Defense Information Systems Agency (DISA)

Joint Interoperability & Engineering Organization (JIEO)

Center for Computer Systems Engineering (JEXF)

DII COE Distributed Applications Series

Recommendations for Using DCE, DCOM, and CORBA Middleware

April 13, 1998

DISA/JIEO/JEXF Points of Contact:

POC:	**David Diskin**	diskind@ncr.disa.mil
or:	**Sherrie Chubin**	chubins@ncr.disa.mil

The MITRE Corporation
1820 Dolley Madison Boulevard
McLean, Virginia 22102-3481

MITRE Document ID: MITRE-DAS-C1

DII COE Distributed Applications Series

(Note: You are currently reading the shaded document)

Document Title	Contents	Purpose
A. Index Documents		
A1 – Guide to Using the DII COE Distributed Applications Series	Briefly describes the documents in this series and explains their applicability to defense system users.	Read this document if: • You want to know which documents in this series are most relevant to your needs.
B. Understanding Distributed Applications		
B1 – Issues in Building Effective Distributed Applications	Introduces basic issues and terminology for distributed applications. Emphasizes concepts such as total performance and scalability that are especially relevant to defense systems.	Read this document if: • You want an introduction to the overall concepts of distributed applications. • You want a perspective of distributed systems oriented more to DoD issues such as scalability.
B2 – New Perspectives on Distributed Architectures	A more in-depth look at underlying factors that must be dealt with when integrating older systems into new applications, and for obtaining greater flexibility and scalability in new systems.	Read this document if: • You will need to integrate older and non-standard systems into new distributed applications. • You need build or evaluate systems that need to scale up to many or very many components.
B3 – Java and Middleware	A brief look at the possible impact of the network programming language Java on middleware and distributed applications.	Read this document if: • You are concerned about the impact Java could have on selecting middleware and architectures. • You are interested in potential benefits of Java.
B4 – Integration and Use of Middleware Products in the DII COE	An overview of both current support for middleware technologies in the DII COE, and of likely future trends for use.	Read this document if: • You need to use the DII COE for distributed apps • You want to use middleware not yet in the COE
C. Selection and Use of Middleware for Building Distributed Applications		
C1 – Recommendations for Using DCE, DCOM, and CORBA Middleware	Quickly summarizes the key features, advantages, and disadvantages of using DCE, DCOM, or CORBA for distributed defense applications, and provides recommendations for how to use them.	Read this document if: • You need to decide whether to use one or more of these middleware technologies in your application. • You have legacy apps that use one of these three. • You want specific examples of middleware issues.
C2 – A Comparison of Three Middleware Technologies: DCE, DCOM, and CORBA	A detailed comparison of three COTS middleware technologies (DCE, DCOM, and CORBA) that are particularly likely to be needed or encountered when building distributed applications.	Read this document if: • You want a more in-depth understanding of the concept of middleware and its uses in defense systems. • You have legacy apps that use one of these three. • You need to select a middleware technology.

Document Title	Contents	Purpose
C3 – Guide for Building Multiple Middleware Applications	A detailed description of the problem of how to build distributed applications when use of a single middleware product cannot be guaranteed. It includes specifics on mixing of DCE, DCOM, and CORBA.	Read this document if: • You need both Unix and Windows NT platforms. • You already use DCE, DCOM, or CORBA. • You need a general strategy for dealing with more than one middleware product (e.g. due to legacy). • You need specifics on bridging between any of the DCE, DCOM, and CORBA middleware products.
D. Guidance for Distributed Application Architects		
D1 – Guide for Building Scalable Distributed Architectures	A brief description of how to identify scalability issues, oriented towards distributed application architects.	Read this document if: • You are working on a large or performance limited distributed application where scaling is critical.
D2 – Guide for Efficient Use of Object Technology in Distributed Applications	A brief description and summary of the advantages object-oriented distributed applications and migrating to them.	Read this document if: • You want a general understanding of the benefits of object technology for distributed systems. • You want to understand available object-oriented tradeoffs when using middleware technologies such as DCE, DCOM, and CORBA. • You need to migrate from non-object-oriented middleware to object-oriented middleware.
D3 – Predicting CORBA Performance Through Prototyping	Provides guidance on how to analyze the performance implications of building distributed applications with CORBA, or with other middleware products such as DCE or DCOM.	Read this document if: • You need to estimate the performance impact of using CORBA or other middleware products as part of a COE-based distributed application. • You want to understand what major factors impact performance in distributed applications.
D4 – DII COE Guidelines for Using CORBA, DCOM, and DCE	Provides specific guidelines for DII COE use of CORBA, DCOM, and DCE, with a primary focus on CORBA. Addresses features down to the IDL syntax level.	Read this document if: • You will be developing distributed DII COE applications using CORBA, DCOM, or DCE. • You need an open, interoperable design strategy.
E. Advanced Architecture Examples		
E1 – A Cellular Proxy Architecture for Battlespace Integration	Applies principles from the documents to the specific problem of distributed tactical systems.	Read this document if: • You are interested in examples of applying scaling and other concepts to a difficult environment.

Executive Summary

A *distributed application* is an application whose software components reside on more than one computer in a network, with the network typically composed of diverse computers and operating systems (a *heterogeneous* network). Middleware is software that simplifies the construction of distributed applications by providing standardized mechanisms that distributed components can use to communicate over a network. From both a historical and technological viewpoint, three of the most important middleware technologies are the Distributed Computing Environment (DCE), the Distributed Component Object Model (DCOM), and the Common Object Request Broker Architecture (CORBA).

Another increasingly important middleware technology is Java Remote Method Invocation (RMI). RMI was originally developed solely for use between distributed components written in the Java programming language, but its relevance to other more generic middleware technologies is growing due to more use of Java in distributed systems, and because there is a concerted effort underway to integrate RMI with CORBA. Java and RMI are mentioned briefly in this document as they relate to middleware, but for more detailed information on Java and RMI please see the index of Distributed Application Series documents at the front of this document.

DCE is best described as an influential early middleware product that subsequently fell behind in the marketplace due to its lack of standardized support for object-oriented languages. Although still a part of many legacy systems, DCE is unlikely to play a major role in the development of newer and (especially) Internet-based distributed systems. Many of the most innovative features of DCE have in the past few years been incorporated into other middleware products such as DCOM and CORBA, and DCE lacks the market support needed for its products to keep pace effectively with either DCOM or CORBA.

DCOM is a relatively new Microsoft-sponsored technology for Windows NT platforms that promises to play a major role in PC-based server systems when Window NT 5.0 begins to significantly penetrate that market in late 1998 or 1999. It is deployed in Windows NT 4.0 systems and is available for Windows 95 systems, but does not appear to be used extensively. The reason is that as of early 1998, the features of DCOM that deal with conveying and supporting service requests between remote components are still too immature to easily support development of large distributed systems. At present DCOM is more important not for its distribution features, but rather for its accompanying specification of how to interact with software components (objects) in a Windows system. This local-only aspect of the DCOM standard, called COM, is deeply embedded in Windows systems and provides an important interoperability target for distributed applications that need to incorporate both Windows and non-Windows computer systems into their networks.

As of early 1998, the middleware product that is the most widely deployed and actively used on Windows-based PCs is CORBA. In 1997 CORBA was distributed to millions of PCs as part of the popular Netscape Communicator browser, which uses a Java-compatible version of the CORBA-compliant Borland VisiBroker to provide communications to remote objects via the Internet. In addition to this de facto standardization of CORBA as the mostly widely used and

readily accessible middleware product for PCs, the CORBA language for describing interfaces to software components is now an international standard (ISO/IEC DIS 14750). Membership in the CORBA sponsoring organization (the Object Management Group, or OMG) includes about 780 vendors, developers, and user organizations. The international standardization of the CORBA interface language makes CORBA attractive for Internet based applications, and the large size of the OMG means that CORBA products can be found on most computer systems.

CORBA is also unique because it is being integrated at the underlying protocol levels both with the Internet and with Java, a language designed specifically for creating distributed applications. Integration of CORBA with the Internet allows developers to use the existing Internet infrastructure to build efficient distributed applications faster and more flexibly. Integration of CORBA with Java makes it significantly easier for developers to use the simpler and more network-oriented features of Java to integrate legacy software into new distributed applications.

As of early 1998, general recommendations for using these three middleware technologies are:

- **Use Distributed Computing Environment (DCE) only for legacy software.** From a market perspective, DCE has been significantly weakened by its belated and incomplete support for object-oriented languages. With object-oriented languages continuing to grow in importance in both commercial applications and on the Internet, this has left DCE in a poor situation compared to both CORBA and DCOM. DCE thus is a poor choice for a middleware technology except when it is already being used in a legacy system. DCE is not an appropriate choice for building entirely new distributed systems.

- **Use CORBA.** The *de facto* position of CORBA as the most widely distributed and used middleware product in PCs (via Netscape Communicator) makes it an excellent choice for use in low-end Windows platforms. Furthermore, the broad availability, open standards, support for a wide range of platforms and programming languages, and mature support for object-oriented software make CORBA an especially good choice for integrating diverse types of systems. Other factors that make CORBA a strong choice are its accompanying long-term architecture (the OMA) for defining and integrating supporting services, and its ongoing integration with Java (see below).

- **Use only one CORBA vendor unless interoperability can be verified.** The greatest current weakness of CORBA is the slow pace of its efforts to make CORBA products from different vendors interoperate with each other. There has been significant progress in this area in the last couple of years, but at present the safest strategy for using CORBA is still to pick a single vendor and use that vendor consistently for a given application.

- **Use integrated CORBA/Internet/Java products whenever possible.** One particularly promising middleware technology trend is the ongoing integration of CORBA, the Internet, and Java. An explicit example of this trend is the Netscape Communicator browser and its bundled VisiBroker Java software. The integration of these three technologies is centered on a CORBA message protocol (that is, a well-defined style of exchanging messages) known as the IIOP, or Internet Inter-ORB Protocol. The IIOP is a simple but flexible CORBA protocol that allows objects written in many different programming languages to communicate directly with each other over the Internet. As of early 1998, both the design and marketing position of IIOP make it the best candidate for becoming a universal protocol for linking diverse types of objects over an Internet-style network.

- **Use middleware bridges into COM.** DCOM is not at present a major factor in building distributed systems, but the closely associated but local-only COM technology is already widely used in Windows 95 and NT to define the interfaces to many types of software objects. Middleware vendors who provide effective bridges to COM thus also provide valuable access to software components available in low-cost PC systems, and also minimize the possible future impact of more robust versions of DCOM.

