# Controlling Quality-of-Service in a Distributed Real-time and Embedded Multimedia Application 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

*An increasingly important and challenging problem for distributed real-time and embedded (DRE) systems is control and adaptation of resources to maintain the best possible application performance in the face of changes in load and available resources. This paper presents two contributions to R&D activities in the DRE domain. First, we describe the structure and functionality of an advanced middleware platform (based on our QuO adaptive middleware framework and the TAO Real-time CORBA middleware suite) for developing DRE applications that can adapt to changes in resource availability to meet quality of service (QoS) requirements. Second, we present the results of a case study of a DRE multimedia application for Unmanned Aerial Vehicle (UAV) video distribution we have developed using this middleware platform in conjunction with QoS-enabled operating systems (such as Real-time Linux) and networking protocols (such as IntServ and DiffServ). We describe the design of the UAV multimedia application using our middleware platform and report empirical results showing how adaptive behavior and end-to-end resource management techniques are used 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 middleware infrastructure can effectively coordinate resource allocation end-to-end and adapt application behavior to continue to meet QoS requirements over changing environments.*


## 1 Introduction

### 1.1 Emerging Trends and Technologies

Next-generation distributed real-time and embedded (DRE) systems must collaborate with multiple remote sensors, provide on-demand browsing and actuation capabilities for human operators, and respond flexibly to unanticipated situational factors that arise at run-time [5,28]. For example, new and planned emergency response systems are incorporating Unmanned Air Vehicles (UAVs) to send video images to processes that distribute the video to the proper control stations, which in turn contain video displays and other video processing applications, such as automatic target recognition (ATR), that analyze the video and trigger appropriate responses, including revised tasking for the UAV-based video sensors. In these types of DRE systems, end-to-end control and adaptation of various application quality of service (QoS) properties (such as latency, jitter, throughput, dependability and security) 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. Controlling the end-to-end real-time behavior of such DRE systems is a crucial dimension of their delivered QoS, as is adaptively managing the tradeoffs among competing demands and optimizations.

The recent focus on user control over QoS aspects [4,6,10,12,24] stems from technology advances in research areas such as resource allocation policies, synchronization of streams in DRE multimedia applications, and assured communication in the face of high demand over shared resources. The focus on QoS has led to the development of 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*, which 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 [17]. Figure 1 illustrates the following two middleware layers that are central to the focus of this paper:

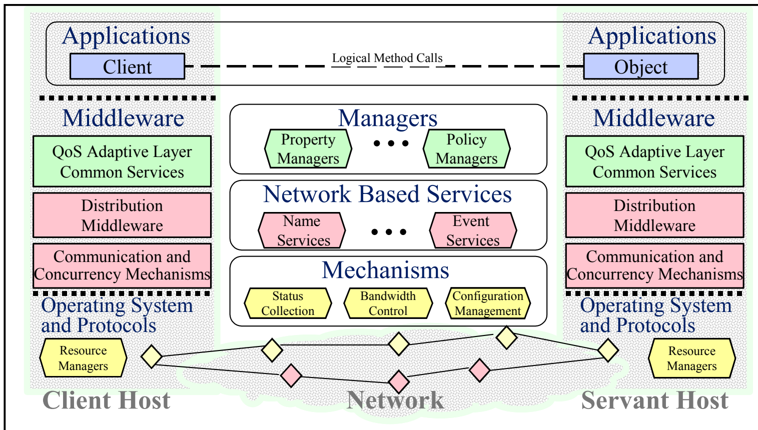


**Fig. 1.** Layers of Middleware

- ***Distribution middleware*** – This layer of middleware encapsulates lower-level operating system and networking mechanisms to provide a higher level programming model that automates common distributed programming tasks, such as connection management, data transfer, parameter (de)marshaling, request and endpoint demultiplexing, concurrency control, and some forms of error handling. Examples of commercial-off-the-shelf (COTS) distribution middleware include CORBA [16], Java RMI [25], and Microsoft's COM+ [2].

- ***QoS adaptive middleware*** – This emerging layer of middleware helps to bridge the gap between (1) an application's QoS needs across its multiple parts and (2) the middleware services and infrastructure that provide QoS.

