Operating System Support for High-Performance, Real-time CORBA

Aniruddha Gokhale, Douglas C. Schmidt, Tim Harrison and Guru Parulkar
Department of Computer Science,
Washington University
St. Louis, MO 63130, USA.
Phone: (314) 935-6160 Fax: (314) 935-7302
E-mail: {gokhale, schmidt, harrison, guru}@cs.wustl.edu

Abstract

A broad range of applications (such as avionics, telecommunication systems, and multimedia on demand) require various types of real-time guarantees from the underlying middleware, operating systems, and networks to achieve their quality of service (QoS). In addition to providing real-time guarantees and end-to-end QoS, the underlying services used by these applications must be reliable, flexible, and reusable. Requirements for reliability, 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 suitable for latency-sensitive real-time applications, including both hard real-time systems (e.g., avionics), and constrained latency systems (e.g., teleconferencing).

This paper describes key changes that must be made to the CORBA specifications, existing CORBA implementations, and the underlying operating system to develop real-time ORBs (RT ORBs). RT ORBs must deliver real-time guarantees and end-to-end QoS to latency-sensitive applications. While many operating systems now support real-time scheduling, they do not provide integrated solutions. The main thesis of this paper is that advances in real-time distributed object computing can be achieved only by simultaneously integrating techniques and tools that simplify application development; optimize application, I/O subsystem, and network performance; and systematically measure performance to pinpoint and alleviate bottlenecks.

1. Introduction

An emerging class of distributed applications require real-time guarantees. These applications include telecommunication systems (*e.g.,* call processing), avionics control systems (*e.g.,* mission control for fighter aircraft), and multimedia applications (*e.g.,* video-on-demand and teleconferencing). In addition to requiring real-time guarantees, these applications must be reliable, flexible, and reusable.

The Common Object Request Broker Architecture (CORBA) is an emerging distributed object computing infrastructure being standardized by the Object Management Group (OMG) [17]. CORBA is designed to support the production of flexible and reusable distributed services and applications. Many implementations of CORBA are now widely available. However, these implementations incur significant overhead that makes them unsuitable for latency-sensitive real-time applications. Key sources of overhead include excessive data copying, inefficient presentation layer conversions, inappropriate internal buffering mechanisms, unoptimized demultiplexing strategies, and many levels of function calls.

Our previous studies measuring the throughput and latency performance of CORBA [8, 9, 10, 20] precisely pinpoint many sources of overhead in existing CORBA implementations. Our results strongly suggest that the only way to ensure end-to-end real-time QoS guarantees for CORBA applications is to integrate the network, transport protocols, operating system, and middleware.

This paper describes an integrated architecture that combines networks, transport protocols, operating systems, and CORBA middleware. To develop this architecture, we propose the changes to operating systems, transport protocols, and current CORBA specifications and implementations required to provide real-time end-to-end QoS guarantees to applications. The real-time guarantees comprise both *hard real-time* applications (where guaranteeing the required QoS is crucial *e.g.,* avionics control), as well as *latency constrained* applications (where certain scheduling and error tolerances are allowed *e.g.,* teleconference and video-on-demand).

The paper is organized as follows: Section 2 outlines the key CORBA middleware and operating system components,

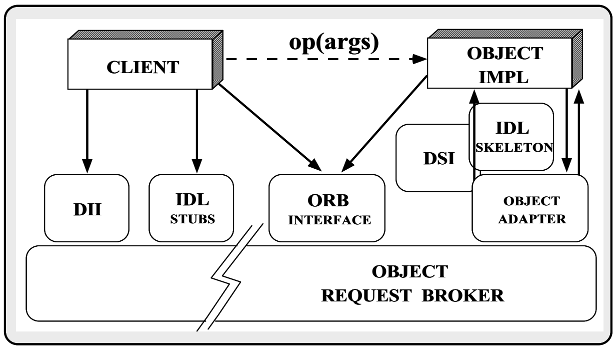

Figure 1. Components in the CORBA Model

policies, and mechanisms required to provide real-time end-to-end QoS guarantees for distributed applications; Section 3 briefly describes results of our previous studies indicating various overheads in existing CORBA implementations [8, 9, 10]; Section 4 describes the changes required in the operating systems, transport protocols and CORBA implementations to support real-time CORBA; and Section 5 presents concluding remarks.

