Flexible and Adaptive QoS Control for Distributed Real-time and Embedded Middleware

Richard E. Schantz[1], Joseph P. Loyall[1], Craig Rodrigues[1], Douglas C. Schmidt[2], Yamuna Krishnamurthy[3], Irfan Pyarali[3]

[1]BBN Technologies
Cambridge, MA
{schantz, jloyall, crodrigu}@bbn.com
[2] Vanderbilt University
Nashville, TN
schmidt@ isis-server.isis.vanderbilt.edu
[3] OOMWorks, LLC
Metuchen, NJ
{yamuna,irfan}@oomworks.com

Abstract. Computing systems are increasingly distributed, real-time, and embedded (DRE) and must operate under highly unpredictable and changeable conditions. To provide predictable mission-critical quality of service (QoS) end-to-end, QoS-enabled middleware services and mechanisms have begun to emerge. However, the current generation of commercial-off-the-shelf middleware lacks adequate support for applications with stringent QoS requirements in changing, dynamic environments. This paper provides two contributions to the study of adaptive middleware to control DRE applications. It first describes how priority- and reservation-based OS and network QoS management mechanisms can be coupled with standards-based, off-the-shelf distributed object computing (DOC) middleware to better support dynamic DRE applications with stringent end-to-end real-time requirements. It then presents the results of experimentation and validation activities we conducted to evaluate these combined OS, network, and middleware capabilities. Our work integrates currently missing low-level resource control capabilities for end-to-end flows with existing capabilities in adaptive DRE middleware and sets the stage for further advances in fine-grained precision management of aggregate flows using dynamic adaptation techniques.

1 Introduction and Background

Emerging trends. 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]. The distributed computing infrastructure for

This work was supported by the Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory under contracts F30602-98-C-0187 and F33615-00-C-1694.

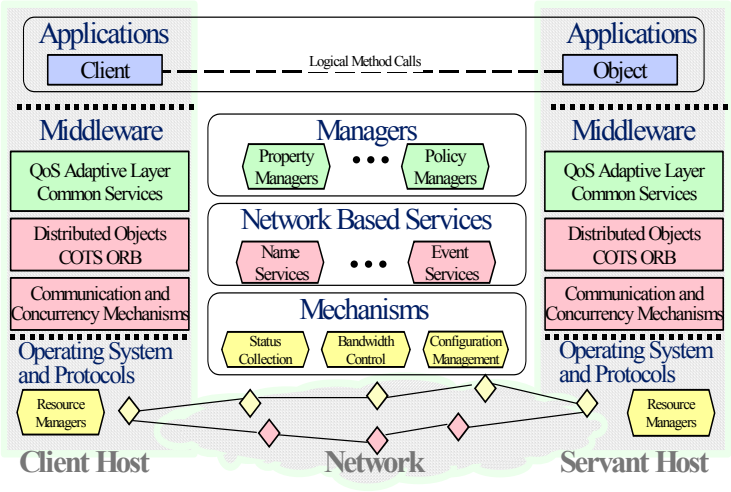

Fig. 1. Layers of Middleware

these systems must be sufficiently flexible to support varying workloads at different times during an application lifecycle, yet maintain highly predictable and dependable behavior. Controlling the real-time behavior of such distributed computing systems is one important dimension of the delivered *quality of service* (QoS).

The recent focus on user control over QoS aspects stems from technology advances in historically challenging research areas, such as allocation policies, synchronization of streams in distributed multimedia applications, and assured communication in the face of high demand. The focus on QoS aspects has led to the development of a number of proposed and implemented improvements to commonly available distributed computing infrastructures. When coupled with software that can recognize and react to environmental changes, these improvements form the basis for constructing appropriate adaptive behavior for next-generation DRE systems.

An overview of COTS middleware. Requirements for faster development cycles, decreased cost, and reusable solutions motivate the use of *middleware* [25]. Figure 1 illustrates the key middleware layers related to the focus of this paper:

- ***Distribution middleware*** – This layer encapsulates concurrency, communication, and distribution mechanisms to provide a higher level programming model that automates common programming tasks, such as parameter (de)marshaling, request demultiplexing, and error handling. At the heart of this infrastructure middleware resides some form of Object Request Broker (ORB), such as CORBA [21], Java RMI [32], or Microsoft's COM+ [3].
- ***QoS adaptive middleware*** – This emerging layer of middleware bridges the gap between an application's QoS needs across its multiple parts and the middleware services and infrastructure that provides QoS. It provides the abstractions necessary to adapt to changing conditions and requirements for applications that can operate in a wide variety of environments and changing conditions. An example of QoS adaptive middleware is the Quality Objects (QuO) framework [34].

Towards an adaptive COTS middleware solution. As network and endsystem performance continues to increase, so too does the demand for more control and manageability of their resources through the middleware interface. In particular, next-generation DRE systems present end-to-end real-time QoS requirements over shared resources and with workloads that can vary significantly at run-time. In turn, this increases the demands on end-to-end system resource management and control, which makes it hard to simultaneously (1) coordinate the management of multiple end-to-end resources and (2) mediate the (possibly conflicting) resource needs across multiple applications, with individual resource mechanisms or managers. In addition, the mission-critical processing aspects of next-generation DRE systems require that they (1) respond adequately to both anticipated and unanticipated operational changes in their run-time environment and (2) ensure that critical capabilities acquire the necessary resources.

Meeting these increasing demands of next-generation DRE systems 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 applications and (3) dynamically adjust resource allocation in response to changing requirements and conditions.

