

CORBA-as-Needed: A Technique to Construct High Performance CORBA Applications

Hui Dai¹, Shivakant Mishra¹, and Matti A. Hiltunen²

¹ Department of Computer Science, University of Colorado
Campus Box 0430, Boulder, CO 80309-0430

² AT&T Labs-Research
180 Park Avenue, Florham Park, NJ 07932

Abstract. This paper proposes a new optimization technique called CORBA-as-needed to improve the performance of distributed CORBA applications. This technique is based on the observation that in many cases the client and the server of a distributed application run on compatible computing platforms, and do not need the interoperability functionality of CORBA. CORBA-as-needed dynamically determines if the interoperability functionality is needed for a specific application invocation, and bypasses this functionality if it is not needed. Performance measurements from a prototype implementation in omniORB show that CORBA-as-needed achieves a very significant performance improvement.

1 Introduction

CORBA [2] has been successfully used in the development of a large number of distributed applications. However, it suffers from an important deficiency. There is a significant performance overhead in using CORBA to construct distributed object applications [3, 4]. Therefore, it is difficult to construct high performance distributed object applications using CORBA. The main source of this performance overhead is the large one-way communication delay incurred by CORBA communication methods, compared to the one-way communication delays of transport-level protocols such as UDP or TCP. We propose a technique called CORBA-as-needed that allows distributed applications to bypass CORBA interoperability functionality whenever the client and the server happen to run on compatible platforms. It allows client server applications to first detect if the interoperability functionality of CORBA is indeed needed for a particular invocation, and bypass it if this functionality is not necessary.

In this paper, we present the design, implementation, and performance evaluation of CORBA-as-needed technique in omniORB [1]. We explore four different design alternatives for incorporating CORBA-as-needed in the current CORBA architecture. These design alternatives are called service approach, integration approach, CORBA wrapper approach, and pluggable ORB module approach. These alternatives differ from one another in the exact layer of CORBA architecture where CORBA-as-needed is incorporated. We provide a thorough analysis

of the four design alternatives, including their pros and cons. We have implemented a prototype of CORBA-as-needed using the pluggable ORB module approach in omniORB[1]. We describe this implementation and a detailed performance evaluation by comparing the performance of several distributed object applications implemented using our prototype and omniORB [1]. This performance comparison shows that CORBA-as-needed is a very useful technique that can improve the latency and scalability of distributed object applications by as much as 25% when clients and servers run on compatible computing platforms. Also, performance measurements show that this technique has insignificant performance overhead when the clients and servers run on incompatible platforms. CORBA-as-needed is a generic technique that can be used in association with other optimization techniques that researchers have used in the past to improve the performance of CORBA applications.

2 CORBA As Needed

While the ability to operate in a heterogeneous distributed computing environment is an important requirement for modern distributed applications, it is important to note that a majority of the applications do in fact use operating systems, network protocols, and hardware platforms that are compatible with one another. The main idea of this paper is to design a system that dynamically recognizes the situations when the extra support for interoperability provided by CORBA is not needed, and allows applications to simply bypass the interoperability functionality by using a standard TCP/UDP communication mechanism for those situations. In other words, interoperability support of CORBA is used by applications only when it is really needed.

The CORBA-as-needed technique identifies operating conditions when a client and a server are running on compatible computing platforms at the time when the client initiates a connection to the server. If the client and the server are determined to be running on compatible computing platforms, the ORB redirects all future communication requests to the lower transportation layer without passing through the rest of ORB core that implements interoperability functionality. If the client and the server are determined to be running on incompatible platforms, all future communication requests are directed through ORB core in the usual manner.

2.1 Design Alternatives

There are at least four different design alternatives to incorporate CORBA-as-needed technique in the current CORBA architecture. These are (1) service approach, (2) integration approach, (3) CORBA wrapper approach, and (4) pluggable ORB module approach.

In the service approach (Figure 1), CORBA-as-needed is implemented as a new CORBA service called *bypass service*. The bypass service uses a similar

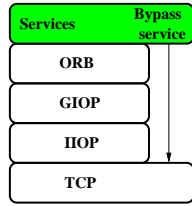


Fig. 1. Service Approach

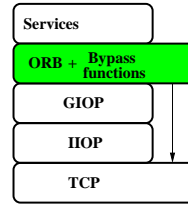


Fig. 2. Integration Approach

interface as other services in CORBA, but uses TCP or UDP as the underlying communication mechanism. The main advantage of this design alternative is that it provides for an easy integration in the CORBA service framework, and can take advantage of other CORBA services such as security. The key problem with this design is that a client needs to have an explicit knowledge of the server's computing environment, so that it can invoke the bypass service whenever appropriate. Thus, this design does not preserve the transparency of the CORBA-as-needed technique from the application. Another problem is the potential performance overhead. In an earlier research project, we observed that providing additional functionality such as group communication as a CORBA service can result in a very significant performance overhead [4].

