A survey of techniques and guidelines for improving the performance of CORBA-based distributed systems

Pekka Kähkipuro
University of Helsinki
Department of Computer Science
P.O. Box 26 (Teollisuuskatu 23)
FIN-00014 University of Helsinki, FINLAND
Phone: +358 9 708 44157, fax +358 9 708 44441
E-mail: Pekka.Kahkipuro@cs.Helsinki.FI

Abstract

This paper provides a survey of the available methods for improving the performance of CORBA-based systems. The methods are divided into two groups. The first group contains technical solutions for improving the CORBA architecture and for implementing it more efficiently. The second group consists of application-level guidelines. Some of them attempt to overcome the observed bottlenecks of today's CORBA products, while others try to optimize the usage of system resources. The purpose of the paper is to give an integral view of the existing CORBA-related performance studies, and to provide a starting point for a more comprehensive framework for evaluating and improving the performance of CORBA-based systems.

1 Introduction

When Object Management Group (OMG) introduced the first version of the CORBA architecture in 1991, the focus of the OMG was on the use of object-orientation for overcoming the complexity and heterogeneity of software systems. Today, the CORBA has evolved into a mature technology, and the focus has moved towards higher-level facilities and frameworks that exploit the underlying CORBA technology and facilitate the development of industrial software [Orf97]. Consequently, an increasing number of ready-made software is now based on CORBA technology, and the question of performance is becoming an important issue for the implementers and users of CORBA technology. However, the complexity of the CORBA specification and differences in the CORBA implementations make if difficult to estimate and improve the performance of a given system. Consequently, there is a need for design guidelines and implementation techniques that would allow developers to estimate and improve the performance of CORBA-based systems.

In this paper, we examine current research results that address the performance of CORBA-based systems. The results can be divided into the following two categories:

- technical solutions that identify and repair bottlenecks within the CORBA specification or inside specific CORBA-based middleware implementations, and
- application-level guidelines that help designers to create well-performing systems by avoiding known bottlenecks and by optimizing the usage of system resources.

The former results are related to the advances of CORBA technology, such as the development of real-time CORBA systems, while the latter results are associated with software engineering research. This dichotomy may be confusing for a system designer that is looking for ways to predict the performance of a particular system, or for an application developer that is trying to find cost-effective ways to deal with system saturation.

In this paper, individual performance issues are not treated separately. Instead, we present the performance issues of CORBA-based applications in an integrated way, so that the relationships between the issues are also discussed. In addition, we believe that performance issues should be considered together with other aspects of system design. This perspective also determines the goals of performance engineering: On one hand, it should help to predict the performance of a particular system design and, on the other hand, it should assist in determining the most cost-effective ways for dealing with performance problems [Men94].

The rest of this paper is structured as follows. Section 2 provides an overview of the CORBA environment. Section 3 discusses the performance of the CORBA architecture and the current ORB implementations. Section 4 concentrates on CORBA-based applications. Section 5 concludes the discussion.

2 Overview of the CORBA architecture

CORBA is an object-oriented middleware architecture that supports the development of distributed systems in heterogeneous environments. The architecture consists of the following elements [OMG97a]:

- *The CORBA object model* provides the semantic foundations for the rest of the architecture. The model is based on the traditional object model where objects offer services to their clients through well-defined interfaces.
- *The OMG Interface Definition Language* (IDL) allows system developers to specify object interfaces in a way that is independent of any specific programming language or operating environment.
- *Language mappings* indicate how interfaces specified with OMG IDL are used in a particular programming language environment. A language mapping defines how IDL data types and interfaces are represented, and how operations are invoked and implemented. Version 2.1 of the CORBA specification contains language mappings for C, C++, Smalltalk, Cobol, and Ada.
- *The Object Request Broker* (ORB) implements the run-time environment for CORBA compliant applications. The main purpose of the ORB is to provide mechanisms for clients to access services offered by object implementations.

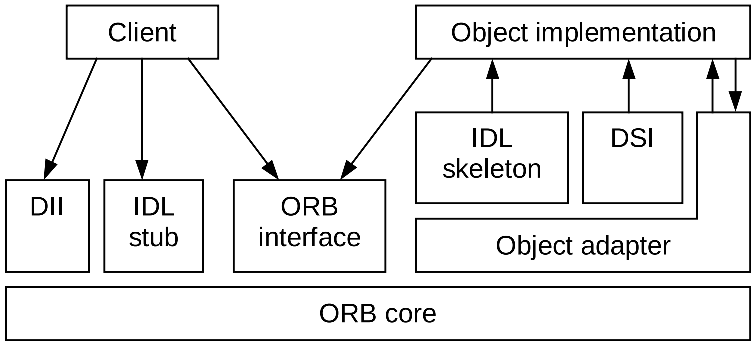

Figure 1. The components of the Object Request Broker.