Our prior work has explored many dimensions of QoS-enabled adaptive middleware design and performance, including QoS frameworks, QoS specification and measurement, inserting adaptive behavior into applications, QoS aspects for dependability and survivability, scalable event processing, request demultiplexing, connection management and explicit binding architectures, asynchronous and synchronous concurrent request processing, and IDL stub/skeleton optimizations. This paper focuses on a previously unexamined dimension of QoS-enabled adaptive middleware: *the integration of priority- and reservation-based OS and network QoS management mechanisms with standards-based COTS DOC middleware.* This integration is essential since it enables a new generation of flexible DRE applications that (1) have more precise control over their end-to-end resource management strategies, (2) can be more easily reconfigured and adapted to dynamically changing network and computing environments, and (3) help mature the emerging standards-based COTS infrastructure.

Paper organization. The remainder of this paper is organized as follows: Section 2 outlines related work on adaptive DRE middleware, including the technologies created during our earlier work on standards-based COTS adaptive DRE middleware that form the basis for the work described in this paper; Section 3 describes the emerging priority and reservation-based resource management mechanisms needed to support dynamic end-to-end QoS management using middleware; Section 4 provides an example DRE application in which we have integrated these resource management services with our earlier adaptive DRE middleware; Section 5 describes empirical results obtained by systematically measuring the behavior of our adaptive DRE middleware in representative application scenarios; and Section 6 presents concluding remarks.

2 Related Work

Distributed object computing (DOC) is the most advanced, mature, flexible paradigm available today for the development of next-generation DRE systems [10]. DOC software architectures are composed of objects that can be distributed or collocated throughout a wide-range of networks and interconnects, thereby shielding applications from many distributed computing complexities. Since conventional DOC middleware historically failed to support more stringent end-to-end application requirements, an increasing body of research has focused on techniques that specify, measure, control, and adapt QoS. This section reviews optimizations and enhancements we and others have made to conventional DOC middleware programming models and implementations so they can support DRE QoS properties and simultaneously allow flexible control and adaptation of key application QoS aspects.

2.1 Our Earlier DRE Middleware Efforts

Our earlier DRE middleware work has focused on TAO and QuO, which leverage Real-time CORBA [20, 21] to provide efficient, scalable, and predictable DRE middleware structures and services, and adaptive QoS management policies, respectively. These technologies serve as the underlying context for adding the specific resource management mechanisms described in Section 3 to manage and control end-to-end DRE performance.

Overview of TAO. TAO [26] is a high-performance distribution middleware targeted for DRE applications with deterministic QoS requirements, as well as best-effort requirements. TAO supports the standard OMG CORBA [21] and Real-time CORBA [20] specifications, whose implementation in TAO ensures efficient, predictable, and scalable QoS behavior for high-performance DRE applications. The following are some of the optimizations in TAO:

- ***Optimized IDL Stubs and Skeletons*** –TAO's IDL compiler generates stubs/skeletons that can selectively use highly optimized compiled and/or interpretive marshaling/demarshaling [8], thereby allowing application developers to trade off time and space, which is crucial for high-performance DRE applications.
- ***Real-time ORB*** –TAO's real-time Object Adapter uses perfect hashing and active demultiplexing [22] optimizations to dispatch servant operations in constant time, regardless of the number of active connections, servants, and operations defined in IDL interfaces and TAO's real-time ORB Core [27] uses a multi-threaded, preemptive, priority-based connection and concurrency architecture [8] to provide an efficient and predictable CORBA protocol engine.
- ***Run-time Scheduler*** – TAO's run-time scheduler maps application QoS requirements (such as bounding end-to-end latency and meeting periodic scheduling deadlines) to ORB endsystem/network resources (such as CPU, memory, network connections, and storage devices) using either static and/or dynamic [7] real-time scheduling strategies.

Overview of QuO. The Quality Objects (QuO) framework [1, 16, 30] is a QoS adaptive layer of middleware that runs on existing DOC middleware (such as Real-time CORBA and Java RMI) and supports distributed applications that can specify (1)

their QoS requirements, (2) the system elements that must be monitored and controlled to measure and provide QoS, and (3) the behavior for adapting to QoS variations that occur at run-time. To achieve these goals, QuO provides middleware-centric abstractions and policies for developing DOC applications. Key components provided by QuO to support the above operations include:

- ***Contracts*** **–** The operating regions and service requirements of the application are encoded in *contracts*, which describe the possible states the system might be in, as well as which actions to perform when the state changes.
- ***Delegates*** **–**Delegates are *proxies* that can be inserted into the path of object interactions transparently, but with woven in QoS aware and adaptive code. When a method call or return is made, the delegate checks the system state, as recorded by a set of contracts, and selects a behavior based upon it.
- ***System Condition Objects*** **–** System condition objects are *wrapper facades* that provide consistent interfaces to infrastructure mechanisms, services, and managers. System condition objects are used to measure and control the states of resources, mechanisms, and managers that are relevant to contracts.

Our recent work [31] integrating the TAO Real-time CORBA ORB with QuO enables a managed end-to-end path through middleware services. The work reported in Sections 3 though 5 of this paper extends end-to-end middleware control of QoS through the OS and network layers, as well.

2.2 Other Adaptive DRE Middleware Efforts

Meta-programming 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 [4] use containers together with aspect-oriented software development (AOSD) [14] 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 [24].

de Miguel [17] 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.

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 [2]. This middleware model concentrates on dynamic composition of objects through open-binding [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.

3 Managing End-to-End Real-time QoS via Middleware-mediated Resource Management Mechanisms

End-to-end QoS requires management and control of the processing resources on nodes in a DRE system and the network resources that connect them. A number of mechanisms for managing these individual resources are emerging, such as mechanisms for (1) prioritizing competing network traffic using standard Internet technologies and (2) reserving prespecified amounts of processor time on COTS host computers. These mechanisms are *necessary* conditions for establishing end-to-end QoS, but they are not *sufficient* by themselves. To achieve end-to-end QoS, therefore, individual resources must be managed in a coordinated manner. Management of an individual resource (e.g., CPU or network connection) will not enable predictable performance if the other complementary resources along an end-to-end path are constrained, unmanaged, or even managed in an uncoordinated manner.

