Middleware Techniques and Optimizations for Real-time, Embedded Systems

Douglas C. Schmidt
{schmidt}@cs.wustl.edu
Computer Engineering Department
University of California, Irvine
Irvine, California, 92697

1 Introduction: Why We Need Middleware for Real-time Embedded Systems

Due to constraints on footprint, performance, and weight/power consumption, real-time, embedded system software development has historically lagged mainstream software development methodologies. As a result, real-time, embedded software systems are costly to evolve and maintain. Moreover, they are often so specialized that they cannot adapt readily to meet new market opportunities or technology innovations.

To further exacerbate matters, a growing class of real-time, embedded systems require end-to-end support for various quality of service (QoS) aspects, such as bandwidth, latency, jitter, and dependability. These applications include telecommunication systems (*e.g.,* call processing and switching), avionics control systems (*e.g.,* operational flight programs for fighter aircraft), and multimedia (*e.g.,* Internet streaming video and wireless PDAs). In addition to requiring support for stringent QoS requirements, these systems are often targeted at highly competitive markets, where deregulation and global competition are motivating the need for increased software productivity and quality.

Requirements for increased software productivity and quality motivate the use of Distributed Object Computing (DOC) *middleware* [1]. Middleware resides between client and server applications and services in complex software systems. The goal of middleware is to integrate reusable software components to decrease the cycle-time and effort required to develop high-quality real-time and embedded applications and services.

Middleware simplifies application development by providing a uniform view of heterogeneous networks, protocols, and OS features. Figure 1 illustrates the various layers of middleware that can reside between (1) the underlying OS and protocol stacks and (2) the applications. These layers include the following capabilities:

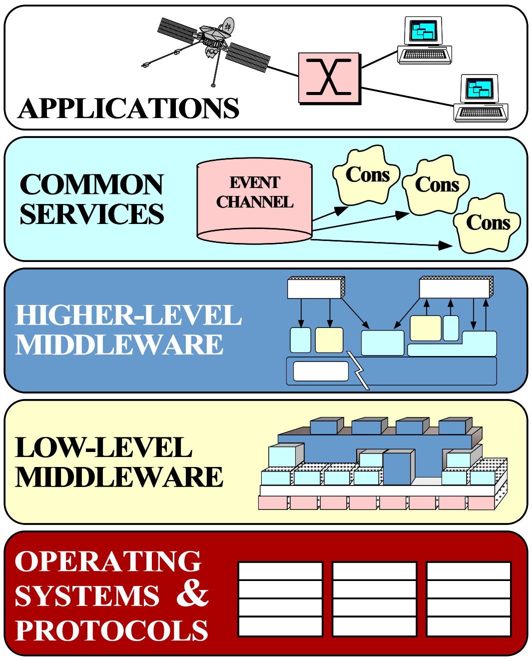

Figure 1: Layers of Middleware

Low-level middleware: which encapsulates core OS communication and concurrency services to eliminate many tedious, error-prone, and non-portable aspects of developing and maintaining distributed applications using low-level network programming mechanisms, such as sockets. Common examples of low-level middleware include the Java Virtual Machine (JVM) [2] and the ADAPTIVE Communication Environment (ACE) [3].

Higher-level middleware: which builds upon the lower-level middleware to automate common network programming tasks, such as parameter marshaling/demarshaling, socket and request demultiplexing, and fault detection/recovery. At the heart of higher-level middleware are Object Request Brokers (ORBs), such as OMG's Common Object Request Broker Architecture (CORBA) [4], Microsoft's Distributed

COM (DCOM) [5], JavaSoft's Remote Method Invocation (RMI) [6].

Common services: which are distributable components that provide domain-independent capabilities that can be reused by many applications. Common examples of services [7] include naming services, transaction services, event services, and security services.

In theory, these middleware layers can significantly simplify the creation, composition, and configuration of communication systems, *without* incurring significant performance overhead. In practice, however, technical challenges have impeded the development and deployment of efficient middleware for real-time, embedded systems, as described next.