- *The interoperability architecture* specifies how different ORB products can cooperate. At the heart of the architecture, the GIOP and IIOP protocols provide the means for sending requests and replies between technology boundaries. The GIOP defines the transfer syntax for requests and replies, and the IIOP specifies how GIOP messages are transmitted over TCP/IP. The interoperability architecture also contains a mapping between CORBA and the Component Object Model (COM) from Microsoft.

Figure 1 shows the main components of the Object Request Broker. The ORB core delivers operation requests and replies between clients and object implementations. It also offers general-purpose operations for applications through the ORB interface. The IDL stubs and skeletons hide communication details from the clients and object implementations. They are generated automatically from IDL specifications. The Dynamic Invocation Interface (DII) allows clients to create arbitrary requests at run-time so that they can invoke operations for which there is no IDL stub available. The Dynamic Skeleton Interface (DSI) is the server-side counterpart for the DII, and it allows object implementations to receive arbitrary operation requests for which there is no IDL skeleton. The Object adapter is responsible for activating object implementations and individual objects. Version 2.1 of the CORBA specification describes only one object adapter, the Basic Object Adapter (BOA), but it anticipates the emergence of additional object adapters, such as an adapter for object-oriented databases.

The ORB offers only an elementary functionality for system developers and, therefore, the OMG has produced higher-level specifications on top of the basic CORBA. The *CORBAservices* simplify common programming tasks with ready-made building blocks, such as the naming service and the transaction service [OMG97c], while the CORBAfacilities define more extensive frameworks to be used in CORBA-based applications [Orf97].

3 Performance issues of the CORBA architecture

The current version of the CORBA architecture and the underlying object model impose several restrictions on the implementation of efficient systems. On one hand, the communication model of the CORBA is not sufficient for all application domains and, on the other hand, there is currently no support for managing the quality of service—a vital requirement in performance-sensitive applications and real-time programming.

Insufficient communication model. The current request/reply communication model of CORBA provides only weak support for other important communication models, such as the transmission of multimedia streams, the exchange of large binary objects, and the use of asynchronous message-oriented communication. Consequently, additional specifications and non-standard techniques are used for coping efficiently with such needs. For example, the Multimedia System Services (MSS) architecture from the International Multimedia Association (IMA) extends the CORBA for dealing with multimedia issues that are not supported by the CORBA specification [Bla97]. Similarly, the efficient exchange of large amounts of data through invocation-oriented CORBA interfaces requires special fine-tuning [Lec96] or the direct use of lower-level communication mechanisms (e.g. TCP/IP) [Har96].

The lack of multiparty communication in the current CORBA specification is not a serious problem for traditional client/server applications, but modern systems, such as those using multimedia [Bla97] or those providing fault-tolerance [Maf97], need reliable multicast facilities. Multiparty communication can be implemented at the application level, but such solutions cannot easily use the available low-level broadcast and multicast protocols that may provide significant performance improvements.

Lack of support for QoS and real-time features. Currently, CORBA has no support for the management of quality of service (QoS). This reduces the usability of CORBA in mission-critical applications. For example, clients have no standard way of indicating the relative priority of their operation requests. Also, facilities are needed for specifying the attributes of operation invocations, such as the reliability and the level of transaction support. However, QoS cannot be easily included in the existing CORBA architecture unless the underlying object model is changed [Orf97]. In distributed real-time systems, additional features are needed for controlling the ORB at run-time. For example, a mechanism is needed to cooperate with the transport layer to optimize the flow control under different operating conditions [Sch97]. Such functionality requires even more changes to the current communication semantics of CORBA.

Although the overall architecture of CORBA is an important source of performance problems, the current ORB implementation techniques also need to be addressed. The rest of this section discusses three of them: unpractical data representations and protocols, inefficient stubs and skeletons, and clumsy server implementations.