It provides the abstractions necessary to adapt to changing conditions and requirements for applications that can operate in a wide variety of DRE system environments and changing conditions. An example of QoS adaptive middleware is the Quality Objects (QuO) framework [27].

### 1.2 Towards Adaptive Middleware for DRE Systems

As computing and networking performance continues to increase, so too does application demand for more control over computing and networking resources through the middleware interface. In particular, the DRE multimedia applications outlined in Section 1.1 have stringent requirements (such as the need for streaming data transport, time-sensitive performance, and demanding QoS characteristics oriented toward human factors) and characteristics (such as workloads that can vary significantly at run-time) that need focused support from middleware. In turn, this increases the demands on end-to-end system resource management, where it has historically been hard to coordinate multiple end-to-end resource needs simultaneously and mediate resource needs across multiple applications. In addition, the mission-critical processing aspects of next-generation DRE multimedia applications require that they (1) continue to respond adequately during both anticipated and unanticipated operational changes in their run-time environment and (2) ensure that critical applications acquire the necessary resources at the expense of less critical applications.

Meeting the growing demands of DRE multimedia applications motivates the need for adaptive middleware-centric QoS management abstractions and techniques. 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 middleware R&D efforts on these topics have focused on The ACE ORB (TAO) [19] and the Quality Objects (QuO) framework [11], which leverage Real-time CORBA [15] to provide efficient, scalable, and predictable middleware structures and services, and adaptive QoS management policies, respectively, in supporting end-to-end DRE system QoS requirements. TAO is a high-performance distribution middleware layer targeted for DRE applications with both deterministic and statistical QoS requirements, as well as best-effort requirements. The QuO framework is a QoS adaptive middleware layer that runs on existing middleware and supports DRE 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 extends our earlier work with TAO and QuO by combining these advanced adaptive middleware frameworks with multimedia middleware services (such as the CORBA Audio/Video Streaming Service [13]), real-time operating systems (such as Real-time Linux [21]), and QoS-enabled networking protocols (such as Int-Serv [26] and DiffServ [8]) to develop robust DRE multimedia applications that can adapt to changes in resource availability toward meeting their QoS requirements. Our approach is presented in the context of a DRE multimedia application for Unmanned Air Vehicles (UAV) video distribution, in which 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. We discuss distinct behaviors 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 present and analyze empirical results we gathered to evaluate this application in the context of an open experimentation platform (OEP)[1] developed to evaluate these technologies in operational systems. Our results show how adaptation can be controlled effectively by applying integrated resource management end-to-end and by superimposing application-level policies managed via middleware to regulate performance problems caused by processor and/or network load. Our UAV application case study also provides insight into emerging 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.

### 1.3 Paper Organization

The remainder of this paper is organized as follows: Section 2 describes a case study in which TAO, QuO, and associated technologies have been applied to develop an adaptive DRE multimedia application for UAV video distribution. Sections 3 and 4 discuss how adaptive middleware-mediated policies and mechanisms can be integrated to coordinate the multiple layers of resources used by DRE multimedia applications, such as those appearing in our UAV video distribution case study. Section 5 analyzes the results of empirical tests we conducted to evaluate these mechanisms and approaches. Section 6 presents concluding remarks and outlines future work.

## 2 Applying Managed QoS to DRE Systems: the UAV Case Study

This section presents a case study of a DRE multimedia application for UAV video distribution, where multi-

---

[1] An OEP is a hardware/software laboratory capability 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 (http://www.emulab.net) to host the UAV application OEP environment.

layer resource management mechanisms are coordinated via 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. The resulting application architecture shown in Figures 2 and 3 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