2 Historical Limitations of Middleware for Real-time, Embedded Systems

Following in the tradition of Remote Procedure Call (RPC) [8] toolkits, such as Sun RPC [9] and OSF DCE [10], DOC ORBs are well-suited for conventional request/response-style applications running on low-speed networks [11]. Until recently, however, the QoS specification and enforcement features of conventional DOC middleware and ORBs, as well as their efficiency, predictability, and scalability, have not been suitable for applications with hard real-time requirements, *e.g.*, avionics mission computing, and stringent statistical real-time requirements, *e.g.*, teleconferencing. In particular, conventional DOC ORB specifications and implementations have been characterized by the following deficiencies:

Lack of QoS specification and enforcement: Conventional DOC ORBs have not defined APIs that allow applications to specify their end-to-end QoS requirements. Likewise, standard DOC ORB implementations have not provided support for end-to-end QoS enforcement between applications across a network. For instance, CORBA provides no standard way for clients to dynamically schedule requests with various deadlines. Likewise, there are no means for DCOM or RMI clients to inform a server the priority of various operations.

Lack of real-time features: Conventional DOC ORBs have not provided certain key features necessary to support time-constrained real-time programming. For instance, CORBA, DCOM, and RMI do not require an ORB to notify clients when transport layer flow control occurs. Therefore, it is hard to write portable and efficient real-time applications that will not to block when ORB endsystem and network resources are temporarily unavailable. Likewise, conventional DOC ORBs do not propagate exceptions stemming from missed deadlines from servers to clients, which makes it hard to write applications that behave predictably when congestion in the communication infrastructure or end-systems causes deadlines to be missed.

Lack of performance optimizations: Conventional ORBs often incur significant throughput and latency overhead [11]. This overhead stems from excessive data copying, non-optimized presentation layer conversions, internal message buffering strategies that produce non-uniform behavior for different message sizes, inefficient demultiplexing algorithms, long chains of intra-ORB virtual method calls, and lack of integration with underlying real-time OS and network QoS mechanisms [12].

Lack of memory footprint optimizations: Conventional ORBs have historically been developed for desktop applications and general-purpose client/server systems. In such systems, memory is often abundant. Thus, DOC middleware can range in size from 2 to 10 megabytes without undue affect on system performance or cost. In contrast, many real-time and embedded systems have tight constraints on memory footprint due to cost, power consumption, or weight restrictions.

Although some operating systems, networks, and protocols now support real-time scheduling, they do not provide integrated end-to-end solutions for real-time, embedded systems that possess stringent QoS requirements. In particular, QoS research at the IPC and OS layers has not necessarily addressed key requirements and usage characteristics of DOC middleware, such as CORBA, DCOM, or RMI. For instance, research on QoS for communication systems has focused largely on policies for allocating network bandwidth on a per-connection basis. Likewise, research on real-time operating systems has focused largely on avoiding priority inversions and non-determinism in synchronization and scheduling mechanisms for multi-threaded applications. In contrast, the programming model for developers of DOC applications focuses largely on invoking remote operations on distributed objects.

3 Tutorial Topics: Optimizing Middleware for Real-time, Embedded Systems

Determining how to map the results from QoS work at the IPC and OS layers to DOC middleware is currently the focus of many active research projects, such as the DARPA Quorum project [13], the QuO project at BBN [14], and the TAO [15] and TMO [16] projects at Washington University and UC Irvine. One topic of this tutorial, therefore, is to summarize the design techniques and optimization patterns necessary to overcome the historical limitations with middleware described above in order to meet end-to-end QoS requirements for real-time, embedded systems. Figure 2 illustrates the key optimization points for real-time, embedded system middleware.

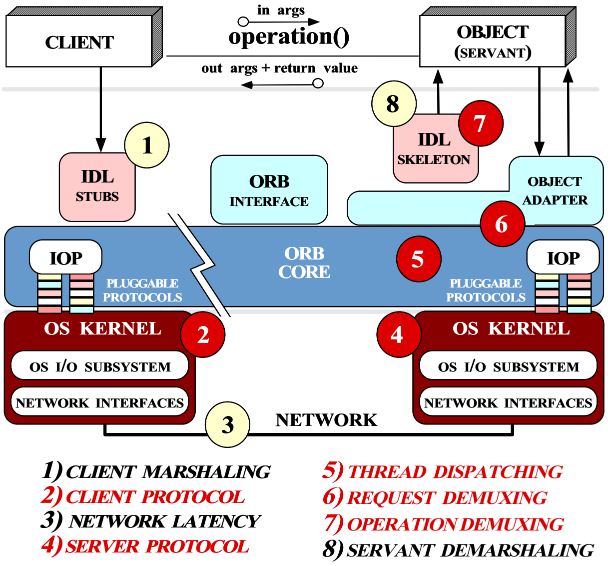

Figure 2: Optimization Points for Real-time, Embedded System Middleware

A decade of intensive research illustrates that meeting the QoS needs applications requires much more than single-point solutions, such as creating new programming languages or building real-time scheduling into ORBs. Instead, it requires a *vertically* (*i.e.*, network interface ↔ application layer) and *horizontally* (*i.e.*, end-to-end) integrated architecture that provides end-to-end QoS support at multiple levels of an entire distributed system. To adequately capture the rich interactions among the various levels, therefore, this tutorial also focuses on the following topics:

- The enchancements required to existing DOC specifications (such as OMG CORBA) that can enable applications to define their Quality of Service (QoS) requirements to ORB endsystems.
- Key architectural patterns required to build real-time ORB endsystems that can enforce deterministic and statistical end-to-end QoS support to applications and services.
- Strategies for integrating I/O subsystem architectures and optimizations with DOC middleware to provide high bandwidth and low latency support to distributed real-time, embedded applications.
- Overview of relevant OMG CORBA standards, such as Real-time CORBA, Minimum CORBA, the Messaging specification, and the Audio/Video Streaming service, that address QoS requirements.

The design techniques and optimization patterns covered in the tutorial are based on TAO [17], which is an open-source, real-time ORB that provides end-to-end QoS support over a wide-range of networks and embedded system interconnects. The architecture of TAO is illustrated in Figure 3.

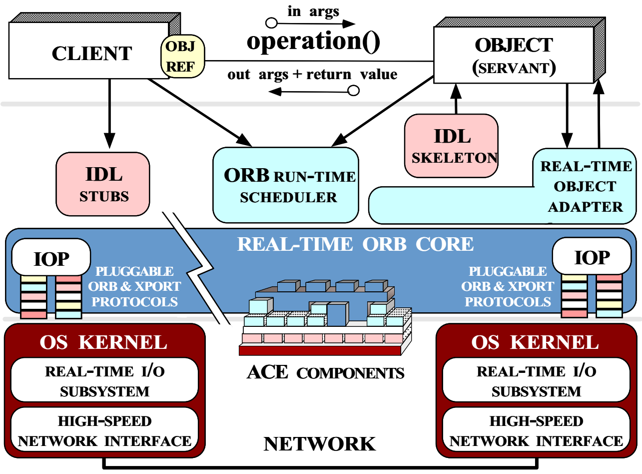

Figure 3: Overview of the TAO Real-time CORBA Architecture

TAO is currently deployed on many projects at many organizations and companies, including Quorum, Boeing, Lockheed, Lucent, Motorola, Nortel, Raytheon, SAIC, and Siemens, where it is used for real-time and embedded avionics, telecommunications, medical, and simulation systems. Complete source code, documentation, and technical papers on TAO are available at `www.cs.wustl.edu/~schmidt/TAO.html`.

4 Overview of Real-time CORBA

A central focus of the tutorial is on the Real-time CORBA standard [18]. Figure 4 illustrates the key features in Real-time CORBA that will be covered in this tutorial. These features include the following:

End-to-end priority propagation: Conventional CORBA ORBs provide no standard way for clients to indicate the relative priorities of their requests to an ORB. Conversely, in Real-time CORBA, the priority at which a client invokes an operation can propagate with the request from sender to receiver. This feature helps minimize end-to-end priority inversion, which is important to bound latency for embedded systems with hard real-time requirements.

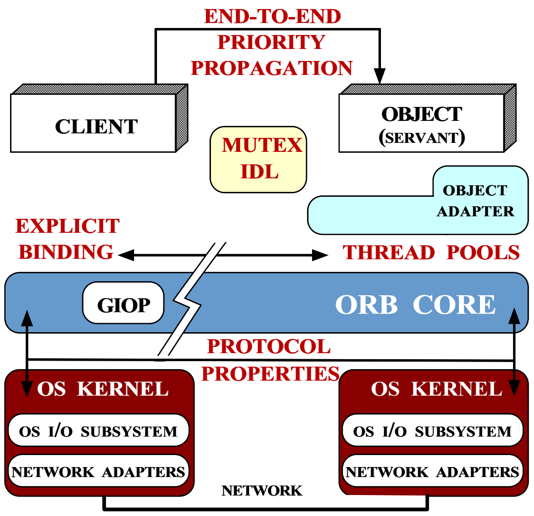

Figure 4: Components and Features in Real-time CORBA

Protocol properties: DOC middleware has traditionally treated the underlying network and bus protocols as a "black box." While this may be sufficient for applications with "best-effort" QoS requirements, it's inadequate for applications with deterministic and/or statistical QoS requirements. Therefore, the Real-time CORBA specification defines a standard set of interfaces that allow applications to select and configure properties of the underlying communication protocols.

