Strategies for Migrating from DCE to CORBA

Douglas C. Schmidt and Aniruddha Gokhale
schmidt@cs.wustl.edu and gokhale@cs.wustl.edu
Department of Computer Science
Washington University, St. Louis, Missouri, 63130

Abstract

Communication software middleware and distributed services for next-generation applications must be reliable, flexible, reusable, and capable of providing the necessary quality of service to applications like multimedia and telecommunication systems running over high-speed networks. Requirements for reliability, flexibility, and reusability motivate the use of object-oriented middleware like the Common Object Request Broker Architecture (CORBA).

This document provides an indepth analysis of DCE and CORBA in terms of their key similarities and differences. It evaluates the advantages of porting DCE applications to CORBA, and describes the key areas where the porting effort is most likely to face the hardest problems.

1 Introduction to CORBA

CORBA is an evolving standard for distributed object computing [11]. The CORBA standard defines a set of components that allow client applications to invoke operations (`op`) with arguments (`args`) on object implementations. Flexibility is enhanced by using CORBA since object implementations can be configured to run locally and/or remotely with minimal impact on their implementation or use.

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 operations on object implementations. Accessing the services of a remote object should be transparent to the caller. Ideally, this should be as simple as calling a method on an object, *i.e.,* `result = obj->op(args)`. The remaining components in Figure 1 help to support this location transparency.

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

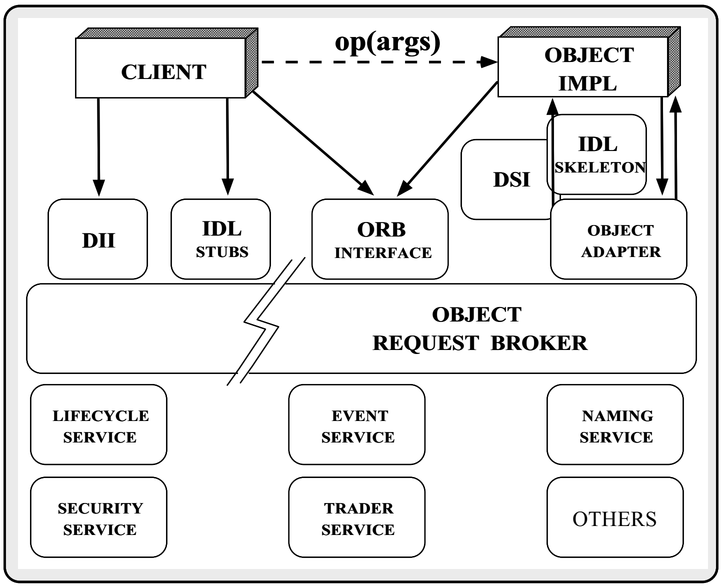

Figure 1: Components in the CORBA Distributed Object Computing Model

• **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 [13].

• **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:** The Object Adapter 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 efficient access to non-remote objects).

• **Higher-level Object Services:** These services include the CORBA Object Services [10] such as the Name service, Event service, Object Lifecycle service, and the Trader service. The OMG is currently defining wide range of higher-level object services.

In general, CORBA enhances conventional procedural RPC middleware (such as OSF DCE and ONC RPC) by supporting object-oriented language features (such as encapsulation, interface inheritance, parameterized types, and exception handling) and advanced design patterns [5] for distributed communication. In principle, these features and patterns enable complex distributed and concurrent applications to be developed more rapidly and correctly. In practice, however, existing CORBA implementations, and the OMG CORBA standardization effort itself, is still relatively immature, as described below.

2 DCE vs. CORBA

Both DCE and CORBA support the development and integration of applications in heterogeneous distributed environments. This section summarizes the main features of DCE and CORBA and describes their key similarities and differences [3].

2.1 Key DCE/CORBA Similarities

The key similarities between DCE and CORBA are described below:

• **Simplify common network programming tasks:** Both DCE and CORBA are designed to simplify common tasks of building distributed applications such as service registration, location, and activation, demultiplexing, framing and error-handling, parameter (de)marshalling, and operation dispatching.

• **Support for heterogeneous environments:** Both DCE and CORBA shield application developers from differences in programming languages, operating systems, computer hardware (particularly instruction byte ordering), and networking protocols.

• **Use of Interface Definition Languages (IDLs):** Both DCE and CORBA support the definition of service components, using high-level interface definition languages. The main purpose of an IDL is to separate interface from implementation. In general, this separation of concerns makes it possible to:

- Improve the modularity and specification of software components;
- Transparently distribute implementation across process and host boundaries;
- Write language-independent applications.

• **Automatically generated stubs and skeletons:** Implementations of DCE and CORBA provide IDL compilers that automatically translate IDL definitions into client-side stubs and server-side skeletons. Stubs serve as *proxies* [5] that interact with the underlying runtime systems to allow clients to access services defined by servers. Skeletons integrate application-specific code with automatically generated code that performs demarshalling, demultiplexing, and dispatching of client requests to target object implementations.

• **Synchronous request/response communication:** Both DCE and CORBA support synchronous request/response communication. In this approach, the client calls an operation on the server. The client blocks until the server completes the operation, at which point `out` or `inout` parameters and/or a return value is passed back to the client. In theory, synchronous request/response communication helps shield client applications from knowledge of whether the target object implementation is local or remote. In practice, it is often difficult to completely hide the use of distribution from clients due to differences in performance and reliability [2].