Unpractical data representation and protocols. Data representations within ORBs are typically implied by language mappings that result from tradeoffs between different requirements. For example, CORBA strings are mapped to the standard C language null-terminated string to achieve simplicity in programming. For communication purposes, however, this is not optimal because the characters must be explicitly counted when marshaling parameters into a network message [Eid97]. ORB implementations must support standard language mappings, but they can internally avoid inefficient representations, although unnecessary conversions are also a potential source of overhead [Sch97].

It has been observed that the current implementations of the IIOP protocol, such as the reference implementation from SunSoft [Sun95], can be substantially improved with well-known protocol implementation enhancement techniques [Gok98]. In addition, it is possible to use the GIOP over transport layer protocols that are more efficient and better suited for performance-sensitive applications than the TCP/IP [Sch97]. Moreover, the use of a single invocation protocol may be too restrictive for some systems. Crane proposes a technique for dynamically selecting the elements of a protocol depending on the interface being accessed [Cra97]. For example, interactions across ORB boundaries might be implemented with the standard IIOP protocol, while interactions within a single ORB use an optimized non-standard protocol.

Inefficient stubs and skeletons. Efficient memory management is essential when request parameters are marshaled into message buffers and when they are unmarshaled into the receiver's memory space. Typical non-optimized stubs and skeletons check the amount of memory repeatedly before every atomic data item is marshaled or unmarshaled. An optimized solution should check and reserve memory for each message as a whole [Eid97]. During repeated operation requests where the data changes only little from one invocation to another (e.g. in multimedia streams), additional optimizations can be effectuated by caching the previous request parameters and thereby reusing previously allocated memory blocks [Sch97]. Currently, the OMG specifications provide no support for such optimizations.

Efficient memory management must be accompanied with efficient copying of data. Copying the request parameters component by component is typically less efficient than copying a block of binary data, especially if the component size is not the same as the machine word size [Eid97]. If efficient block copies are desired, however, the IDL precompiler is no longer a straightforward translation. Instead, the precompiler must separately optimize the code for each stub and skeleton. In some cases, the copying of data can be avoided completely by using appropriate communications hardware that can access directly the application memory through the main system bus [Sch97]. However, this solution requires that the ORB implementation replace the communications facilities offered by the operating system.

A straightforward translation of IDL specifications into stubs and skeletons may create unnecessarily long chains of function calls, especially for complex data structures. Careful elimination of unnecessary function calls (by using inline function implementations) in stubs and skeletons may speed up significantly the processing of complex data types [Eid97]. In many cases, this technique also reduces the size of the stub or skeleton because the inline function implementation can be optimized for each invocation.

Clumsy server implementations. The demultiplexing of operation requests at the server side may be a source of serious performance problems [Gok97]. In a straightforward ORB implementation that follows the conceptual structure of the CORBA specification, operation requests are routed through a number of distinct layers:

1. The protocol stack implementation routes network messages to the ORB core,
2. The ORB core routes the resulting request to the object adapter,
3. The object adapter routes the message to the object implementation,
4. The object implementation routes the message to the correct skeleton and, finally,
5. The object skeleton invokes the requested operation.

In a typical ORB implementation, steps 2, 3 and 4 can be integrated so that the demultiplexing of operation requests can be effectuated with much less overhead [Gok97, Eid97].

In the current CORBA specification, the Basic Object Adapter is responsible for activating object implementations and individual objects through a complex protocol. The protocol starts when an object reference is created and object-specific reference data is passed to the BOA. When a client invokes an operation on an object that is currently inactive, the protocol continues through several steps until the object implementation can examine the reference data in order to recover the state of the object, say, from a database. As a result, both BOA and the object implementation maintain persistent state information about the objects in the system, but there is no standard way for optimizing the usage and maintenance of this information. Consequently, straightforward BOA implementations have problems with large amounts of CORBA objects and, for example, Sun suggests that CORBA objects should not be used as frequently as normal C++ objects [Sun97].

4 Performance issues of CORBA-based applications

This section discusses application-level guidelines that are designed to improve or guarantee the performance of CORBA-based systems. A common feature in these solutions is the attempt to use existing CORBA implementations as they are. The guidelines fall into two groups. The first group provides design solutions for known performance bottlenecks in the CORBA architecture and in the ORB products. However, these guidelines seldom discuss the underlying technical issues in the CORBA architecture that are the primary cause for the performance problem. The second group of guidelines is intended for optimizing the usage of system resources and for maximizing the amount of parallel execution. Guidelines in this group are typically needed in situations where the standard functionality of the ORB run-time environment uses excessively system resources or blocks applications unnecessarily.