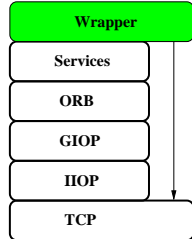


Fig. 3. Wrapper Approach

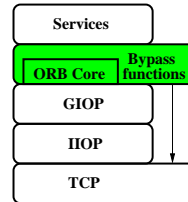


Fig. 4. Pluggable ORB Module Approach

In the integration approach (Figure 2), an existing ORB is extended to include the CORBA-as-needed functionality. The main advantage of this approach is that it can provide very good performance. Since all state information is available in the ORB, the implementation is also simpler. However, there are several problems associated with this approach. First, this approach violates the basic philosophy behind the design of CORBA: the ORB should provide minimal functionality to allow interoperability between heterogeneous objects. By integrating additional functionality in ORB, the functionality of ORB are overloaded. A more serious problem with this approach is that it may result in the loss of portability and interoperability. Integrating additional functionality in ORB forces applications to be ORB dependent. Application clients and servers are

required to use the same ORB implementation (or compatible implementations of the additional functionality).

In the CORBA wrapper approach (Figure 3), the CORBA-as-needed technique is implemented as a separate system module that intercepts all CORBA command invocations between an application and a CORBA ORB. When a CORBA client initiates a new connection with a CORBA server, the system module intercepts the request and forwards it to the ORB. The ORB then calls the named service to get the appropriate information and returns it to the system module. If the system module determines that the client and the server are running on compatible platforms, it redirects all the following communication requests to the transport layer and bypasses the ORB. The main advantage of this approach is its simplicity. It provides a clean separation between an ORB, CORBA services, and the CORBA-as-needed technique. The problem is again the potential for significant performance overhead. Because the system module is a separate module that sits between a CORBA application and an ORB, it may impose significant performance overhead.

Finally, in the pluggable ORB module approach (Figure 4), the CORBA-as-needed is implemented as a separate module that can be plugged into an ORB. The ORB interface to the application is kept untouched. This design approach is based in the idea of pluggable protocols that has been used in many middleware systems, including CORBA [6], to improve a middleware's performance and provide additional functionality. When a client initiates a connection with the server, the ORB activates a component to determine the computing environment of the requested object. If the server's computing environment is determined to be compatible with the client's environment, all future communication are redirected through a TCP/IP connection bypassing the original ORB core. The main advantage of this approach is that it is expected to provide good performance. In addition, because the CORBA-as-needed technique is implemented as a separate module, the existing ORB code is minimally affected. We have used this approach in our prototype implementation.

2.2 Pluggable ORB Module

Figure 5 illustrates the communication model of the ORB in which CORBA-as-needed has been implemented as a pluggable ORB module. On a client's first invocation, the extended ORB first locates the server (location service), determines the computing environment of the server (environment service), and then establishes a logical communication channel either by using the GIOP/IIOP (if the client's and the server's computing platforms are incompatible), or the underlying TCP/IP protocol (if the client's and the server's computing platforms are compatible). An important observation is that this entire process of determining if the interoperability functionality of CORBA is indeed needed, establishing an alternate connection, and facilitating all future communications is transparent to both the client and the server.

Figure 6 illustrates the operational details of CORBA-as-needed implemented as a pluggable ORB module. There are five important components here: (1)

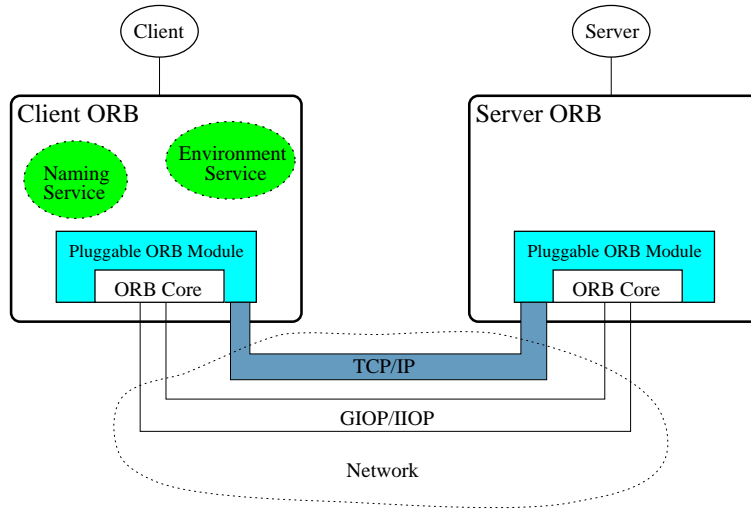


Fig. 5. CORBA-as-Needed as a pluggable ORB Module.