• **Oneway communication:** CORBA supports "oneway" (send-only) calls, where the server does not return any information to the client (*e.g.,* as part of the operation's return value or inout/out parameters). In DCE, oneway operations can be achieved using "maybe" semantics, which are a special case of "idempotent" operations.

• **Common request path:** Figure 2 shows the general path that CORBA and DCE implementations use to transmit requests from client to server for remote operation invocations. The client code invokes the IDL compiler-generated stubs to

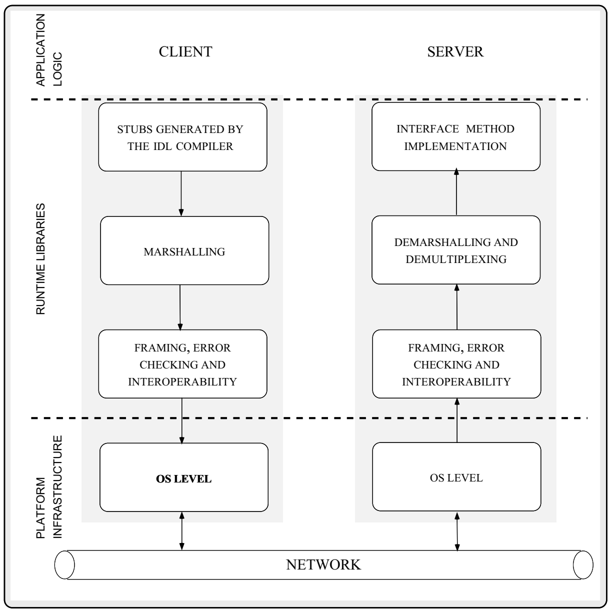

Figure 2: General Path of CORBA and DCE Requests

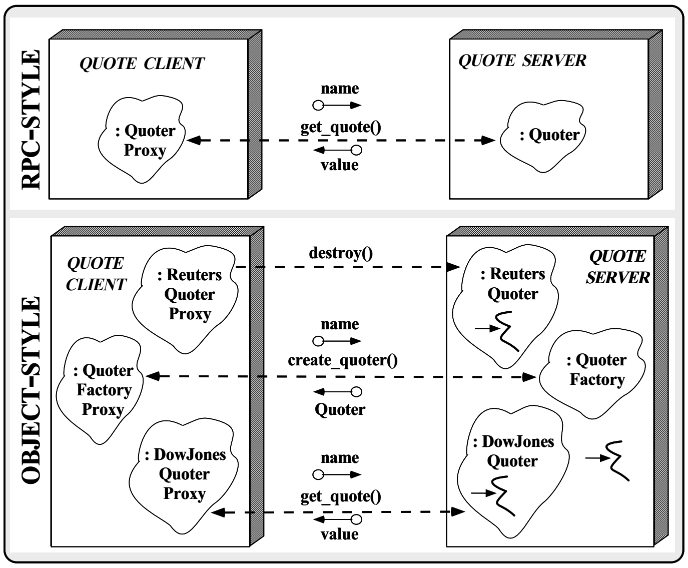

Figure 3: RPC-style vs. Object-style Communication

access the services of the server. After the client invokes the stub, it blocks until it receives a response from the server. The presentation conversion layer encodes the request data into a common data representation. The runtime system then packetizes the encoded data by adding framing information. This may include packet headers and trailers, checksums for data integrity, encrypted data for security and information for interoperability. The runtime system uses the underlying network software provided by the operating system and device drivers to send the packets to the destination.

On arrival at the server, the network software passes the request to the runtime system. The runtime system removes the framing information and passes the request to the presentation conversion layer. This layer converts the encoded data into the native format of the host machine (if necessary) and passes it over to the message demultiplexer. The demultiplexer dispatches the request to the appropriate server stub generated by the IDL compiler.

The response traces the reverse path of the incoming request message through the server and client. When the response reaches the client, it unblocks the client stub that is waiting for the reply.

• **Higher-level services:** Both DCE and CORBA build upon their core communication infrastructure (called the "executive" in DCE and the "ORB" in CORBA) to provide higher-level distributed services. Common services provided by both CORBA and DCE include a time service, event service, and naming and directory services.

2.2 Key DCE/CORBA Differences

The following bullets describe the key differences between DCE and CORBA.

• **Programming model:** One important difference between DCE and CORBA is that DCE was designed to provide a procedural programming model, whereas CORBA was designed to provide an object-oriented programming model. This difference in programming models is analogous to the difference between the C and C++ programming models. For instance, while it is possible to implement object-oriented programs using C, the effort required to do so is high and the results are often hard to program and error-prone. In contrast, the effort required to implement object-oriented programs with C++ is much lower, because the language supports the features directly.

The difference between DCE's procedural programming model and CORBA's object-oriented programming model is often overstated, however. In particular, there are extensions to DCE that provide an OO veneer (such as OODCE [4] and DCE Objects [1]). There are, however, a number of restrictions inherent in using DCE in an object-oriented manner. To illustrate these, consider the following ways in which the DCE and CORBA programming models differ:

- *Support for multiple inheritance of interfaces and polymorphism* – CORBA provides these features to support the specialization and reuse of existing interfaces. Developers can use inheritance to form new composite interfaces, which can be implemented flexibly using polymorphism. In contrast, DCE does not directly support interface inheritance or polymorphism of implementation.