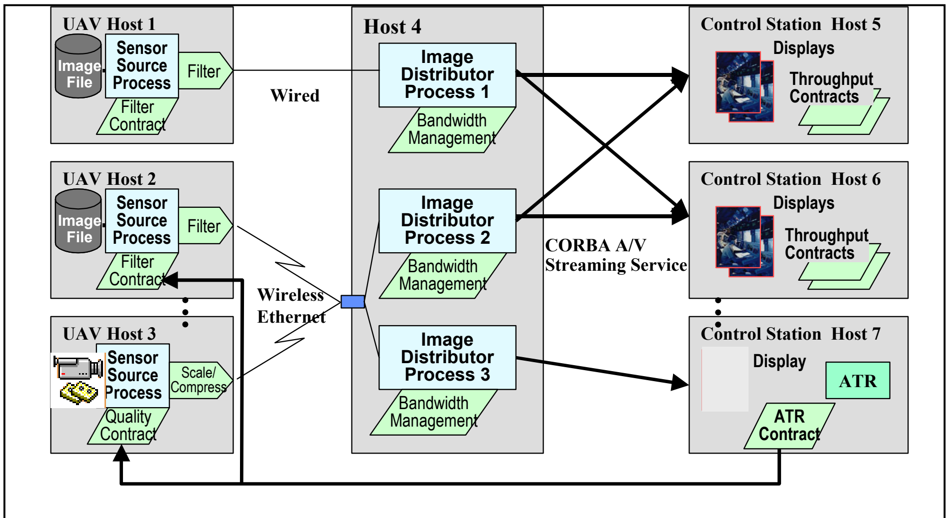


**Fig. 2.** Architecture of the UAV Application Suite

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.

Our UAV application suite uses the QuO and TAO middleware outlined in Section 1.1 to manage QoS 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. It also exhibits a wide variety of characteristics (such as constrained resources, varying conditions and configurations, and varying data and processing characteristics) that are representative of many

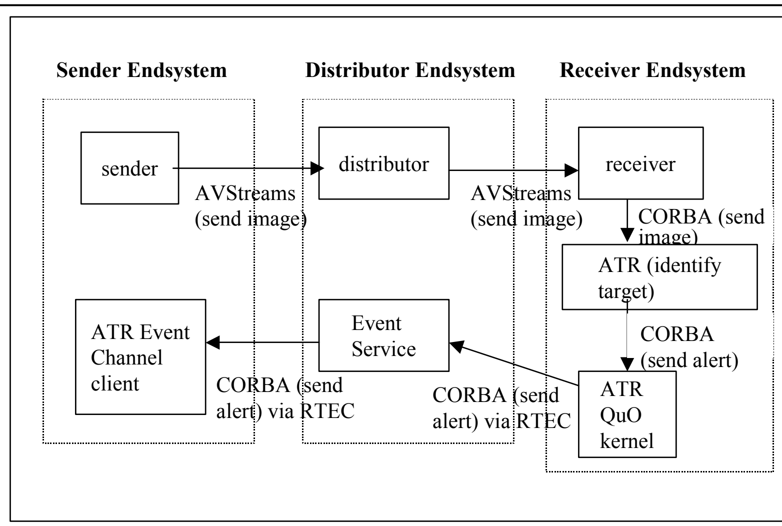


**Fig. 3.** A Detailed Interaction Diagram for a Single end-to-end Stream

DRE 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 (the automated target recognition (ATR) process shown in Figure 3), 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 operation DRE applications, especially in the domains of pervasive computing, remote sensing, hazardous operating environments, and automated process control.

In the context of our UAV application, managing real-time end-to-end QoS requires supporting and coordinating the following measures of operational effectiveness:[2]

- ***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 UAV application uses variable frame rates 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 so there is no more current image.
- ***Minimal jitter.*** Controlling the smoothness of the video can have greater impact on the perceived quality than the frame rate. Minimizing jitter requires control all along 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 impor-

[2] We are continually evaluating our technology approach against these and other measures of effectiveness. Section 5 reports quantitative results that validate empirically whether we are achieving this operational effectiveness.

tant 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, in our UAV application it is not acceptable to 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 not acceptable to drop arbitrary frames or to attempt to retransmit lost frames continuously.

All remote operation calls in our UAV application are made via the TAO real-time ORB [19,20]. The TAO implementation of the CORBA Audio/Video (A/V) Streaming Service [14] 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 [23] that are distributed throughout the UAV 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.