As of early 1998, associated general predictions of events that could impact use of these three middleware technologies are:

- **Expect impacts from DCOM when Windows NT 5.0 is released.** Microsoft is strongly committed to DCOM and has already announced new initiatives that should make it more powerful and useful on both its native Windows systems and on other operating systems. The most important event for DCOM will be the full commercial release of Windows NT 5.0 in late 1998 or 1999. NT 5.0 will include an advanced set of middleware-oriented services (called Active Directory) that should make DCOM much more powerful and easier to use. Overall, it is still too early to tell how such future versions of DCOM will compete with the growing CORBA/Internet/Java collection of multi-vendor middleware efforts.

- **Expect to see more use of Java in distributed applications.** Java provides features that make applications more portable and more scalable, particularly when it is accessed though the IIOP. Java is currently the best overall candidate for "mobile objects" that can be flexibly reassigned to new platforms to increase efficiency during everyday use of a distributed application. Due to the many market, technical, and legal factors that are affecting Java, it is difficult to predict exactly how large a role Java will play in the future of middleware, although a fairly major role looks fairly well assured as of early 1998.

- **Expect to see continued expansion in the use of scripting languages.** Contrary to the minimalist implications of the term "scripting," scripting languages such as Perl, Visual Basic, Javascript, Python, and tcl/Tk are actually among the most powerful programming languages in existence. [1] They are particularly useful for integrating legacy software, since much of their power comes from an open and highly accommodating programming style that does not require legacy components to meet the internal programming conventions of new software. In terms of their relationship to middleware technologies such as CORBA, scripting languages are best understood as complementary technologies that can be used

[1] "Scripting: Higher-Level Programming for the 21st Century" (March 1998) , by John K. Ousterhout, *IEEE Computer*, Vol. 31, No. 3, March 1998, pp. 23-30. Public abstract at: http://computer.org/computer/co1998/r3023abs.htm

to bring legacy software into a middleware framework rapidly and easily. For example, Perl or Python can be used to convert the inputs and outputs of a legacy system into IIOP messages, so that the resulting combination looks like a CORBA-compliant server.

- **Expect to see XML used for sharing middleware data.** XML is a new World Wide Web Consortium (W3C) standard for representing complex data using human-readable labels and structuring.[1] It is relevant to middleware because it provides a convenient and well-standardized way to present, transport, and preserve complex data objects without "attaching" non-portable software methods to them. For example, a software object written in Java can present all or part of its internal data structure as a language-independent XML object that can then be stored in a database, or directly interpreted by components written in languages such as C++. In a heterogeneous distributed application, XML thus can be used to make it easier to store and share complex data across diverse types of platforms and components.

- **Expect change.** The Internet has drastically increased the rate at which new technologies impact the market place. Approaches that emphasize open, well-standardized message protocols (e.g., the CORBA IIOP and the recent XML standard) are generally the best choices in such situations, as opposed to selections of specific tools or functions.

[1] "Extensible Markup Language (XML)" (W3C, March 1998), http://www.w3.org/XML/

Document History

Release	**Release Date**		**Authors and Reviewers**
1.6	1998-04-13	*Authors:*	Terry Bollinger (MITRE) David Diskin (DISA) Sherrie Chubin (DISA)
		Reviewers:	David Diskin (DISA) Sherrie Chubin (DISA) Roger Duncan (MITRE)

Table of Contents

List of Figures

List of Tables

1. Introduction: Key Features of Middleware Technologies

All middleware technologies share certain features, since they all deal with the same problem of how to make it easier to construct and integrate distributed applications. This introduction provides a brief overview of several key features and concepts that are useful for understanding and comparing middleware products.

1.1 Distributed Applications and Middleware

A *distributed application* is an application whose software components reside on more than one computer in a network, with the network typically composed of diverse computers and operating systems (a *heterogeneous network*). *Middleware* is software that simplifies the construction of distributed applications by providing standardized mechanisms that distributed components can use to communicate over a network. From both a historical and technological viewpoint, three of the most important middleware technologies are the *Distributed Computing Environment (DCE)*, the *Distributed Component Object Model (DCOM)*, and the *Common Object Request Broker Architecture (CORBA).*

1.2 Interface Definition Languages (IDLs)

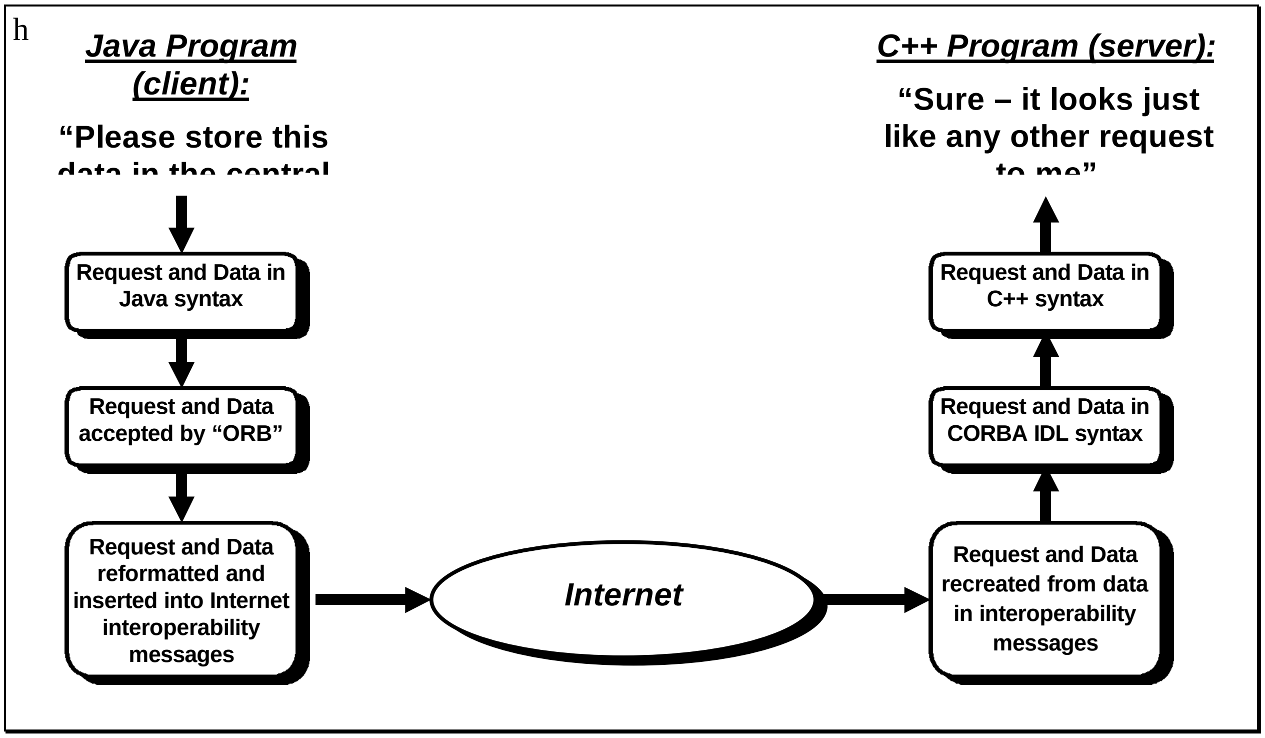
An *interface definition language* is just what it sounds like: a computer language for describing the externally accessible operations of a software component. IDLs look like abbreviated programming languages, ones that only provide enough features to permit a programmer to name operations (that is, programming procedures, functions, or methods), and to describe what kinds of parameters need to be sent and returned when an operation is used. Thus an IDL is valuable not because it lets programmers do anything new, but because it provides a *standardized* way of defining the services that are available from a software component. Once such a standardized interface has been defined for a software component, that component can in principle be called or used by any other component on any other platform, even if the other component was written in a different programming language.

A second and subtler advantage of an IDL is that it provides a consistent way to hide diverse communication methods from software components that use them. With an IDL interface into the middleware, application components can be largely indifferent as to whether another component is located on the same or a different platform, even though the underlying communication mechanisms are very different for local and remote access.

1.3 Interoperability Protocols

An IDL defines common interfaces for components, but it says nothing about how requests and responses to those interfaces will be conveyed as messages across a network. This conveying of service requests and responses is the second basic function of a middleware technology, and it is accomplished by defining an *interoperability protocol* that specifies what types of messages will be exchanged and how their contents should be interpreted. The complexity of a middleware interoperability protocol is directly related to the complexity of its IDL, since an IDL that allows complex data structures to be defined will need to be supported by an interoperability protocol that "understands" how to convert such data structures to and from network messages. The overall process of converting data structures into messages is called *marshaling,* and the recovery of the data from such messages is called (not too surprisingly) *unmarshaling.* An example of the overall process of using a CORBA interoperability protocol on the Internet is shown in Figure 1.

Figure 1. Java to C++ Via an Internet Interoperability Protocol


h
Java Program (client):
"Please store this
C++ Program (server):
"Sure – it looks just like any other request
Request and Data in Java syntax
Request and Data accepted by "ORB"
Request and Data reformatted and inserted into Internet interoperability messages
Internet
Request and Data recreated from data in interoperability messages
Request and Data in CORBA IDL syntax
Request and Data in C++ syntax

1.4 Message Brokers

In addition to showing how an interoperability protocol is used to transfer requests and data across a network, Figure 1 also introduces the important question of what, exactly, is responsible for providing the address information needed to send a message across a network. The simplest approach to this problem is a "direct mail" strategy in which each component knows the full address of every component to which it needs to send a message. This approach is closely analogous to the way ordinary postal mail works, since anyone who wants to mail a letter must first know the physical address of the intended recipient.

A direct-mail strategy has the advantage of being efficient, since every message goes directly to the intended recipient without having to pass through any intermediate steps. However, it also has important disadvantages. At the network level, it is inflexible because individual components remain locked to an old address even when the receiving component has moved to a new location. At the individual component level, a direct-mail approach is burdensome because it requires each component to store and maintain all the addresses it needs. This is wasteful, since many components use the same addresses and thus could share them instead of keeping separate copies of them. A direct-mail approach is even more burdensome for legacy components that lack features for storing or using network addresses. Trying to force legacy components to understand network addresses may require major code re-writes, and thus can significantly limit the range of components available for integration into new distributed applications.

