

Design and Implementation of QoS enabled OO Middleware

Vishal Kachroo, Yamuna Krishnamurthy
and Fred Kuhns
{vishal,yamuna,fredk}@cs.wustl.edu
Department of Computer Science
Washington University
St. Louis, MO 63130, USA

Ronald G. Akers, Pradeep Avasthi
Surender Kumar and Vidya Narayanan
rona@rsch.comm.mot.com,
{CCOF59, CSK037, CVN065}@email.mot.com
Motorola
Schaumburg, IL 60196, USA*

Abstract

The current interest in the commodity Internet for commercial purposes has helped to fuel R&D into advanced networks and distributed applications. Much of this research is addressing a common problem of scaling the internet to accommodate traffic from advanced applications requiring stringent QoS guarantees.

This research has progressed along three different axes: 1) providing scalable and high performance core networking technologies, 2) defining architectures and protocols which provide explicit end-to-end quality-of-service (QoS) guarantees to applications and 3) developing advanced applications, distributed middleware and software development frameworks. Advanced testbeds such as Abilene for Internet2 provide an environment for these three areas to converge and produce the next generation distributed applications and infrastructure. This integration effort must occur at all levels, from hardware and internetworking protocols, to application definition and development frameworks.

This paper provides three contributions to the R&D efforts on the development of frameworks and architectures for QoS-enabled distributed applications. First, it motivates an object-oriented (OO) QoS-enabled application programming interface (API) that we developed to provide a uniform interface to applications, regardless of the underlying networking technology or QoS architecture. Second, common patterns of use and a general abstraction of QoS parameters are presented to describe the QoS-enabled API design and class structure. Third, we describe a QoS-enabled pluggable protocol framework that has been applied to the CORBA Audio/Video Streaming Service.

Keywords: CORBA-based Multimedia Streaming, QoS-enabled OO Middleware, Pluggable Protocols

*This work was supported in part by Motorola Commercial Government and Industrial Solutions Sector, Motorola Laboratories, DARPA contract 9701516, Boeing and Sprint.

1 Introduction

Motivation: Advanced, large-scale applications targeted for the next generation internet are motivating advances in scalable and adaptive high-speed networks and middleware. These applications require an increasingly broad range of features to support various quality of service (QoS) aspects, such as predictable performance, secure communications, availability and fault tolerance. Adaptive applications are also being developed [1] which request service guarantees and are able to adjust to changing demands and available resources. Applications range from traditional military applications such as avionics mission computing and tactical command and control systems to medical, manufacturing and commercial applications such as tele-immersion, process control and electronic commerce.

Large-scale, performance sensitive applications: One of the most demanding applications on the horizon is tele-immersion [2] which is the combination of teleconferencing, telepresence, and virtual reality. Tele-immersion will place stringent demands at all levels along the end-to-end path for distributed applications. It will require real-time, predictable behavior from the local endsystem in order to interact with the physical world within specific delay bounds or to present an image, or other stimuli, in real-time to users distributed across an internet [2]. Likewise, the network must provide predictable performance for low latency, high bandwidth applications [3].

Private users of the Next Generation Internet will increasingly rely on interactive applications and electronic commerce. These applications will include video conferencing tools, IP telephony, video-on-demand, electronic commerce and other advanced distributed applications such as tele-immersion. These emerging application scenarios will require varying levels of service guarantees from the network. The end-to-end Quality of Service (QoS) received by the applications will directly translate into a users perceived worth of the new applications and related services. For example, if a video

conference application routinely delivers packets late it will have a relatively low value to the user.

Corporations and health care providers already rely on computing and networking technology to provide services to their internal and external customers. Along with ubiquitous networking comes the opportunity to converge traditionally disparate services and facilities. For example, data and voice networking converge resulting in one network infrastructure to provide video, voice and data services. Virtual reality and tele-immersion will enable novel services such as virtual simulations, virtual classrooms, tele-medicine and even remotely performed medical surgical procedures. Expertise will be concentrated to reduce costs but still made available to remote locations.

The characteristics of the next-generation systems outlined above present resource requirements that can vary significantly at run-time. In turn, this increases the demands on end-to-end system resource management, thereby making it hard to simultaneously (1) create effective resource managers using traditional statically constrained allocators and schedulers and (2) achieve reasonable resource utilization. In addition, the mission-critical aspects of these systems require that they respond adequately to changing situational features in their run-time environment.

Meeting the increasing demands of advanced NGI applications motivates the need to integrate adaptive patterns and techniques at all levels of the distributed system.

Adaptive endsystems: Advances in operating system technology and techniques such as dynamically loadable network modules [4] provides an opportunity to dynamically configure endsystems to meet application requirements. Using technology developed for network devices endsystems are able to identify unique processing requirements within the kernel and load any necessary processing modules dynamically. These modules can be loaded either from the local host or from across the network.