Application-level performance guidelines are often presented in the form of *design patterns*. Design patterns provide a structured way for documenting and publishing software design expertise. In addition, the use of common object-oriented terminology makes it possible to use the same design pattern across application domains. Most of the patterns discussed in this presentation have been specifically developed for CORBA, but a large number of other patterns may be useful for the designer of CORBA-based systems [Gam94, Cop96]. Note that performance questions are not the only thing to consider when applying the proposed patterns. Design patterns typically list several forces that affect their usage.

At least the following performance issues have been addressed in the literature of design patterns and other architectural guidelines:

- Increasing the amount of parallel execution,
- Reducing the number system components,
- Reducing the amount of communication, and
- Improving the architecture of CORBA-based systems.

In the rest of this section, we shortly discuss each issue, and also indicate the underlying technical motivations in the CORBA architecture.

Increasing parallelism. The CORBA specification does not discuss the use of threads or similar programming-level techniques for increasing parallelism in ORB implementations. As a result, multiple threads are rarely used in CORBA-based applications. In addition, the use of synchronous operation invocations (i.e. the standard way for invoking IDL operations) further reduces the amount of parallel execution. To increase parallelism, a number of application-level techniques have been proposed. First, the synchronous operation invocations can be replaced with fully or partially asynchronous communication [Mow97]. The purpose of these techniques is to allow the clients and the servers to proceed in parallel. For example, the server can invoke a callback routine on the client when the requested operation has been completed, or the server can continue the processing of the client's request after returning the results to the client.

Another technique for increasing parallelism is to use threads with synchronous communication. At the server side, the object adapter can launch a new thread for each incoming request. This technique is transparent to the application code apart from the need to synchronize the simultaneous access to the object's internal state. At the client side, however, threads require more work from the application programmer. Consequently, a special technique has been proposed to hide the use of threads in CORBA clients [Hel96]. When the client application invokes an operation, a new thread is launched transparently to wait for the server's reply. The client's main thread continues to execute the application code until it attempts to use the results of the pending invocation. If the server has already delivered the results, the client application code simply accesses them. However, if the server is still executing the request, the client blocks until the results become available.

Reducing the number of system components. Generated IDL stubs and skeletons increase the size of applications because they are usually linked to the application executables. Compiler optimization techniques can reduce the size of the stubs and skeletons [Eid97] but, still, every operation and attribute in an interface increases the application size. Therefore, special guidelines have been created for reducing the number of interfaces, operations, and attributes in CORBA-based applications. For example, an interface can offer two generic operations, *get* and *set*, for the client applications to access any number of attributes [Mow97]. In the same way, it is possible to reduce the number of operations and interfaces by abstracting away differences that exist between them. Such generic interfaces and operations often require mechanisms for specialization. In this case, context objects can be used to provide information about the request without the context object being part of the argument list for the request [DEC96]. Another alternative is to use the *any* data type as a placeholder for a generic parameter that changes its type from one invocation to another. However, the *any* data type entails the use of meta-information at run-time and this may incur a serious degradation of performance [Zie96].

Large number of objects and object implementations in a single server node is a typical cause for performance problems [Sun97]. Special techniques can be used to increase the number of supported objects, such as the use of efficient hashing algorithms for finding the appropriate method [Gok97], and the correct division of work between the object implementations and the object adapter [OMG97b]. However, application-level techniques are still needed for reducing the number of objects and object implementations. A straightforward method for reducing object implementations is to use the *shared server* activation policy [OMG97a]. This policy is easy to implement with object-oriented programming languages, but additional programming effort is needed if the server is implemented with a non-object-oriented programming language, such as C or COBOL [Mow97]. A related optimization technique is to combine multiple object implementations into one operating system process.

The Names Library in the CORBA Naming Service [OMG97c] uses a common technique for reducing the number of CORBA objects: if an object does not need the full CORBA functionality, it is converted into a lightweight *pseudo-object*. Operations are invoked on pseudo-object in the normal way, except that the pseudo-objects do not support remote invocations. This way, the pseudo-objects do not use the ORB, but the clients can still use them through CORBA-like interfaces. The drawback of this technique is that it may require the manual modification of IDL stubs [Mow97].

