A High-performance Endsystem Architecture for Real-time CORBA

Douglas C. Schmidt, Aniruddha Gokhale, Timothy H. Harrison, and Guru Parulkar

{schmidt,gokhale,harrison,guru}@cs.wustl.edu
Department of Computer Science, Washington University
St. Louis, MO 63130, USA

This paper will appear in the feature topic issue on Distributed Object Computing in the IEEE Communications Magazine, Vol. 14, No. 2, February 1997.

Abstract

Many application domains (such as avionics, telecommunications, and multimedia) require real-time guarantees from the underlying networks, operating systems, and middleware components to achieve their quality of service (QoS) requirements. In addition to providing end-to-end QoS guarantees, applications in these domains must be flexible and reusable. Requirements for flexibility and reusability motivate the use of object-oriented middleware like the Common Object Request Broker Architecture (CORBA). However, the performance of current CORBA implementations is not yet suited for hard real-time systems (e.g., avionics) and constrained latency systems (e.g., teleconferencing).

This paper describes the architectural features and optimizations required to develop real-time ORB endsystems that can deliver end-to-end QoS guarantees to applications. While some operating systems, networks, and protocols now support real-time scheduling, they do not provide integrated solutions. The main thrust of this paper is that advances in real-time distributed object computing can be achieved only by systematically pinpointing performance bottlenecks; optimizing the performance of networks, ORB endsystems, common services, and applications; and simultaneously integrating techniques and tools that simplify application development.

1 Introduction

An increasingly important class of distributed applications require stringent quality of service (QoS) guarantees. These applications include telecommunication systems (*e.g.,* call processing and switching), avionics control systems (*e.g.,* operational flight programs for fighter aircraft), multimedia (*e.g.,* video-on-demand and teleconferencing), and simulations (*e.g.,* battle readiness planning). In addition to requiring QoS guarantees, these applications must be flexible and reusable.

The Common Object Request Broker Architecture (CORBA) is a distributed object computing middleware standard defined by the Object Management Group (OMG) [1]. CORBA is intended to support the production of flexible and reusable distributed services and applications. Many implementations of CORBA are now available. Two of the most popular ORBs include IONA's Orbix, which is widely considered as the market leader, and Visigenic's VisiBroker, which is included in Netscape 4.0.

Our experience using CORBA on telecommunication, avionics, and medical projects [2] indicates that it is well-suited for request/response applications over lower-speed networks (such as Ethernet and Token Ring). However, CORBA is not well-suited for performance-sensitive real-time applications, for the following reasons:

• **Lack of standard QoS policies and mechanisms:** The CORBA specification defines neither the policies nor mechanisms for providing end-to-end QoS guarantees, *i.e.,* from application to application across a network. For instance, there is no standard way for clients to indicate the relative priorities of their requests. Likewise, there are no means for clients to inform the ORB how frequently an isochronous object service should execute.

• **Lack of real-time features:** CORBA does not define key features that are necessary to support real-time programming. For instance, there is no standard way to invoke CORBA requests asynchronously, the CORBA specification does not require the ORB to notify clients when transport layer flow control occurs, and no threading model is defined by the CORBA specification. As a result, it is hard to write a standard CORBA application that is guaranteed not to block indefinitely when endsystem and network resources are temporarily unavailable.

• **Lack of performance optimizations:** Existing ORBs incur significant run-time throughput and latency overhead [3, 4, 2, 5]. These overheads include excessive data copying, non-optimized presentation layer conversions, internal message buffering strategies that produce non-uniform latency, inefficient demultiplexing algorithms, long chains of intra-ORB virtual function calls, and lack of integration with underlying OS and network QoS mechanisms.

This paper describes an integrated *ORB endsystem architecture* for constructing real-time ORBs (RT ORBs) that can ensure end-to-end QoS for applications. QoS guarantees can be both *deterministic* (*e.g.,* hard real-time avionics applications where meeting QoS guarantees is crucial) and *statistical* (*e.g.,* latency constrained applications like teleconference and video-on-demand where minor fluctuations in scheduling and reliability guarantees are tolerated).

This paper is organized as follows: Section 2 outlines the features and optimizations required in ORB endsystems to provide end-to-end QoS guarantees for applications; Section 3 summarizes results from our studies [3, 4, 2, 5] of performance overhead in existing CORBA implementations; Section 4 describes the feature enhancements and optimizations we are developing to support high-performance, real-time ORB endsystems; and Section 5 presents concluding remarks.