Core networking technologies: During the past decade, there has been substantial R&D emphasis on *high-speed networking* and *performance optimizations* for network elements [5] and protocols [6]. These effort have paid off such that networking products are now available off-the-shelf that can support Gbps on every port, *e.g.*, Gigabit Ethernet and ATM switches. Moreover, OC-12 (622 Mbps) ATM connectivity in WAN backbones are becoming standard and OC-48 (2.4 Gbps) is being deployed for advanced networks such as Abilene [7] and Advanced Technology Demonstration Network (ATDnet) [8]. There are already plans to deploy OC-192 (9.6Gbps) within these backbones as it becomes practical.

Advanced architectures for modern high-performance routers and switches are being designed and constructed to support novel approaches for providing QoS. For example,

the Active Network Node (ANN) [9] project at Washington University is using the Washington University Gigabit Switch (WUGS) [10] switch with the Smart Port Cards (SPC) [11] to provide a robust environment to support active networking and QoS research and development.

QoS architectures and models: The various real-time applications demand QoS assurance at the endsystem and network resource levels. Providing QoS guarantees at both these levels ensures true end-to-end QoS. There is extensive ongoing research at both these levels. AQUA (Adaptive Quality of service Architecture) [12] is a resource-management architecture, at the endsystem level, in which applications and the OS cooperate to dynamically adapt to variations in resource requirements and availability. AQUA manages the CPU and network-I/O resources in an integrated fashion to provide predictable QoS. At the network resource level the current Internet supports only best-effort service, irrespective of user expectations. Moreover, application heterogeneity dictates that there be service heterogeneity and service differentiation. QoS architectures and models have been proposed to address the end-to-end QoS challenge. For example, the IETF has several working efforts directed to defining an architecture and proposing necessary protocols and infrastructure requirements. These working groups include Differentiated Services (DiffServ) [13], Integrated Services (IntServ) [14] and Integrated Services over Specific Link Layers (ISSLL) [15]. Additionally, the Internet2 QoS working group has proposed a testbed for Ip differentiated servers call the QBone [16]. These all support the allocation of resources to provide different levels of guarantees to applications.

IntServ is defined in RFC 1633 [17] and is intended to provide QoS transport over IP internets. IntServ effort uses RSVP (Resource ReSerVation Protocol) [6] for signaling resource requirements. IntServ requires flow classification and forwarding state for each active flow at each router along each QoS path. ISSLL is intended to provide QoS transport for IP over specific networking technologies.

As an alternative, the *Differentiated Services* (DiffServ) [18] working group started to address perceived scalability and implementation issues associated with IntServ. DiffServ aggregates flows into service classes rather than maintaining per flow state. Moreover, QoS requirements are specified out-of-band, removing the necessity for a signaling protocol such as RSVP. Packet classification is based on the setting of a few bits in the IP header.

Active network nodes: Recent advances in active network technologies [9] offer new flexibility to dynamically configure QoS management mechanisms and policies at the network switch level [19]. Using Field-Programmable Gate Array (FP GA) technologies in switch hardware offers the ability to reprogram switch hardware components that are either

physically inaccessible or that cannot be taken off-line for system performance reasons. Active network nodes with downloadable plug-in capabilities offer similar reprogramming capabilities for switch software. Taken together, these technologies offer significant flexibility for in-band and out-of-band QoS management at the network switch level.

Network protocols: Network protocols for enforcing and coordinating QoS management activities between network nodes and endsystems are increasingly important. In particular, technologies for efficient, scalable multicast are crucial to manage and constrain network loads. Furthermore, technologies such as group management that are employed by multicast management have further utility in such diverse areas as security, fault tolerance, and distributed resource reservation.

Distributed application development frameworks: During the same time period, there has also been substantial R&D emphasis on object-oriented (OO) communication *middleware*, including open standards like OMG’s Common Object Request Broker Architecture (CORBA) [20], as well as popular proprietary solutions like Microsoft’s Distributed Component Object Model (DCOM) [21] and Sun’s Remote Method Invocation (RMI) [22]. These efforts have paid off such that OO middleware is now available off-the-shelf that allows clients to invoke operations on distributed components without concern for component location, programming language, OS platform, communication protocols and interconnects, or hardware [23]. However, the general lack of support in off-the-shelf middleware for QoS specification and enforcement features; integration with high-speed networking technology; and performance, predictability, and scalability optimizations [24] has limited the rate at which advanced distributed applications have been developed to leverage advances in OO middleware.

Providing QoS to applications: Most existing approaches are highly platform/protocol-specific, however, which makes it hard to develop and deploy portable applications. The different R&D focuses outlined above have not, in general, addressed providing middleware with standard QoS models and interfaces. And very little has been done to provide application developers with a standard programming interface that can leverage the underlying advances to provide end-to-end QoS guarantees.