shared memories, (2) worker daemons, (3) reader daemons, (4) environment service, and (5) ORB. Shared memory is used on both the client and the server side for interprocess communication on the same machine. There are two types of shared memories. SEND_MEM is used by the ORB to store the request information on the client side and the reply message on the server side. The RECV_MEM is used by the reader daemon to store the information received from the other side (client side for a server, and server side for a client). Local ORB watches RECV_MEM for the incoming messages. The shared memory structure includes a flag to indicate any change in the memory made by the ORB or the reader daemon, the address of the servant, and some other support data structures.

Worker daemons monitor SEND_MEM. When the ORB stores a message in SEND_MEM, the worker daemon retrieves it and sends it to the other side. Reader daemon receives messages coming from the other side and stores them in the RECV_MEM. The environment service fetches the computing environment of the server. Finally, the ORB wrapper provides the CORBA-as-needed functionality as a separate module. The following steps are executed when the ORB receives an initial request from a client for connection with a server:

1. ORB checks with the naming and environment services to get the location of the target object as well as its computing environment information.
2. The naming/environment service responds with the location and environment information of the servant.
3. If the computing environments of the client and the server are compatible, the client ORB puts the request message into the SEND_MEM. This message encapsulates the target class name, method name, arguments to the target object, and server's address. It then sets the flag to indicate that there is new information in SEND_MEM.

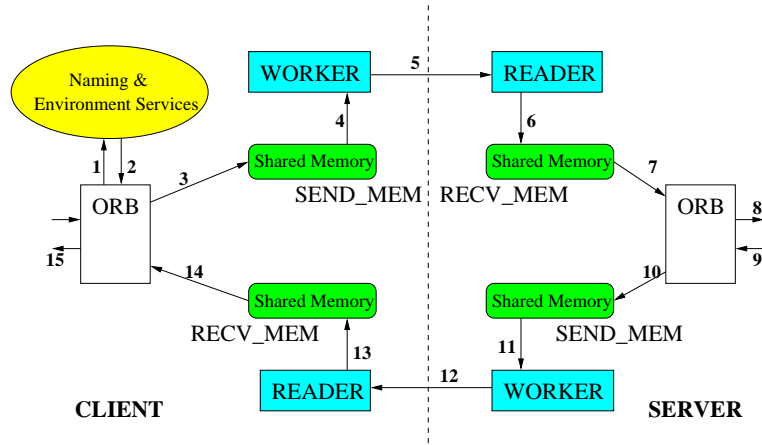


Fig. 6. Invocation steps in CORBA-as-needed prototype.

4. The worker daemon notices the change in the SEND_MEM flag. It reads the new message from SEND_MEM.
5. The worker daemon sends the request to the reader daemon on the server side.
6. The server side's reader daemon receives the request message and puts it in the server side's RECV_MEM, and sets the flag.
7. The ORB on the server side notices the change in RECV_MEM's flag, and reads in the request message from RECV_MEM.
8. The ORB calls the dispatch mechanism of the servant.
9. The servant returns the result of the request to the server side's ORB.
10. The server side's ORB puts the reply message in the SEND_MEM. This message encapsulates the result of the request. It then set the flag to indicate that there is new information in SEND_MEM.
11. The server side's worker daemon notices the change in the SEND_MEM's flag, and retrieves the reply message from SEND_MEM.
12. The server side's worker daemon sends the reply message to the reader daemon on the client side.
13. Client side's reader daemon receives the reply message and puts it in the client side's RECV_MEM, and sets the flag.
14. The ORB notices the change in the RECV_MEM's flag, and retrieves the reply message from RECV_MEM.
15. The ORB returns the result to the client.

3 Performance

In order to understand the effect of CORBA-as-needed technique on the performance of a client server application, we have measured the performance of

our CORBA-as-needed prototype and compared it with the performance of omniORB. In our performance measurement, we have concentrated on the overall performance of a client server application.