This section describes four emerging mechanisms for managing resources in a DRE system: two each for managing OS and network resources. One pair of mechanisms is predominantly based on a priority paradigm and the other pair is predominantly based on a reservation paradigm. We also discuss how we have enhanced the TAO and QuO middleware to combine and coordinate these mechanisms toward achieving complete end-to-end QoS management capabilities. Section 4 describes the application context for this work and Section 5 then reports our latest experimentation and validation work in using the resource management mechanisms described below separately and in combination.

3.1 Priority-based OS Resource Management

CORBA (as well as other existing standards-based COTS middleware) has historically lacked features to provide fine granularity allocation, scheduling, and control of key host OS resources necessary to ensure and coordinate predictable platform proc-

essing behavior. The Real-time CORBA (RT-CORBA) 1.0 specification [20] defines standard features that support end-to-end predictability for operations in fixed-priority CORBA applications. RT-CORBA (and the TAO implementation) now includes standard interfaces and QoS policies that allow applications to configure the following types of resources:

- ***Processor resources*** via priority mechanisms, standardized ways of handling thread pools and intra-process mutexes, and a global scheduling service;
- ***Communication resources*** via protocol properties and explicit bindings; and
- ***Memory resources*** by bounding buffering requests and the size of thread pools.

Applications typically configure these real-time QoS policies along with other policies when they invoke standard ORB operations. For instance, when an object reference is created using a QoS-enabled RT-CORBA object adapter, the object adapter ensures that any server-side policies that affect client-side requests are embedded within a tagged component in the object reference. This enables clients who invoke operations on such object references to honor the policies required by the target object.

Strict control over the scheduling and execution of processor resources is essential for correct execution of fixed-priority DRE applications. RT-CORBA enables client and server applications to (1) determine the priority at which CORBA invocations will be processed and (2) allow servers to pre-define pools of threads to service incoming invocations in a standard, ORB independent manner.

The fine-grained control of various aspects of ORB implementations is important for predictable behavior. However, establishing a global task priority mechanism that can be mapped to existing lower-level OS priorities and propagated across platforms can be viewed as the key element of RT-CORBA for enabling coordinated end-to-end behavior in a standard and interoperable COTS manner.

3.2 Priority-based Network Resource Management

Due to its pervasiveness (and the associated cost/availability ramifications), Internet technology is coming to predominate the communication infrastructure for many types of systems. A historic limitation of the Internet technology for DRE applications has been its exclusive reliance on a "best effort" style of resource management. The Internet Engineering Task Force (IETF) realized that Internet Protocol (IP) on its own did not satisfy the requirements for these types of applications, and set up a working group to develop new mechanisms to augment basic IP/TCP. As a result, network resource management capabilities based on Internet technologies are slowly emerging that are more in line with the requirements of DRE computing environments.

In IP networks, data packets contain just enough information for intermediate routers to forward a packet to its destination. Without emerging network traffic management extensions, all IP packets are treated the same and forwarded with "best effort" QoS. If a router between source and destination receives network traffic at a rate faster than it can process, it will drop packets arbitrarily, which is a condition known as *network congestion*. When TCP packets are dropped, the data is lost, and the source host must retransmit the data to the destination host, thereby incurring latencies that are unacceptable to many DRE applications.

The Differentiated Services (DiffServ) architecture [12] provides different types or levels of service for IP network traffic. Individual traffic flows can be made more resistant to dropping (and hence get preferential delivery) by setting the value of each IP packet's DiffServ field with an appropriate value. 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 (PHB) and (2) two bits of Explicit Congestion Notification (ECN). A DSCP is added to data packet headers to specify the expected type of service. DiffServ-enabled routers and other network elements use the DSCP to differentiate the network traffic.

We have implemented two enhancements to the RT-CORBA support in TAO that leverage DiffServ capabilities. First, we provided an efficient and flexible way of setting the DSCP by extending the ORB 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, we provide a mechanism to map RT-CORBA priorities to DiffServ network priorities. The TAO ORB provides a priority-mapping manager that supports installation of a custom mapping to override the default mapping. Figure 2 depicts how these individual mechanisms are integrated into an end-to-end priority configuration.

3.3 Reservation-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. An alternative approach is to reserve sufficient resources *a priori* for the estimated need. While this approach may seem a more natural fit with meeting real-time demands, it is usually more complicated to implement this type of resource management strategy effectively because of issues associated with metering and meeting allocation guarantees while also supporting a priority mechanism for practicality.

The TimeSys Corporation has applied a reservation approach to resource management by implementing a CPU reservation feature in the Linux kernel. The TimeSys Linux kernel is based on resource kernel (RK) [28] work done by the Real-time and

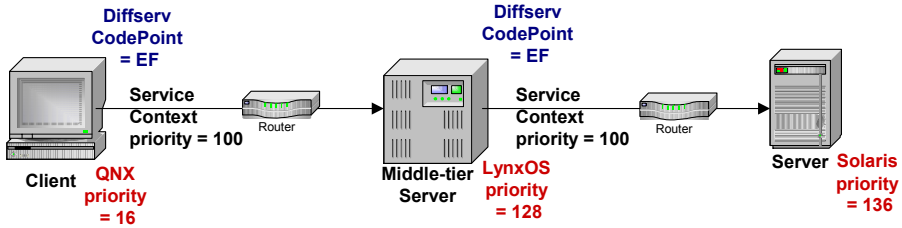

Fig. 2. Example Priority Propagation in RT-CORBA + DiffServ

Multimedia Systems Laboratory at CMU. An application (or more precisely, a middleware proxy for the application) running on top of the TimeSys resource kernel can specify its QoS requirements for timeliness, and the underlying resource kernel will manage the OS resources so that these requirements can be met.