Application developers need a standardized framework and interfaces which allow for QoS specification and to receive guarantees from the underlying network and QoS infrastructure. There have been several attempts [25] at designing and implementing a unified QoS API that leverages the QoS features available in networks and end-systems. Our QoS API (1) provides a simple interface for the users to QoS enable their applications, (2) hides the underlying platform/protocol specific issues of a QoS implementation, and (3) is integrated

with middleware like CORBA, so the application not only continues to benefit from the middleware for distribution but also gets QoS guarantees through the standard middleware APIs.

The remainder of this paper is organized as follows: Section 2 provides an overview of CORBA; Section 3 describes the work that we have done to provide QoS APIs for applications to guarantee QoS both at the network and at the ORB end-system layer; Section 4 describes how these APIs have been used in ORB services like the Audio/Video service built on top of The ACE ORB (TAO); and Section 5 presents concluding remarks.

2 Synopsis of CORBA

CORBA Object Request Brokers (ORBs) allow clients to invoke operations on distributed objects without concern for object location, programming language, OS platform, communication protocols and interconnects, and hardware [26]. Figure 1 illustrates the key components in the CORBA reference model [27] that collaborate to provide this degree of portability, interoperability, and transparency.¹ Each component in the

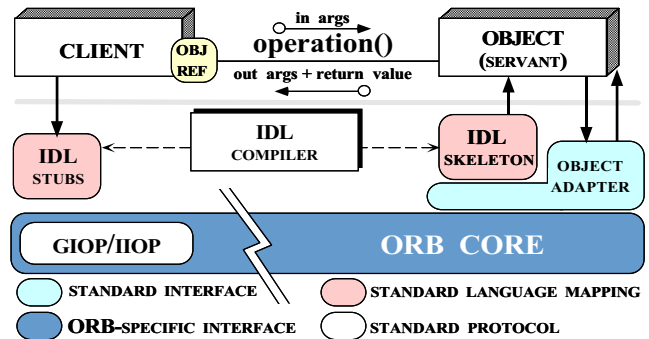


Figure 1: Key Components in the CORBA 2.x Reference Model

CORBA reference model is outlined below:

Client: A client is a *role* that obtains references to objects and invokes operations on them to perform application tasks. Objects can be remote or collocated relative to the client. Ideally, a client can access a remote object just like a local object, *i.e.*, `object→operation(args)`. Figure 1 shows how the underlying ORB components described below transmit remote operation requests transparently from client to object.

Object: In CORBA, an object is an instance of an OMG Interface Definition Language (IDL) interface. Each object is identified by an *object reference*, which associates one or

¹This overview only focuses on the CORBA components relevant to this paper. For a complete synopsis of CORBA’s components see [27].

more paths through which a client can access an object on a server. An *object ID* associates an object with its implementation, called a servant, and is unique within the scope of an Object Adapter. Over its lifetime, an object has one or more servants associated with it that implement its interface.

Servant: This component implements the operations defined by an OMG IDL interface. In object-oriented (OO) languages, such as C++ and Java, servants are implemented using one or more class instances. In non-OO languages, such as C, servants are typically implemented using functions and structs. A client never interacts with servants directly, but always through objects identified by object references.

ORB Core: When a client invokes an operation on an object, the ORB Core is responsible for delivering the request to the object and returning a response, if any, to the client. An ORB Core is implemented as a run-time library linked into client and server applications. For objects executing remotely, a CORBA-compliant ORB Core communicates via a version of the General Inter-ORB Protocol (GIOP), such as the Internet Inter-ORB Protocol (IIOP) that runs atop the TCP transport protocol. In addition, custom Environment-Specific Inter-ORB protocols (ESIOPs) can also be defined.

OMG IDL Stubs and Skeletons: IDL stubs and skeletons serve as a “glue” between the client and servants, respectively, and the ORB. Stubs implement the *Proxy* pattern [28] and provide a strongly-typed, *static invocation interface* (SII) that marshals application parameters into a common message-level representation. Conversely, skeletons implement the *Adapter* pattern [28] and demarshal the message-level representation back into typed parameters that are meaningful to an application.

IDL Compiler: An IDL compiler transforms OMG IDL definitions into stubs and skeletons that are generated automatically in an application programming language, such as C++ or Java. In addition to providing programming language transparency, IDL compilers eliminate common sources of network programming errors and provide opportunities for automated compiler optimizations [29].

Object Adapter: An Object Adapter is a composite component that associates servants with objects, creates object references, demultiplexes incoming requests to servants, and collaborates with the IDL skeleton to dispatch the appropriate operation upcall on a servant. Object Adapters enable ORBs to support various types of servants that possess similar requirements. This design results in a smaller and simpler ORB that can support a wide range of object granularities, lifetimes, policies, implementation styles, and other properties.