# 3 Resource Management for DRE Multimedia Applications

This section describes the various priority- and reservation-based OS and network resource management mechanisms we have integrated and evaluated within our QoS management framework for DRE multimedia applications based on QuO and TAO. The 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 DRE 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) has 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 [15] 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 standard Real-time CORBA interfaces and QoS policies. As a result, DRE 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 [18] describes how these priority-based OS resource management mechanisms have been applied to UAV mission computing systems via the TAO Real-time CORBA ORB.

**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 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 [22] 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 pre-empted.

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 higher-level middleware since it retains the end-to-end perspective to set the lower-level OS resources appropriately. We have worked with the University of Utah to develop a CORBA-based CPU reservation manager 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 results of applying reservation-based OS resource management within our UAV multimedia application context described in Section 2.

**Priority-based network resource management.** The Internet Engineering Task Force (IETF) Differentiated Services (DiffServ) architecture [8] 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 differentiate the network traffic.

We have implemented enhancements to TAO and QuO that 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 be propagated 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 UAV 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 this 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) [26], 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. Our earlier work [18] describes how IntServ reservation-based network resource management mechanisms were applied to UAV applications via the CORBA A/V Streaming Service provided with TAO.

### 3.2 A QoS Management Framework for DRE 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 DRE multimedia applications by integrating the TAO and QuO middleware outlined in Section 1.1 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 outlined in Section 3.1 (such as bandwidth or CPU reservation or priority access to available CPU and network resources) 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 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 dynamically change the resource consumption of an individual DRE application. By intelligently modifying the approach to the application 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 often 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 some key adaptive strategies used by our UAV video distribution application.

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

This section describes in detail 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 the application.

## 4.1 Using Adaptation to Meet UAV Application QoS Requirements

A bottleneck may occur in our UAV 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[3], we use adaptation techniques 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.

---

[3] See [9] for a discussion of approaches to bottleneck detection.

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 4, and [4] 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.

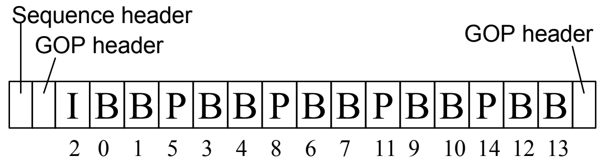


**Fig. 4.** Sequence of frames in MPEG file

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. On the other hand, 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 prior experimental results [1] showed the ability of individual technologies to (1) manage 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.[4] 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 3 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 [7] to an Event Channel client program. When this client program 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 $7^{th}$ stream, out of the 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). The CORBA object in the ATR that received the frames was encapsulated by a QuO delegate that was 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

[4] In contrast, our earlier experiments introduced artificial load on targeted processors or links. Our experiments reported in this paper produced more realistic loading of the entire system end-to-end.

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:

- 850 MHz Intel Pentium III processor
- 512MB PC133 ECC SDRAM
- 4 Intel EtherExpress Pro 10/100Mbps Ethernet ports (Experimental network)
- 1 Intel EtherExpress Pro 10/100Mbps Ethernet port (Control network)
- 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 the following:

- Red Hat Linux 7.3
- TimeSys v3.1 (selected nodes)
- FreeBSD 4.8 on "router" nodes, modified to support QoS for network traffic using the (ALTQ) extensions
- TAO v.1.3.3
- QuO v.3.0.11
- 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 5 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 largely dominated by controlling network behavior.

**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.

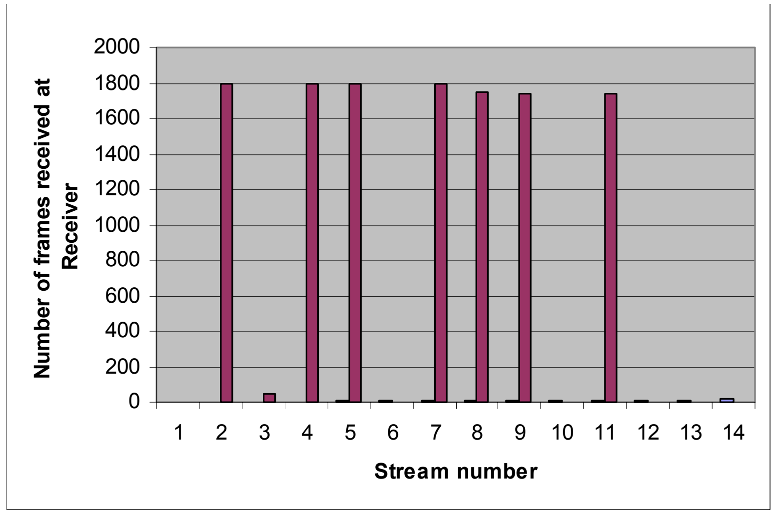


**Fig. 5.** Number of frames received; Stream 7 uses a CPU reservation and Diffserv Priority

Figure 6 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.