2 Endsystem Requirements for Real-time CORBA

CORBA Object Request Brokers (ORBs) allow clients to invoke methods on target object implementations without concern for:

- *Object location* – target objects can be located locally or remotely;
- *Programming language* – such as C/C++, Java, Ada95, and Smalltalk;
- *OS platform* – such as Win32, OS/2, UNIX, and MVS;
- *Communication protocols and networks* – such as TCP/IP, IPX/SPX, FDDI, ATM, and Fast Ethernet;
- *Hardware* – such as RISC vs. CISC.

The components in Figure 1 support this level of transparency.

Steve Vinoski's article [6] (in this issue of the *IEEE Communications Magazine*) describes the CORBA middleware components (*e.g.,* the Object Request Broker, IDL Stubs and Skeletons, Object Adapter, etc.). However, an ORB endsystem is more than just middleware – it also contains the network adapters, operating system I/O subsystem, communication protocols, and common object services. The remainder of this section outlines the requirements of RT ORB endsystems:

• **Policies and mechanisms for specifying end-to-end application QoS requirements:** ORB endsystems that provide end-to-end QoS must allow applications to specify their QoS requirements at a high level with a small number of parameters (typically throughput, latency, and reliability). For instance, video-conferencing groupware may require high throughput and low latency. In contrast, high reliability may be more important requirements for electronic fund transfer systems. QoS specification is not addressed by the current

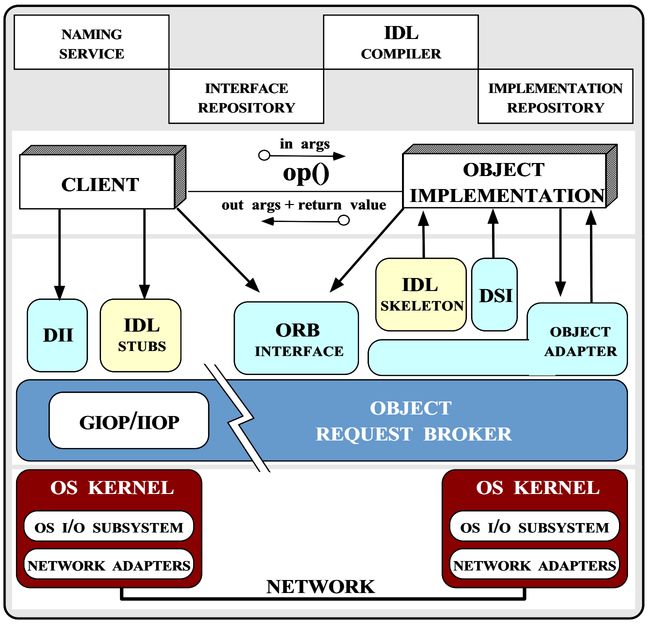

Figure 1: ORB Endsystem Components for Real-Time CORBA

CORBA specification. Section 4.4 outlines our mechanism for allowing applications to specify their QoS parameters on a per-request or per-object basis.

• **Optimized real-time operating system and network:** Regardless of the ability to specify QoS requirements, ORBs cannot deliver end-to-end QoS guarantees to applications without network and OS support for predictable I/O operations. Therefore, ORB endsystems must be capable of scheduling resources such as CPUs, memory, storage throughput, network adapter throughput, and network connection bandwidth and latency. For instance, OS scheduling mechanisms must allow high priority CORBA requests to run to completion and prevent them from being blocked indefinitely by lower priority operations [7, 8]. Section 4.1 describes an OS I/O subsystem and network adapter that can provide end-to-end gigabit data rates and ∼10 msec latency to CORBA applications.

• **Optimized real-time communication protocols:** The throughput, latency, and reliability requirements of multimedia applications like teleconferencing are more stringent and varied than those found in traditional applications like remote login or file transfer. Likewise, the channel speed, bit-error rates, and services (such as isochronous and bounded-latency delivery guarantees) of networks like ATM exceed those offered by traditional networks like Ethernet. Therefore, ORB endsystems must provide a range of communication protocols that can be customized and optimized for specific application requirements and network/host environments. Section 4.2 outlines optimizations for the CORBA General Inter-ORB Protocol (GIOP) [1], which specifies the request format and transmission protocol that enables interoperability among heterogeneous ORBs.

• **Optimized real-time request demultiplexing and dispatching:** ORB endsystems must demultiplex and dispatch incoming CORBA requests to the appropriate method of the target object. In conventional ORBs, demultiplexing occurs at multiple layers. Layered demultiplexing is often inappropriate, however, for real-time applications. In addition to increasing overhead, the layered demultiplexing and dispatching mechanisms in conventional ORBs neither schedule nor prioritize demultiplexing behavior. Therefore, the ORB endsystem must provide mechanisms (such as packet filters [9], de-layered protocol stacks [10], direct demultiplexing [11], and real-time upcalls [7]) that perform CORBA request demultiplexing and dispatching efficiently and predictably. Section 4.3 outlines a de-layered demultiplexing mechanism and a real-time upcall mechanism that processes CORBA requests predictably regardless of the number of active connections, application-level target object implementations, and operations defined in IDL interfaces.