3 Unified Quality of Service API

A generic QoS Architecture is shown in Figure 2, with different QoS technologies like RSVP, Diffserv, MPLS, COPS and Bandwidth Brokers (BB) working cooperatively to enable end-to-end QoS. As the different QoS protocols become more and more mature, the need for a QoS API that applications can use as a unified interface to the underlying QoS protocols has grown tremendously. Firstly, the applications would like to shield themselves from the protocol specific details. Secondly, although the QoS protocols provide a mechanism for allocating resources between two end systems, they are not sufficient to address the translation required from the application level QoS parameters to the network level QoS parameters. Thirdly, the adaptive applications require a uniform mechanism to get notified of changes to the available resources so they can renegotiate the QoS. All these considerations motivate the design of a unified QoS API that addresses these issues in a platform/protocol independent way.

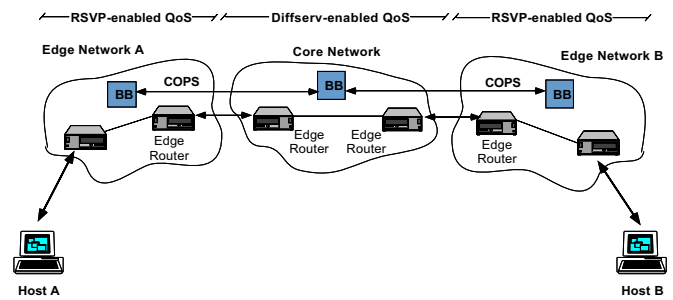


Figure 2: Generic QoS Architecture

Once the unified API is developed, it can be exposed to the applications through middleware like CORBA. This allows the applications to get QoS guarantees through standard middleware APIs and at the same time leverage all the usual features of a middleware. The API can also be integrated with higher level middleware services. At this level, the API would have added functionality like binding QoS to specific application data flows or translating standard QoS flow requirements to network QoS.

With this motivation we have designed and implemented a unified QoS API in the Adaptive Communication Environment (ACE) [30] to abstract the two separate implementations of IntServ available. They are, the GQoS implementation on Windows 2000 from Microsoft Corp.[31] and the RSVP API (RAPI) implementation on UNIX from Sun Microsystems Inc [32]. The interface provided by these to the application developers is totally disparate from each other. Our effort, in this work, has been to abstract out the common functionality from these implementations from an application developer’s perspective and to design APIs driven by these abstractions.

The unified ACE QoS API enables the users to QoS enable their applications without bothering about the underlying platform or QoS protocol implementation. These APIs are further exposed to the ORB, thereby, empowering the ORB with network level QoS guarantees. The APIs will also be used by the ORB services like the Audio/Video service to provide QoS to applications that make use of such services.

3.1 ACE QoS API

The ACE QoS API is built along the abstraction principles of ACE. It extends the existing ACE APIs by making them QoS aware and also introduces new APIs that allow the applications to manage QoS sessions. The API allows the application to set up QoS sessions which could be unicast or multicast. This can be done through a request for the appropriate session object to the session factory. Using the Factory Pattern relieves the application from managing the lifetime of QoS session objects. Once created, the QoS session object is added to the list of session objects that a socket has subscribed to. The Reactor pattern is used to separate the concern of listening for QoS events from processing these events.

3.2 Challenges in Designing the ACE QoS API

The QoS API for ACE was designed after abstracting the common patterns in the various QoS implementations currently available. During this process we kept the following challenges in mind :

Portability : The API should abstract the user from the underlying platform dependent implementation.

QoS Parameters : The key QoS parameters should be identified and the way to represent these should be orthogonal to the implementation.

Extensibility : The API should allow future implementations of QoS to be easily plugged in.

Further, the API should be capable of specifying QoS for a unicast or a multicast session. It should allow binding of flows to reservations. Also, there should be a mechanism to query for the QoS currently in place for a given unicast or a multicast session. A socket may join multiple multicast sessions. The user should be able to specify and query for QoS for each such session. The API must allow the application to set QoS for a new or an ongoing session. Finally, there should be a mechanism for the application to be notified of the changes in the QoS state like rejection of a request for QoS or changes to the existing set of reservations for a particular session. All these

requirements were kept in mind for designing the ACE QoS API.

3.3 Meeting the Design challenges for the ACE QoS API

The ACE QoS API addresses the previously described challenges in the following way :

Portability : The API shields the user from the underlying platform dependent implementations. We have so far integrated GQoS and RAPI into ACE. The applications or the ORB, however, program to an interface that hides the details of these implementations.

QoS Parameters : The API abstracts the data structures used to represent the various QoS parameters in different implementations. The advantage of this is that the application is presented with a simplified, easy to use view of these parameters.

Extensibility : The design of the API takes care that new implementations of QoS can be easily plugged in without changes to the application code that uses the API.