- *Accessing distributed resources via Object References* – In CORBA, Object References are "first class" entities that can be passed to clients throughout a network and used to flexibly access server objects. DCE does not provide this degree of flexibility without additional programming effort.

- *Object-style vs. RPC-style communication* – Figure 3 illustrates the difference between RPC-style communication (supported by DCE) and Object-style communication (supported by CORBA) [15].[1] There are several benefits to Object-style communication:
  - *Customized quality of service* – Clients can use a factory to create different types of product objects that support a range of functionality or performance characteristics (such as real-time quality of service, high-bandwidth, etc.) tailored to their individual needs.
  - *Flexible lifecycle control* – Object-style communication gives clients more flexibility to control the lifecycle of object implementations, compared with RPC-style communication. For instance, servers accessed via RPC-style interfaces often must make inefficient or non-robust assumptions about the lifecycle of clients that access their services.
- *Ability to associate cohesive operations into modular and reusable components* – CORBA's programming model encourages the association of related operations to form modular and reusable components. Although it is possible to achieve much the same effect in DCE via developer conventions, the standard DCE programming model is not as conducive to supporting OO design and implementation.

• **Communication model:** CORBA supports the "deferred synchronous" communication, which separates the send operation from the server's reply. DCE does not support deferred synchronous operations, though it is possible to approximate this to some extent using multiple threads. However, DCE supports the notion of "idempotent" operations, which can be used to optimize duplicate detection on a server. CORBA does not provide support for idempotent operations.

• **Interface Definition Languages (IDLs):** CORBA IDL is designed to allow interoperability between a range of target languages (such as C, C++, Smalltalk, Java, Ada, and COBOL). In contrast, DCE IDL is focused primarily on C (and C++ to the extent that the type system of C is a subset of C++).

• **Type systems:** An important consequence of CORBA's emphasis on language independence is that its type system is much simpler (and inherently more restrictive) than DCE. In particular, there is no direct support for:

- *Passing pointers or structures containing pointers*
- *Streaming data – e.g.,* via the DCE `pipe` mechanism.
- *Conformant arrays – i.e.,* arrays of varying sizes.

On the other hand, unlike DCE, CORBA support the "any" type, which allows clients and servers to pass arbitrary data values whose type is determined at run-time.

[1] Note that CORBA also supports RPC-style communication.

• **Dynamic invocation:** CORBA supports the dynamic invocation of requests that can be created and called at run-time. The correctness of these requests can be checked at run-time using the CORBA Interface Repository. In contrast, DCE does not provide support for dynamic invocation or interface repositories.

• **Interface and Implementation Repository:** CORBA provides an interface repository that stores information present in the IDL files. Applications can query an interface repository for information about the interfaces. This feature is useful for tools such as browsers and debuggers that have no prior knowledge of the interfaces offered by a server. By querying the interface repository, they can find the services offered by different servers and construct requests dynamically.

In addition, CORBA supports an implementation repository that the ORB uses to map client requests to object implementations. An implementation repository holds information that allows the ORB to locate and activate implementations of objects. In addition, it is useful to store other information such as resource allocation, security, debugging information, etc. DCE does not define an interface or implementation repository explicitly.

• **Component identity:** In DCE, all components (*e.g.,* IDL definitions, IDL implementations, servers, etc.) are identified by "universal unique identifiers" (UUIDs). CORBA has no notion of a UUID. Instead, components in CORBA are "identified" via Object References, which only grant access to a CORBA object, but provide no notion of unique identify. For more information on the pros and cons of this issue see [14] and [9], respectively.

• **Infrastructure services:** DCE defines a multi-threading API that is part of its core "executive" infrastructure, whereas CORBA does not define a standard API for multi-threading. Therefore, it is not possible to write portable CORBA multi-threaded applications. Likewise, DCE also defines a security service (based on Access Control Lists) in its core infrastructure, whereas CORBA defines this as a higher-level service.

• **Higher-level services:** The higher-level services defined by DCE and CORBA are different. For instance, DCE defines a distributed file service (DFS) in its *extended services* component, whereas CORBA does not provide this service. On the other hand, the OMG has completed specifications for a much wider range of distributed services, including *Concurrency Control*, *Event Service*, *Externalization Service*, *Life Cycle Services*, *Naming Service*, *Persistent Object Service*, *Query Service*, *Relationship Service*, *Transaction Service*, *Licensing Service*, *Property Service*, *Security Service*, *Time Service*, *Trading Service*.
In addition, the OMG is currently defining standards for even higher-level, application-specific services, known as *CORBA facilities*. These facilities will cover domains such as user-interface, compound documents, and task management. As described in Section 4.1, however, the CORBA services and facilities are not well defined at this point.