For CPU resources, TimeSys Linux allows an application to specify its timeliness requirements by specifying parameters for *compute time* and *period*. If the resource kernel can allocate resources that meet these requirements, it grants the 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. Reserving appropriate slices of resources on each of the participating platforms is an alternative to priority based end-to-end management of host processing.

Although TimeSys Linux provides COTS mechanisms for reserving OS CPU resources, the QuO and TAO middleware are responsible for determining who gets the reserved capacity, how much, and for how long. These policy decisions are performed via the higher-level middleware since it retains the end-to-end perspective to set the lower-level OS resources appropriately. We are working with the University of Utah to develop a CORBA-based CPU reservation manager that will (1) be the local agent for setting up reservations on a host and (2) translate various representations of reservation specification into the particular style supported by the TimeSys Linux [23].

3.4 Reservation-based Network Resource Management

Setting DSCPs as discussed in Section 3.2 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. Just as for the OS-level resource reservations discussed in Section 3.3, 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 Internet Engineering Task Force (IETF) established a working group to develop a new reserved capacity mechanism to augment IP. The result was the Resource Reservation Protocol (RSVP) specified in RFC 2205 [33], also commonly referred to as IntServ (for Integrated Services). Whereas the DiffServ mechanisms outlined in Section 3.2 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.

RSVP specifies a *signaling* protocol, whereby an application can request a level of service, such as bandwidth, for a certain network flow between a source and destination host. Each router between the source and destination host receives this signaling information, and allocates enough resources to meet the required QoS. The resource reservation is stored in each router so that it can be updated or deleted dynamically.

We have integrated the IntServ mechanisms described above into the QuO and TAO middleware outlined in Section 2.1, where we use them to coordinate end-to-end reservation allocation strategies. The QuO contracts contain the information about the specification of the required reservations. From our prior work on developing intermediate level common distribution services, we utilize the CORBA A/V Streaming

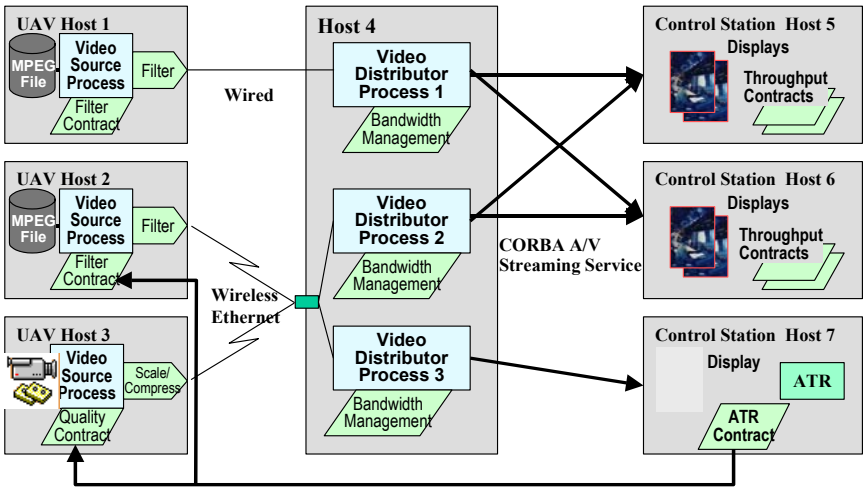

Fig. 3. Architecture of the Evaluation Application Suite

Service [19] to set up the (video stream) paths between the communicating CORBA objects. Integrated with that is the ability to attach an RSVP reservation to the underlying network connection as it is setup by the A/V Streaming Service.

4 Applying Managed QoS in DRE Applications

We are applying and evaluating the multi-layered managed QoS approach and mechanisms described in Section 2.1 and Section 3 to complex challenge problems in the avionics and remote sensor processing domains. Certain experimentation platforms involve high speed mobile airborne vehicles, whereas others reside on relatively fixed or slower moving ground platforms. The relevant QoS management includes trading off sensor quality and timeliness, and coordinating resource usage among competing applications, to satisfy changing mission requirements under dynamic, and potentially hostile, environmental conditions.

Figure 3 illustrates the architecture for the application suite that is motivating the specific directions for our work currently being developed and undergoing evaluation. It represents collections of three-stage pipelines that apply QuO, TAO, and TAO's implementation of the CORBA Audio/Video Streaming Service [18] alongside other relevant technologies under investigation to the following three stages:

1. ***Sensor sources***, (hosts 1-3) including processes with live camera feeds and those that replay from a file, which send video images to
2. ***Distributor processes***, (host 4) which are responsible for distributing the video to
3. ***Multiple receivers***, (hosts 5-7) including human-oriented video displays and CPU-intensive image processing software.

This application presents a wide variety of characteristics representative of many or most DRE applications involving constrained resources, varying conditions, and configurations, and varying data and processing characteristics, including:

- ***Varying data formats***, including MPEG and PPM, with different data sizes and compression characteristics.
- ***Varying network transports***, including wireless, LAN, and WAN, with variable and constrained bandwidth over both noisy and private channels.
- ***Varying image processing algorithms***, including image display and image recognition processes (the ATR – automated target recognition – process in Figure 3), with different CPU usage patterns.
- ***Varying granularities*** of real-time deadlines, ranging from microseconds to milliseconds and seconds.

In particular, managing real-time end-to-end QoS in this context 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 generally perceptible at above approximately 20 fps. Lower frame rates are visibly less smooth, but are viewable as long as other qualities, such as data fidelity and jitter, are controlled. Our DRE application can use 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 that the end viewer see an accurate and timely view of what the sensor is collecting, which implies a minimal latency requirement. Studies have indicated that humans can perceive a delay of more than 100-200 ms, so this provides a lower bound for timeliness requirement in cases where the video is meant for human viewing and precision action. In cases where the image is being processed automatically, the important threshold is for the latency to be low enough that there is no more current image. In the case of MPEG-1 where I-frames (full content frames) are two fps, that means a minimum latency of 500 ms.