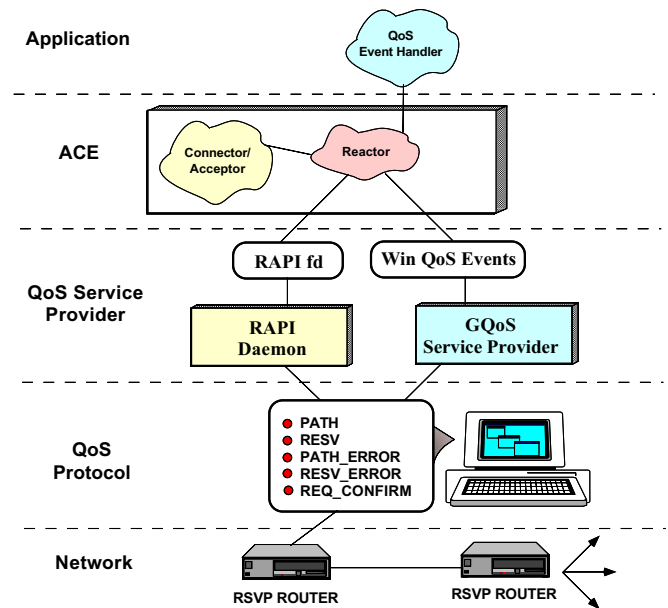


Figure 3: ACE QoS Event Handling

As mentioned earlier, a major part of any QoS mechanism is to provide the application with an ability to get notified about establishment of QoS state for the specified flows and get updates on changes to this state. In RSVP this is done through RSVP events. As shown in Figure 3, the ACE QoS API uses

the elegant Reactor pattern and a clever event handling mechanism to receive and handle these events uniformly for different implementations. The Reactor allows a server to decouple the event demultiplexing/dispatching from event handling. The synchronous event demultiplexer takes care of the event demultiplexing and an initiation dispatcher is used to call back the previously registered event handlers. The application instantiates a Reactor or uses the global singleton Reactor to listen for QoS events. A QoS Event Handler is registered with the reactor to handle the QoS events. This handler is responsible for updating the QoS state of a QoS session. It does that by calling a `rapi_dispatch ()` in case of UNIX and `WSAIocctl (SIO_GET_QOS)` in case of Win2K. It then proactively retrieves the QoS parameters from its QoS session object that was updated. Thus, the QoS Event Handler provides a mechanism to handle the asynchronous QoS events in a synchronous way. The application may decide not to get the QoS immediately after the occurrence of a QoS event but later when it needs to. It could also run the event handling mechanism in a different thread of control and hence be able to completely get QoS in a synchronous way. This model is motivated by the Asynchronous-Synchronous pattern that is already used in ACE for other similar problem contexts.

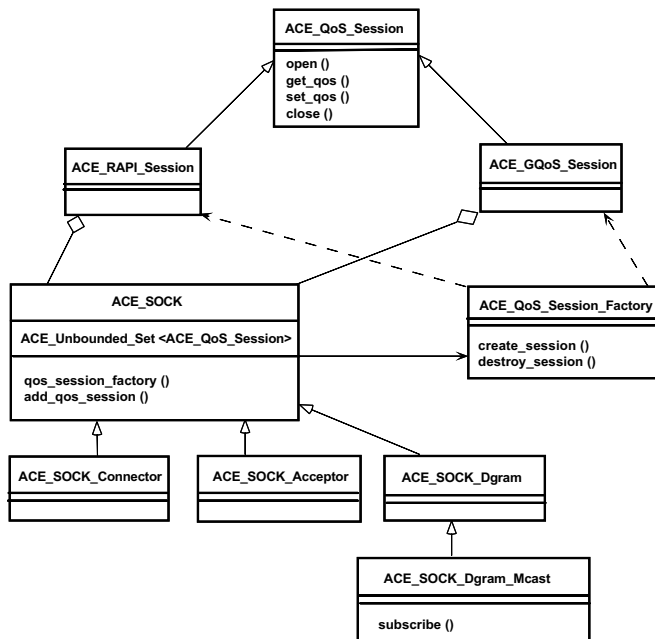


Figure 4: ACE QoS Class Design

The API defines the concept of a QoS session. This abstraction is orthogonal to the underlying socket e.g. a QoS session can be created before a socket is opened for data transmission that would use the QoS session. Figure 4 sketches the class design for the ACE QoS classes.

4 Enabling QoS in the CORBA Audio/Video Streaming Service

The QoS API defined in Section 3.1 provides an abstraction level to the underlying operating system, QoS implementation and networking infrastructure. While this is sufficient for some development environments, others require higher levels of abstraction which allow for the transport support of disparate protocols and models. The TAO based CORBA Audio/Video (A/V) Streaming Service [33], a multi-protocol and multimedia streaming service, is one such environment. Multimedia applications like Video-on-Demand and Video Conferencing can be developed using the A/V Service framework.