Reducing the amount of communication. The amount of remote communication is a typical target for optimizations in distributed systems because of the limitations in network bandwidth and latency. There are several techniques for reducing remote communication in CORBA-based systems. We shortly discuss four of them. First, the conversion of CORBA objects into pseudo-objects not only reduces the number of objects, but it also eliminates remote access to them. Second, the client stubs can be modified to maintain a local cache of remote objects [Mow97]. This way, access to the remote object is minimized but the client can still maintain the abstract view provided by the IDL interface. Third, an application-level object can be divided into two half-objects that use an optimized protocol for maintaining the necessary state information in both halves [Mes94]. This way, both half-objects can locally support non-optimized CORBA interfaces, but the overall design can avoid most of the related overhead.

The fourth technique for reducing the communication overhead is to replicate CORBA objects in the network. This way, clients can invoke operations locally while the replicated servers synchronize in the background. Some ORB implementations provide ready-made solutions for creating multiple implementations for a single object reference [DEC96], but standard CORBA facilities can also be used for object replication [Mow97]. An important use of replication is the implementation of high availability through *virtual synchrony* [Maf97]. This technique uses a group abstraction mechanism for ensuring that the request of a client succeeds if there is at least one operational replica in the group. In process-oriented applications with very little persistent data, virtual synchrony can implement reliable system behavior with much less overhead compared to traditional ACID type transaction processing.

Improving the system architecture. The CORBAservices have been designed to support general-purpose needs in different application domains. Hence, they cannot be optimized for all domains. Several application-level solutions have been proposed for simplifying and optimizing the CORBAservices for particular application environments. A domain-specific layer between the client and a general-purpose service reduces the number of operations requests and the amount of data to be transferred. Consequently, such layers can improve the performance of the system. For example, the context-relative CORBA Naming Service can be hidden under a simplified interface that uses a single naming context [Mow97]. In some cases, it may even be necessary to create domain-specific service implementations. For example, the requirements of real-time applications necessitate a special version of the CORBA Event Service [Har97].

Finally, we discuss the reuse of system architectures for improving application performance. If the architecture of a CORBA-based system is carefully designed and similar architectures have been used earlier, system developers have means for estimating the performance characteristics of the resulting system. In addition, they may be able to pinpoint and remove potential bottlenecks before the system is implemented. These observations suggest that *application frameworks* should be used for implementing performance-sensitive systems [Mow95]. Although most frameworks within the area of distributed computing have been designed for controlling the complexity [Cop95], it has been observed that they also reduce system-level communication traffic and improve overall performance [Mow97].

5 Concluding remarks

In this paper, we have reviewed the state-of-the-art solutions for improving the performance of CORBA-based systems. The presented solutions can be divided into two groups. The first group focuses on the CORBA architecture and ORB implementations, while the second concentrates on application-level guidelines for avoiding known bottlenecks and for optimizing resource usage. It was observed that design patterns were used extensively for documenting the application-level guidelines.

While presenting the current research results, we observed several relationships between the different performance issues. In particular, we have indicated how the performance problems within the CORBA architecture give rise to application-level guidelines. This way, the paper provides hints for finding more than one solution for a given performance problem. These observations could be used as a starting point for a more comprehensive framework for supporting the performance analysis of CORBA-based systems. However, such a framework requires further classification of the observed performance issues. This classification should pay attention to a number of relevant issues that have not been addressed in this paper, such as the type of the target application, the selected CORBA environment, and the tradeoffs between various user requirements. Moreover, precise analysis is needed for characterizing the relationships between the observed performance problems, the possible solutions, and the consequences entailed by the solutions.

References

[Bla97] Blair, Gordon, Stefani, Jean-Bernard, Open Distributed Processing and Multimedia, Addison-Wesley, Great Britain, 1997.

[Cop96] Coplien, James, Schmidt, Douglas (eds.), Pattern Languages of Program Design, Addison-Wesley, Reading, Massachusetts, USA, 1994.

[Cra97] Crane, Stephen, Dulay, Naranker, A Configurable Protocol Architecture for CORBA Environments, Proceedings of the Third International Symposium on Autonomous Decentralised Systems, Berlin, Germany, April 1997.

[DEC96] Digital Equipment Corporation, ObjectBroker—Designing and Building Applications, Maynard, Massachusetts, USA, May 1996.