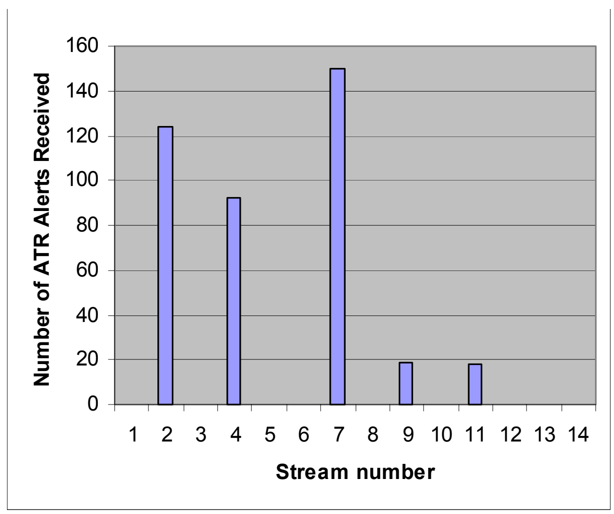


**Fig 6.** ATR alerts successfully detected; Stream 7 uses a CPU reservation and Diffserv Priority

**Measurement 3: Receiver frame latency.** For this measurement, the time that elapsed when an image was transmitted from the sender to the receiver was recorded. Stream 7 was prioritized with DiffServ and CPU Reservations. Figure 7 charts the average latency for frames received (a lower number is better for this chart, in contrast with the previous). The figure shows streams #1,4,6 with average latency per frame delivered lower than for the prioritized stream 7. However, only streams #2,4,5,7,8,9,11 had a significant number of successful frame deliveries (from figure 5) 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 likely indicating a more controlled outcome expected from applying course grain resource management strategies to ensure outcome.

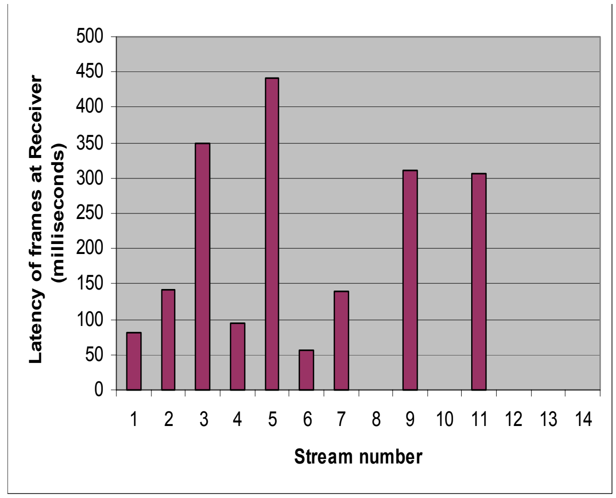


**Fig 7.** Latency of images; Stream 7 uses a CPU reservation and Diffserv Priority

### 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 5. 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 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 client. This was 21% better than the next best stream (Stream 2), which 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. Being able to selectively control these end to end behaviors as they wind through 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 longer run 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 Concluding Remarks

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

During the past two years we have enhanced, applied, and evaluated these middleware-mediated QoS management technologies in the context of an open experimentation platform (OEP) that embodies complex challenge problems associated with DRE 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 empirical results presented in this paper showed how integrated resource management techniques can be effective in sustaining predictable QoS results under very heavy competing load.

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. Other combinations have been tried and are continuing to be integrated and evaluated, toward a more comprehensive analysis of the tradeoffs, effectiveness, and widespread availability of these middleware-mediated OS and network resource management mechanisms. Our future work will present the results of these efforts as well as the results of experiments that combine the middleware-mediated managed resource approach with the adaptation approach used to dynamically change application profiles, and combine design time analytic approaches with runtime adaptive behavior approaches.