The QoS requirements of such multimedia applications depend on three main factors. First, the application class like interactive and non-interactive. Interactive applications require real-time responses and hence a predictable delivery at constant bit rate. Non-Interactive applications on the other hand are less stringent on the response time but may have high throughput demands. Second, depending on the media type like audio and video, appropriate streaming protocols should be used in order to preserve the reliability of delivery needed by audio and tolerate the unreliability for video. Third, the application specific requirements like tolerance and adaptation to changes in network resource availability. In order to provide such QoS to multimedia applications built using the A/V Service we need to enable QoS in the A/V Streaming Service.

Overview of CORBA A/V Streaming Service: The A/V Streaming Service is a distributed multimedia streaming CORBA service that facilitates the creation of data, video and audio streams between two or more media devices. Its implementation conforms to the CORBA-based A/V specification that provides the interfaces and semantics needed to control and manage A/V streams. These streams are terminated by stream endpoints that can be distributed across networks and are controlled by the Stream Control interface which acts as a stream manager. The A/V Streaming Service was designed to address the concerns of CORBA developers who wanted to leverage the language and platform flexibility of CORBA but could not accommodate the overhead imposed when transferring data through CORBA method calls.

The A/V Streaming Service combines the flexibility of CORBA and the efficiency of other protocols such as UDP, TCP and RTP through a pluggable protocol framework. The management and stream set-up is done through CORBA methods, while the data transfer is through UDP and other lower level protocols. Although the main intention of this service is to stream audio and video, the streams that are set up can be used for any type of data.

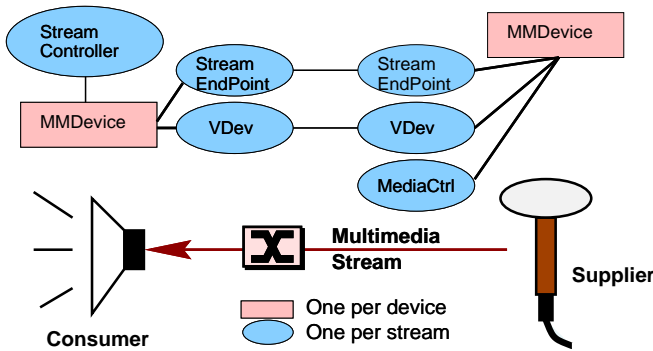


Figure 5: A/V Streaming Service Components

Support for QoS in A/V: The A/V specification has provided interfaces and semantics to enable end-to-end QoS for the individual flows. It allows the application developer specify the required application or network level QoS parameters like video frame rate or audio sample rate. These QoS parameters are specified using the CORBA Property Service [33] as name, value pairs. The A/V specification enforces the support for a mandatory set of Network QoS parameters like token bucket, peak-bandwidth and token rate among others. It also provides methods for negotiating QoS between two peer media devices and also modifying the QoS if there is a violation in the initially guaranteed QoS or the specified QoS cannot be met. Though the A/V specification provides interfaces to specify and modify QoS it leaves it up to the A/V implementation to enforce the end-to-end QoS.

4.1 Using the ACE QoS API to implement QoS in the A/V specification:

In the discussion on the QoS support provided by the A/V specification in Section 4 we saw that the specification leaves it to the implementation to enforce end-to-end QoS. For our TAO based A/V Service implementation we have designed a framework based on the QoS APIs that we have developed and described in Section 3.1. This framework provides an easy-to-use interface that abstracts the underlying QoS specific details to the A/V Service. Using this framework we can build QoS support into the A/V Service and this will allow the application developers get end-to-end quality of service by just specifying the QoS that they require for the different flows which will be translated, enforced and modified by the QoS enabled A/V Service, transparent to the developer. The framework comprises three main components namely the QoS mapping component, the QoS Monitoring and Adaptation component and the QoS-based Transport API component. We go on to describe each of these components in detail.