What is needed to resolve these difficulties is a new software entity that serves as a director or dispatcher of messages – that is, as a *message broker*. (In the case of object-oriented middleware, such message brokers are often called *Object Request Brokers*, or *ORBs*.) The idea of a message broker is to allow simple descriptive names to be used as the addresses of components. The message broker translates those names into network addresses without the senders or receivers ever knowing each other's real network locations. The result would be the postal-mail equivalent of being able to address a letter by simply writing a person's name on the envelope, and then letting the post office do the work of looking up the physical address of that person.

It should be noted that except in a few unusual cases, it is neither practical nor desirable for a distributed application to use a "pure" message broker model in which every message is routed through the address-lookup portion of the broker. Since distributed components often send hundreds or thousands of messages to the same recipients, repeated lookups in such cases could be both very slow and wasteful of resources. A well-designed message broker instead provides a fast local storage (called a *cache*) for keeping recently retrieved and other relevant network addresses. The addresses kept in this fast local storage can then be used to "direct mail" all subsequent messages to the same remote components. In a good broker design this switch between explicit remote lookup of an address and local use of a cache can be made transparent, so that from the perspective of a software component its requests are all handled in the same way.

Historically, the concept of a message broker and recognition of its importance has come about rather slowly in middleware. This may be due in part to the simplicity and conceptual appeal of direct-mail messaging models, which correspond to the way people use everyday messaging services such as telephones and postal mail. However, in the long term the message broker concept provides a number of fundamental benefits. At the design level it provides a good way to separate the complexity of the network from the functionality of a distributed application, and at the operations level it enables more reliable and scalable performance by making it easier to reconfigure an application in response to increased network loads or isolated network failures. In both cases the message broker concept significantly increases the range of possible uses of middleware and distributed applications.

1.5 Supporting Services

Once a middleware technology has its IDL and interoperability protocol in place, it can use those features to provide a range of *supporting services* that make it easier to build distributed applications. A common example would be a name server, which is a service that helps distributed components locate each other on a network. Supporting services are not absolutely necessary for middleware, but in practice good supporting services can make a very substantial difference in how easy it is to develop flexible, extensible distributed applications.

1.6 Tool and Componentware Support

As with supporting services, supporting tools and componentware (that is, prefabricated components ready for incorporation into new applications) are not absolutely necessary in a middleware technology, but they can make a substantial difference in how easy it is to develop new distributed applications. The most powerful tools and componentware are ones that make the process of distributing application components across a network as “transparent” as possible. That is, they hide as much of the complexity of the middleware itself as possible, so designers can keep their focus and efforts on building application functionality.

1.7 Support for Objects in the IDL

An object is a resource (e.g., a data structure) that can only be accessed by using a previously defined set of operations (procedures or functions). This explicit association of resources and operations makes it easier to control and protect resources, and also forces developers to put more thought up front into how to create a set of operations that makes access to the resource fast and convenient. From a design perspective, this kind of association of resources and operations can be done using any programming language. However, only *object-oriented* languages support explicit association of resources and operations at the level of the source code, so that correctly associated use of resources and operations can be verified at compilation. In addition to this basic capability to associate resources and operations explicitly, most object-oriented languages also provide a variety of mechanisms that make it easier to create new objects and new types of objects. Examples of these mechanisms include *object classes* for creating new objects of a specific type, and *inheritance* for defining new object classes as more specific versions of other, more generic object classes.

Object-oriented programming languages such as C++ and Java have become quite popular in recent years, and for this reason alone it is important that middleware products support object-oriented software components. However, object-oriented technologies also provide a direct benefit to distributed applications by making resources more readily available on a network. The operations of the resource object can simply be “published” or made known to other platforms in the network, which can then use the middleware to access that resource. On the other hand, if there is no well-defined set of operations there will be a tendency to treat each new request to use the resource as a custom programming task. Such a case-by-case approach is costly, inflexible, and in the long term unreliable, since it results in a proliferation of different ways to access the same resource. Defining resources as objects avoids such problems and helps ensure consistency.

Ideally, the IDL of a middleware product should support the declaration of objects in much the same way that object-oriented programming language does. An IDL that does not provide explicit object specification can still be used to convey service requests between object-oriented components, but it requires additional custom programming and the resulting application is less likely to be portable.

2. Annotated Descriptions of DCE, DCOM, and CORBA

This section provides annotated overviews of the three middleware technologies DCE, DCOM, and CORBA. The annotations are in the form of hyperlinked footnote references that provide Internet references to many of the more important documents from which this comparison was developed. If you are reading this document in a version of Word that supports Internet browsing, you can view any referenced document by simply double-clicking on the blue underlined portion of the footnote. If your copy of Word does not support Internet browsing, or if you are using a printed copy of this document, the Internet address (called a Universal Resource Locator, or URL) of each reference is also provided. To use the URL reference, copy or type it into the URL field of your Internet browser.

2.1 DCE – Distributed Computing Environment

Historically, DCE[1] was the first major multi-vendor middleware technology. DCE is defined and promoted by The Open Group[2], previously known as the Open Systems Foundation (OSF). A good source of introductory and general information about DCE is the DCE Frequently Asked Questions (FAQ) page,[3] which is referenced on The Open Group home page and maintained by Jon Mauney (jon@mauney.com). DCE has had significant competition from CORBA and DCOM in recent years, but it is still widely used in many systems and platforms.[4] DCE has good security features based on Kerberos,[5] and in the past DCE products on different platforms and from different vendors have generally shown better compatibility and interoperability than have comparable CORBA products.[6] DCE was supposed to become object-oriented in early 1996 with the release of the DCE 1.2.1 standard, which specifies an object interface for the C++ language (only).[7] In practice, however, DCE vendor support for C++ and other object language interfaces has been fragmented and incomplete. As a result, object-oriented developers using DCE are forced either to choose one of several incompatible object-oriented extensions to DCE, or to

[1] DCE FAQ Question 1.01: What is DCE? (August 1997),
http://www.camb.opengroup.org/tech/dce/info/faq-mauney.html#Q1_01

[2] The Open Group home page (March 1998) , http://www.opengroup.org/

[3] DCE Frequently Asked Questions (August 1997) ,
http://www.camb.opengroup.org/tech/dce/info/faq-mauney.html

[4] DCE FAQ Question 1.03: What platforms support DCE? (August 1997) ,
http://www.camb.opengroup.org/tech/dce/info/faq-mauney.html#Q1_03

[5] "Kerberos User's Guide SG-2409 9.0" (December 1997) ,
http://www-lc.llnl.gov:8080/library/all/SG-2409

[6] DCE FAQ Question 1.08: What is the relationship between DCE and CORBA? (August 1997),
http://www.camb.opengroup.org/tech/dce/info/faq-mauney.html#Q1_08

[7] OSF DCE 1.2.1 New Features (November 1995),
http://www.camb.opengroup.org/tech/dce/info/papers/osf-dce-ds-1195.htm

create their own custom (and thus incompatible) object interface extensions to DCE.[1] For distributed applications written in object-oriented languages such as C++, this makes DCE complex to use and greatly reduces the portability and interoperability of the resulting applications.

2.2 DCOM – Distributed Component Object Model

DCOM[2] is the Microsoft response to DCE[3] and CORBA[4] middleware technologies. Rather than being viewed as an entirely new middleware technology, DCOM is perhaps best understood as an application of DCE technology (see above) to the problem of how to distribute Microsoft document components over a network.[5] Microsoft document components and their associated GUI controls (e.g., ActiveX[6] controls) are described internally to Windows systems by using a proprietary, object-oriented Microsoft technology called Component Object Model (COM).[7] [8] DCOM (Distributed COM) extends the range of COM technology by transparently (that is, without requiring any changes to the components) conveying requests to components on other platforms through use of the underlying DCE communication protocols.[9] Since COM is object-oriented, DCOM is object-oriented in the same sense as other object-oriented extensions[10] of DCE. However, the COM object interfaces used by DCOM are generally more complex to create and read than those of CORBA[11] [12], and as a result are more likely to be generated by commercial

[1] DCE FAQ Question 2c-02: Can I use DCE from C++? (August 1997) , http://www.camb.opengroup.org/tech/dce/info/faq-mauney.html#Q2_20

[2] DCOM (January 1998), http://www.microsoft.com/com/dcom.htm

[3] DCE FAQ Question 1.01: What is DCE? (August 1997), http://www.camb.opengroup.org/tech/dce/info/faq-mauney.html#Q1_01

[4] "What IS CORBA????" (OMG, October 1997), http://www.omg.org/about/wicorba.htm

[5] "Microsoft Complies With DCE – For Now" (*Information Week*, 29 July 1996) , http://www.techweb.com/se/directlink.cgi?IWK19960729S0051

[6] ActiveX™ Controls (January 1998), http://www.microsoft.com/com/activex.htm

[7] COM Technologies home page (March 1998) , http://www.microsoft.com/intdev/com/

[8] COM versus CORBA: A Decision Framework (April 1998) , "http://www.quoininc.com/quoininc/COMCORBA.html#COM versus CORBA: A Decision Framework"

[9] "DCOM Technical Overview" (Microsoft, November 1997) , http://www.microsoft.com/ntserver/library/dcomtec.exe

[10] DCE FAQ Question 2c-02: Can I use DCE from C++? (August 1997) , http://www.camb.opengroup.org/tech/dce/info/faq-mauney.html#Q2_20

[11] "DCOM and CORBA Side by Side, Step by Step, and Layer by Layer " (September 1997), http://www.cs.wustl.edu/~schmidt/submit/Paper.html

[12] "From CPP to COM" (Microsoft, October 1995), http://premium.microsoft.com/msdn/library/techart/html/cpptocom.htm

tools from Microsoft or other vendors than coded manually.[1] Although convenient for developers who are using such tools, this approach can complicate the use of a technology such as DCOM if DCOM is not fully integrated with the tools that create the COM interfaces, or if DCOM is used on platforms that do not support comparable generation of COM interfaces.

