adding the complexity of multiple competing streams over shared and constrained resources increases the complexity. The scale of the problem space does not end there, however, since the complexity continues to scale if the number of UAVs can change at runtime, and if the runtime conditions (including possible mission modes) can change. We have had success in managing end-to-end QoS for multiple, competing streams, as described in this paper. Up to now, however, we have done so only for fixed (or bounded) numbers of UAVs and for a fixed set of mission modes which are known a priori. Allowing these two dimensions to scale unconstrained adds a new set of challenges for us to address. Our future work focuses on a more comprehensive analysis of the tradeoffs, effectiveness, and widespread availability of middleware-mediated OS and network resource management.

References

[1] D. Le Gall. MPEG: a Video Compression Standard for Multimedia Applications. *Communications of the ACM*, April 1991.

[2] J. Zinky, D. Bakken, and R. Schantz, “Architectural Support for Quality of Service for CORBA Objects”, Theory and Practice of Object Systems, vol. 3, num. 1, 1997.

[3] D. Schmidt, David Levine, and S. Mungee, “The Design and Performance of the TAO Real-Time Ob-

ject Request Broker", *Computer Communications* 21(4), April 1999.

[4] S. Mungee, N. Surendran, Y. Krishnamurthy, and D. Schmidt, "The Design and Performance of a CORBA Audio/Video Streaming Service," Design and Management of Multimedia Information Systems: Opportunities and Challenges, Idea Publishing Group, 2000.

[5] R. Schantz and D. Schmidt, "Middleware for Distributed Systems: Evolving the Common Structure for Network-centric Applications," Encyclopedia of Software Engineering, Wiley and Sons, 2002.

[6] T. Fitzpatrick, G. Blair, G. Coulson, N. Davies, P. Robin, "Supporting Adaptive Multimedia Applications through Open Bindings," International Conference on Configurable Distributed Systems, Maryland, 1998.

[7] Eric Eide, Tim Stack, John Regehr, and Jay Lepreau "Dynamic CPU Management for Real-Time, Middleware-Based Systems," 10th IEEE Real-Time and Embedded Technology and Applications Symposium, May 25 - May 28, 2004, Toronto, Canada.

[8] R. Schantz, J. Loyall, C. Rodrigues, D. Schmidt, Y. Krishnamurthy, and I. Pyarali, "Flexible and Adaptive QoS Control for Distributed Real-time and Embedded Middleware," The ACM/IFIP/USENIX International Middleware Conference, June 2003, Rio de Janeiro, Brazil.

[9] C. D. Gill, J. M. Gossett, D. Corman, J. P. Loyall, R. E. Schantz, M. Atighetchi, and D. C. Schmidt, "Integrated Adaptive QoS Management in Middleware: An Empirical Case Study," Proceedings of the 10th IEEE Real-time Technology and Application Symposium (RTAS '04), Toronto, CA, May 2004.

[10] F. Kon, F. Costa, G. Blair, and R. Campbell, "The Case for Reflective Middleware," CACM, June 2002.

[11] Object Management Group, "Control and Management of Audio/Video Streams, OMG RFP Submission (Revised), OMG Technical Document 98-10-05", Oct 1998, Framingham. MA.

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

[13] Object Management Group, Real –Time CORBA 2.0: Dynamic Scheduling Specification, OMG Final Adopted Specification, September 2001.

[14] TimeSys Corporation. *TimeSys Linux R/T User's Manual*, 2.0 edition, 2001.

[15] IETF, An Architecture for Differentiated Services, *www.ietf.org/rfc/rfc2475.txt.*

[16] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala, "RSVP: A New Resource ReSerVation Protocol," *IEEE Network*, September 1993.

[17] Timesys Corporation. Predictable Performance for Dynamic Load and Overload, Version 1.0. www.timesys.com/files/whitepapers/Predictable_Performance_1_0.pdf, 2002.

[18] G. Blair, G. Coulson, P. Robin, M. Papathomas, "An Architecture for Next Generation Middleware," Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed SysProcessing, London, England, 1998.

[19] J. Loyall, D. Bakken, R. Schantz, J. Zinky, D. Karr, R Vanegas, and K. Anderson. "QoS Aspect Languages and Their Runtime Integration." Lecture Notes in Computer Science, Vol. 1511, Springer-Verlag. Proceedings of the Fourth Workshop on Languages, Compilers, and Run-time Systems for Scalable Computers (LCR98), May 1998, Pittsburgh, PA.