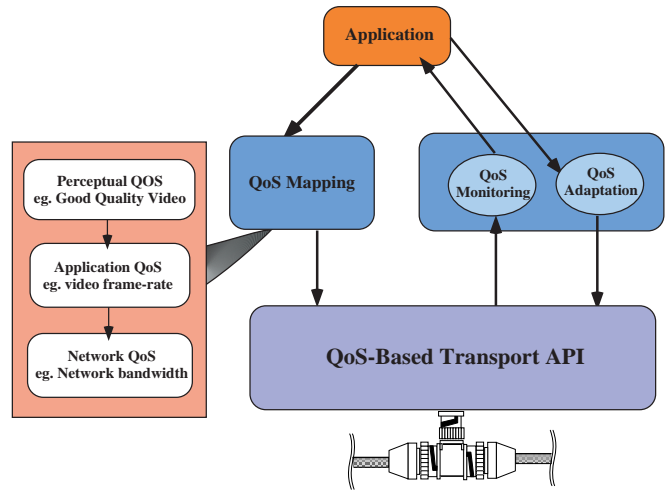


Figure 6: QoS Components

QoS mapping: QoS Mapping is an essential component of the framework that facilitates the translation of QoS parameters between different levels in the protocol stack. In the present scenario it helps to translate between the application level QoS to network level QoS. This allows the application developer to specify QoS as a perceptual quality, for example, the application developer can specify the video quality by the frame rate for a video flow. It is the responsibility of the QoS mapper to translate the frame rate to bandwidth requirements at the network level. QoS mapping is essential both during resource allocation and during renegotiations. Good mapping rules need to be identified to avoid reservation of too much or too little resources.

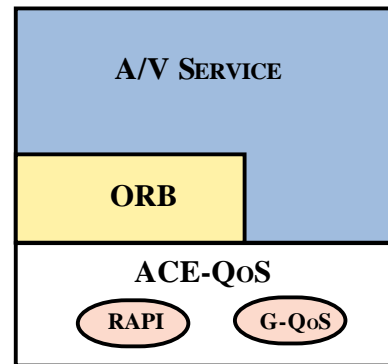


Figure 7: QoS-Based Transport API

QoS monitoring and adaptation: These components are essential for applications that require QoS guarantees but are more flexible in their needs. The monitoring component measures the end-to-end QoS of the flows over a finite period of

time. If there are any violations in the QoS guaranteed then it notifies the application of the available resources at that time. The application can then decide if the QoS available is sufficient for its needs and if not it renegotiates the QoS. This renegotiation is done by the QoS adaptation component. Adaptation can be at the transport (eg. flow control), application (eg. MPEG-II coding rate adaptation) and at the signalling (eg. QoS renegotiation) levels. Various adaptation algorithms can be used. This requires an extensible design of the QoS Adaptation component.

QoS-Based transport API: This is the most important component as it enforces end-to-end QoS by reserving network resources as per the user requirements. It should provide calls for provisioning, control (renegotiation and violation notification) and media transfer. The ACE QoS APIs provide all these functionalities. Figure 7 shows that the A/V Service is built over TAO and ACE for flow control processing and media transfer respectively. This allows the A/V Service leverage the ACE QoS APIs. As mentioned earlier in the paper the ACE QoS API is an abstraction of the G-QoS and RAPI APIs. It uses these underlying APIs to provision the specified QoS to individual flows. The QoS required by the application is translated from application level to network level parameters by the QoS mapping component and passed through the generic QoS mechanism designed as part of the ACE QoS API. The ACE QoS API provides means of detecting and notifying QoS violations that can be used by the QoS Monitoring and Adaptation layers for renegotiation of QoS between peer media devices. The ACE QoS APIs provide an abstraction of the underlying QoS protocols and other network details.

4.2 Design Challenges for Enabling QoS in CORBA A/V Streaming Service

We would need to provide a generic application level QoS API framework with the components identified in Section 4.1 that can be used for enforcing QoS in different applications. This API will mediate between the application and the underlying ACE QoS APIs to enforce end-to-end QoS for individual flows. The A/V Service supports different flow protocols. So the API that we design should provide a common interface to the different flow protocols. This requires a good understanding of the different flow protocols. The QoS mapping, monitoring and adaptation components should be extensible to facilitate the addition of new algorithms for realizing the functionalities for different use cases.

5 Concluding Remarks

In order to build performance-sensitive applications, such as video-on-demand, teleconferencing, a generic unified QoS

API is required that abstracts the platform specific details of the underlying QoS implementation. Applications built using the CORBA Audio/Video Streaming Service, for instance, could leverage these APIs for getting end-to-end QoS guarantees.

The paper identifies the different components that would be required to build a QoS framework. This QoS Framework would expose the QoS APIs to the application through the ORB. It can be used to provide QoS guarantees to the different flows in TAO's A/V Streaming Service.