To understand the role of DCOM in the middleware market it is important to make a clear distinction between COM and DCOM. COM is a mature and broadly deployed document and component structuring technology used in essentially all Windows 95 and NT platforms,[2] whereas DCOM is an immature middleware technology[3] with a web site that as of early 1998 provides only one unambiguous example of its use in a commercial application.[4] In such a situation there are strong marketing incentives for emphasizing the success of the target COM technology while skipping somewhat lightly over the status and success of DCOM *per se*. As a result, many technical descriptions of DCOM often tend to end up being more about the strength and maturity of COM,[5] than the status of DCOM. Such presentations can blur important issues, however, since COM as it currently exists is not a middleware technology at all, but rather an interface technology that can in principle be used by any middleware technology to gain access to software components on local Windows platform.[6] The real advantage that DCOM gains from its close ties to COM is an "inside track" on the data needed to build an effective interface to such COM components. Other middleware technologies such as CORBA are dependent on potentially incomplete or ambiguous published specifications of COM interfaces, and thus may not be able to track changes to COM standards as quickly as DCOM. This advantage does not appear to have had much impact on DCOM as of early 1998, but it is a factor that clearly needs to be taken into account when looking at the long-term potential of DCOM as a middleware product.

As of March 1998 it is still difficult to find specific examples of commercial use of DCOM. The Microsoft web site that promotes DCOM[7] includes a document with three examples of DCOM-based architectures, but one of the examples is labeled as hypothetical and the other two having no specifics on the companies that developed them or their current implementation status.[8] The

[1] "COM+: Building on the Success of the Component Object Model" (Microsoft, November 1997), http://www.microsoft.com/com/slides/complus.zip

[2] COM: Component Object Model (Microsoft, February 1998) , http://www.microsoft.com/com/

[3] "Real-Time DCOM's Immediate Future Now Uncertain" (Jensen, November 1997) , http://www.realtime-os.com/noteworthy/rt-dcom.html

[4] "DCOM Cariplo Home-Banking Case Study" (Microsoft, Nov 1997), http://www.microsoft.com/com/wpaper/dcomhome.zip

[5] "DCOM Business Case" (Microsoft, March 1998), http://www.microsoft.com/ntserver/guide/dcom.asp

[6] IONA CORBA-COM Bridge announcement: OrbixCOMet™ Desktop (DDDD) , http://www.iona.com/news/pressroom/msoft.html

[7] DCOM (January 1998), http://www.microsoft.com/com/dcom.htm

[8] "DCOM Solutions in Action" (November 1997) , http://www.microsoft.com/com/wpaper/dcomsol.zip

Microsoft DCOM site also provides one detailed, web-verifiable, company-specific case study[1] of a home banking system in Italy[2] that uses DCOM. In contrast, as of March 1998 the CORBA web site[3] lists 53 specific examples of companies with CORBA distributed application success stories.[4] In terms of vendor support, two vendors currently support DCOM:[5] Microsoft,[6] and more recently Software AG.[7] Microsoft provides all tool and support work for DCOM on Windows 95 and NT, while Software AG develops products for non-Windows operating systems such as Unix and OS/390.[8] Microsoft clearly views the DCOM porting activity of Software AG as an important component of their long-term ability to make DCOM into a full middleware product that can interoperate with other non-Windows operating systems. For example, Microsoft includes Software AG products as part for their long-term strategy (called COM+)[9] for making COM easier to use and more interoperable with other operating systems.

DCOM is nominally an open standard by virtue of a quasi-association[10] with the DCE standards of The Open Group.[11] However, for all practical purposes the definition and future direction of DCOM are best understood as being one of a group of interrelated technologies that are all controlled by the Microsoft Corporation.[12] These COM-related technologies[13] include DCOM itself, COM+,[14] the Microsoft Transaction Server (MTS);[15] and ActiveX™ Controls.[1] The fact

---

[1] “DCOM Cariplo Home-Banking Case Study” (Microsoft, Nov 1997), http://www.microsoft.com/com/wpaper/dcomhome.zip

[2] Home Banking web site (Cariplo Bank in Italy) implemented using DCOM technology (October 1997) , http://www.cariplo.it/HomeBanking.htm

[3] OMG (CORBA) Home Page (February 1998) , http://www.omg.org/

[4] CORBA Success Stories (November 1997) , http://www.corba.org/index.html

[5] “DCOM Business Case” (Microsoft, March 1998), http://www.microsoft.com/ntserver/guide/dcom.asp

[6] Microsoft home page (March 1998) , http://www.microsoft.com/

[7] Software AG home page (March 1998) , http://www.softwareag.com/corporat/default.htm

[8] “Software AG's Latest DCOM Runs On IBM OS/390” (*Computer Reseller News*, November 1997), http://www.techweb.com/se/directlink.cgi?CRN19971103S0197

[9] “COM+: Building on the Success of the Component Object Model” (October 1997), http://www.microsoft.com/com/slides/complus.zip

[10] “Real-Time DCOM's Immediate Future Now Uncertain” (Jensen, November 1997) , http://www.realtime-os.com/noteworthy/rt-dcom.html

[11] DCOM-DCE meeting minutes (November 1996) , http://www.opengroup.org/public/tech/dce/nov96/OOminutes.html#dcom

[12] “Microsoft Complies With DCE – For Now” (*Information Week*, 29 July 1996) , http://www.techweb.com/se/directlink.cgi?IWK19960729S0051

[13] COM Technologies home page (March 1998) , http://www.microsoft.com/intdev/com/

[14] COM+ (November 1997), http://www.microsoft.com/com/complus.htm

[15] MTS (March 1998), http://www.microsoft.com/com/mts.htm

that DCOM uses a slight variation of the DCE protocol for conveying messages between remote components[2] means that it is possible to communicate between DCE and DCOM components without complex message conversions. DCOM supports object-oriented languages, although as of early 1998 the range of object-oriented languages supported by DCOM (C++ and Java) was considerably smaller than that of CORBA (C++, Java, Smalltalk, Ada95, Objective Cobol, and others).

2.3 CORBA – Common Object Request Broker Architecture

CORBA[3] is a mature, object-oriented middleware technology defined by the Object Management Group, or OMG.[4] As of early 1998 the OMG had a total membership of over 780 vendors, developers, and end users.[5] CORBA was created after DCE but well before DCOM, and it benefits from a number of "lessons learned" from DCE. In contrast to DCE, CORBA has been object-oriented since its inception, and in contrast to DCOM, it has always been platform and language independent. These characteristics make CORBA particularly well-suited for rapidly integrating diverse legacy software and systems to build new distributed applications.[6] Notable features of CORBA include: support of and interoperability across a wide range of computing platforms, and programming languages, and CORBA products; [7] support by hundreds of vendors, developers, and users; compatibility with object-oriented approaches and languages; selection of its Interface Description Language (IDL) by the International Standards Organization (ISO) as a "universal" language for describing interfaces to software components[8] (see ISO/IEC DIS 14750[9]); a central role in the Object Management Architecture, or OMA,[10] which is the exceptionally broad and long-term OMG vision of for distributed services and how to promote integration and interoperability among distributed systems; broad use and acceptance by the Unix community; and the recent (1997) distribution of client portions of CORBA technology (the

[1] ActiveX™ Controls (January 1998), http://www.microsoft.com/com/activex.htm

[2] DCE FAQ Question 2c-01: Will Windows NT communicate with DCE? (August 1997), http://www.camb.opengroup.org/tech/dce/info/faq-mauney.html#Q2_19

[3] Object Resource Lists (by author of "The CORBA Reference Guide") (Alan Pope, February 1998), http://www.qds.com/people/apope/Corba/ap_resources.html

[4] OMG Home Page (February 1998) , http://www.omg.org/

[5] OMG Member Company Listings (March 1998), http://www.omg.org/cgi-bin/memlist.pl

[6] NDF Tracker97 Project (February 1998), http://www.corba.org/gov.htm#usdsfg

[7] The ORB Interoperability Showcase (OMG, June 1997), http://corbanet.dstc.edu.au/

[8] "OMG pushes to become standards body, touts CORBA" (Computer Reseller News, July 1997), http://www.techweb.com/se/directlink.cgi?CRN19970728S0027

[9] ISO/IEC DIS 14750 (March 1998), http://www.iso.ch/cate/d25486.html

[10] "OMA Executive Overview" (OMG, October 1997), http://www.omg.org/about/omaov.htm

Borland VisiBroker[1]) to over 30 million Windows and PC users of the popular Netscape Communicator Internet browser.[2] The OMG web site gives examples of how 53 companies with highly diverse applications areas are successfully using CORBA technology in both new and legacy integration applications.[3]

Due in large part to the acceptance of its IDL as an international standard, CORBA has become a *de facto* rallying point for interoperability technologies being spun off by the Internet. Recent examples of the impact of CORBA include:

- The ongoing merger of Java and CORBA standards for distributed application.[4] [5] [6]
- The adoption of CORBA IDL as the programming interface to XML,[7] which is a much more extensible replacement for Internet HTML that promises to make structured data much more readily available on the Internet.
- The increasing trend towards use of CORBA in transaction-based systems that emphasize reliable database access.[8] [9] [10]
- Support for CORBA from other middleware organizations such as The Open Group, which defines and promotes DCE middleware standards. Even though it is nominally "competing" with the OMG on middleware standards, The Open Group has initiated a

---

[1] "Borland's VisiBroker ORB Surpasses 30 Million Licenses Deployed Worldwide" (March 1998), http://www.borland.com/visibroker/press/1998/visi30m.html

[2] Netscape press release on 68 million users of Commu Marketplace (December 1997), http://www.netscape.com/newsref/pr/newsrelease538.html

[3] "CORBA Success Stories" (OMG, Novermber 1997) , http://www.corba.org/index.html

[4] Java, RMI and CORBA (OMG, June 1997), http://www.omg.org/news/wpjava.htm

[5] "JavaSoft concedes it's not an all-Java world" (Computer Reseller News, July 1997), http://www.techweb.com/se/directlink.cgi?CRN19970721S0060

[6] Review of book "Instant CORBA" (Orfali & Harkey, February 1998), http://www.micromail.com/titles/7666.html

[7] Document Object Model (Core) Level 1 (W3C, December 1997), http://www.w3.org/TR/WD-DOM/level-one-core-971209.html

[8] Traditional middleware players embrace object-request brokers (Information Week, June 1996), http://www.techweb.com/se/directlink.cgi?IWK19960603S0076

[9] "BEA Objectbroker presentation" (March 1998), http://www.beasys.fr/products/objectbroker.htm

[10] Press quotes on BEA products: TUXEDO, Jolt, ObjectBroker, MessageQ (March 1998), http://www.beasys.fr/products/quotes.htm

program called VSOrb[1] whose purpose is to help ensure adherence of CORBA products to CORBA standards.