2. Applying the CORBA Model for Real-time Applications

This section outlines the primary components that constitute a standard CORBA 2.0 Object Request Broker (ORB). In addition, we describe the policies and mechanisms that must be developed to achieve real-time CORBA implementations.

2.1. Components in the CORBA Model

Figure 1 illustrates the primary components in the CORBA architecture. The responsibility of each component in CORBA is described below:

• **Object Implementation:** This defines operations that implement a CORBA IDL interface. Object implementations can be written in a variety of languages including C, C++, Java, Smalltalk, and Ada.

• **Client:** This is the program entity that invokes an operation on an object implementation. Accessing the services of a remote object should be transparent to the caller. Ideally, it should be as simple as calling a method on an object, *i.e.,* `obj− >op(args)`. The remaining components in Figure 1 help to support this level of transparency.

• **Object Request Broker (ORB):** When a client invokes an operation, the ORB is responsible for finding the object implementation, transparently activating it if necessary, delivering the request to the object, and returning any response to the caller.

• **ORB Interface:** An ORB is a logical entity that may be implemented in various ways (such as one or more processes or a set of libraries). To decouple applications from implementation details, the CORBA specification defines an abstract interface for an ORB. This interface provides various helper functions such as converting object references to strings and vice versa, and creating argument lists for requests made through the dynamic invocation interface described below.

• **CORBA IDL stubs and skeletons:** CORBA IDL stubs and skeletons serve as the "glue" between the client and server applications, respectively, and the ORB. The transformation between CORBA IDL definitions and the target programming language is automated by a CORBA IDL compiler. The use of a compiler reduces the potential for inconsistencies between client stubs and server skeletons and increases opportunities for automated compiler optimizations.

• **Dynamic Invocation Interface (DII):** This interface allows a client to directly access the underlying request mechanisms provided by an ORB. Applications use the DII to dynamically issue requests to objects without requiring IDL interface-specific stubs to be linked in. Unlike IDL stubs (which only allow RPC-style requests), the DII also allows clients to make non-blocking *deferred synchronous* (separate send and receive operations) and *oneway* (send-only) calls.

• **Dynamic Skeleton Interface (DSI):** This is the server side's analogue to the client side's DII. The DSI allows an ORB to deliver requests to an object implementation that does not have compile-time knowledge of the type of the object it is implementing. The client making the request has no idea whether the implementation is using the type-specific IDL skeletons or is using the dynamic skeletons.

• **Object Adapter:** This assists the ORB with delivering requests to the object and with activating the object. More importantly, an object adapter associates object implementations with the ORB. Object adapters can be specialized to provide support for certain object implementation styles (such as OODB object adapters for persistence and library object adapters for non-remote objects).

• **Higher-level Object Services (not shown):** These services include the CORBA Object Services [16] such as the Name service, Event service, Object Lifecycle service, and the Trader service. There is currently no explicit support for real-time guarantees in the CORBA 2.0 specification, although there is a domain-specific Task Force in the OMG that is focusing on specifying real-time CORBA.

2.2. Issues for High Performance, Real-Time CORBA

The following Section discusses challenges that must be addressed to develop high performance CORBA implementations that provide real-time end-to-end QoS guarantees to applications.

• **Tradeoffs between high-performance and real-time predictability:** A key theme underlying this section is the fact that requirements for high performance often conflict with requirements for real-time predictability. In particular, real-time scheduling policies often rely on the predictability of system operations like scheduling, demultiplexing, and message buffering. However, certain optimizations (such as "one-back caching" for request demultiplexing) can increase performance while decreasing operation predictability. In some cases, bounding the worst case operation time is sufficient to guarantee real-time requirements.