## Acknowledgements

This work is sponsored by DARPA and US AFRL under contract nos. F33615-00-C-1694 and F30602-98-C-0187. Thanks to Tim Stack, Eric Eide and colleagues at the University of Utah and to Gautam Thacker and colleagues at Lockheed Martin Advanced Technology Labs (ATL) for their help with technology integration and experimentation reported here.

## References

[1] "PCES UAV Phase I Experiments Summary Report," BBN Technologies Technical Report, September 2003, http://www.dist-systems.bbn.com/projects/AIRES/UAV/experimentation/results/phaseI/bbn_summary_phaseI_experiments.doc, BBN Technologies, "Quality Objects (QuO)", http://www.dist-systems.bbn.com/papers.

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

[3] 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.

[4] D. Le Gall. MPEG: a video compression standard for multimedia applications. *Communicationsof the AC*M, April 1991.

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

[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] T. Harrison., D. Levine, and D. Schmidt, "The Design and Performance of a Real-time CORBA Event Service," *OOPSLA '97*, Atlanta, GA, October 1997.

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

[9] Karr DA, Rodrigues C, Loyall JP, Schantz RE, Krishnamurthy Y, Pyarali I, Schmidt DC. Application of the QuO Quality-of-Service Framework to a Distributed Video Application. Proceedings of the International Symposium on Distributed Objects and Applications, September 18-20, 2001, Rome, Italy.

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

[11] J. Loyall, R Schantz, J.. Zinky,and D. Bakken, "Specifying and Measuring Quality of Service in Distributed Object Systems", Proceedings of the 1st IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC), April 1998.

[12] 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.

[13] 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.

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

[15] Object Management Group, “Realtime CORBA Joint Revised Submission”, OMG Document orbos/99-02-12, March 1999.
[16] Object Management Group, Real –Time CORBA 2.0: Dynamic Scheduling Specification, OMG Final Adopted Specification, *September 2001*, http://cgi.omg.org/docs/ptc/01-08-34.pdf.
[17] 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.
[18] 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.
[19] D. Schmidt, D.Levine, and S. Mungee, “The Design and Performance of Real-Time Object Request Brokers,” Computer Communications, April 1998.
[20] D. Schmidt, S. Mungee, S. Flores-Gaitan, and A. Gokhale, “Software Architectures for Reducing Priority Inversion and Non-determinism in Real-time Object Request Brokers”, Journal of Real-time Systems, special issue on Real-time Computing in the Age of the Web and the Internet, Kluwer, 2001.
[21] TimeSys Corporation. *TimeSys Linux R/T User’s Manual*, 2.0 edition, 2001.
[22] Timesys Corporation. Predictable Performance for Dynamic Load and Overload, Version 1.0. http://www.timesys.com/files/whitepapers/Predictable_Performance_1_0.pdf, 2002.
[23] R. Vanegas, J. Zinky, J. Loyall, D. Karr, R. Schantz, and D. Bakken, “QuO's Runtime Support for Quality of Service in Distributed Objects", Proceedings of Middleware 98, the IFIP International Conference on Distributed Systems Platform and Open Distributed Processing, Sept 1998.
[24] N. Wang, D. Schmidt, A. Gokhale, C. Gill, B. Natarajan, C. Rodrigues, J. Loyall, R. Schantz. "Total Quality of Service Provisioning in Middleware and Applications," Microprocessors and Microsystems spec. issue on "Middleware Solutions for QoS-enabled Multimedia Provisioning over the Internet", 2003.
[25] A. Wollrath, R. Riggs, and J. Waldo, “A Distributed Object Model for the Java System”, USENIX Computing Systems, MIT Press, vol. 9, num. 4, Nov/Dec 1996.
[26] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala, “RSVP: A New Resource ReSerVation Protocol,” *IEEE Network*, September 1993
[27] 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.
[28] D. Corman, J. Gossett, D. Noll, “Experiences in a Distributed, Real-Time Avionics Domain - Weapons System Open Architecture, ISORC, Washington DC, USA, April 2002.