Collectively, such trends indicate that CORBA will continue to play an important role in middleware and in the current and future integration of software using Internet technologies.

[1] CORBA Validation - the VSOrb Test Technology (The Open Group, January 1998) , http://www.opengroup.org/public/vsorb/

3. Middleware Comparisons and Recommendations

The paragraphs below summarize key features of DCE, DCOM, and CORBA, describe various comparative advantages and disadvantages of each, and provide general recommendations for when it is appropriate or inappropriate to use each of the three technologies.

3.1 DCE – Distributed Computing Environment

DCE is best described as an influential early middleware product that subsequently fell behind in the marketplace due to its lack of standardized support for object-oriented languages. Although still a part of many legacy systems, DCE is unlikely to play a major role in the development of newer and (especially) Internet-based distributed systems. Many of the most innovative features of DCE have in the past few years been incorporated into other middleware products such as DCOM and CORBA, and DCE appears to lack the market support needed for its products to keep pace effectively with either DCOM or CORBA.

The organization that defines and promotes DCE is The Open Group, formerly known as the Open Software Foundation (OSF), and it includes members from both industry and governments. It is interesting to note that The Open Group recently set up a new program called VSOrb to promote the creation of test suites that will validate the conformance of CORBA vendor products to the standards of the CORBA standards organization, the Object Management Group (OMG). The fact that the organization that sponsors DCE is now working to promote standardization of CORBA products demonstrates the importance with which even the promoters of DCE view CORBA.

3.1.1 Key Features of DCE

The key features of DCE are:

- **Proven multi-platform interoperability**. DCE has demonstrated mature multi-platform interoperability in applications on a wide variety of Unix and non-Unix platforms.

- **A mature IDL.** DCE has a mature, well-defined interface definition language (IDL), with interface specification capabilities that are roughly comparable to the declarative features of the procedural (that is, non-object-oriented) programming language C.

- **A lack until recently of any standard way to specify objects in its IDL.** Prior to late 1997, DCE lacked any standardized way to specify objects to its IDL and required users to develop their own approaches to conveying object-oriented service requests.

- **A mature set of relatively low-level support services.** DCE has a suite of mature but relatively low-level, operating-system-like services that are available to its users. These include distributed security services, synchronization of system clocks, and (optionally) file systems that work transparently across both networks and operating systems.

- **A mature interoperability protocol that is also used by DCOM.** Microsoft chose to use the DCE interoperability protocol in its own DCOM middleware product, making it fairly straightforward for DCE components to communicate with DCOM components.

- **Support for object-like access to remote components.** Although it is not explicitly object-oriented, DCE does support object-like design by requiring that all procedures used to access a remote component be explicitly declared and registered before they can be used by other components in a distributed application. This promotes good design, and helps simplify any subsequent conversion to fully object-oriented middleware.

3.1.2 Advantages of DCE

As a technology for integrating and creating distributed applications, the advantages of DCE are:

- **Proven interoperability.** DCE provides proven, verifiable interoperability across a wider range of operating system than either CORBA or DCOM, and has generally had a tighter approach to verifying interoperability claims than CORBA. CORBA vendors have also tended to produce unique variations and extensions in their products that help them "lock in" customers, but which can seriously damage the ability to interoperate between CORBA products. As of early 1998, DCOM is best understood as being a proprietary product of Microsoft for use on their own operating systems (Windows 95 and NT), although there is at least one significant joint effort underway by Software AG and Microsoft to port DCOM to Unix operating systems.

- **Support for a wide range of current and legacy operating systems.** DCE is supported by a wide range of vendors, including vendors of older legacy operating systems that are less likely to attract the attention of vendors of new middleware products. DCE also has good support on Unix operating systems, and is readily available for Windows.

- **Mature services for increasing distributed application reliability.** Although they are relatively low-level, the maturity and distributed operating system focus of DCE services make them quite useful for increasing the overall reliability and security of an application. The Kerberos-based security service of DCE in particular has been praised for making distributed applications more secure, although this particular service has also been criticized for not being scalable to large applications.

3.1.3 Disadvantages of DCE

The main disadvantages of DCE are:

- **Weak history of support for object-oriented languages.** For application developers using object-oriented languages such as C++, Smalltalk, or Java, DCE has been a weak choice for many years because of its historical lack of a standardized way to describe the interfaces to software objects. The lack in DCE of a standardize approach for describing object interfaces means that developers using object-oriented languages for distributed applications must spend additional development time to achieve interoperability. They must either devise custom standards for describing objects using DCE protocols, or (worse) make their distributed objects look like procedural objects when they are invoked between platforms. Creating custom standards for representing objects over DCE protocols is certainly feasible and has been done numerous times, but such an approach sharply limits the portability and interoperability of the resulting distributed applications. This is because only DCE components that "understand" the same customized object interface will be able to interact with them. In late 1997 this situation was helped somewhat for C++ developers by the release of the DCE 1.2.1 standard, which provides a fully standardized DCE approach for interfacing to C++ objects. Compared to CORBA, however, DCE 1.2.1 object support features are both belated and highly restrictive, since CORBA has had object interfaces for many years and supports interfaces to many other languages besides C++. Thus DCE 1.2.1 is unlikely to regain any market share for DCE, since CORBA provides more flexible object-oriented alternatives that have been in place for several years now. It will more likely be used primarily in legacy systems where some new C++ coding is needed.

- **Weak market position relative to CORBA and DCOM.** At present, DCE is more likely to be found on operating systems with weak market positions. This is due to a difficult combination of the strength of object-oriented CORBA on many of the same systems as DCE, and the strong focus of Microsoft on its own object-oriented but otherwise DCE-like DCOM technology. Additionally, the rapid growth of object-oriented middleware to help support the Internet market is rapidly shutting out older non-object approaches. These factors diminish the chances of DCE increasing its market share. It is more likely that DCE will become a legacy technology used primarily on installed bases.

- **Low level of support services.** Although valuable, the existing set of DCE services is quite low level when compared to the plans of CORBA and DCOM. Perhaps even more importantly, rapid growth and integration of CORBA in the Internet market may shut out DCE from playing an active role in new services that may result from the synergy of the Internet with object-oriented CORBA and Java developments.

- **Difficulty in scaling some services to larger applications.** The Kerberos-based security service of DCE has been criticized for not scaling to larger sizes, and The Open Group has responded by opening up the range of security methods that can be used with DCE. The Kerberos-based security features of DCE are still an attractive feature in comparison to more complex DCOM and CORBA methods, but they generally should only be used when a distributed application will definitely not require scaling beyond Kerberos limits.

- **Weak tool and componentware support.** Both DCE and CORBA are relatively weak in tool and componentware support when compared to DCOM, which has the advantage of being sponsored by the same company that is developing both the operating systems and development tools that will be used with DCOM. The relatively weak market position of DCE is likely to make it difficult for DCE to make much headway in this area. In contrast to DCE, CORBA appears to be much better situated for obtaining new tool and componentware support, as demonstrated by products such as Borland's JBuilder 2 and Black and White's OrbixBuilder for Visual Café.

3.1.4 Recommendations for DCE

Although historically important, it is unlikely that DCE will be able to gain sufficient lost ground to obtain the levels of new vendor and tool support likely to be seen for both CORBA or DCOM. Recommendations for using DCE as middleware thus are as follows:

- **Avoid using DCE for new applications on new platforms.** Despite its maturity and good set of basic services, the market situation for DCE makes it a weak choice for developing new distributed applications that run on new (e.g., Solaris or Windows NT) host platforms. For these cases CORBA, DCOM, or a combination of CORBA and DCOM with bridging would provide better access to object-oriented software technology, an expanding set of tools and componentware, and a wider range of services.

- **Avoid using DCE for new object-oriented applications.** Although DCE has now added object-oriented syntax to its IDL, it is far behind both CORBA and DCOM in terms of vendor and language support for object-oriented distributed computing. The new object-oriented features of DCOM would be better used to upgrade DCE legacy systems, rather than to create entirely new object-oriented applications.

- **Consider using DCE to continue supporting purely legacy applications.** In situations where an existing DCE-based application needs to be supported or modestly extended, continued use of DCE should be considered as long as there are no overriding requirements to switch to or integrate into a different middleware product. The cost and risks associated with conversion in such cases make it unlikely that the conversion effort would be worthwhile in comparison to the costs of simply continuing to use DCE.

- **Plan for eventual replacement if you do use DCE.** If new DCE software is definitely required for a particular application, it should in general be designed to use small, well-defined sets of operators (remote procedures) to access DCE resources. These sets of operators should be as complete as possible (e.g., they should include operators for overall control of the resource, as well as for use of the resource). This approach provides the proven benefits of resource encapsulation for the new DCE code, and will also make any subsequent transition to object-oriented middleware easier to accomplish.

3.2 DCOM – Distributed Component Object Model

DCOM has an unusual history as a middleware technology. It began as a Microsoft document structuring technology called Object Linking and Embedding (OLE), and was later transformed into a more generic object-oriented technology called Component Object Model, or COM. COM is partway to being middleware in that it defines a generic object-oriented interface definition language, but not an interoperability protocol. Thus COM can be used to define common interfaces between software components, but it can link to those components only if they reside on the same computer. To create the true middleware product DCOM, Microsoft then extended COM with the addition of a re-engineered version of the DCE interworking protocol. As of early 1998, the most recent major DCOM development was Microsoft's announcement in late 1997 of the next generation of DCOM technology, which will be called COM+. COM+ will add CORBA-like language interfaces to DCOM and will be used in an overall strategy for distributed systems design that Microsoft calls Distributed Network Architecture, or DNA.

DCOM promises to play a major role in PC based server systems when Window NT 5.0 begins to significantly penetrate that market sometime in 1998. As of early 1998, however, the features of DCOM that deal with conveying and supporting service requests between remote components are still too immature to easily support development of large distributed systems. At present DCOM is more important not for its distribution features, but rather for its relationship to the older COM interface model. COM is deeply embedded in Windows systems and provides an important compatibility target for distributed applications that need to incorporate both Windows and non-Windows host systems into their networks.

3.2.1 Key Features of DCOM

The key features of DCOM as of late 1997 are:

- **Support for object-oriented language interfaces.** DCOM supports programming objects, although its origins as a document structuring concept tends to result in significantly longer and less intuitive object definitions than those of CORBA. Microsoft has already stated its intent in COM+ (the next generation of DCOM) to provide much more CORBA-like style of interface definition in future releases.