Minimal jitter. Controlling the smoothness of the video can have greater impact on the quality of human perception than the frame rate. Controlling the jitter requires control all along the end-to-end path, since it can be affected by changes in the rate at which video is sent, latency of video delivery, and the rate at which frames are displayed. Some typical strategies for reducing jitter, such as buffering, are not as useful in real-time video because of the need for the video to be timely.

Image quality. The image must be of high enough quality, comprising the image size, pixel depth, etc., for the purpose it is being used. In the case of human viewing, this means the video must be large enough and clear enough to discern the detail 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 system and application directives, must control and coordinate the QoS so that the necessary allocations and tradeoffs 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.

Achieving end-to-end QoS requires managing the resources, including CPU and network bandwidth, along the entire path from source to sink. It requires making tradeoffs that consider user requirements, e.g., whether timeliness, fidelity, etc. is the

dominant characteristic. In our application architecture, we encode QoS measurement, control, and adaptation directives and policies in QuO contracts that are distributed throughout the application and 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.

By incorporating the OS and network mechanisms described in Section 3 that provide lower level resource control capabilities, we can integrate QuO contracts, services such as CORBA Audio/Video Streaming Service, and the underlying TAO Real-time CORBA middleware to establish task and network priorities and reservations end-to-end. These new integrated capabilities complement previous work [13] in which the data/processing characteristics (such as changing the frame rate or image size) were modified as well to satisfy operational requirements.

5 Empirical Results

Section 3 described mechanisms and services useful for managing CPU and network resources. Using the QuO and TAO middleware outlined in Section 2.1 to integrate and control these CPU and network resources, we are developing two complementary approaches to end-to-end QoS management:

- ***Priority-based*** – Using TAO's standard support for CORBA priorities to map to OS priorities and to network priorities based on Real-time CORBA and DiffServ.
- ***Reservation***-based – Reserving CPU cycles and network bandwidth based on the TimeSys Linux resource kernel and the IntServ mechanism.

This section describes the results of systematic experiments we have conducted in the context of the application described in Section 4 to evaluate the integration of priority- and reservation-based techniques using standards-based DRE middleware to manage predictable QoS end-to-end.

5.1 Priority-based End-to-End Adaptive QoS

As discussed in Sections 3.1, RT-CORBA supports the preservation of priorities across threads of activities in DRE applications by (1) mapping the importance of various application activities to corresponding operating system thread priorities and (2) propagating these priorities across the multiple hosts that the activity spans. RT-CORBA is less explicit about the communication transport and network, however. As described in Section 3.2, our approach is to use the RT-CORBA priority not only for mapping to thread priorities and application scheduling requirements, but also to map to DiffServ network priorities for end-to-end predictability and performance.

Empirical results for prioritization. We conducted a set of experiments to evaluate the improvement in predictability and performance when using RT-CORBA priorities to map to thread priorities and network DSCPs. The experiments consisted of two video senders transmitting video to two receiver servants under the following conditions:

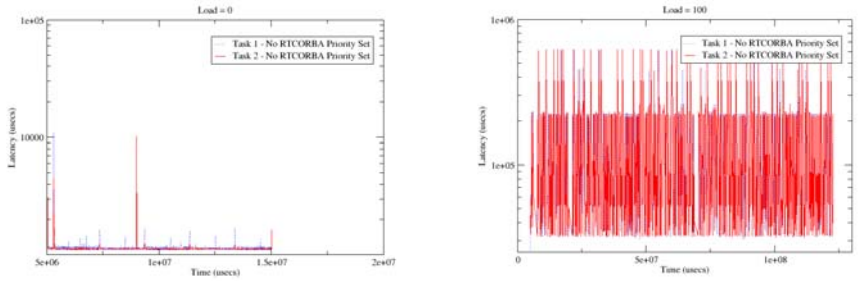

Fig. 4. No DSCP set on either task and no thread priorities; (a) no extra CPU or network load and (b) with extra CPU load and network traffic.

- ***Control runs***, using senders with no priorities and no network management, with and without contending CPU load and network traffic.
- ***Experimental runs*** with extra CPU load and network traffic, using thread priorities; DiffServ priorities; and thread priorities and DiffServ priorities combined.

This experiment tests the hypothesis that combined management of thread and network resources results in improved performance and predictability compared with no resource management and management of either one individual resource alone.

The testbed consisted of 4 PCs with 1 GHz AMD Athlon processors and 512 MB RAM, running Redhat Linux 7.1, with a 100 M bits-per-second (bps) Ethernet network. The sender machine hosted two identical tasks playing the role of video senders, generating GIOP messages at the rate of approximately 1.2 Mbps, the approximate bandwidth of MPEG video at 30 frames-per-second (fps). The receiver machine hosted two servants activated in two separate Portable Object Adapters (POAs), which dispatch operations to servants. A third machine played the role of a DiffServ-enabled router, while a fourth machine generated cross traffic in excess of 90 Mbps. We introduced continuous additional CPU load by making 100 consecutive calls to a function checking for large prime numbers.

Figure 4 illustrates the control runs, with both sender tasks having no RTCORBA (thread) priority and no network management of the video stream traffic. Figure 4a indicates the results with no extra load introduced, either in the CPU or in the network. As long as resources are plentiful, the latency of the video traffic for both streams was approximately 1.5 ms, with occasional spikes to approximately 10 ms. When extra network and CPU load was introduced, as illustrated in Figure 4b, performance and predictability degraded significantly. Latency fluctuates widely between a few milliseconds to nearly a second for both streams.