• **Real-Time OS and network scheduling:** Without operating system and network layer support for predictable I/O operations, RT ORBs cannot provide real-time guarantees to applications. Therefore, the underlying operating system and network must provide resource scheduling mechanisms [22] that provide real-time guarantees to CORBA middleware and applications. For instance, the operating system must deliver scheduling mechanisms that allow high priority tasks to run to completion. Furthermore, real-time tasks should be given precedence at the network level to prevent them from being blocked by lower priority applications [11].

• **Light-weight transport mechanisms:** Current reliable transport protocols (such as TCP) are relatively heavy-weight in that they support functionality (such as adaptive retransmissions and delayed acknowledgments) that yields excessive overhead and latency for real-time applications. Likewise, unreliable transport protocols (such as UDP) lack certain functions such as congestion control, end-to-end flow control, and rate control, which cause excessive congestion and missed deadlines in networks and endsystems. Furthermore, different applications have different QoS requirements, so multiple transport mechanisms may be necessary. One solution is to have the operating system provide a set of lightweight real-time implementations of transport protocols [21], which can be customized for specific application requirements and network/host environments.

• **Efficient and predictable demultiplexing:** Incoming CORBA requests must be demultiplexed to the appropriate method of the target object implementation. In contemporary CORBA implementations, demultiplexing occurs at multiple levels, with no ability to schedule or prioritize demultiplexing behavior. For instance, operating systems demultiplex incoming TCP/IP packets multiple times to the appropriate network and transport layer protocols. Then, CORBA Object Adapters demultiplex the packet to an appropriate target object and IDL skeleton, which finally demultiplexes the request to the appropriate method of the target object implementation.

Experience [10, 23] has shown that layered demultiplexing can be inappropriate for latency-sensitive applications. Hence, the operating system must provide mechanisms (such as a packet filters [15, 7] or delayered protocol stacks [1]) to perform CORBA request demultiplexing with minimal overhead. Moreover, request demultiplexing mechanisms must provide consistent QoS performance regardless of the number of protocols, application-level target object implementations, and operations defined by the IDL interfaces of these objects. Optimized demultiplexing paths can increase ORB performance and predictability of demultiplexing algorithms can enable real-time guarantees.

• **Reduced data copying:** The operating system device drivers, protocol stacks, and CORBA middleware must collaborate to provide efficient buffer management schemes that reduce and/or eliminate data copying. On modern RISC hardware, data copying consumes a significant amount of CPU, memory, and I/O bus resources [6]. For real-time applications, the memory management mechanisms used by the OS and ORB must behave predictably, irrespective of user buffer sizes and endsystem workload.

• **Efficient presentation layer conversions:** Presentation layer conversions transform application-level data units from differences in byte order, alignment, and word length. In addition, conversions are necessary due to different encodings used by the protocols at various layers. 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 marshalling code is efficient, but may require excessive amounts of memory. In contrast, interpreted marshalling code is slower, but more compact. CORBA implementations for performance sensitive systems must be flexible to select optimal choices between (1) using compiled marshalling code for data types that are used heavily and (2) interpreted marshalling routines for data types that are used infrequently. For real-time applications, the ORB must be able to make worst case guarantees for both interpreted and compiled marshalling operations.

3. Overhead in Current CORBA Implementations

[10] describes results of our experiments measuring the latency and scalability of two widely used CORBA implementations - Orbix 2.0 and ORBeline 2.0. In that paper, we illustrate how latency-sensitive CORBA applications that use many target objects are not supported efficiently by