- **Ready availability on Windows operating systems.** DCOM comes free with Windows NT 4.0 and can be downloaded free as an add-on to Windows 95.[1] This is both an advantage and a disadvantage, since it allows DCOM to take advantage of "insider" operating system expertise. However, it also reinforces the perspective that DCOM is largely a single-vendor product.

- **Minimal availability outside of Windows platforms.** DCOM is not at present a major middleware technology outside of Windows platforms. Third-party vendors such as Software AG have shown an interest in implementing DCOM on other operating systems such as Unix, but as of early 1998 the capability and actual deployment of such tools was quite modest. Compared to DCOM on its native Windows platforms, such third-party implementations of DCOM on non-Windows platforms are likely to be higher in cost and less well supported by development tools, and are also likely to have difficulty keeping current with the rapidly changing technology path of DCOM within Microsoft.

- **Ability to intercommunicate (bridge) easily with DCE.** DCOM uses a slight variation of the same interoperability protocol as DCE, which makes it relatively easy to interface or bridge DCOM to legacy DCE applications.

[1] COM versus CORBA: A Decision Framework (April 1998) ,
"http://www.quoininc.com/quoininc/COMCORBA.html#COM versus CORBA: A Decision Framework"

3.2.2 Advantages of DCOM

The main advantages of DCOM as of early 1998 are:

- **Strong tool and system support.** Microsoft is very strongly committed to DCOM. One recent example of the depth of this commitment has been the recoding of significant portions of the 5.0 release to their NT operating system to use DCOM, which in NT 5.0 goes by the new moniker of COM+. For developers, this support translates into serious long-term tool and development support for DCOM applications, as well as assurances that DCOM will be well supported for Windows and NT operating system platforms.

- **Lower cost of DCOM-compatible networks.** Because it is distributed as an integral part of Windows (in particular Windows NT) operating systems, distributed applications using DCOM on Microsoft-only network have a substantial cost advantage over third-party middleware on Windows platforms. Additionally, programming tools such as Visual C++ include integrated support for basic DCOM programming paradigms, making it significantly easier to integrate DCOM middleware into new applications when using those tools. Finally, the Intel platforms that typically host Windows operating systems have increased in power and dropped in cost much more rapidly than comparable Unix systems, so that the total cost of an Intel and Windows based DCOM network can often be much lower than a comparable Unix network.

- **Easier interfacing to object-oriented languages than DCE.** DCOM supports programming objects, which simplifies its interface to object-oriented languages such as C++ in comparison to DCE. However, the code required to describe an object in DCOM tends to be longer and less intuitive than comparable code for CORBA. This is offset during initial development by the language-level tool support available for DCOM on Windows platforms, but can still be an issue during support of the resulting code.

- **Good separation of interfaces and implementations.** Like CORBA, DCOM provides a clear separation between the interface to an object (how it is called or used) and the implementation of an object (how it is coded). This allows flexibility during development, since objects can be implemented in different ways or even in different languages for different nodes in a network. It can also be a problem if implementations fail to provide the correct functionality to an interface.

3.2.3 Disadvantages of DCOM

DCOM also suffers from several significant disadvantages:

- **Platform dependence.** At present (early 1998), the only operating systems on which DCOM is strongly supported is Windows 95 and NT. Microsoft has publicly expressed a strong interest in helping in helping one vendor, Software AG, to port DCOM to Unix and other operating systems. The level of commitment of Microsoft to this effort is indicated by the fact that they list the Software AG DCOM porting work as a significant component of their overall long-term strategy (called COM+) for COM and DCOM. However, it is worth noting that the initial (November 1997) price of the Software AG port of DCOM to the OS/390 mainframe operating system was $200,000.[1] This is a rather mind-boggling figure for cost-conscious PC and Windows NT users who are accustomed to receiving COM and DCOM technology bundled with NT for free, although presumably the Unix ports will cost far less when they are finally released. Unless prices for DCOM ports drop drastically and the technical exchange relationship between Microsoft and Software AG becomes very close, it is unlikely that non-Windows DCOM products will even approach the cost and operating system advantages that COM and DCOM enjoy on their native Windows operating systems. Pragmatically, DCOM thus should be viewed in 1998 as primarily platform-specific middleware for Windows, although it should be watched carefully for the arrival of new third-party products or initiatives from Microsoft that may make it a more genuinely multi-platform and affordable.

- **Sole source lock-in.** As of early 1998, a decision to use DCOM as the only middleware for a distributed application is tantamount to deciding to host the application on a purely Windows based network, with the possible addition of a few costly legacy systems using Software AG products. This is a consequence both of the relative immaturity of current (pre-NT 5.0 release) distribution features of DCOM, and the early state of efforts by third-party vendors to produce DCOM tools for other operating systems. Since Microsoft has final control on all DCOM standards, technology, and marketing strategies, a decision to use only DCOM for a distributed application should not be made lightly, especially if the expected target for the distributed application is a heterogeneous network.

- **Complicated non-intuitive programming style for objects and interfaces.** Due in part to its ancestry as a document structuring technology, DCOM uses a style of object and interface specification that is more complex and less intuitive than the programming-oriented styles of languages such as Java or Smalltalk. As a consequence, hand-coded DCOM interfaces tend to be larger, less intuitive, more manually intensive, and more error prone than the equivalent CORBA interfaces, which in contrast are visually simpler and much more Java-like.[2] Microsoft has compensated for this problem by providing a rich set of tools and components to simplify creation of DCOM interfaces. These tools include DCOM support embedded within Microsoft's Visual J++, Visual C++, and Visual Basic programming tools, and numerous components based on COM interfaces. Also, Microsoft has stated its intention to add language-independent interfaces to its next generation of DCOM technology, COM+.[3]

---

[1] ActiveX™ Controls (January 1998), http://www.microsoft.com/com/activex.htm

[2] "DCOM and CORBA Side by Side, Step by Step, and Layer by Layer " (September 1997), http://www.cs.wustl.edu/~schmidt/submit/Paper.html

[3] "COM+: Building on the Success of the Component Object Model" (October 1997), http://www.microsoft.com/com/slides/complus.zip

3.2.4 Recommendations for DCOM

As of early 1998, DCOM is a technology in rapid transition. It has the massive technological and marketing support of Microsoft, but is unlikely to become a major player in the construction of new distributed applications until Windows NT 5.0 servers achieve significant deployment and the anticipated COM+ and DNA enhancements of DCOM materialize. In the meantime, CORBA is making rapid inroads into the Windows PC marketplace through the massive distribution of the CORBA compliant Borland VisiBroker tool as part of the popular Netscape Communicator browser, although the actual level of usage of the VisiBroker component of Communicator cannot be readily determined. When in the past Microsoft promoted proprietary networking schemes in competition with more open Internet standards, the results were generally in favor of the more open standards. It is too early to tell if this will also be the case for DCOM and CORBA, but caution in relying solely on proprietary and rapidly changing DCOM middleware is certainly advisable, particularly for applications that reside on highly heterogeneous networks.

As of early 1998, the recommendations for using DCOM are as follows:

- **Consider DCOM mostly for experimental use aimed at pure NT 5.0 networks.** Any impact that DCOM has on the middleware market is most likely to appear first in pure Windows NT 5.0 networks, which will provide stronger DCOM support and also make more extensive use of DCOM within the operating system itself. DCOM will also benefit greatly when Active Directory technology in NT 5.0 becomes widely available. Using DCOM thus may be appropriate for projects that will run only Windows NT 5.0 when it becomes available (possibly not until the end of 1998 or later).

- **Avoid DCOM as the only middleware product for heterogeneous networks.** Current levels of support for DCOM on non-Windows operating systems do not easily justify the use of DCOM for heterogeneous networks. The current (early 1998) relative immaturity of the distributed communication features of DCOM also works against it for use in complex network environments that require multi-vendor support and a high level of adaptability to unique circumstances. (Note: This is a rapidly-changing area, and new products and changes to DCOM may make heterogeneous use of DCOM or its future incarnations such as COM+ easier in the future – e.g., sometime in 1999 or 2000. At present, however, both pure CORBA and bridging between multiple middleware products appear to be more viable middleware approaches for heterogeneous networks that require the use of both Unix and Windows.)

- **Avoid using DCOM to integrate legacy systems (except for DCE legacy systems).** As of early 1998, DCOM does not have the flexibility or range of platform implementations needed to make it appropriate for integrating legacy (e.g., Cobol, Ada, or C++) software into new network applications. In contrast, CORBA is a much better choice for such integration activities because of its broad platform and vendor support, and because of its cleaner and more understandable object model. One important exception to this general rule is that when the legacy system already uses DCE and the new portion of the network application consists of Windows-based PCs, DCOM may provide easier integration of the DCE components into the new Windows-based platforms than would CORBA in the same situation. This is possible because DCOM uses a communication protocol that is very close to that of DCE, so that a minimum of new software development should be needed to bridge between the two. Even in this case the tradeoffs of using CORBA versus DCOM for the integration should be carefully considered, however, especially if the new components of the system include both Unix and Windows operating systems.

- **Consider bridging in heterogeneous networks that require the use of DCOM.** When DCOM is required for the NT portions of a heterogeneous network, the possibility of using a middleware bridge should be considered strongly. The alternative of attempting to use DCOM across a heterogeneous network is much less attractive and in many cases may simply not be feasible. Also, the trend of CORBA vendors towards providing good bridges to COM has accelerated, with many new products likely in 1998.

3.3 CORBA – Common Object Request Broker Architecture

The Common Object Request Broker Architecture (CORBA) is the oldest major object-oriented middleware standard. Although early versions of CORBA products suffered from incompatible interpretations of CORBA standards, there have been impressive progress and maturing of CORBA standards and products in the last two years (1997 and 1998).[1] This burst of effort to make CORBA products more powerful, standardized, and interoperable appears to be largely the result of competition from DCOM, which the CORBA community has increasingly realized represents a genuine competitive threat that requires both well-defined standards and genuinely interoperable products. CORBA is notable for: its support for an impressively wide range of platforms and programming languages; its compatibility with object-oriented approaches, which are needed to keep distributed applications flexible and scalable; the unusually broad and diverse range of vendors, developers, and users who support it; its focus on an open, long-term integration and interoperability architecture (the Object Management Architecture, or OMA); and its recent unexpected ascent to become one of the most widely deployed middleware products on Windows and PC platforms, which occurred in 1997 through the distribution of the CORBA-compliant Borland VisiBroker as part of the popular Netscape Communicator Internet browser.