Figure 5 illustrates the results of the first set of experimental runs, evaluating the ability of thread priority alone. As mentioned above, the hypothesis is that while the thread priority mechanism defined by the RT-CORBA specification alone might improve performance and predictability, it is only a partial solution and cannot, by itself, lead to end-to-end QoS. To test this hypothesis, we set one sender task as high priority and the other low priority and increased the CPU load. Figure 5a illustrates that the

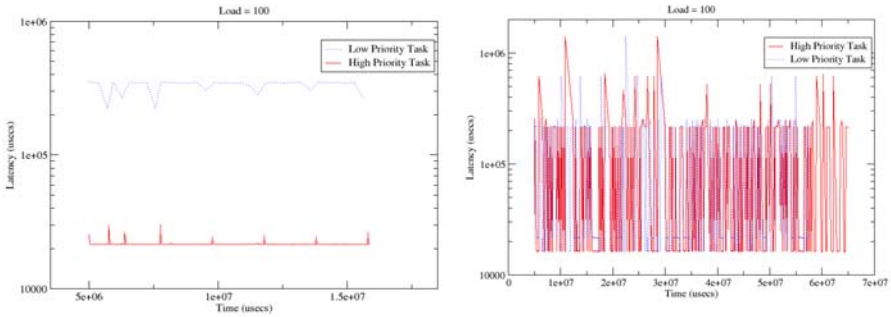

Fig. 5. Tasks with different priorities; no DSCP set on either task; and with introduced CPU load; (a) no traffic congestion and (b) with traffic congestion.

higher priority task (sender 1) exhibits significantly lower latency than the lower priority task (sender 2).

When network traffic is introduced, however, thread priorities are not sufficient to maintain QoS. The system becomes unpredictable even with RTCORBA priorities set, as Figure 5b illustrates. In fact, there is a possibility of a priority inversion when the high priority task finishes after the low priority task. As expected, the RT-CORBA mapping to thread priority has no capability to maintain QoS when the network is the bottleneck.

Figure 6 illustrates the efficacy of Diffserv alone. In the presence of introduced network traffic, Diffserv is sufficient to maintain predictable performance (Figure 6a). However, when extra CPU load is also introduced, the system experiences unacceptable jitter (Figure 6b). In contrast, Figure 7 illustrates the results of the experiment in which we use RT-CORBA priorities for mapping to both thread priorities and DSCPs. In this experiment, both senders get thread priorities and their DSCP set, giving them preferential treatment over the competing CPU load and network traffic, with sender 1

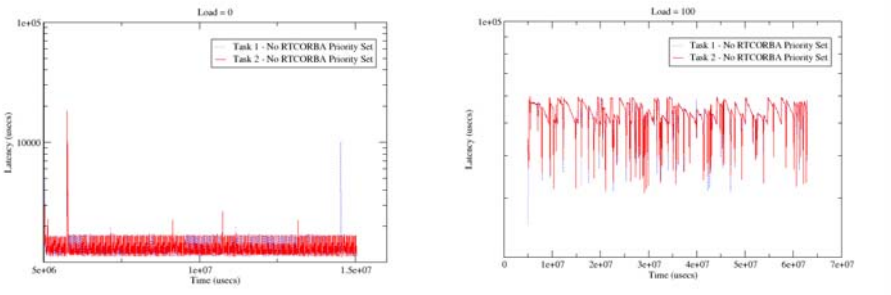

Fig. 6. Tasks with no thread priorities but with DSCP set on either task; and with introduced network traffic; (a) no CPU load and (b) with CPU load.

having the higher priority thread and higher network priority. Both senders become much more predictable, while sender 1's stream exhibits lower latency than sender 2's. Priority-based thread control combined with priority-based DiffServ network management is thus able to provide better end-to-end performance and predictability in the face of CPU and network contention than either can do individually.

5.2 Reservation-Based End-to-End Adaptive QoS

The previous sections described priority-based mechanisms for ensuring end-to-end QoS performance of applications. Priority mechanisms do not always work as well, however, in systems with unpredictable and dynamic load scenarios as they do in more static systems, where the load is periodic and the system is not affected to a significant degree by the external environment [29]. Priorities serve best to reflect relative timing requirements of tasks. Likewise, reservations serve best to reflect the absolute timing requirements of each task, when and if they are known.

Below, we describe experiments conducted with reservation-based resource management mechanisms in the context of the video streaming application and the CORBA Audio/Video Streaming Service outlined in Section 4. These results illustrate how the reservation resource mechanisms described in Sections 3.3 and 3.4 can be used in conjunction with adaptive DRE middleware to provide end-to-end QoS as an alternative to, or in concert with, priority-based techniques. The experiments evaluate network and CPU mechanisms separately and in conjunction with application adaptation, providing another step toward our ongoing research objective of a comprehensive end-to-end capability for QoS adaptive middleware.

Empirical results for network reservations. We conducted a set of experiments on the video delivery application to evaluate the effectiveness of network reservation to increase the predictability and performance of data delivery. Our experiments included two types of RSVP reservations: full reservations (1.2 Mbps, enough to support 30 fps) and partial reservations (670 Kbps). Since partial reservations are not sufficient to support full rate video (i.e., 30 fps), we experimented with using QuO-based

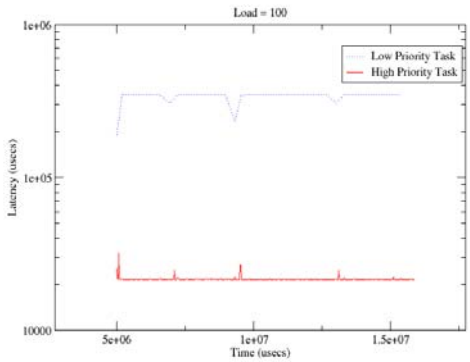

Fig. 7. Task 1 with High Thread Priority and DSCP set and with traffic and CPU load