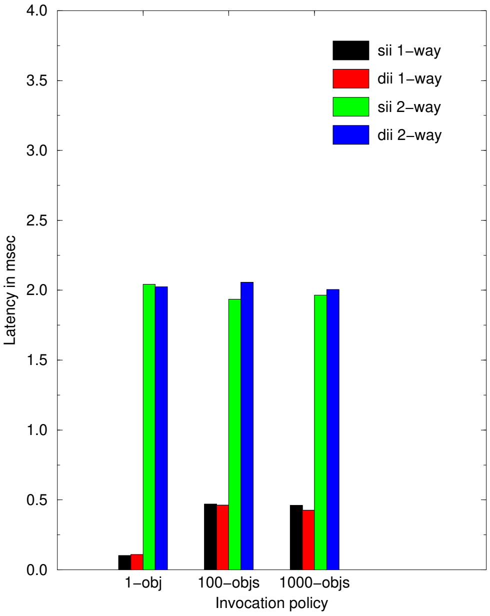

Figure 2. ORBeline: Latency for Sending Parameterless Operation

contemporary CORBA implementations due to (1) inefficient server demultiplexing techniques, (2) improper choice of underlying operating system interfaces, (3) long chains of intra-ORB function calls, (4) excessive presentation layer conversions and data copying, and (5) unpredictable buffering algorithms used for network reads and writes, and (6) general lack of the ability to specify and ensure operation priorities and scheduling.

On low-speed networks, for conventional (*i.e.*, non-real-time) applications, these sources of overhead are often masked. On high-speed networks and for real-time applications, they become a dominant factor limiting end-to-end performance and predictability. If these limitations are not addressed, CORBA will not be adopted for use in performance-sensitive domains.

The following discussion gives a summary of the results from [10] for scalability, latency, and data marshalling tests:

• **Scalability:** The results presented in Figures 2 and 3 illustrate that current implementations of CORBA do not scale well as the number of objects increase by several orders of magnitude. Figure 2 shows that for the ORBeline implementation, the latency of sending parameterless oneway operations increased 4 times as the number of objects went from 1 to 100, and then remained stable as the number of objects increased to 1,000. The latency for the twoway static and dynamic invocation was almost 20 times that of

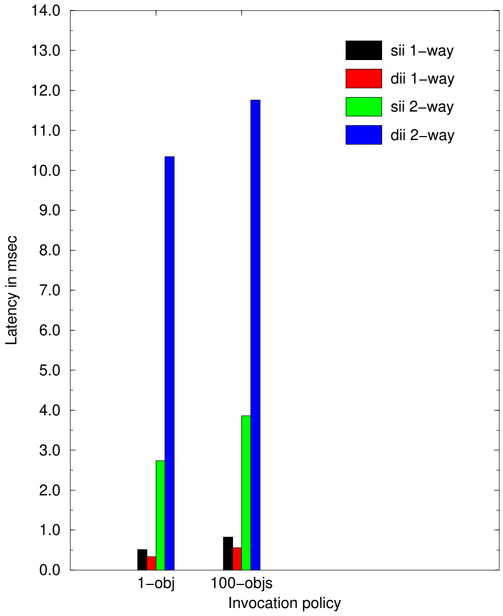

Figure 3. Orbix: Latency for Sending Parameterless Operation

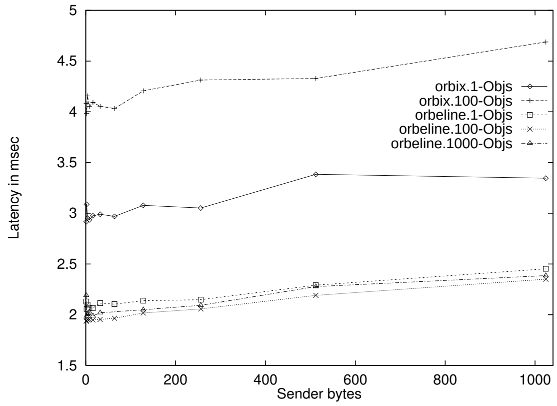

Figure 4. Latency for Sending Octets Using Twoway IDL Stubs

the oneway case for a single object, whereas the latency was 4 to 5 times higher for the 100 and 1,000 object case.

• **Latency:** Figure 3 illustrates the latency for sending parameterless operations using Orbix. The latency of Orbix oneway dynamic invocations was slightly less than that of the static invocation. Latency increased roughly 1.2 to 1.6 times for the oneway and twoway cases as the number of