• **Optimized memory management:** On modern RISC hardware, data copying consumes a significant amount of CPU, memory, and I/O bus resources. Therefore, multiple layers in an ORB endsystem (*e.g.,* the network adapters, I/O subsystem protocol stacks, Object Adapter, and presentation layer) must collaborate to minimize data copying [12]. Section 4.1 outlines our zero-copy memory management mechanism, which behaves predictably and efficiently irrespective of user buffer sizes and endsystem workload.

• **Optimized presentation layer:** Presentation layer conversions transform application-level data into a portable format that masks byte order, alignment, and word length differences. There are many techniques for reducing the cost of presentation layer conversions. For instance, [13] describes the tradeoffs between using compiled versus interpreted code for presentation layer conversions. Compiled marshaling code is efficient, but requires excessive amounts of memory, which is problematic in many embedded real-time environments. In contrast, interpreted marshaling code is slower, but more compact. Section 4.4 outlines how RT ORB endsystems must support worst case guarantees for both interpreted and compiled marshaling operations.

It is important to recognize that requirements for high performance may conflict with requirements for real-time determinism. For instance, real-time scheduling policies often rely on the predictability of endsystem operations like thread scheduling, demultiplexing, and message buffering. However, certain optimizations (such as using self-organizing search structures to demultiplex CORBA requests) can increase the average performance of operations, and yet decrease the predictability of any given operation. Therefore, our ORB endsystem is designed with an open architecture that allows applications to select the appropriate tradeoffs between average-case and worst-case performance. Moreover, where possible, we use algorithms and data structures that can optimize for both. For instance, de-layered demultiplexing paths can increase ORB performance *and* predictability.

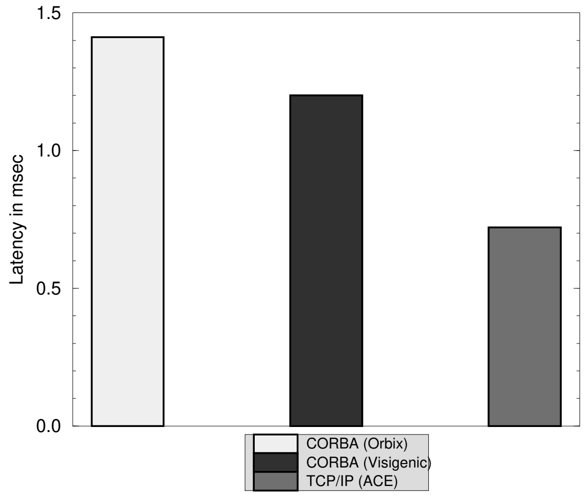

Figure 2: Latency for Invoking Parameterless Twoway Operations over ATM

3 Performance Overhead in Current CORBA Implementations

Real-time CORBA applications are not supported efficiently by conventional CORBA implementations. The primary sources of overhead include (1) excessive presentation layer conversions and data copying, (2) inefficient server demultiplexing techniques, (3) unpredictable buffering algorithms used for network reads and writes, (4) long chains of intra-ORB virtual function calls, and (5) improper choice of underlying OS system calls [3, 4, 2, 5].

Figures 2, 3, and 4 illustrate the latency for sending parameterless requests, octets, and structs between a client and a target object over an otherwise unused 155 Mbps ATM network. These tests were conducted using a modified version of the TTCP benchmarking tool. The TTCP tool was enhanced to support three communication mechanisms: two widely used implementations of CORBA (Orbix 2.1 and VisiBroker 2.0) and an implementation written in ACE [14] (which is a lightweight C++ network programming framework implemented directly atop sockets and TCP/IP).

The results of the TTCP benchmarks over ATM were as follows:

• **Latency for parameterless invocations:** Figure 2 illustrates the latency of the three communication mechanisms for invoking parameterless twoway operations. This figure shows the baseline latency for VisiBroker and Orbix is approximately twice that of the TCP/IP implementation written with ACE.