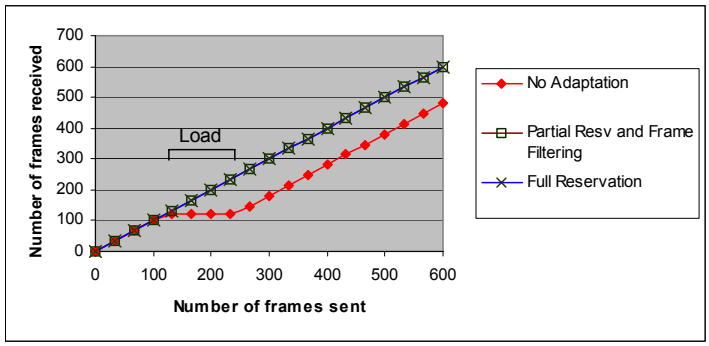

Fig. 8. Predictability of Image Delivery using Network Reservation

frame filtering also, i.e., reducing the video frame rate to a rate that the network (and network reservation) can support. In these experiments, we conducted experiments in every one of the following possible combinations:

1. No frame filtering and no reservation
2. No frame filtering and partial reservation
3. No frame filtering and full reservation
4. Frame filtering and no reservation
5. Frame filtering and partial reservation
6. Frame filtering and full reservation (This combination is included for completeness only. With a full reservation, no frame filtering is necessary.)

The sender/distributor and receiver boxes were 750 MHz Pentium III laptops with 512MB RAM, with 10 Mbps Ethernet between them. The video sender sent MPEG-1 video (approximately 1.2 Mbps for 30 fps) for 300 seconds. After 60 seconds elapsed, an extra 43.8 Mbps network load was generated for 60 seconds, then discontinued. The frame filtering cases dynamically reacted to network load by filtering frames down to 10 fps or 2 fps, whichever the network would support.

Figure 8 summarizes the predictability of video delivery in experiments 1, 5, and 6. With no adaptation, almost all of the frames sent while the system was under load were lost. With a partial reservation and frame filtering, the middleware dropped less important intermediate frames, but successfully delivered all full content frames (i.e., *I-frames*). With a full reservation, all frames were delivered successfully.

Table 1 describes the predictability (number of frames delivered), performance (latency), and jitter (standard deviation) under load of all the experimental cases. It illustrates that network reservation greatly increases the predictability and performance, and reduces the jitter when the system is heavily loaded.

Empirical results for CPU reservations. As with the priority-based experiments, end-to-end QoS reservations need to consider image processing as well as image delivery. To measure the ability to control the predictability of the CPU processing requirements of the application, we constructed an experiment where image frame data was transmitted from a client program to a CORBA middleware-based image processing server, simulating the automatic target recognition (ATR) function shown in Figure 3 of our application architecture. In the image-processing server, we ran three dif-

Table 1. Summary of Network Reservation Experimental Results

	Under Load		
	% Frames Delivered	**Average Latency**	**Standard Deviation**
No Adaptation	0.83%	324 ms	NMF
Partial Reservation	43.9%	742 ms	190.6
Full Reservation	100%	190 ms	42.3
No Reservation; Frame Filtering	95.04%	276 ms	146.6
Partial Reservation; Frame Filtering	99.18%	187 ms	143.6
Full Reservation; Frame Filtering	100%	171 ms	63.5

ferent computationally intensive edge detection algorithms (Prewitt, Sobel, and Hirsch algorithms [11]) from the Tools for Image Processing library [9].

We ran the experiments by sending many images to the image-processing server in the following different runs:

- Two *control* runs, which processed images with (1) no competing CPU load and (2) competing CPU load (and no CPU management) and
- An *experimental* case, where images were processed with competing CPU load and with CPU reservations added.

We ran these experiments on a Pentium-III 850 MHz machine with 480 megabytes of RAM, running Red Hat Linux 7.1 with version 2.4.7-timesys-3.0.145 of the Timesys real-time Linux kernel. We used four images in PPM format, 400x250 pixels, 300,060 bytes, and in RGB color. The sender continuously sent the images to the receiver via a CORBA interface. The receiver processed the image by invoking the Kirsch, Prewitt, and Sobel edge detection algorithms in sequence. We executed the algorithms without load, with competing CPU load, and with competing CPU load and a CPU reservation, and recorded the time that each algorithm took to process the image. The results are summarized in Table 2.

These results show that under load, the execution time of the edge detection algorithms increased significantly – Kirsch by +41%, Prewitt by +13%, and Sobel by +30%. The execution times of the edge detectors varied more than when there was no load, as illustrated by the higher standard deviations. This result may be explained by the fact that the load added was variable and not sustained.

Adding a CPU reservation reduced the execution time under load to values that are comparable to those exhibited with no load. The variability in the execution times was also much less than in the experiment with load but with no CPU reservation. CPU reservations therefore did its job as expected to reduce the latency and increase the predictability of executing tasks when there is competing CPU load.

Table 2. Summary of CPU Reservation Experimental Results

Algorithm	No Load		Competing CPU Load		CPU Load & CPU Reservation	
	Av. Proc. Time (ms)	Std. Dev.	Av. Proc. Time (ms)	Std. Dev.	Av. Proc. Time (ms)	Std. Dev.
Kirsch	44.3	0.08	56.5	11.8	44.5	0.87
Prewitt	44.6	0.09	51.5	9.2	44.7	0.16
Sobel	45.3	0.12	59.8	12.9	45.7	0.84

Our short-term future plans involve combining CPU and network reservation mechanisms in a middleware controlled end-to-end QoS management capability for our video streaming application.

6 Concluding Remarks

This paper describes recent advances we have made towards developing adaptive DRE systems with end-to-end QoS management by integrating and testing a variety of emerging operating system and network resource management mechanisms with our earlier work on standards-based COTS DRE middleware and adaptive QoS management frameworks. As our work becomes more completely integrated as common middleware – and is complemented with appropriate resource management binding and scheduling services and policies – it is enabling a new generation of flexible DRE applications that (1) have more precise control over their end-to-end resource management strategies and (2) can be easily reconfigured and adapted to dynamically changing network and computing environments.