[Eid97] Eide, Eric, Frei, Kevin, Ford, Bryan, Lepreau, Jay, Lindstrom, Gary, Flic: A Flexible, Optimizing IDL Compiler, Proceedings of the ACM SIGPLAN '97 Conference on Programming Language Design and Implementation (PLDI), Las Vegas, Nevada, June 1997.

[Gam94] Gamma, Erich, Helm, Richard, Johnson, Ralph, Vlissides, John, Design Patterns: elements of reusable object-oriented software, Addison-Wesley, Reading, Massachusetts, USA, 1994.

[Gok97] Gokhale, Aniruddha, Schmidt, Douglas, Evaluating the Performance of Demultiplexing Strategies for Real-time CORBA, Proceedings of GLOBEGOM '97, Phoenix, Arizona, November 1997.

[Gok98] Gokhale, Aniruddha, Schmidt, Douglas, Principles for Optimizing CORBA Internet Inter-ORB Protocol Performance, to appear in the Proceedings of the 31th Hawaii International Conference on System Sciences, Volume VII, Hawaii, USA, 1998.

[Har96] Harrison, Timothy, Schmidt, Douglas, Design and performance of an Object-Oriented Framework for High-Speed Electronic Medical Imaging, USENIX Computing Systems Journal, Vol. 9, No. 3, November/December 1996.

[Har97] Harrison, Timothy, Levine, David, Schmidt, Douglas, The Design and Performance of a Real-time CORBA Event Service, Proceedings of the OOPSLA '97, Atlanta, Georgia, USA, October 1997.

[Hel96] Hellemans, Patrick, Steegmans, Frank, Vanderstraeten, Hans, Zuidweg, Han, Implementation of Hidden Concurrency in CORBA Clients, Proceedings of the International Workshop TreDS '96, Springer-Verlag, Aachen, Germany, October 1996.

[Lec96] Leclerc, M., Linnhoff-Popien, C., Lipperts, S., Müssener, T., Wegmann, T., CORBA-Based Data Transfer for Financial Risk Management, Proceedings of International Workshop TreDS '96, Springer-Verlag, Aachen, Germany, October 1996.

[Maf97] Maffeis, Silvano, Schmidt, Douglas, Constructing Reliable Distributed Communication Systems with CORBA, IEEE Communications Magazine, Vol. 35, No. 2, February 1997.

[Men94] Menascé, Daniel, Almeida, Virgílio, Dowdy, Larry, Capacity Planning and Performance Modeling, Prentice Hall, New Jersey, 1994.

[Mes94] Meszaros, Gerard, Pattern: Half-object + Protocol (HOPP), Pattern Languages of Program Design, Addison-Wesley, Reading, Massachusetts, USA, 1994.

[Mow95] Mowbray, Thomas, Zahavi, Ron, The Essential CORBA: Systems Integration Using Distributed Objects, John Wiley & Sons, Inc., New York, USA, 1995.

[Mow97] Mowbray, Thomas, Malveau, Raphael, CORBA Design Patterns, John Wiley & Sons, Inc., New York, USA, 1997.

[OMG97a] Object Management Group, The Common Object Request Broker: Architecture and Specification, Rev. 2.1, August 1997, Framingham, Massachusetts, USA.

[OMG97b] Object Management Group, ORB Portability Joint Submission, OMG document orbos/97-05-15, May 1997, Framingham, Massachusetts, USA.

[OMG97c] Object Management Group, CORBAservices: Common Object Services Specification, Framingham, Massachusetts, USA, July 1997,

[Orf97] Orfali, Robert, Harkey, Dan, Edwards, Jeri, Instant CORBA, John Wiley & Sons, Inc., New York, 1997.

[Sch97] Schmidt, Douglas, Gokhale, Aniruddha, Harrison, Timothy, Parulkar, Guru, A High-Performance End System Architecture for Real-Time CORBA, IEEE Communications Magazine, Vol. 35, No. 2, February 1997.

[Sun95] SunSoft, Inc., SunSoft Inter-ORB Engine, Release 1.1, June 1995. Available from ftp://ftp.omg.org/pub/interop/iiop.tar.Z.

[Sun97] Sun Microsystems, Inc., NEO Performance Tips, Palo Alto, California, USA, 1997. Available from http://www.sun.com/.

[Zie96] Zielinski, K., Uszok, A., Steinder, M., Crossing Technological Domains Using the Inter-ORB Request Level Bridge—Preliminary Performance Study, Proceedings of the International Workshop TreDS '96, Springer-Verlag, Aachen, Germany, October 1996.