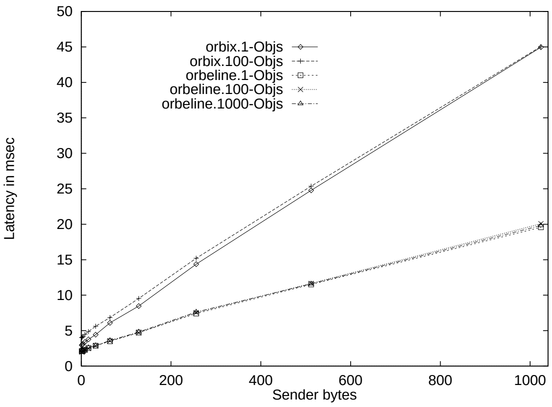

Figure 5. Latency for Sending Structs Using Twoway IDL Stubs

objects increased from 1 to 100. The latency for twoway static invocation was 5 to 6 times that of the oneway case and the latency of twoway dynamic invocation was roughly 30 times that of the oneway case. These results show that for sending untyped data using the static invocation interface, the latency for Orbix was roughly 1.5 to 2 times that of ORBeline.

• **Data Marshalling:** Figures 4 and 5 illustrate for both Orbix and ORBeline the latency for sending `octets` and `structs`, respectively, using the IDL compiler generated stubs. For sending richly-typed data, as the sender buffer size increases, the latency for Orbix increases rapidly compared to that of ORBeline. For smaller sender buffer sizes (1, 2, and 4 bytes), the DII latency of Orbix was roughly 5 to 6 times that of ORBeline. With increasing buffer sizes, the latency increases by roughly 12 to 14 times that of ORBeline. For the SII case, the latency for Orbix increased by roughly 3 to 4 times that of ORBeline for sending richly-typed data.

Note that neither ORB provides interfaces or mechanisms for specifying or delivering end-to-end Quality of Service or real-time guarantees. In section 4 below, we discuss an architecture that addresses the primary sources of overhead discovered in ORBeline and Orbix so the real-time guarantees can be made.

4. An Open Architecture for Real-Time CORBA

This section describes operating system and CORBA middleware policies and mechanisms we are developing to implement a real-time ORB. Figure 6 illustrates the optimizations we are incorporating into the CORBA model shown in Figure 1. These include a *high-performance, real-time I/O system* that replaces conventional operating system I/O subsystems; a set of *flexible and adaptive communication protocols* that can provide real-time guarantees; and a *streamlined CORBA implementation* that optimizes many of the overheads present in current ORBs.

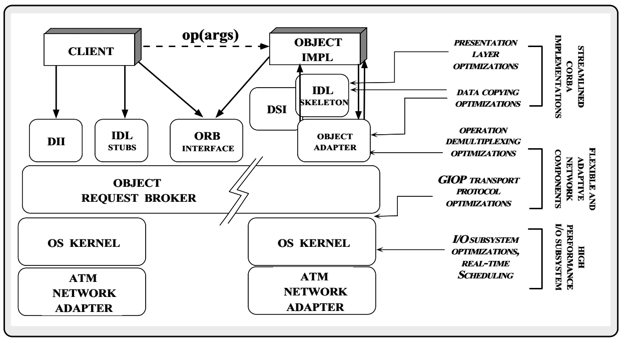

Figure 6. Proposed Optimizations for Real-Time CORBA

4.1. High-Performance, Real-Time I/O System

To ensure end-to-end real-time QoS, we are developing a high-performance network I/O system. This I/O system enhances conventional operating systems (such as SOLARIS with the following components: (1) a Universal Continuous Media (UCM) I/O interface, (2) a zero-copy buffer management system, and (3) a periodic protocol processing and data delivery system using real-time upcalls (RTU). Figure 7 illustrates these components.

The Universal Continuous Media I/O (UCM I/O) combines multiple types of I/O into a single abstraction. This "universal" I/O mechanism is necessary to support new high-speed networks, and high-bandwidth multimedia applications and devices. A detailed design of the UCM I/O and buffer management system for ENOS (Experimental Network Operating System) has been completed [3]. We are now integrating the UCM I/O and buffer management system into our real-time CORBA system. This integration involves (1) interfacing UCM I/O with CORBA and (2) implementing an integrated buffer management system that handles many types of I/O efficiently.

For periodic delivery of CORBA requests, we have created a prototype implementation of real-time upcalls (RTUS). An RTU is an operating system mechanism that provides QoS guarantees to protocols and applications implemented in user-space. Experimental results [12] indicate that it can deliver and process requests with performance that exceeds many real-time thread packages of existing operating systems [14, 24].