[20] D. Conan, E. Putrycz, N. Farcet, M. DeMiguel, "Integration of Non-Functional Properties in Containers," Proc. of the 6th International Workshop on Component-Oriented Programming, Budapest, Hungary, 2001.

[21] R. Schantz, J. Loyall, M. Atighetchi, P. Pal, "Packaging Quality of Service Control Behaviors for Reuse," *Proceedings of the 5th IEEE International Symposium on Object-oriented Real-time distributed Comput-ing (ISORC 2002)*, April 29 - May 1, 2002, Washington, DC.

[22] M. deMiguel, "QoS-Aware Component Frameworks," Proceedings of the 10th International Workshop on QoS (IWQoS), Miama Beach, Florida, May 2002.

[23] Gary Duzan, Joseph Loyall, Richard Schantz, Richard Shapiro, and John Zinky, "Building Adaptive Distributed Applications with Middleware and Aspects," International Conference on Aspect-Oriented Software Development (AOSD '04), Lancaster, UK, March 22-26, 2004.

[24] Nanbor Wang, Douglas C. Schmidt, Aniruddha Gokhale, Christopher D. Gill, Balachandran Natarajan, Craig Rodrigues, Joseph Loyall, and Richard E. Schantz, "Total Quality of Service Provisioning in Middleware and Applications," The Journal of Microprocessors and Microsystems, Elsevier, vol. 26, number 9-10, January 2003.

[25] Joseph Loyall, Richard Schantz, David Corman, James Paunicka, Sylvester Fernandez. "A Distributed Real-time Embedded Application for Surveillance, Detection, and Tracking of Time Critical Targets," Real-time and Embedded Technology and Applications Symposium (RTAS), San Francisco, CA, March 7-10 2005.

[26] Joseph Loyall, Jianming Ye, Sandeep Neema and Nagabhushan Mahadevan, "Model-Based Design of End-to-End Quality of Service in a Multi-UAV Surveilance and Target Tracking Application," Second RTAS Workshop on Model-Driven Embedded Systems (MoDES '04), Torinto, Canada, May 25-28, 2004.

[27] Douglas C. Schmidt and Carlos O'Ryan, "Patterns and Performance of Distributed Real-time and Embedded Publisher/Subscriber Architectures," Journal of Systems and Software, Special Issue on Software

Architecture -- Engineering Quality Attributes, October 2002.

[28] T. F. Abdelzaher, J. Stankovic, C. Lu, R. Zhang, and Y. Lu, “Feedback Performance Control in Software Services,” IEEE: Control Systems, 23(3), June 2003.

[29] L. Abeni, L. Palopoli, G. Lipari, and J. Walpole, “Analysis of a Reservation-based Feedback Scheduler,” In *IEEE Realtime Systems Symposium*, Dec. 2002.

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

[31] X.Wang, H.-M. Huang, V. Subramonian, C. Lu, and C. Gill, “CAMRIT: Control-based Adaptive Middleware for Realtime Image Transmission,” In *Proc. of the 10th IEEE Realtime and Embedded Tech. and Applications Symp. (RTAS)*, Toronto, Canada, May 2004.

[32] T. Cucinotta, L. Palopoli, L. Marzario, G. Lipari, and L. Abeni. Adaptive Reservations in a Linux Environment. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 238–245, 2004.

[33] L. Abeni and G. C. Buttazzo. Hierarchical QoS Management for Time Sensitive Applications. In *IEEE Real Time Technology and Applications Symposium*, 2001.

[34] Praveen K. Sharma, Joseph P. Loyall, George T. Heineman, Richard E. Schantz, Richard Shapiro, Gary Duzan. Component-Based Dynamic QoS Adaptations in Distributed Real-Time and Embedded Systems. International Symposium on Distributed Objects and Applications (DOA), Agia Napa, Cyprus, October 25-29, 2004.

[35] Prakash Manghwani, Joseph Loyall, Praveen Sharma, Matthew Gillen, and Jianming Ye. End-to-End Quality of Service Management for Distributed Real-time Embedded Applications. The Thirteenth International Workshop on Parallel and Distributed Real-Time Systems (WPDRTS 2005), Denver, Colorado, April 4-5, 2005.