- **Interoperability and portability:** The DCE specification and its various implementations were designed *a priori* to be interoperable and portable. In contrast, the original CORBA 1.x specification was not sufficiently detailed to ensure interoperability and portability of CORBA implementations. Although CORBA 2.0 addresses this weakness via the UNO specification [12], many CORBA implementations do not yet implement UNO robustly. In addition, non-portable aspects of the CORBA server-side Object Adapter specification remain including:

  - *Non-portable mapping of skeletons onto implementations* – There is no standard way to map the automatically generated IDL skeletons onto application-specific target object operation implementations.
  - *Incomplete Object Adapter Interface* – The existing interface for the Basic Object Adapter in the CORBA 2.0 standard is woefully incomplete. Therefore, each ORB vendor has added non-standard features in order to make it possible to utilize important OS platform resources such as threads or dynamic linking.

This lack of specificity in the CORBA 2.0 specification makes it impossible to develop portable server implementations. However, the client-side CORBA 2.0 specification does support the development of relative portable clients.

- **Context:** The notion of context in DCE and CORBA is different. Contexts in DCE are used to maintain server states during a series of logically related requests from a single client. The runtime system understands the information stored in the contexts. DCE contexts in a distributed application is analogous to a file handle in a local application. These contexts are maintained by the stubs and the RPC runtime libraries and not by the application code. In contrast, CORBA contexts are opaque to the runtime system. They are used to carry user information along with the request and are similar to UNIX "environment variables." Programmers responsible for managing and interpreting CORBA context information.

The key differences between DCE and CORBA are summarized in Table 2.2.

3 Advantages of Moving to CORBA

3.1 Simplifying Software Development

CORBA offers an object-oriented distributed computing environment. The following are the advantages of using CORBA:

- The use of object oriented technology makes it reliable, flexible, extensible and reusable;
- CORBA allows flexible client-server relationships and consumer/supplier roles that can be interchanged easily and flexibly;
- CORBA allows easier integration of services. Since CORBA IDL can be inherited, it is possible to integrate higher level services easily;
- CORBA's use of an object request broker obviates the need to discover a server and route requests to it;
- CORBA servers need not be active at all times. They can be activated by daemons whenever a request arrives;
- CORBA supports both synchronous, as well as quasi-asynchronous (*i.e.,* deferred synchronous) communications styles;
- CORBA IDL supports inheritance from interfaces;
- CORBA IDL allows interoperability between a wide range of target languages;
- The OMG has specified a wide range of distributed services and higher-level application-specific facilities for CORBA.

3.2 Marketability

Although, CORBA is an emerging technology for distributed systems, it has already attracted a large customer base. A number of customers are demanding that their products be "CORBA compliant". Efforts are also underway in the OMG to achieve interworking between OLE/COM and CORBA. Since OLE/COM is the *de facto* standard for modern desktop computing, it is crucial that the distributed object computing technology interoperate with OLE/COM.

3.3 Support from Third Party Vendors

One of the main strengths of CORBA, relative to DCE, is its current force in the marketplace. There are over a dozen ORB vendors who are actively developing CORBA products on many OS and hardware platforms. Table 2 shows the different CORBA implementations available in the market, along with their URL addresses. Amongst these, IONA'S ORBIX is the most widely used ORB, which an estimated 80% share of the marketplace.

These ORBs are now available for most OS and hardware platforms. The ubiquity of ORB implementations increases competition, which should result in better quality and lower cost implementations. In contrast, the DCE marketplace is much leaner, and there are fewer vendors who are actively developing and enhancing DCE products.

4 Concerns with Moving to CORBA

This section focuses on problems that may arise when using CORBA to support current Bellcore MediaVantage architecture and requirements.

4.1 Immaturity of Higher-level Services and Facilities

CORBA defines a number of higher-level services mentioned in Section 2.2 and a number of higher-level application-specific facilities. The higher-level services have been recently adopted, whereas the facilities are still being discussed

Category	Sub Category	CORBA	DCE
Programming Aspect	Model	Object-oriented	Procedural
	Interface inheritance	supported (including mult. inheritance)	not supported
	IDL design	interoperability between many languages	only for C (C++ minimal)
Communication Aspect	Model	object style	RPC style
	Idempotent ops	not supported	supported
	Component Identification	object references	unique ids (UUID)
	Dynamic Invocation	supported	not supported
	Deferred Synchronous	supported	not supported
	Repository (impl and i/f)	supported	not supported
	Interoperability	optional (UNO specs)	required
	contexts	opaque to runtime system	used solely by runtime system
Services	Infrastructure	no thread API	built in thread API
	security	external	built-in
Market Force	Vendor support	active, large	restricted
	Customer base	large	restricted

Table 1: Summary of Key Differences between DCE and CORBA

ORB	URL
Orbix, Iona Tecnologies	http://www-usa.iona.com/www/Orbix/index.html
CORBUS, BBN	http://www.bbn.com/offerings/corbus.html
ILU, Xerox PARC	ftp://ftp.parc.xerox.com/pub/ilu/ilu.html
ORBeline, Post Modern Computing	http://www.pomoco.com/orbinfo.html
PowerBroker, Expersoft	http://www.expersoft.com/home_pag.htm
CHORUS/COOL, Chorus	http://www.chorus.com/Products/Cool/index.html
HP ORB Plus, HP Labs	http://www.dmo.hp.com/cgi-bin/fe.pl/gsy/orbplus.html
DOME, Object Oriented Technologies	http://www.octacon.co.uk/onyx/external/oot.co.uk
NEO, Sun	http://www.sun.com/sunsoft/neo
ObjectBroker, DEC	http://www.dec.com/info/objectbroker
Electra, Cornell University	http://www.cs.cornell.edu/Info/People/maffeis/electra.html
SOM, IBM	http://www.software.ibm.com/objects/somobjects
DAIS, ICL	http://www.icl.com/dais
Fresco, Faslabs	http://www.faslabs.com/fresco/HomePage.html
OpenBase, Prism Technologies	http://www.prismtech.co.uk

Table 2: Available Vendor ORBs

by the various OMG domain-specific task groups. As a result, most implementations of CORBA either do not support these services and facilities, or provide minimal and possibly non-interoperable support.