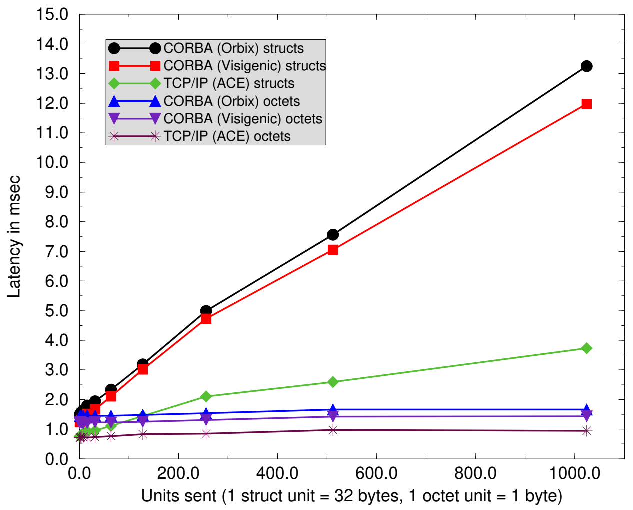

Figure 3: Latency for Sending Octets and Binary Structs over ATM

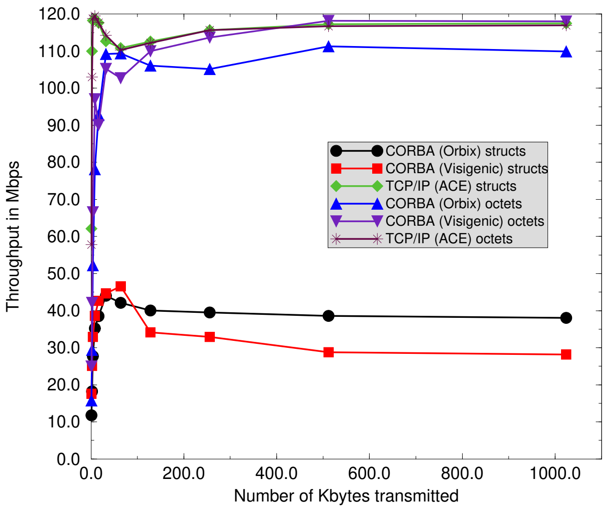

Figure 4: Throughput for Sending Octets and Binary Structs over ATM

• **Latency for untyped octet buffers and binary structs:** Figure 3 illustrates the latency for sending relatively small buffers of untyped `octets` and `structs` (containing binary data in the form of `shorts`, `longs`, and `doubles`) The buffers ranged from 1 to 1,024 units in powers of two (where `octet` units are 1 byte each and `struct` units are 32 bytes each). For `octets`, the performance curves are similar to those in Figure 2, with the latency of the the CORBA implementations roughly twice as high as the ACE implementation. For binary structs, the CORBA latency was more than four times higher than the ACE version as the buffers increased in size.

• **Throughput for untyped octet buffers and binary structs:** Figure 4 illustrates the throughput for sending large streams of untyped `octets` and binary `structs`. The data streams were transmitted in buffer sizes ranging from 1 K to 1,024 K bytes by powers of two. For untyped `octets`, the ACE implementation outperforms the CORBA implementations by ∼10-15% for buffer sizes upto 8 K. Beyond that, the performance of all three implementations is similar, with Orbix performing ∼5-10% below ACE and VisiBroker. For binary `structs`, however, the throughput performance of the CORBA versions is substantially lower than the ACE version. Both CORBA implementations achieved only 26-30% of the throughput of the ACE implementation.

In general, the CORBA implementations performed poorly when sending binary `structs` because of excessive copying and marshaling/demarshaling overhead, many layers of virtual function calls, and excessive writes resulting from non-optimal fragmentation of request buffers. In the following section, we discuss an RT ORB architecture called TAO (The ACE ORB) that alleviates these sources of overhead. In addition, the TAO architecture addresses a more fundamental issue – that of providing mechanisms to specify and deliver end-to-end QoS guarantees to applications.

4 A High-performance ORB Endsystem for Real-Time CORBA

This section describes network interface, operating system, communication protocol, and CORBA middleware mechanisms we are developing to implement a high-performance ORB endsystem for real-time CORBA called TAO. Figure 5 illustrates the components in the TAO architecture. These include a *Gigabit I/O subsystem* that optimizes conventional OS I/O subsystems to execute at Gigabit rates over high-speed ATM networks; a suite of *real-time GIOP/IIOP protocols* that provide efficient and predictable transmission of requests using standard CORBA interoperability protocols; a *real-time Object Adapter* that schedules and dispatches CORBA requests in real-time; and a *application-specific components* that optimize key sources of overhead in current ORBs and provides features that support end-to-end QoS guarantees to applications and higher-level CORBA services.