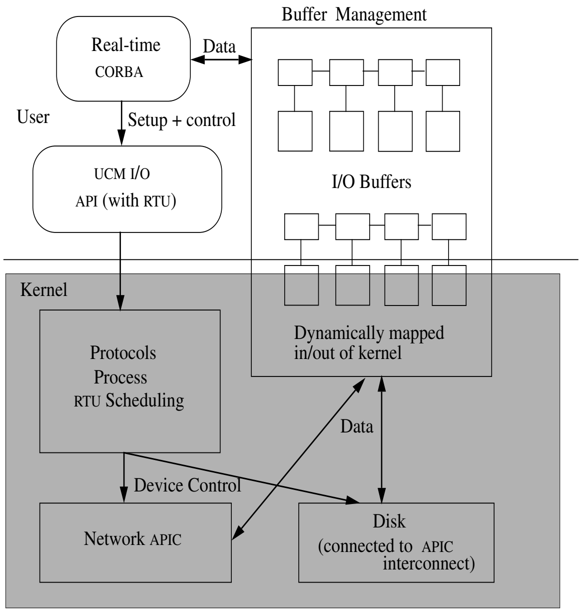

Figure 7. Important High-performance I/O Subsystem Components

4.2. Flexible and Adaptive Real-Time Communication Protocols

Conventional CORBA implementations utilize inflexible, static policies for selecting the transport protocol used to deliver requests and responses. To enhance flexibility, our real-time CORBA system will integrate adaptive communication protocols underneath CORBA to optimize run-time selection of configurations of lightweight General Inter-ORB Protocols (GIOP). The GIOP specification [18] supports transport protocol level interoperability between CORBA implementations.

Conventional ORBs implement the GIOP as a layer above TCP/IP networks (shown in Figure 9(a)). To operate efficiently over high-speed networks that support real-time QoS, real-time CORBA will provide a suite of lightweight transport protocols [21]. These transport protocols will optimize the CORBA GIOP for high-speed networks (*e.g.,* ATM LANs and ATM/IP WANs) and can be customized for specific application requirements (shown in Figure 9(b)).

For instance, when applications do not require complete reliability (*e.g.,* teleconferencing or certain types of imaging), REAL-TIME CORBA will omit transport layer retransmission and error handling to run directly atop ATM or ATM/IP. The GIOP transport layer tightly integrates the underlying ATM/IP infrastructure via techniques such as ALF/ILP [2], our high-performance real-time I/O subsystem [3, 12], and APIC [5].

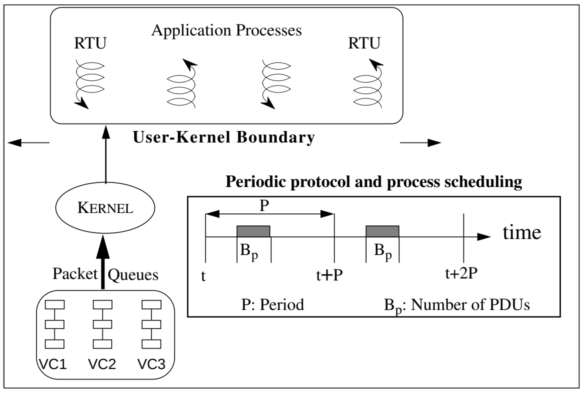

Figure 8. The Real-Time Upcall (RTU) Architecture

4.3. Streamlined Real-Time CORBA Implementation

The recent development of high-speed networks has increased the importance of optimizing memory- and bus-intensive communication software tasks such as demultiplexing remote operations, data movement and presentation layer conversions [19]. Therefore, our real-time CORBA system will optimize the ORB implementation to guarantee real-time responses for the following operations:

• **Remote operation demultiplexing:** A GIOP-compliant CORBA request message contains the identity of its remote object implementation and its intended remote operation. The remote object implementation is typically represented by an object reference and the remote operation is typically represented as a string or binary value. Conventional ORBs use the OBJECT ADAPTER and IDL skeletons to demultiplex request messages to the appropriate method of the object implementation in two steps (shown in Figure 9(c)): (1) the OBJECT ADAPTER uses the object reference in the request to locate the appropriate object implementation and associated IDL skeleton and (2) the IDL skeleton locates the appropriate method and performs an upcall, passing along the demarshalled parameters in the request.