3.1 Computing Environment

The performance measurements were done over two different computing environments: LAN and WAN. In the LAN computing environment, the client and the server were run on two separate HP75 PCs. Both of them have PIII, 450 MHz CPU, with 128 MB RAM. These two PCs are connected by a 10 Mbps Ethernet. Both of these PCs were running Linux. In the WAN computing environment, the client was run on an HP75 PC in the University of Colorado, Boulder, while the server was run on a dual-processor, Linux cluster with PIII, 1 GHz CPUs and 256 MB of RAM in the University of Wyoming, Laramie. The two PCs were connected by the Internet.

We constructed a simple client-server application for all our measurements. Both the client and the server are written in C++. The application is a simple echo string application, where the client invokes a method called `echoString` (`string str`) on an object `Echo` that resides on the server. When invoked, this method simply returns the string parameter `str` back to the client.

3.2 Performance Index

We have measured *application latency* to evaluate the effect of CORBA-as-needed on the performance of a CORBA application. Latency is defined as the time interval between the moment a client ORB sends out a request until it gets the reply. Latency consists of two parts: the overhead of the ORB core or pluggable ORB module implementing CORBA-as-needed, and the time spent in the network transmission.

3.3 Performance Results

We measured the latency for five different string sizes: 200 B, 1 KB, 4 KB, 10 KB, and 100 KB. Figure 7 shows the latency measured for these five string sizes. The latency reported in this figure is an average latency over 200 invocations of the method `echoString()`. As we can see from these measurements, CORBA-as-needed results in the improvement of the application latency by as much as 25%. There are two important observations we make from these measurements. First, the percentage improvement in the application latency increases with increase in the size of the string parameter. The main reason for this increase is marshalling and unmarshalling of the string parameter performed by omniORB. Larger the size of the string, longer it takes to do this. As observed in [5], the main source of performance overhead is marshalling and unmarshalling of function parameters. CORBA-as-needed avoids this marshalling and unmarshalling if the client and the server are running on compatible computing platforms.

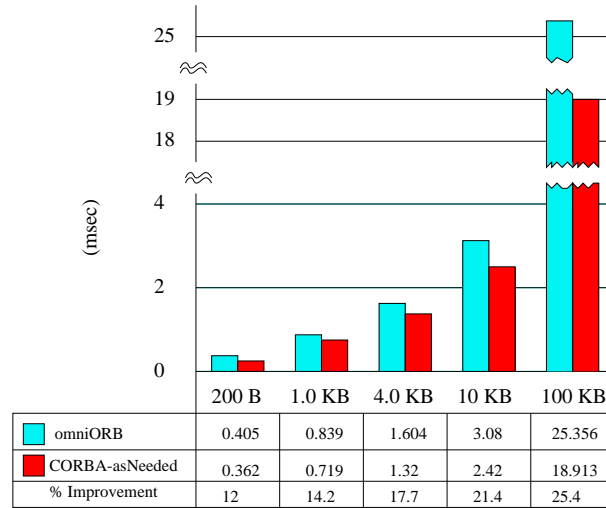


Fig. 7. Application Latency in LAN.

The second observation we make from our performance measurement is that the application latency of CORBA-as-needed was slightly higher for the first invocation compared with the rest of the invocations. Actually, the same was true for omniORB as well. The main reason is that in the first invocation, a connection with the server needs to be established between the client and the server. CORBA-as-needed also interacts with naming and environment services at this time. For all future invocations, both omniORB and CORBA-as-needed reuse the connection established for the first invocation.

Application latency for the WAN environment is shown in Figure 8. Again, the latency reported in this figure is an average latency over 200 invocations of the method `echoString()`. As we can see, the CORBA-as-needed does result in a significant improvement in application latency in WAN computing environment as well. We noticed that the two observations we made from the latency measurement in the LAN environment were also present in the WAN computing environment. These are the increasing improvement in application latency with increase in string size, and a slightly larger latency for the first invocation. The anomaly of lower improvement in application latency when the string size was 10 K, as compared to when the string size was 1 K or 4 K is attributed to the unstable communication characteristics of the Internet, i.e. the communication delays are rather variant in the Internet.