4.1 Gigabit I/O Subsystem

To implement end-to-end QoS guarantees, we are developing a high-performance network I/O subsystem. At the heart of this subsystem is a daisy-chained interconnect comprising a number of ATM Port Interconnect Controller (APIC) chips [15]. APIC is designed to sustain an aggregate bi-directional data rate of 2.4 Gbps.

Our Gigabit I/O subsystem builds on the APIC to enhance conventional operating systems with a zero-copy buffer management system. At the device level, the APIC interfaces directly with the main system bus and other I/O devices to transfer CORBA requests between endsystem buffer pools and ATM virtual circuits without incurring additional data copying. The buffer pools for I/O devices support "direct demultiplexing" of periodic and aperiodic CORBA requests into memory shared among user- and kernel-resident threads.

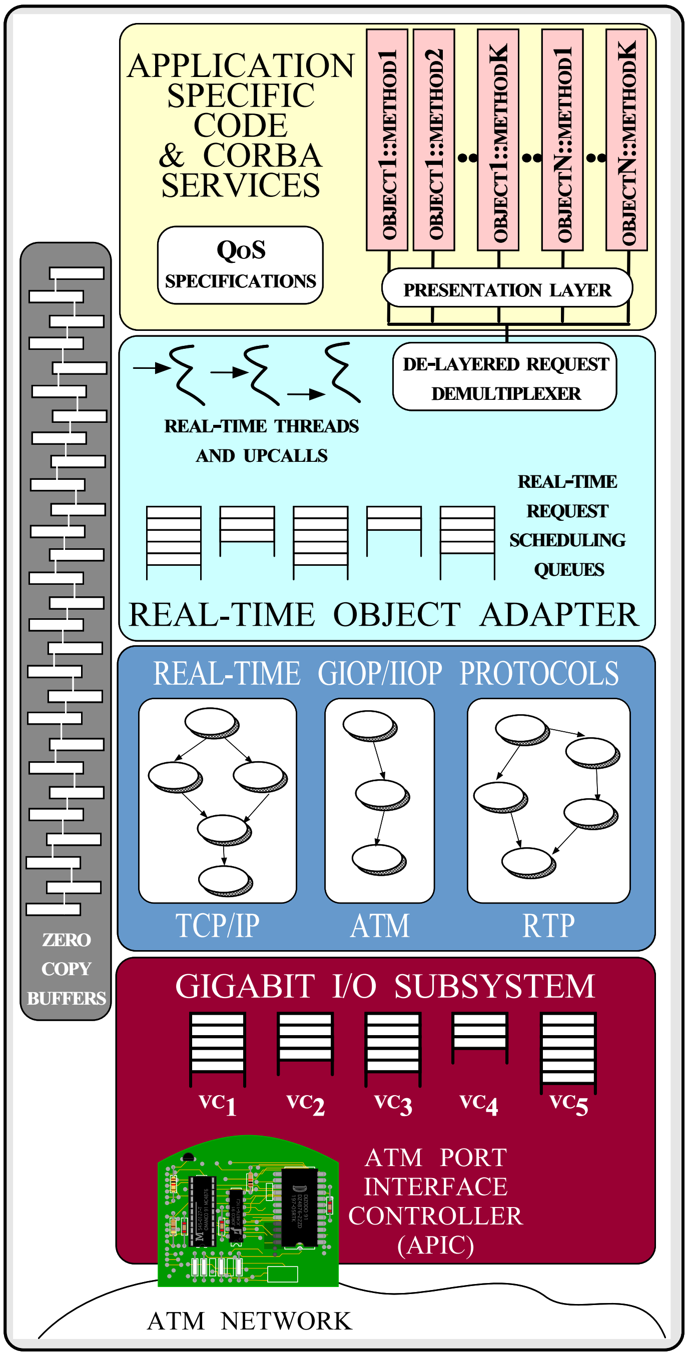

Figure 5: A High-performance ORB Endsystem for Real-time CORBA

4.2 Real-time GIOP/IIOP Protocols

The CORBA General Inter-ORB Protocol (GIOP) [1] specifies the request format and transmission protocol that enables ORB-to-ORB interoperability. Conventional CORBA implementations utilize inflexible, static strategies for selecting the GIOP implementation. For instance, ORBs commonly implement the GIOP using the Internet Inter-ORB Protocol (IIOP), which is layered above TCP/IP.

Performance-sensitive real-time applications often cannot tolerate the latency overhead and jitter of TCP, which supports functionality (such as adaptive retransmissions, deferred transmissions, and delayed acknowledgments) that can cause excessive overhead and latency for real-time applications. Likewise, unreliable protocols like UDP lack functionality (such as congestion control, end-to-end flow control, and rate control), which leads to excessive congestion and missed deadlines in networks and endsystems.