Thread pools: Prior to the Real-time CORBA specification, there was no standard way to write multi-threaded CORBA servers. However, many embedded systems, particularly real-time systems that are rate monotonically scheduled, use multi-threading to (1) distinguish various classes of service and (2) support thread preemption to prevent unbounded priority inversion. Therefore, the Real-time CORBA specification defines a standard thread pool model that allows server to pre-allocate threads and set thread attributes, such as stacksize and default priority levels.

Explicit binding: The original CORBA specification only supported *implicit binding*, where the resources and "path" between a client and its server object were established implicitly after the first invocation on the server. This binding model is inadequate for many real-time applications, however, because it defers object/server activation and resource allocation until run-time, thereby increasing latency and jitter. Likewise, implicit bindings often *multiplex* multiple client threads in a process through a shared network connection to the corresponding server process, which can yield substantial priority inversion [19]. To avoid these problems, the Real-time CORBA specification defines an explicit binding mechanism that enables clients to *pre-establish* non-multiplexed connections to client, as well as to map the priority of a client request to the appropriate non-multiplexed connection.

Mutex IDL: Since earlier versions of CORBA did not define a threading model, there was also no standard, portable way to ensure consistency between the internal synchronization mechanisms used by an ORB and an application. For real-time applications, however, it's necessary to ensure consistency between synchronization mechanisms to enforce priority inheritance and priority ceiling protocols [20]. Therefore, the Real-time CORBA specification defines a set of locality constrained mutex operations that ensure consistency between synchronizers used by the ORB and its applications.

5 Concluding Remarks

DOC middleware is a promising paradigm for decreasing the cost and improving the quality of real-time embedded software systems by increasing the flexibility and modularity of reusable components and services. Meeting the QoS requirements of real-time, embedded systems requires more than object-oriented design and programming techniques, however. It requires an integrated architecture that delivers end-to-end QoS support at multiple levels in real-time and embedded systems. The design techniques and optimization patterns described in this tutorial address this need with policies and mechanisms that span network adapters, operating systems, communication protocols, ORB middleware, and common application services.

The future of middleware for real-time and embedded systems is very promising. Based on current trends in many domains, real-time system development strategies will continue to migrate towards those used for "mainstream" systems, thereby achieving lower development cost and faster time-to-market. In particular, the flexibility and adaptability offered by Real-time CORBA makes it very attractive for use in real-time systems. An increasing number of highly optimized, interoperable, and standards-compliant implementations of Real-time CORBA are now available. Over the next several years, fierce competition between ORB suppliers will drive real-time, embedded middleware to become a commodity, much in the same way that CPUs, operating systems, and protocols have become a commodity.

6 About the Presenter

Dr. Schmidt is an Associate Professor of Computer Engineering at the University of California, Irvine. Previously, he was an Associate Professor and Director of the Center for Distributed Object Computing in the Department of Computer Science and in the Department of Radiology at Washington University in St. Louis, Missouri, USA. His research focuses on design patterns, implementation, and experimental analysis of object-oriented techniques that facilitate the development of high-performance, real-time distributed object computing systems on parallel processing platforms running over high-speed ATM networks and embedded system interconnects. Dr. Schmidt has published widely in top IEEE, ACM, IFIP, and USENIX technical conferences and journals. His publications cover a range of experimental systems topics including high-performance communication software systems, parallel processing for high-speed networking protocols, real-time distributed object computing with CORBA, and object-oriented design patterns for concurrent and distributed systems.

Dr. Schmidt has served as guest editor for feature topic issues on Distributed Object Computing for the IEEE Communications Magazine and the USENIX Computing Systems Journal, and served as co-guest editor for the Communications of the ACM special issue on Design Patterns and the special issue on Object-Oriented Frameworks. In addition, he has co-editored the first volume of the *Pattern Languages of Program Design* series by Addison-Wesley and the *Object-Oriented Application Frameworks: Applications & Experiences* series by Wiley. Dr. Schmidt has also served as the editor of the C++ Report magazine, as well as the Patterns++ section of C++ Report, where he also co-authors a column on distributed object computing.