The type of demultiplexing scheme used by an ORB can impact performance significantly. Excessive demultiplexing layers are expensive, particularly when a large number of operations appear in an IDL interface, or a large number of objects exist on a host. To minimize this overhead, real-time CORBA will utilize delayered demultiplexing [23] (shown in Figure 9(d)). The packet filters [15, 7] provided by the operating system kernel will be modified to incorporate CORBA request demultiplexing. Because packet filters are kernel-resident, the demultiplexing process can be optimized, thereby providing low-latency guarantees.

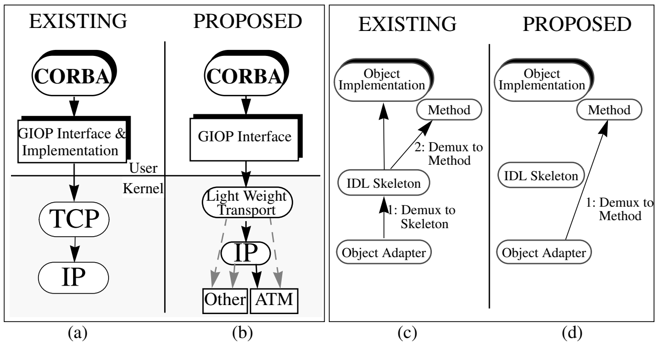

Figure 9. Transport Protocol and Demultiplexing Optimizations in Real-Time CORBA

• **Data movement:** Conventional implementations of CORBA suffer from excessive data copying [8]. For instance, IDL skeletons generated automatically by a CORBA IDL compiler do not generally know how the user-supplied upcall method will use the parameters passed to it from the request message. Therefore, they use conservative memory management techniques that dynamically allocate and release copies of messages before and after an upcall, respectively. These memory management policies are important in some circumstances (*e.g.,* to protect against corrupting internal CORBA buffers when upcalls are made in parallel applications that modify their input). However, this strategy needlessly increases memory and bus overhead for streaming applications (such as satellite surveillance and teleconferencing) that consume their input immediately without modifying it.

Our real-time CORBA system is designed to minimize and eliminate data copying at multiple points. For instance, the buffer management system described in section 4.1 allows CORBA requests to be sent and received to and from the network without incurring any data copying overhead. In addition, Integrated Layer Processing (ILP) [2] can be used to reduce data movement. Since ILP requires maintaining ordering constraints, we are applying compiler techniques (such as control and data flow analysis [4]) to determine where ILP can be employed effectively.

• **Presentation layer:** Our real-time CORBA system will produce and configure multiple marshalling and demarshalling strategies for CORBA IDL definitions, each applicable under different conditions (such as time/space tradeoffs between compiled vs. interpreted CORBA IDL stubs and skeletons). Using dynamic linking, it is possible to include an appropriate marshalling stub for a given data type based on its run time usage by a CORBA application. This can be used to 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]). For example, to avoid the time and space overhead of dynamically linking marshalling stubs for operations that are performed infrequently, real-time CORBA can choose to use interpreted marshalling stubs. Dynamic linking reduces application resource utilization and allows compiled and/or interpreted code to be added or removed at run-time.

Parameter marshalling involves accessing and moving data. Therefore, it is necessary to employ efficient buffer and memory management schemes that minimize overhead. For example, REAL-TIME CORBA will cache certain types of request information. Caching will be employed when certain types of application data units (ADUs) are transferred sequentially in "request chains." In cases where ADUs contain a large number of subparts that remain constant, only a few vary from one transmission to the other. In such cases, it is not necessary to marshal the entire ADU every time. Marshalling overhead can be reduced significantly by having real-time CORBA cache the marshalled information for the constant subparts and only marshal the varying quantities. This type of optimization requires flow analysis [4] of the application code to determine which information can be cached. Our real-time CORBA system will utilize these techniques to achieve efficient marshalling and minimal data copying.