Most cost effective, near term solutions to end-to-end system QoS management need to rely on underlying COTS components, such as operating systems, networks, and CPUs. To provide an effective end-to-end solution, however, mechanisms for managing resources at these architectural levels must be integrated with middleware. Until recently, capabilities for managing these resources outside of the narrow host or intra-network perspective were either missing, primitive, or non-standard. The attention being focused on QoS issues for distributed systems has prompted activities that have led to a variety of new mechanisms for low level resource control. Accordingly, our current R&D activity is focused on two topics:

1. How to integrate these emerging individual mechanisms effectively to form a consistent end-to-end control model and

2. How to integrate these low-level mechanisms with higher levels of middleware that can provide end-to-end and system-wide policies and strategies needed to complete our vision.

While this paper indicates that we are making significant progress, we have not yet achieved our final integration milestones. The results in Section 5 illustrate how we have made fundamental progress in separately supporting the two dominant paradigms: priority-and reservation-based approaches. We have demonstrated the effectiveness of implementations of these mechanisms (sometimes integrated, sometimes in isolation) to do what they were intended to do. We have made progress on integrating these low-level controls with higher-level middleware abstractions and policy mechanisms intended to unify the individual elements of the system. All of that work is continuing. After we have integrated, tested, and validated all of the individual mechanisms end-to-end, as well as with the upper levels of middleware for individual flows, we will then focus on the aggregate management policies and adaptive behavior that these levels of middleware are intended to enable. When all of these attributes are in place, we can then provide a more detailed decomposition of the roles and interfaces required to provide a complete service.

Ultimately, we contend that priority- and reservation-based approaches will both have their place in complex DRE application systems. It will therefore be important not only to provide both a consistent end-to-end priority and reservation capability stand-alone, but to characterize, understand, and manage the patterns of their individual use and the interplay and interactions between them.

One promising research direction is to combine priority-based mechanisms in conjunction with reservation mechanisms, using the priority paradigm to drive who gets reservations and to what degree. Since OS reservation allocation mechanisms are newer and have been applied less often than priority mechanisms, they also lag priority-based mechanisms in coordinating and standardizing the reservation mechanisms across distributed systems. Our future work aims at redressing this imbalance.

QuO and TAO software are available in open-source format at http://www.dist-systems.bbn.com/tech/QuO and http://deuce.doc.wustl.edu/Download.html.

References

1. BBN Technologies, “Quality Objects (QuO)”, http://www.dist-systems.bbn.com/papers.
2. 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 Processing, London, England, 1998.
3. D. Box, Essential COM, Addison-Wesley, Reading, MA, 1997.
4. 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.
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. C. Gill, D. Levine, and D. Schmidt, "The Design and Performance of a Real-Time CORBA Scheduling Service”, Real-Time Systems, The International Journal of Time-Critical Computing Systems, special issue on Real-Time Middleware, vol. 20, num. 2, 2001.

8. A. Gokhale and D. Schmidt, "Optimizing a CORBA IIOP Protocol Engine for Minimal Footprint Multimedia Systems", Journal on Selected Areas in Communications special issue on Service Enabling Platforms for Networked Multimedia Systems, vol. 17, num. 9, Sep 1999.
9. S. Grigorescu, C. Grigorescu, and A. Jalba. Tools for Image Processing, Version 0.0.1. http://www.cs.rug.nl/˜cosmin/tip/, 2002.
10. M. Henning and S. Vinoski, "Advanced CORBA Programming With C++", Addison-Wesley, 1999.
11. S. Hlavac and R. Boyle *Image Processing, Understanding, and Machine Vision, 2nd edition*. PWS Publishing Company, Pacific Grove, CA, 2nd edition, 1999.
12. IETF, An Architecture for Differentiated Services, *http://www.ietf.org/rfc/rfc2475.txt*
13. 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.
14. G. Kiczales, "Aspect-Oriented Programming", Proceedings of the 11th European Conference on Object-Oriented Programming, Jun 1997.
15. F. Kon, F. Costa, G. Blair, and R. Campbell, "The Case for Reflective Middleware," CACM, June 2002.
16. 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.
17. M deMiguel, "QoS-Aware Component Frameworks," The 10th Int'l Workshop on QoS, Florida, 2002.
18. 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.
19. Object Management Group, "Control and Management of Audio/Video Streams, OMG RFP Submission (Revised), OMG Technical Document 98-10-05", Oct 1998, Framingham. MA.
20. Object Management Group, "Realtime CORBA Joint Revised Submission", OMG Document orbos/99-02-12, March 1999.
21. 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.
22. I. Pyarali, C. O'Ryan, D. Schmidt, N. Wang, V. Kachroo, and A. Gokhale, "Applying Optimization Patterns to the Design of Real-time ORBs", Proceedings of the 5th Conference on Object-Oriented Technologies and Systems, San Diego, CA, May 1999.
23. J. Regehr and J. Lepreau. "The Case For Using Middleware To Manage Diverse Soft Realtime Schedulers," In *Proc. of the International Workshop on Multimedia Middleware*, Ottawa, Canada, October 2001.
24. 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 Computing (ISORC 2002)*, April 29 - May 1, 2002, Washington, DC.
25. 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.
26. D. Schmidt, D.Levine, and S. Mungee, "The Design and Performance of Real-Time Object Request Brokers," Computer Communications, April 1998.
27. 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.
28. TimeSys Corporation. *TimeSys Linux R/T User's Manual*, 2.0 edition, 2001.
29. Timesys Corporation. Predictable Performance for Dynamic Load and Overload, Version 1.0. http://www.timesys.com/files/whitepapers/Predictable_Performance_1_0.pdf, 2002.
30. 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.
31. 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.
31. 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.
33. L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala, "RSVP: A New Resource ReSerVation Protocol," *IEEE Network*, September 1993
34. 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.