In addition, it is important to recognize that the existing specifications of CORBA services have very weakly defined semantics. This makes it impossible to purchase non-trivial, plug-compatible services from multiple vendors and expect them to work together in a robust, efficient, and semantically meaningful manner.

4.2 Inter-ORB Portability and Interoperability

The Open Systems Foundation (OSF) supplied a reference implementation to vendors of DCE. To ensure interoperability, the only condition for vendor implementations of DCE was to achieve interoperability with the reference implementation. The Object Management Group (OMG) did not supply any reference implementation to the vendors. In addition, earlier versions of the CORBA specification were not precise enough to ensure interoperability. Instead, the OMG chose to let the vendors develop their ORB implementations and experiment with the new technology. Interoperability was added to CORBA version 2.0 through the Universal Networks Object (UNO) proposal [12].

4.3 Security

DCE defines a security model that provides:

- Protection levels for various levels of security;
- Authentication services;
- Authorization services;
- Data privacy and integrity services using cryptography and checksums.

CORBA also defines a security model that provides a number of services for ensuring security. The CORBA security model provides the following features:

- **Identification** and **authentication** – ability to identify and authenticate users and objects using the system;
- **Authorization** and **access control** – ability to provide authorization and access control to individual or groups of users and objects;
- **Security accounting** – ability to make users accountable for their security related actions;
- **Security of communications** – ability to provide secure communication over unreliable links. In addition, this also includes preserving the integrity of the data;
- **Security administration** – functionality to manage and administer the various security policies used.

This model supports the authentication and access control security requirements of the Bellcore products. This security model was adopted very recently, however, and is not yet available with currently available CORBA implementations.

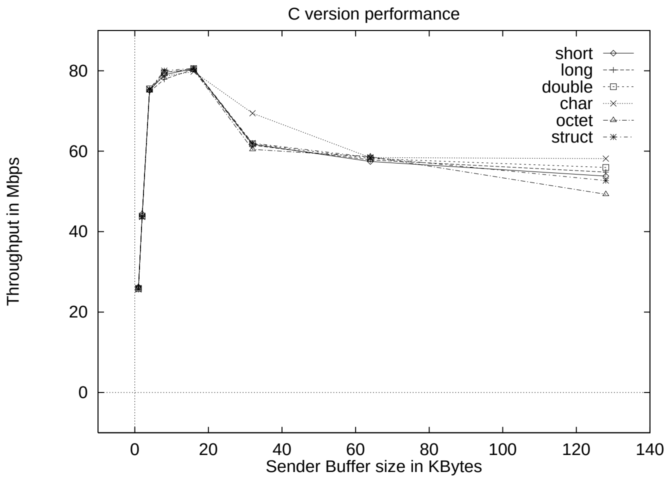

Figure 4: Blackbox Throughput Performance of Low-level UNIX Sockets using C version of TTCP

4.4 Scalability and Performance

Our studies measuring the performance of CORBA for throughput, latency and scalability [6, 7, 8] indicate that current implementations of CORBA suffer from the following sources of overhead that degrade their performance:

- Inefficient presentation layer conversions;
- An excessive number of data copying operations;
- Non-optimal choice of internal buffers that leads to discontinuous latency behavior;
- Inefficient choice of request demultiplexing strategies;
- Long chains of virtual function calls.

As a result, our performance measurements reveal that:

- Static stubs and the Dynamic invocations perform poorly compared to low level networking software (such as sockets);
- The DII performs worse than the static stubs;
- CORBA implementations were unable to scale to a large number of objects;
- Latency for invoking client-side requests increased with the number of objects handled by a server;
- Latency for sending different data types was different.

Figures 4, 5, and 6 depict the throughput performance of three versions of the `TTCP` benchmarking test suite - a low-level socket version implemented in C, IONA's ORBIX version, and Visigenic's VISIBROKER FOR C++ (formerly known as ORBELINE) version, respectively.