The preceding optimizations are not always suitable for applications with real-time constraints because they optimize for the common case. Although performance in the average will be better, the performance for worst case scenarios may be unacceptable to application real-time requirements. In addition, since static scheduling policies often consider only worst-case execution, resource utilization can be decreased. As a result, these optimizations can only be employed under certain circumstances, *e.g.,* for soft deadlines or when the worst case scenarios are still sufficient to meet hard deadlines.

5. Concluding Remarks

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

- Real-time OS and network scheduling;
- Lightweight presentation layer based on compiler analysis and efficient buffer management schemes;
- Lightweight data copying using efficient zero-copy buffer management schemes and UCM I/O;

- Efficient demultiplexing of CORBA requests using delayered demultiplexing and kernel-level packet filters;
- Customized, light-weight transport protocol implementations.

The integration of these optimizations comprise a high-performance, real-time architecture that we are developing to implement the CORBA standard. This architecture is designed to provide both hard real-time and constrained-latency guarantees to applications.

References

[1] M. Abbott and L. Peterson. Increasing Network Throughput by Integrating Protocol Layers. *ACM Transactions on Networking*, 1(5), October 1993.

[2] 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)*, pages 200–208, Philadelphia, PA, Sept. 1990. ACM.

[3] 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)*, pages 83–86, Durham, New Hampshire, Apr. 1995.

[4] 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.

[5] 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*, pages 179–187, Boston, USA, April 1995. IEEE Computer Society Press.

[6] 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)*, 7(4), July 1993.

[7] 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*, pages 53–59, Stanford University, California, USA, August 1996. ACM Press.

[8] A. Gokhale and D. C. Schmidt. Measuring the Performance of Communication Middleware on High-Speed Networks. In *Proceedings of ACM SIGCOMM '96*, pages 306–317, Stanford, CA, August 1996. ACM.

[9] 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, November 1996. IEEE.

[10] A. Gokhale and D. C. Schmidt. Evaluating Latency and Scalability of CORBA Over High-Speed ATM Networks. In *Submitted to IEEE INFOCOM 1997*, Kobe, Japan, April 1997. IEEE.

[11] R. Gopalakrishnan and G. Parulkar. Quality of Service Support for Protocol Processing Within Endsystems. In W. E. et. al., editor, *High-Speed Networking for Multimedia Applications*. Kluwer Academic Publishers, 1995.

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

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

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

[16] Object Management Group. *CORBAServices: Common Object Services Specification, Revised Edition*, 95-3-31 edition, Mar. 1995.

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

[18] Object Management Group. *Universal Networked Objects*, TC Document 95-3-xx edition, Mar. 1995.

[19] S. W. O'Malley, T. A. Proebsting, and A. B. Montz. USC: A Universal Stub Compiler. In *Proceedings of the Symposium on Communications Architectures and Protocols (SIGCOMM)*, London, UK, Aug. 1994.

[20] I. Pyarali, T. H. Harrison, and D. C. Schmidt. Design and Performance of an Object-Oriented Framework for High-Performance Electronic Medical Imaging. In *Proceedings of the 2^{nd} Conference on Object-Oriented Technologies and Systems*, Toronto, Canada, June 1996. USENIX.

[21] D. C. Schmidt, B. Stiller, T. Suda, A. Tantawy, and M. Zitterbart. Language Support for Flexible, Application-Tailored Protocol Configuration. In *Proceedings of the 18^{th} Conference on Local Computer Networks*, pages 369–378, Minneapolis, Minnesota, Sept. 1993. IEEE.

[22] J. A. Stankovic, M. Spuri, M. D. Natale, and G. Buttazzo. Implications of Classical Scheduling Results for Real-Time Systems. *IEEE Computer*, 28(6):16–25, June 1995.

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

[24] H. Tokuda, T. Nakajima, and P. Rao. Real-Time Mach: Towards Predictable Real-time Systems. In *USENIX Mach Workshop*. USENIX, October 1990.