To enhance flexibility and performance, therefore, TAO supports the run-time configuration of lightweight GIOP implementations. These protocols optimize the CORBA GIOP for high-speed networks (*e.g.,* ATM LANs and ATM/IP WANs) [16] and can be customized for specific application requirements. For instance, certain applications that do not require complete reliability (*e.g.,* teleconferencing or certain types of imaging). In this case, TAO can selectively omit transport layer functionality (such as retransmissions and end-to-end error handling) and run directly atop ATM or ATM/IP. Our GIOP transport layer tightly integrates the underlying ATM/IP infrastructure via techniques such as ALF/ILP [17], our Gigabit I/O subsystem [18, 19, 20, 21], and the APIC [15] network adapter.

4.3 Real-Time Object Adapter

The Object Adapter is the component in CORBA that associates object implementations with the ORB and delivers requests to the appropriate target object. In addition to its standard duties, the Object Adapter in TAO is responsible for real-time scheduling and dispatching of the following ORB operations:

- **Real-time upcalls (RTU):** Support for periodic data transfer and protocol processing is critical for multimedia and other delay-sensitive applications. To support periodic delivery of CORBA requests, we have implemented a real-time upcall (RTU) mechanism [7]. RTUs are an OS scheduling mechanism that provides rate monotonic QoS guarantees to protocols and CORBA applications. When used in conjunction with the APIC's zero-copy buffering and direct demultiplexing, RTUs can significantly reduce synchronization and context switching overhead for isochronous multimedia applications.

• **De-layered demultiplexing optimizations:** A standard GIOP-compliant CORBA request contains the identity of its remote object implementation and remote operation. The remote object implementation is represented by an object reference and the remote operation is represented as a string. Conventional ORB endsystems demultiplex CORBA requests to the appropriate method of the object implementation using the following steps (shown at the top of Figure 6):

- *Steps 1 and 2* – The OS protocol stack demultiplexes the incoming CORBA request multiple times (*e.g.,* through the data link, network, and transport layers) to the ORB's Object Adapter;
- *Steps 3 and 4* – The Object Adapter uses the addressing information in the request to locate the appropriate target object implementation and associated IDL skeleton;
- *Step 5* – The IDL skeleton locates the appropriate method, demarshals the request buffer into method parameters, and performs the method upcall.

Demultiplexing requests through all these layers is expensive, particularly when a large number of operations appear in an IDL interface and/or a large number of objects are managed by an ORB. To minimize this overhead, TAO utilizes de-layered demultiplexing [22] (shown at the bottom of Figure 6). This approach uses pre-negotiated demultiplexing keys to map CORBA requests directly to object/method tuples that perform application-level real-time upcalls. To further reduce the number of demultiplexing layers, the APIC can be programmed to directly dispatch CORBA requests associated with ATM virtual circuits. This strategy reduces demultiplexing latency and supports end-to-end QoS on a per-request or per-object basis.

4.4 Application-specific Code and CORBA Services

Certain ORB endsystem features and optimizations depend largely on application characteristics. Chief among these are QoS parameter specifications and presentation layer conversions. TAO uses this application-specific information to guide the selection and/or generation of the following policies and mechanisms to deliver end-to-end QoS to applications:

• **Specifying and mapping QoS requirements:** Specification of QoS requirements is essential to provide performance guarantees. Since applications can have widely varying requirements, a structured and general way to specify QoS is necessary. TAO identifies four application service classes that encompass continuous media, bulk data, low-latency transaction message, and high-bandwidth message streams [7]. These four classes of QoS specifications are defined using high-level parameters; other low-level parameters are derived automatically.