In addition, we noticed that the variation in the application latency between different invocations was much larger than in case of LAN. Again, this is attributed to the unstable communication characteristics of the Internet. Another observation we made was that the improvement in application latency due to CORBA-as-needed was generally lower in WAN than in LAN. This is because the time spent by omniORB in addressing the interoperability issues (marshalling,

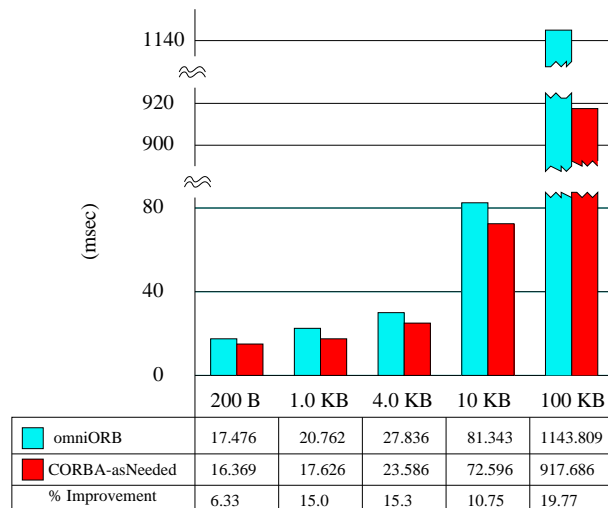


Fig. 8. Application Latency in WAN.

unmarshalling, byte padding, byte alignment, etc.) is constant for a given invocation, and is independent of the network latency. So, as the network latency increases, the percentage of time spent on addressing the interoperability issues, and hence the overhead of omniORB, decreases.

Finally, we also measured the overhead of CORBA-as-needed when the client and the server run on incompatible platforms, i.e. when the CORBA-as-needed technique does not result in improving an application performance. We noticed that the performance overhead was extremely low. It was less than 1% in LAN and less than 0.5% in WAN. As expected, this performance overhead decreased with increase in string size.

4 Discussion

Heterogeneity in modern, distributed computing systems is a fact of life. As a result, the ability to operate in a heterogeneous, distributed computing environment is essential for any current and future distributed computing service. CORBA is a very useful technology, because it enables the construction of distributed applications that can operate in a heterogeneous distributed computing environment. CORBA naturally incurs some performance overhead to provide this support. This extra performance overhead has limited the applicability of CORBA.

In this paper, we have proposed a technique called CORBA-as-needed that can improve the performance of distributed CORBA applications under some common computing scenarios. The main motivation behind this approach is the observation that a majority of the components in modern, distributed computing environments are compatible with one another and can interoperate with one

another. This is despite the fact that most modern, distributed computing environments are heterogeneous in general. So, while CORBA is needed to develop modern, distributed computing applications, its interoperability functionality is not really needed in a significant number of application runs.

Our performance measurements show that CORBA-as-needed does result in a significant performance improvement when the underlying computing platforms of client and server are compatible with one another. This performance improvement is as high as 25% when the client and the server are running on the same subnet, and 20% when they are connected via the Internet. It is important to note that this performance improvement can result in even a much higher performance improvement for more complex distributed applications such as group communication systems.

CORBA-as-needed identifies computing instances when client and server's computing platforms happen to be compatible with one another, and bypasses interoperability functionalities for those instances. This idea can be generalized by identifying other special instances when one or more CORBA functionalities are not needed by the application. As an example, another reason for poor performance of distributed, CORBA applications is that CORBA ORBs typically use TCP as a transport-level protocol. The reason for this is that GIOP specifications assume a reliable, stream-oriented protocol as the underlying transport-level protocol. In one of our earlier projects, we observed that this contributes to a significant performance overhead for applications that do not need TCP functionality [4]. Since, in a LAN environment, UDP is mostly reliable and almost always provides a sequenced message delivery, it is worthwhile to experiment with a pluggable ORB module that allows applications to use UDP in a LAN environment. Another possibility is to use an alternate high performance communication protocol, if such a protocol is available.

References

1. *omniORB: Free High Performance CORBA 2 ORB*. AT&T Laboratories, Cambridge. URL: <http://www.uk.research.att.com/omniORB/>.
2. *The Common Object Request Broker: Architecture and Specification*. Object Management Group, Framingham, MA, 1998.
3. A. Gokhale and D. C. Schmidt. Measuring the performance of communication middleware on high-speed networks. In *Proceedings of SIGCOMM'96*, Aug 1996.
4. S. Mishra, L. Fei, X. Lin, and G. Xing. On group communication support in CORBA. *IEEE Transaction on Parallel and Distributed Systems*, 12(2), February 2001.
5. S. Mishra and N. Shi. Improving the performance of distributed CORBA applications. In *Proceedings of the 2002 IEEE International Parallel and Distributed Processing Symposium*, Fort Lauderdale, FL, April 2002.
6. W. Zhao, L. Moser, and P. Melliar-Smith. Design and implementation of a pluggable fault tolerant CORBA infrastructure. In *Proceedings of the 2002 IEEE International Parallel and Distributed Processing Symposium*, Fort Lauderdale, FL, April 2002.