Figures 7 and 8 depict the the path a CORBA request takes through the client and server using the IDL compiler generated stubs for Orbix and ORBeline, respectively. The figures

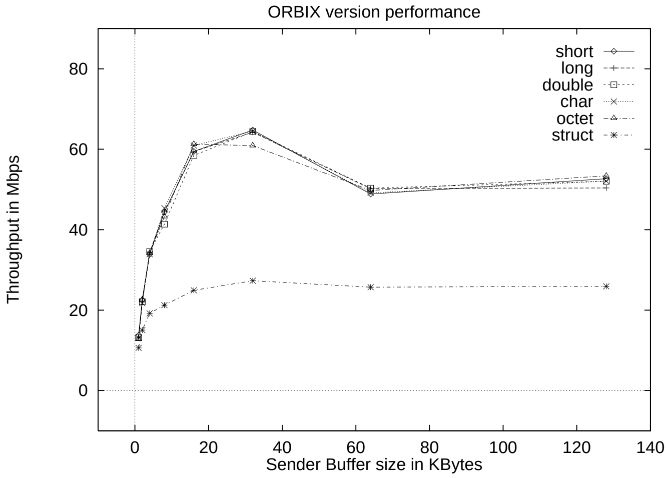

Figure 5: Blackbox Throughput Performance of Orbix version of TTCP

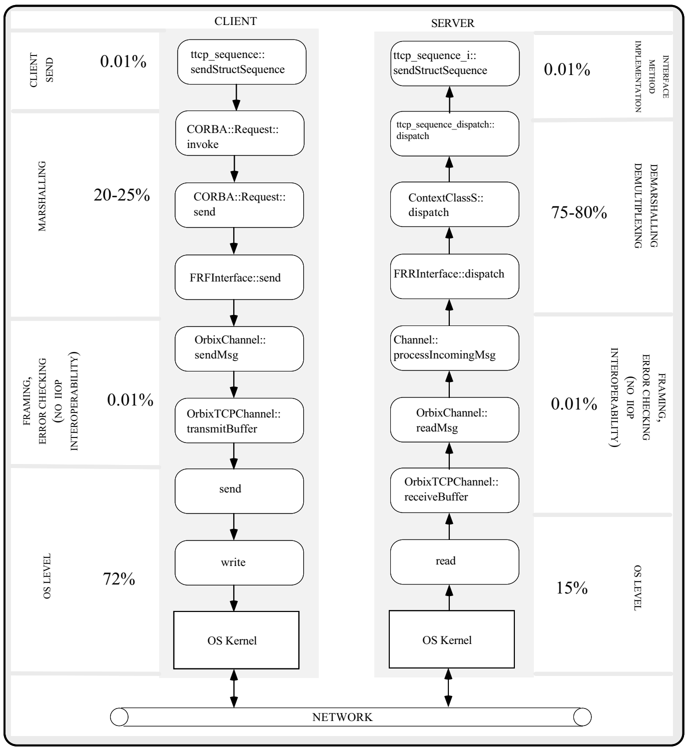

Figure 7: Whitebox Throughput Performance of Orbix version of TTCP

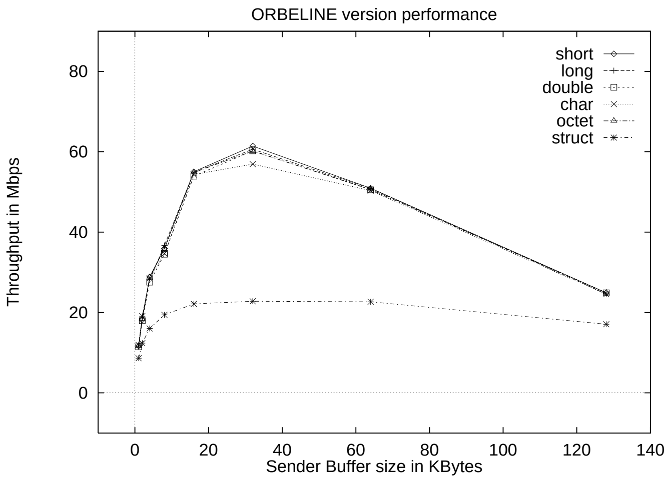

Figure 6: Blackbox Throughput Performance of ORBeline version of TTCP

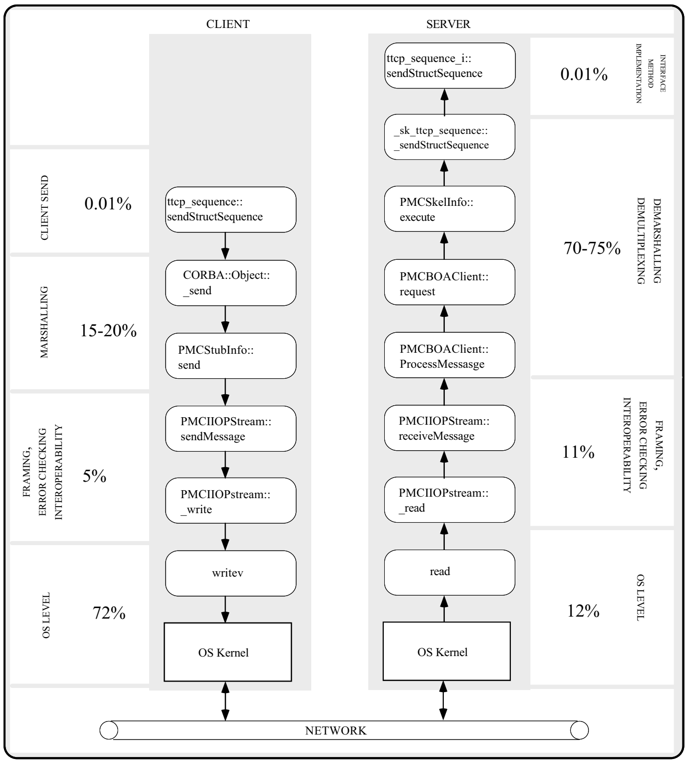

Figure 8: Whitebox Throughput Performance of ORBeline version of TTCP

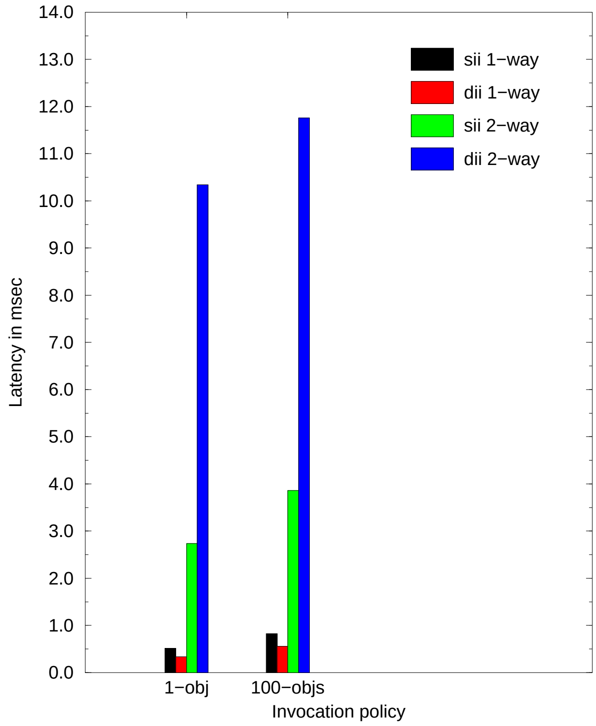

Figure 9: Blackbox Latency Performance of Orbix

also show how these two ORBs implement the generic request path shown in Figure 2. Percentages at the side of each figure indicate the contribution to the total processing time for a oneway call to the `sendStructSequence` method.[2]