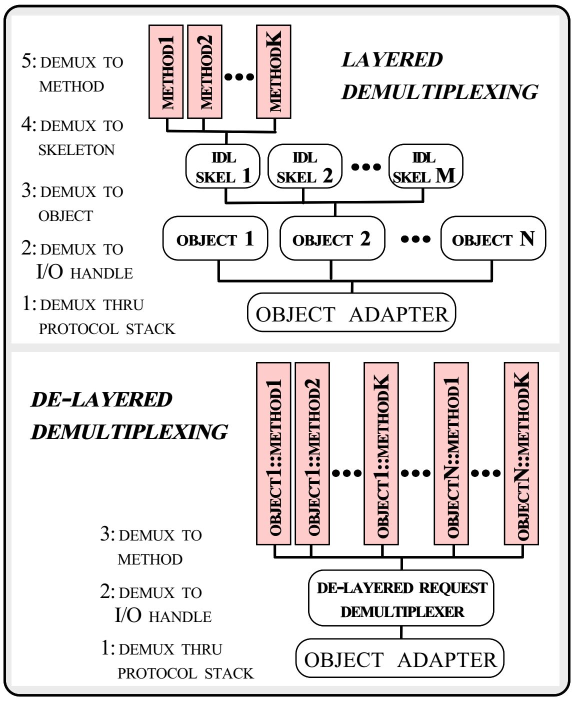

Figure 6: Layered and De-layered CORBA Request Demultiplexing

Several ORB endsystem resources (such as CPU, memory, network connections, and storage devices) are involved in satisfying application QoS requirements. Applications must specify their QoS needs so that the ORB subsystem can guarantee resource availability. For instance, network connection bandwidth must be determined from the specified QoS parameters when establishing connections. Likewise, the amount of computing required to process CORBA requests must be determined so CPU capacity can be allocated accordingly.

In distributed object systems, real-time applications may need to specify QoS parameters on a per-request or per-object basis. For instance, an event service for real-time avionics may choose to propagate navigation events with a higher priority (lower latency) than sensor events. In contrast, a video-on-demand server may only need to specify QoS parameters per-object.

• **Presentation layer optimizations:** TAO can generate and configure multiple strategies for marshaling and demarshaling CORBA IDL types. For instance, based on measures of a type's run-time usage, the TAO can link in either compiled and/or interpreted CORBA IDL stubs and skeletons. This flexibility can achieve an optimal tradeoff between interpreted code (which is slow, but compact in size) and compiled code (which is fast, but larger in size) [13].

Likewise, TAO can cache premarshaled application data units (ADUs) that are used repeatedly. Caching improves performance when ADUs are transferred sequentially in "request chains" and each ADU varies only slightly from one transmission to the other. In such cases, it is not necessary to marshal the entire ADU every time. This optimization requires TAO to perform flow analysis [23, 24] of application code to determine what request fields can be cached.

Although these techniques can significantly reduce marshaling overhead for the common case, real-time applications with static scheduling policies often consider only worst-case execution. As a result, the flow analysis optimizations described above can only be employed under certain circumstances, *e.g.,* for applications that can accept statistical QoS guarantees or when the worst case scenarios are still sufficient to meet deadlines.

5 Concluding Remarks

Currently, there is significant interest in developing high-performance implementations of real-time ORB endsystems. However, meeting these needs requires much more than defining ORB QoS interfaces using CORBA IDL – it requires an integrated architecture that delivers end-to-end QoS guarantees at multiple levels of the entire system. The architecture we describe in this article addresses this need with the following policies and mechanisms spanning network adapters, operating systems, communication protocols, and CORBA middleware:

- Real-time scheduling of OS and network resources;
- A high-performance ATM Port Interface Controller (APIC);
- Efficient zero-copy buffer management that shares CORBA request buffers across OS protection domains;
- Customized real-time implementations of GIOP-compliant communication protocols;
- Real-time Object Adapter implementation with real-time upcalls (RTUs) and de-layered demultiplexing using kernel-level packet filters;
- Lightweight presentation layer based on compiler analysis and efficient buffer management schemes;
- Application interface for specifying end-to-end QoS and mapping this onto individual CORBA requests and/or sessions.

TAO's architecture is designed to offer deterministic and statistical guarantees to real-time applications. More information on real-time CORBA is available through the OMG Realtime Special Interest Group (http://www.omg.org/sigs.htm), which provides a forum for defining CORBA standards applicable to isochronous and real-time domains, and from our CORBA WWW page (http://www.cs.wustl.edu/∼schmidt/corba.html).

Acknowledgments

We like to thank IONA and Visigenic for their help in supplying the CORBA implementations used for the tests in Section 3. Both companies are currently working to eliminate their latency and throughput overheads. We expect their forthcoming releases to perform much better over high-speed ATM networks.

References

[1] Object Management Group, *The Common Object Request Broker: Architecture and Specification*, 2.0 ed., July 1995.

[2] I. Pyarali, T. H. Harrison, and D. C. Schmidt, "Design and Performance of an Object-Oriented Framework for High-Performance Electronic Medical Imaging," *USENIX Computing Systems*, vol. 9, November/December 1996.

[3] A. Gokhale and D. C. Schmidt, "Measuring the Performance of Communication Middleware on High-Speed Networks," in *Proceedings of SIGCOMM '96*, (Stanford, CA), pp. 306–317, ACM, August 1996.