CORBA is an open standard of the Object Management Group, which is a consortium of vendors, developers, and end users that was established in 1989. The goal of the OMG is to encourage the development and standardization of an Object Management Architecture (OMA) that provides broad interoperability between object-oriented distributed components even in highly heterogeneous networks. The CORBA specification forms the core of the OMA architecture, with other OMG standards dealing with a variety of closely related middleware issues. These other OMA standards include the CORBA Internet Inter-ORB Protocol (IIOP) for Internet based communications between software objects, object services to help standardize the creation, support, and use of objects, common facilities for supporting more local support operations such as printing, document management, database and email, and domain interfaces for providing more

[1] The ORB Interoperability Showcase (OMG, June 1997), http://corbanet.dstc.edu.au/

consistent access to software components needed in specific, well-defined application areas such as health care, insurance, and telecommunications.

The breadth of CORBA support is impressive and is also a significant factor in the overall viability of CORBA as a middleware standard. As of early 1998 the OMG had a total membership of over 780 vendors, developers, and end users. The OMG web site indicated that the OMG was aware of 261 different organizations using or developing CORBA applications, and the actual number of organizations using CORBA is probably many times higher given that the OMG figures are based on voluntary information. The most intriguing deployment figure for CORBA technology occurred in 1997, when a significant subset of CORBA was shipped to about 68 million users as an integral part of the popular Netscape Communicator browser. In a few months this event increased the total deployment of CORBA technology from hundreds or at most thousands of systems to over 30 million systems (mostly in PCs) in early 1998.[1] One immediate impact of this deployment has been to increase the overall awareness of CORBA IDL as a standard interface language for cross-platform computing.

3.3.1 Key Features of CORBA

The key features of CORBA are:

- **Support for a broad range of platforms and programming languages.** One of the most notable advantages of CORBA as a middleware technology is its unusually broad range of support for computing platforms and programming languages, which makes it particularly suitable for applications such as integration of legacy software and platforms into new, heterogeneous network applications.

- **Broad and diverse market support.** An unusually broad and diverse range of vendors, developers, and users supports CORBA. Membership in its supporting organization, the Object Management Group, has accelerated in the last few years, indicating that support for this standard is increasing and becoming even broader based.

- **Standardization of a flexible Internet protocol for inter-object communications.** One of the most conspicuous recent contributions from CORBA has been the addition of a new specification, called the Internet Inter-ORB Protocol or IIOP, by which software objects of nearly any type can communicate with each other over the Internet. This addition to CORBA has greatly increased its versatility, both because it makes it much easier to use CORBA over the Internet, and because the IIOP can easily be used to bridge CORBA objects to other types of software objects such as Java components.

- **Support for object-oriented distributed applications.** Object-oriented approaches benefit distributed applications by ensuring that software components can be moved around a network more flexibly and with less risk of losing the connection between data

[1] "Borland's VisiBroker ORB Surpasses 30 Million Licenses Deployed Worldwide" (March 1998), http://www.borland.com/visibroker/press/1998/visi30m.html

and associated functions. The growth of the Internet has increased this need for object-oriented approaches to distributing components, and thus has helped increase interest in existing object-oriented middleware standards such as CORBA.

- **Maturity of its object concepts.** As the first major object-oriented, multi-platform middleware technology, CORBA has had more time to mature than DCOM. CORBA has also benefited from the fact that it originated from a programming-oriented perspective that differs significantly from that of DCOM, which has its roots in a document structuring technology known as Object Linking and Embedding (OLE). It is notable that because of the simplifications it provides, Microsoft has indicated that they will provide a much more CORBA-like, programming-language-independent style of interfacing in its next (COM+) iteration of DCOM technology.

- **Emphasis on network transparency (the ORB concept).** Overviews of CORBA often focus on *object request brokers*, or ORBs. ORBs are usually represented in diagrams as “software switches” that receive all service requests and then route them to appropriate server locations. In practice, however, an ORB is probably better understood simply as the implementation of CORBA by a particular vendor, and the switching role of an ORB as the degree to which that ORB makes the underlying network invisible or “transparent” to application objects. An explicit software switch, which can be a serious performance bottleneck, is not necessarily required. For example, an ORB can generally provide *location transparency*, or the ability to request services from an object without knowing the name of the computer on which the object is located. It achieves this by finding a location once and then having the client host send all services requests directly to the host of that server object. A well-designed ORB combines network transparency and good performance by making such optimizations as automatic and transparent to applications as possible.

3.3.2 Advantages of CORBA

The main advantages of using CORBA are:

- **Support by about 800 vendors, developers, and users.** While Microsoft participates in the OMG, the most important support for CORBA comes from the rest of the very large and diverse membership of the OMG. Vendors appear to feel it is important to have an open object middleware standard if they are to remain independent of any one operating system or operating system vendor. The growth of DCOM thus has encouraged many vendors to more strongly support the CORBA standard, which they feel is less focused on a particular operating system (Windows) and also easier to use than DCOM.

- **Platform independence.** With the participation of hundreds of vendors and developers in the OMG, CORBA is conspicuously platform independent, especially when compared to DCOM and its strong bias towards Windows. As of early 1998, CORBA is also the *de facto* standard for actively used middleware on Windows-based PCs. This occurred in 1997 through the inclusion and distribution of the Borland VisiBroker ORB within the popular Netscape Communicator browser. Additionally, all of the major CORBA vendors provide Windows NT versions of their ORB products, making CORBA a viable option for exclusive use as the middleware in networks that include NT systems. This is especially true when CORBA vendors provide good bridging products for interfacing into the COM objects of Windows operating systems. Since there has been a flurry of new CORBA/COM bridge announcements in early 1998, such as the IONA Orbix announcement of OrbixCOMet Desktop,[1] it is likely that will be a substantial increase in 1998 of off-the-shelf support for creating CORBA-based heterogeneous distributed applications that include PC platforms.

- **Open, public process for creating and approving specifications.** The open process in which OMG specifications are created is itself a significant asset, since it allows any interested vendor or development group both to contribute to the standard and to obtain the most current version of the standard. DCE has similar advantages on this point, but the openness of the specification processes for both CORBA and DCE contrasts sharply with the proprietary nature of the DCOM specification process, which is essentially owned by Microsoft and can change rapidly and unexpectedly based on internal marketing decisions by Microsoft. Even the name of the DCOM technology has been changed or modified on a fairly frequent basis, and even companies such as Software AG that are working closely with Microsoft appear to have difficulty obtaining sufficiently detailed information on DCOM to build fully functional equivalents on other platforms.

- **Long-term definition and support of middleware services.** The Object Management Architecture (OMA) that is built around CORBA provides a more complete and extensive overall roadmap for defining and adding services and domain-specific interfaces than is currently found in any other middleware technology. The defined and planned services of the OMA include a wide range of capabilities for creating, supporting, controlling, and maintaining objects distributed over a network. As of early 1998, the CORBAservices guide[2] lists 15 key services and further describes 12 future object services for further extending these services. The key services are: Naming, Event, Life Cycle, Persistent Object, Transaction, Concurrency Control, Relationship, Externalization, Query, Licensing, Property, Time, Security, and Object Trader. The future object services are: Archive, Backup/Restore, Change Management, Data Interchange, Internationalization, Implementation Repository, Interface Repository, Logging, Recovery, Replication, Startup, and Data Interchange. While such lists help demonstrate the depth and inclusiveness of the OMG vision for CORBA middleware, it should also be noted that many of these services are not yet provided by vendors, and that other CORBA services could eventually be overtaken by events elsewhere in the software industry if they do not become widely available and actively deployed in the next two or three years.

- **Easier to understand, program, and support at the software coding level.** Of the three middleware technologies of DCE, DCOM, and CORBA, CORBA has the cleanest and

[1] Orbix COMet Desktop (February 1998), http://www.iona.com/news/pressroom/msoft.html

[2] Electronic copies of CORBAservices guide (February 1998), http://www.omg.org/corba/sectran1.htm

most easily maintained interface at the software coding level. This interface level is particularly important for creating reliable, supportable software, since any additions or changes to an interface need to be clearly understood by the people maintaining them. Additionally, an understandable programmatic interface is especially important when interfacing to legacy systems and software that cannot be easily accommodated by more abstract approaches such as graphical user interface (GUI) programming or automatic generation of interfaces from programming tools. DCE also has a fairly understandable programming interface, but suffers from being too close to the operating system, so that using DCE correctly requires users of DCE to understand a variety of detailed issues that may not be directly relevant to their application. DCOM has a particularly complex and non-intuitive binary programming interface that is based on C++ virtual functions. The complexity of using this interface means that users rarely work with it at the programmatic level and rely instead on generation of interface code by higher level tools. This approach is fine for rapid early generation of prototype interfaces, but tends to produce interfaces that in the long range are hard to understand and difficult to maintain.

- **Support for object-oriented languages and designs.** CORBA is highly compatible with both an overall object-oriented approach to designing distributed applications and with object-oriented languages in general. An especially promising object-oriented design path for CORBA is found in its continuing integration with both the Internet and Java. This "gang of three" (CORBA/Internet/Java) makes possible a higher level of integration and underlying flexibility than is possible with any one of these technologies by itself. When used together, CORBA provides both universal object-oriented messaging via its Internet Inter-ORB Protocol (IIOP), and also a way to access legacy and compiled systems. The Internet provides a highly robust and already nearly universal network host on which the distributed applications can reside. Finally, Java provides a highly dynamic language suitable both for rapidly creating new interfaces to legacy systems (accessed via IIOP), and the ability to relocate components dynamically on the underlying Internet host.

- **Support of the Internet Inter-ORB Protocol (IIOP) –** One of the most interesting and potentially powerful mechanisms that has been developed by the CORBA community is something called the Internet Inter-Operability Protocol, or IIOP. The IIOP well defined protocol that lets software objects of nearly any type to communicate over the Internet, so that it can be used not only between CORBA objects, but also between CORBA and non-CORBA objects. This makes the IIOP a good *de facto* candidate for universal object communications over the Internet.