Figures 9 and 10 depict the blackbox latency measurements for Orbix and ORBeline, respectively. These figures illustrate the latency for invoking the `sendNoParams_1way` method, which was used to perform a oneway operation containing no parameters. Figures 11 and 12 illustrate the corresponding whitebox latency measurements. These figures depict the path traced by the requests through the client and the server.

We are currently preparing to compare the performance of DCE and CORBA for a representative set of benchmarks over high-speed and low-speed networks. A subsequent version of this paper will present these results.

4.5 Management and Administration

Currently available DCE implementations are interoperable since they adhere to the reference implementation. In contrast, the original CORBA standard did not discuss interoper-

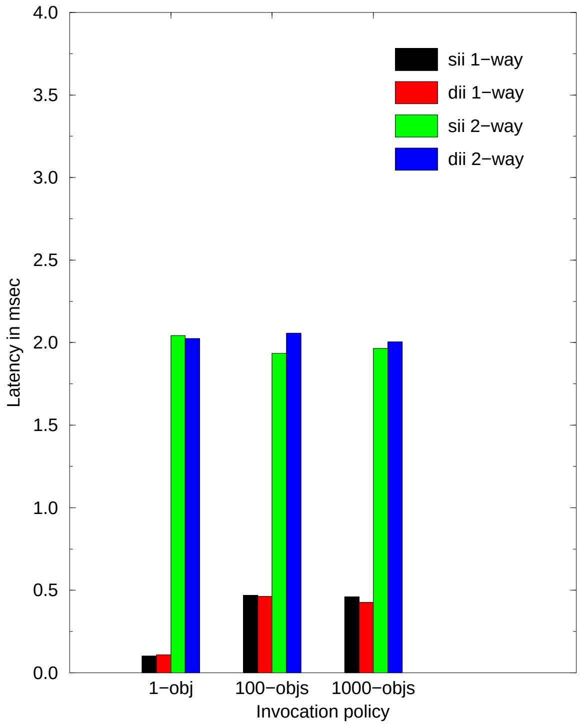

Figure 10: Blackbox Latency Performance of ORBeline

[2]The percentages may not add up to 100 since we have not shown the entire contribution of the OS and network device overhead.