[4] A. Gokhale and D. C. Schmidt, "The Performance of the CORBA Dynamic Invocation Interface and Dynamic Skeleton Interface over High-Speed ATM Networks," in *Proceedings of GLOBECOM '96*, (London, England), pp. 50–56, IEEE, November 1996.

[5] A. Gokhale and D. C. Schmidt, "Evaluating Latency and Scalability of CORBA Over High-Speed ATM Networks," in *Proceedings of the International Confernce on Distributed Computing Systems*, (Baltimore, Maryland), IEEE, May 1997.

[6] S. Vinoski, "CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments," *IEEE Communications Magazine*, vol. 14, February 1997.

[7] R. Gopalakrishnan and G. Parulkar, "Bringing Real-time Scheduling Theory and Practice Closer for Multimedia Computing," in *SIGMETRICS Conference*, (Philadelphia, PA), ACM, May 1996.

[8] S. Khanna and et. al., "Realtime Scheduling in SunOS5.0," in *Proceedings of the USENIX Winter Conference*, pp. 375–390, USENIX Association, 1992.

[9] S. McCanne and V. Jacobson, "The BSD Packet Filter: A New Architecture for User-level Packet Capture," in *Proceedings of the Winter USENIX Conference*, (San Diego, CA), pp. 259–270, Jan. 1993.

[10] M. Abbott and L. Peterson, "Increasing Network Throughput by Integrating Protocol Layers," *ACM Transactions on Networking*, vol. 1, October 1993.

[11] D. R. Engler and M. F. Kaashoek, "DPF: Fast, Flexible Message Demultiplexing using Dynamic Code Generation," in *Proceedings of ACM SIGCOMM '96 Conference in Computer Communication Review*, (Stanford University, California, USA), pp. 53–59, ACM Press, August 1996.

[12] P. Druschel, M. B. Abbott, M. Pagels, and L. L. Peterson, "Network subsystem design," *IEEE Network (Special Issue on End-System Support for High Speed Networks)*, vol. 7, July 1993.

[13] P. Hoschka, "Automating Performance Optimization by Heuristic Analysis of a Formal Specification," in *Proceedings of Joint Conference for Formal Description Techniques (FORTE) and Protocol Specification, Testing and Verification (PSTV)*, (Kaiserslautern), 1996. To be published.

[14] 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.

[15] Z. D. Dittia, J. Jerome R. Cox, and G. M. Parulkar, "Design of the APIC: A High Performance ATM Host-Network Interface Chip," in *IEEE INFOCOM '95*, (Boston, USA), pp. 179–187, IEEE Computer Society Press, April 1995.

[16] G. Parulkar, D. C. Schmidt, and J. S. Turner, "$a^{I}t^{P}m$: a Strategy for Integrating IP with ATM," in *Proceedings of the Symposium on Communications Architectures and Protocols (SIGCOMM)*, ACM, September 1995.

[17] D. D. Clark and D. L. Tennenhouse, "Architectural Considerations for a New Generation of Protocols," in *Proceedings of the Symposium on Communications Architectures and Protocols (SIGCOMM)*, (Philadelphia, PA), pp. 200–208, ACM, Sept. 1990.

[18] C. Cranor, "OS Buffer Management System for Continuous Media I/O," tech. rep., Washington University Department of Computer Science, July 1995.

[19] C. Cranor and G. Parulkar, "Design of Universal Continuous Media I/O," in *Proceedings of the 5th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '95)*, (Durham, New Hampshire), pp. 83–86, Apr. 1995.

[20] R. Gopalakrishnan and G. M. Parulkar, "Efficient User Space Protocol Implementations with QoS Guarantees using Real-time Upcalls," Tech. Rep. 96-11, Washington University Department of Computer Science, March 1996.

[21] R. Gopalakrishnan and G. M. Parulkar, "Real-time Upcalls: A Mechanism to Provide Real-time Processing guarantees," Tech. Rep. 95-06, Dept. of Computer Science, Washington University in St. Louis, 1995.

[22] D. L. Tennenhouse, "Layered Multiplexing Considered Harmful," in *Proceedings of the* 1^{st} *International Workshop on High-Speed Networks*, May 1989.

[23] J.-D. Choi, R. Cytron, and J. Ferrante, "Automatic Construction of Sparse Data Flow Evaluation Graphs," in *Conference Record of the Eighteenth Annual ACE Symposium on Principles of Programming Languages*, ACM, January 1991.

[24] R. Cytron, J. Ferrante, B. K. Rosen, M. N. Wegman, and F. K. Zadeck, "Efficiently Computing Static Single Assignment Form and the Control Dependence Graph," in *ACM Transactions on Programming Languages and Systems*, ACM, October 1991.