References

- [1] J. A. Zinky, D. E. Bakken, and R. Schantz, "Architectural Support for Quality of Service for CORBA Objects," *Theory and Practice of Object Systems*, vol. 3, no. 1, 1997.
- [2] A. Network and I. Services, "National Tele-Immersion Initiative." <http://www.advanced.org/tele-immersion>.
- [3] J. Lanier, "Tele-Immersion: The Ultimate QoS-Critical Application," in *First Internet2 Joint Applications/ Engineering QoS Workshop*, May 1998.
- [4] D. Decasper, Z. Dittia, G. Parulkar, and B. Plattner, "Router plugins - a modular and extensible software framework for modern high performance integrated services routers," Department of Computer Science TR-98-08, Washington University in St. Louis, February 1998.
- [5] C. P. et al., "A fifty gigabit per second ip router," *IEEE Journal of Transactions on Networking*, vol. 6, pp. 237-248, June 1998.
- [6] R. Braden et al, "Resource ReSerVation Protocol (RSVP) Version 1 Functional Specification," *Network Working Group RFC 2205*, pp. 1-112, Sep 1997.
- [7] U. C. for Advanced Internet Development, "Abilene is an advanced backbone for the Internet2 project." <http://www.internet2.edu/abilene/>.
- [8] ATD, "Advanced Technology Demonstration Network." <http://www.atd.net/>.
- [9] D. Decasper, G. Parulkar, S. Choi, J. DeHart, T. Wolf, and B. Plattner, "A Scalable, High Performance Active Network Node," *IEEE Network Magazine*, vol. 13, January/February 1999.
- [10] J. Turner and N. Yamanaka, "Architectural Choices in Large Scale ATM Switches," *ICICE Transactions*, 1998.
- [11] W. N. Eatherton and T. Aramaki, "SPC Specification," Applied Research Lab, Working Notes ARL-WN-98-02, Washington University, St. Louis, 1998.
- [12] R. F. K. Lakshman, Raj Yavatkar, "Integrated CPU and Network-I/O QoS Management in an Endsystem," in *Proceedings of the IFIP Fifth International Workshop on Quality of Service (IWQoS '97)*, 1997.
- [13] D. S. (diffserv), "IETF." <http://www.ietf.org/html.charters/diffserv-charter.html>.

- [14] I. S. (intserv), "IETF." <http://www.ietf.org/html.charters/intserv-charter.html>.
- [15] I. S. over Specific Link Layers (issll), "IETF." <http://www.ietf.org/html.charters/issll-charter.html>.
- [16] I. Q. W. G. Draft, "QBone Architecture (v1.0)," tech. rep., Internet2, August 1999.
- [17] B. Braden, D. Clark, and S. Shenker, "Integrated services in the internet architecture," *Network Information Center RFC 1633*, June 1994.
- [18] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An architecture for differentiated services," *Network Information Center RFC 2475*, December 1998.
- [19] S. Choi, D. Decasper, J. Dehart, R. Keller, J. Lockwood, J. Turner, and T. Wolf, "Design of a Flexible Open Platform for High Performance Active Networks," in *Allerton Conference on Communication, Control, and Computing*, (Monticello, Illinois), September 1999.
- [20] Object Management Group, *The Common Object Request Broker: Architecture and Specification*, 2.2 ed., Feb. 1998.
- [21] D. Box, *Essential COM*. Addison-Wesley, Reading, MA, 1997.
- [22] A. Wollrath, R. Riggs, and J. Waldo, "A Distributed Object Model for the Java System," *USENIX Computing Systems*, vol. 9, November/December 1996.
- [23] S. Vinoski, "CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments," *IEEE Communications Magazine*, vol. 14, February 1997.
- [24] D. C. Schmidt, D. L. Levine, and S. Mungee, "The Design and Performance of Real-Time Object Request Brokers," *Computer Communications*, vol. 21, pp. 294–324, Apr. 1998.
- [25] B. Riddle, A. Adamson, "A QoS API Proposal." Pre-Workshop Draft, May 1998. <http://www.internet2.edu/qos/may98Workshop/html/apiprop.html>.
- [26] M. Henning and S. Vinoski, *Advanced CORBA Programming With C++*. Addison-Wesley Longman, 1999.
- [27] Object Management Group, *The Common Object Request Broker: Architecture and Specification*, 2.3 ed., June 1999.
- [28] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley, 1995.
- [29] E. Eide, K. Frei, B. Ford, J. Lepreau, and G. Lindstrom, "Flick: A Flexible, Optimizing IDL Compiler," in *Proceedings of ACM SIGPLAN '97 Conference on Programming Language Design and Implementation (PLDI)*, (Las Vegas, NV), ACM, June 1997.
- [30] D. C. Schmidt and T. Suda, "An Object-Oriented Framework for Dynamically Configuring Extensible Distributed Communication Systems," *IEE/BCS Distributed Systems Engineering Journal (Special Issue on Configurable Distributed Systems)*, vol. 2, pp. 280–293, December 1994.
- [31] Microsoft Corp., "QOS_SPEC.DOC revision 3.1." ftp.microsoft.com/bussys/WINSOCK/winsoc2/gqos_spec.doc, September 1998.
- [32] Sun Microsystems Inc., "Solstice Bandwidth Reservation Protocol version 1.0." www.sun.com/software/bandwidth/rsvp/spec.html.
- [33] S. Mungee, N. Surendran, and D. C. Schmidt, "The Design and Performance of a CORBA Audio/Video Streaming Service," in *Proceedings of the Hawaiian International Conference on System Sciences*, Jan. 1999.