Dr. Schmidt served as the general chair of the IFIP/ACM International Conference Middleware 2000, as well as the program chair for the 1996 USENIX Conference on Object-Oriented Technologies and Systems (COOTS) and the 1996 Pattern Languages of Programming conference. He has presented over 150 keynote addresses, invited talks, and tutorials on reusable design patterns, concurrent object-oriented network programming, and distributed object systems at many conferences including OOPSLA, USENIX COOTS, ECOOP, IEEE LCN, ACM PODC, IEEE ICNP, and IEEE GLOBECOM.

In addition to his academic research, Dr. Schmidt has over a decade of experience building object-oriented communication systems. He is the chief architect and developer of the ADAPTIVE Communication Environment (ACE), which is a widely used, freely-available object-oriented framework that contains a rich set of components that implement design patterns for high-performance and real-time communication systems. Dr. Schmidt has successfully used ACE on large-scale projects at Ericsson, Siemens, Motorola, Kodak, Lucent, Lockheed Martin, Boeing, and SAIC. These projects involve telecommunications systems, medical imaging systems, real-time avionic systems, and distributed interactive simulation systems. Dr. Schmidt and the members of his research group in the Center for Distributed Object Computing are currently using ACE to develop a high performance, real-time CORBA ORB endsystem called TAO (The ACE ORB). TAO is the first real-time ORB endsystem to support end-to-end Quality of Service support over ATM networks.

Dr. Schmidt received B.S. and M.A. degrees in Sociology from the College of William and Mary in Williamsburg, Virginia, and an M.S. and a Ph.D. in Computer Science from the University of California, Irvine (UCI) in 1984, 1986, 1990, and 1994, respectively. His Ph.D. advisor was Tatsuya Suda. Dr. Schmidt is a member of the IEEE, ACM, and USENIX.

References

[1] A. Gokhale and D. C. Schmidt, "Techniques for Optimizing CORBA Middleware for Distributed Embedded Systems," in *Proceedings of INFOCOM '99*, Mar. 1999.

[2] J. Gosling and K. Arnold, *The Java Programming Language*. Reading, MA: Addison-Wesley, 1996.

[3] D. C. Schmidt, "ACE: an Object-Oriented Framework for Developing Distributed Applications," in *Proceedings of the* 6^{th} *USENIX C++ Technical Conference*, (Cambridge, Massachusetts), USENIX Association, April 1994.

[4] Object Management Group, *The Common Object Request Broker: Architecture and Specification*, 2.3 ed., June 1999.

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

[6] A. Wollrath, R. Riggs, and J. Waldo, "A Distributed Object Model for the Java System," *USENIX Computing Systems*, vol. 9, November/December 1996.

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

[8] A. D. Birrell and B. J. Nelson, "Implementing Remote Procedure Calls," *ACM Transactions on Computer Systems*, vol. 2, pp. 39–59, February 1984.

[9] Sun Microsystems, "RPC: Remote Procedure Call Protocol Specification," Tech. Rep. RFC-1057, Sun Microsystems, Inc., June 1988.

[10] W. Rosenberry, D. Kenney, and G. Fischer, *Understanding DCE*. O'Reilly and Associates, Inc., 1992.

[11] 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.

[12] A. Gokhale and D. C. Schmidt, "Optimizing a CORBA IIOP Protocol Engine for Minimal Footprint Multimedia Systems," *Journal on Selected Areas in Communications special issue on Service Enabling Platforms for Networked Multimedia Systems*, Sept. 1999.

[13] DARPA, "The Quorum Project." http://www.ito.darpa.mil/ito/research/quorum/index.html, 1999.

[14] B. Technologies, "Quality Objects (QuO)." http://www.dist-systems.bbn.com/papers.

[15] Center for Distributed Object Computing, "TAO: A High-performance, Real-time Object Request Broker (ORB)." www.cs.wustl.edu/∼schmidt/TAO.html, Washington University.

[16] K. Kim and E. Shokri, "Two corba services enabling tmo network programming," in *Fourth International Workshop on Object-Oriented, Real-Time Dependable Systems*, IEEE, January 1999.

[17] 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.

[18] Object Management Group, *Realtime CORBA Joint Revised Submission*, OMG Document orbos/99-02-12 ed., March 1999.

[19] D. C. Schmidt, S. Mungee, S. Flores-Gaitan, and A. Gokhale, "Software Architectures for Reducing Priority Inversion and Non-determinism in Real-time Object Request Brokers," *Journal of Real-time Systems*, To appear 1999.

[20] R. Rajkumar, L. Sha, and J. P. Lehoczky, "Real-Time Synchronization Protocols for Multiprocessors," in *Proceedings of the Real-Time Systems Symposium*, (Huntsville, Alabama), December 1988.