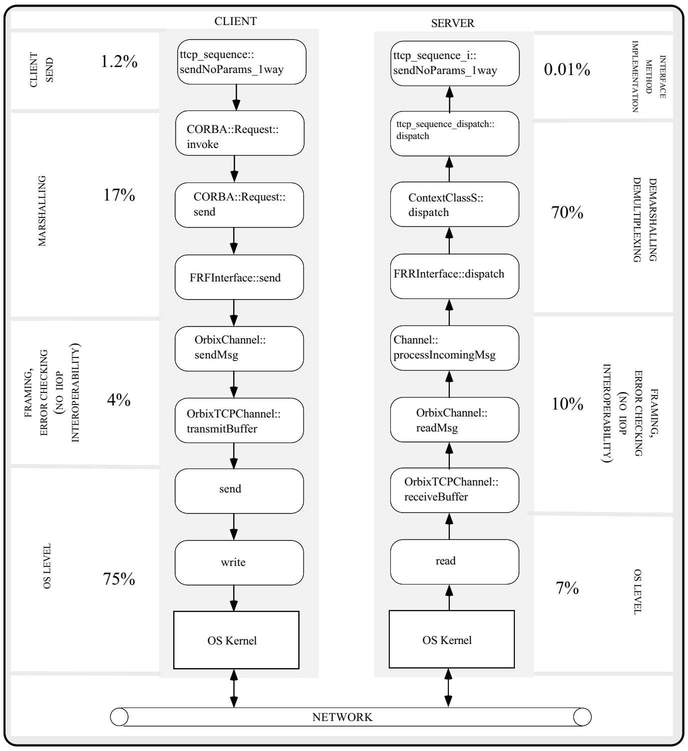

Figure 11: Whitebox Latency Performance of Orbix

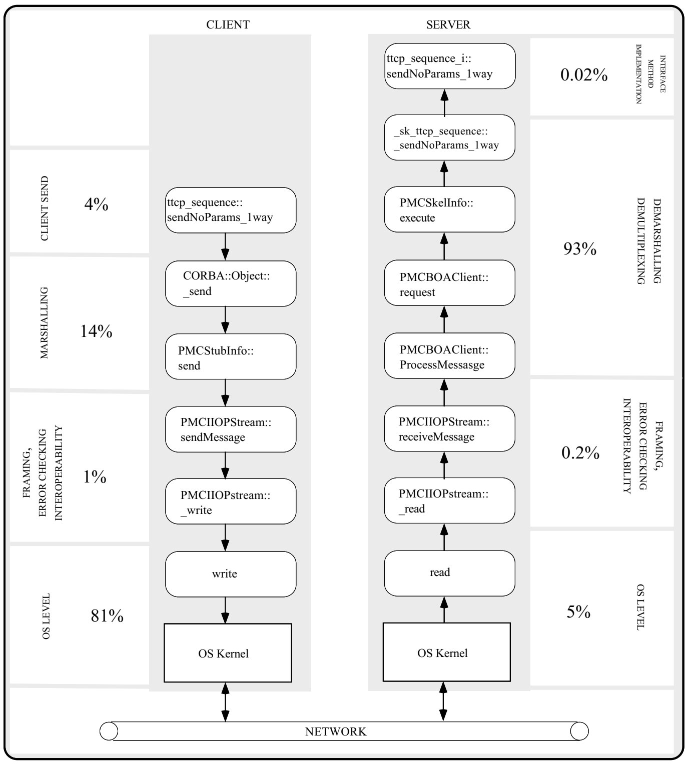

Figure 12: Whitebox Latency Performance of ORBeline

ability and the implementations thus lacked interoperability. Interoperability was adopted in the CORBA 2.0 standard. Since then, interoperability to a certain extent has been added to the implementations. In addition, the CORBA standard does not specify how to manage and monitor such heterogeneous CORBA systems. Due to this, it is difficult to manage and monitor these CORBA systems.

With emerging technologies such as TME 10, the task of managing CORBA systems will become easier. TME 10 is a new product from Tivoli Systems (http://www.tivoli.com) that has become a de-facto industry standard for management of large-scale distributed systems. The principal goals addressed by TME 10 are:

- *Heterogeneity and integration* – The ability to integrate the management of various elements such as networks, operating systems, hardware platforms, databases, applications, and middleware.
- *Scalability* – *i.e.,* able to manage a system containing a large number of servers, desktops, and other network resources;
- *Standardization* – a uniform means for managing distributed systems;

References

[1] Bellcore. *dceObjects Developer's Guide*, Bellcore Document BD-DCEO-DG-R140-001 edition, November 1995.

[2] Kenneth Birman and Robbert van Renesse. *Reliable Distributed Computing with the Isis Toolkit*. IEEE Computer Society Press, Los Alamitos, 1994.

[3] Thomas J. Brando. Comparing DCE and CORBA. Technical Report MP 95B-93, MITRE, March 1995. URL : http://www.mitre.org/research/domis/reports/DCEvCORBA.html.

[4] John Dilley. OODCE: A C++ Framework for the OSF Distributed Computing Environment. In *Proceedings of the Winter Usenix Conference*. USENIX Association, January 1995.

[5] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, 1995.

[6] Aniruddha Gokhale and Douglas C. Schmidt. Measuring the Performance of Communication Middleware on High-Speed Networks. In *Proceedings of SIGCOMM '96*, Stanford, CA, August 1996. ACM.

[7] Aniruddha Gokhale and Douglas 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.

[8] Aniruddha Gokhale and Douglas C. Schmidt. Evaluating Latency and Scalability of CORBA Over High-Speed ATM Networks. In *Submitted to the International Confernce on Distributed Computing Systems*, Baltimore, Maryland, May 1997. IEEE.

[9] William Harrison. *The Importance of Using Object References as Identifiers of Objects: Comparison of CORBA Object.* IBM, OMG Document 94-06-12 edition, June 1994.

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

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

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

[13] Sean W. O'Malley, Todd A. Proebsting, and Allen B. Montz. USC: A Universal Stub Compiler. In *Proceedings of the Symposium on Communications Architectures and Protocols (SIGCOMM)*, London, UK, August 1994.

[14] Michael L. Powell. *Objects, References, Identifiers, and Equality White Paper*. SunSoft, Inc., OMG Document 93-07-05 edition, July 1993.

[15] Douglas Schmidt and Steve Vinoski. Comparing Alternative Programming Techniques for Multi-threaded CORBA Servers. *C++ Report*, 8(7), July 1996.