- **Ability to integrate legacy software applications.** The CORBA community has promoted the concept of integrating legacy systems with custom (and generally non-object-oriented) interfaces into new applications by "wrapping" or hiding the legacy software behind a new set of interfaces that meet CORBA standards, and has provided clear, easily understood programming-level interfaces to make this possible. This is in sharp contrast to DCOM, which currently uses a complex, hard-to-understand programmatic interface that is usually generated by higher level tools instead of being directly coded by programmers. While this generative approach of DCOM is useful for creating initial prototypes on one well-defined platform such as Windows NT, it is not well suited for defining interfaces to components written in many different languages and residing on many different platforms. In contrast, the more readily understandable programming interfaces of CORBA can be created with about the same ease for nearly any type of legacy component, and are also far easier to maintain because they are more easily understood than generated code. Once CORBA interfaces have been created, they can also be interfaced to other systems in a more standard fashion (e.g., via the Internet by using IIOP). This approach is likely to become even more useful as CORBA continues its current trend towards closer integration with both the Internet and Java.

- **Good separation of interfaces and implementations.** As with DCOM, CORBA provides a very clear separation of the interface to an object (how it is called or used) and the implementation of an object (how it is coded). This allows flexibility during development, since objects can be implemented in different ways or even in different languages for different nodes in a network.

- **Strong focus on providing dynamic interface options.** All three of the middleware technologies described here have traditionally tended to rely on "static" or compiled approaches to setting up communication paths between components. While these approaches are powerful and generally efficient, they are also inflexible and can make dynamic recovery and load balancing more difficult. In the case of CORBA, however, the OMG has placed a strong emphasis on also providing "dynamic" interfaces that provide (at the cost of some performance) the ability to create and re-allocate the paths used to communicate between components, so that greater component mobility and system reliability is possible. Dynamic capabilities are also especially useful for adding legacy software into a new application, since they can be used to provide a single "traffic cop" interface into a large set of legacy components, rather than attempting to build separate static interfaces to each such component. The Object Request Broker (ORB) concept of CORBA has from the beginning been oriented towards "brokering" requests in a dynamic fashion, and it provides a well-defined software location for performing the types of message re-direction required for dynamic component interfaces. The OMG has also specifically designed client and server side dynamic interface capabilities in the form of the CORBA Dynamic Invocation Interface (DII) and Dynamic Skeleton Interface (DSI).

3.3.3 Disadvantages of CORBA

Some disadvantages of CORBA is:

- **Less integrated tool support than DCOM.** When compared to the language-level GUI support provided for COM and DCOM by Visual C++, Visual J++, and Visual Basic, CORBA is neither as fully integrated nor as easy to use. On the other hand, it can be argued that based on the relative complexity of using CORBA and DCOM for the same tasks,[1] COM and DCOM might probably not be used much at all if it were not for the

[1] "DCOM and CORBA Side by Side, Step by Step, and Layer by Layer " (September 1997), http://www.cs.wustl.edu/~schmidt/submit/Paper.html

availability of these tools. Perhaps more importantly, generative approaches such as those used by the Microsoft development tools can lead to significant difficulties later in enhancing and maintaining the resulting software, since once any changes are made directly to the generated code it becomes impossible to directly recreate it from the original tool level. When viewed from a full life cycle cost perspective that includes the (usually significantly larger) costs of enhancing and maintaining software after initial development is completed, a technology such as CORBA that provide succinct, readable statements of what is actually being done will generally be superior to generative approaches. This is because the generative approaches tend to increase the complexity and fragility of the code that must be maintained. Finally, the overall market for CORBA components is likely to grow as integration of CORBA with Java proceeds through tools such as the Borland VisiBroker, it is likely that CORBA will benefit from a much larger suite of supporting components and tools than is now available.

- **Weak vendor-to-vendor interoperability of CORBA products.** In contrast to DCE, the ability of CORBA products to interoperate with each other has been historically weak. Protocols have not always matched, services cannot always be used across vendors, and the internal naming conventions have often been incompatible or opaque across vendor products. The result has been a general inability of CORBA products from different vendors to talk with each other. This has weakened the overall value of CORBA, since it means that for all practical purposes CORBA users, like DCOM users, will need to choose a single vendor for their technology. On the other hand, the CORBA standards community is aware of these problems and has been working to increase interoperability across vendors, such as through the relatively new (in early 1998) branding and testing initiative by The Open Group CORBA. The comparatively recent Internet Inter-ORB Protocol (IIOP) CORBA standard has been particularly helpful in this area, since it provides a good basis for well-standardized communictions between CORBA vendors. Application builders who are faced with using multiple CORBA products (e.g., Orbix plus Netscape with an embedded Borland ORB) need to verify that the level of interoperability needed to fully support their intended application has been implemented.[1]

3.3.4 Recommendations for CORBA

As of early 1998, CORBA is a mature object-oriented middleware product that is the most widely deployed for Windows-based PCs, and it is far and away the most dominant middleware platform for Unix systems. CORBA also enjoys unusually strong support from a wide range of vendor, developer and user organizations. Its long-term plan for services provides an unusually broad and well-defined path for service definition and integration. CORBA is also being integrated rapidly both with the Internet and with Java, a language designed specifically for creating distributed applications. Integration of CORBA with the Internet allows developers to use the existing

[1] COM versus CORBA: A Decision Framework (April 1998) ,
"http://www.quoininc.com/quoininc/COMCORBA.html#COM versus CORBA: A Decision Framework"

Internet infrastructure for faster and more flexible creation of new distributed applications. Similarly, integration of CORBA with Java makes it significantly easier for developers to use the simpler and more network-oriented features of Java to integrate legacy software into new distributed applications.

General recommendations for using CORBA are:

- **Use CORBA.** The *de facto* position of CORBA as the most widely distributed and used middleware product for Internet-connected PCs makes it an excellent choice for low end Windows platforms. Furthermore, its broad availability and support on other platforms such as Unix makes it useful for integrating diverse types of components and systems. Its appropriateness for integrating legacy software is further enhanced by its clear, well-defined, and internationally standardized interface description language for specifying the interfaces to software components. The maturity of the object-oriented features of CORBA also make it well-suited to the current trend towards more dynamic distributed software that whose relationship to the underlying network can change in real time.

- **Use only one CORBA vendor unless interoperability can be verified.** The greatest current weakness of CORBA is the slow pace of its efforts to make CORBA products from different vendors interoperate with each other. There has been significant progress in this area in the last couple of years, but at present the safest strategy for using CORBA is still to pick a single vendor and use that vendor consistently for a given application.

- **Use "gang of three" (CORBA/Internet/Java) CORBA products whenever possible.** At present, the most promising overall path for broad integration of applications using the Internet appears to be joint use of CORBA (especially IIOP), Internet technologies, and Java.[1] [2] [3] CORBA provides integration of legacy systems, broad platform interoperability, object-oriented interfaces, and a well-defined path (the ORB) for implementing various forms of network transparency. This support is likely to become increasingly important as part of an overall industry thrust to make distributed applications more scalable, robust, and portable. The Internet provides a robust universal network for hosting distributed applications, and Java provides a dynamic programming "glue" that can be used to develop new interfaces into older legacy systems more rapidly and more effectively. While all CORBA 2.0 and 2.1 products are required to support IIOP, the way in which IIOP is supported can vary significantly from vendor to vendor. The best implementations make

[1] Java, RMI and CORBA (OMG, June 1997), http://www.omg.org/news/wpjava.htm

[2] "JavaSoft concedes it's not an all-Java world" (Computer Reseller News, July 1997), http://www.techweb.com/se/directlink.cgi?CRN19970721S0060

[3] Review of book "Instant CORBA" (Orfali & Harkey, February 1998), http://www.micromail.com/titles/7666.html

good use of features that increase efficiency and reduce needless overhead for operations such as communication with Java objects.[1]

- **For networks that include NT, use CORBA with good COM bridges.** DCOM will be an important force in the upcoming release of Windows NT 5.0. However, for now (early 1998) an approach that relies on CORBA for the network side of distributed applications and CORBA-to-COM bridges for the Windows NT and Windows 95 side is more likely to produce robust, reliable distributed applications. Support for CORBA-to-COM bridges should increase in 1998, as demonstrated by the early 1998 release of products such as the IONA OrbixCOMet Desktop.[2]

[1] "Distributed Object Computing in the Internet Age" (Visigenic, October 1997), http://www.visigenic.com/prod/vbrok/wp.html (Note: Withdrawn by Borland in March 1997)

[2] Orbix COMet Desktop (February 1998), http://www.iona.com/news/pressroom/msoft.html

4. Summary of Recommendations

Use Distributed Computing Environment (DCE) only for legacy software. From a market perspective, DCE has been significantly weakened by its belated and incomplete support for object-oriented languages. With object-oriented languages continuing to grow in importance in both commercial applications and on the Internet, this has left DCE in a poor situation compared to both CORBA and DCOM. DCE thus is a poor choice for a middleware technology except when it is already being used in a legacy system. DCE is not an appropriate choice for building entirely new distributed systems.

Use CORBA. The *de facto* position of CORBA as the most widely distributed and used middleware product in PCs (via Netscape Communicator) makes it an excellent choice for use in low-end Windows platforms. Furthermore, the broad availability and support for CORBA on Unix and other platforms makes it especially appropriate for integrating diverse types of systems. The mature object-oriented features of CORBA also support the current trend towards more dynamic distributed application architectures, since software objects combine data and functionality in a way that makes them safer and easier to move to from platform to platform than comparable non-object-oriented software components.

Use integrated CORBA/Internet/Java products whenever possible. One particularly promising middleware technology trend is the ongoing integration CORBA, the Internet, and Java. An explicit example of this trend is the Netscape Communicator browser and its bundled VisiBroker Java software. The integration of these three technologies is centered on a CORBA message protocol (that is, a well-defined style of exchanging messages) known as the IIOP, or Internet Inter-ORB Protocol. The IIOP is a simple but flexible CORBA protocol that allows objects written in many different programming languages to communicate directly with each other over the Internet. As of early 1998, both the design and marketing position of IIOP make it the best candidate for becoming a universal protocol for linking diverse types of objects over an Internet-style network.

Use middleware bridges into COM. DCOM is not presently a major factor in building distributed systems, but the closely associated local-only COM technology is already widely used in Windows 95 and NT to define the interfaces to many types of software objects. Middleware vendors who provide effective bridges to COM thus also provide valuable access to software components available in low-cost PC systems, and also minimize the possible future impact of more robust versions of DCOM.

Pay close attention to DCOM in Windows NT 5.0. Microsoft is strongly committed to DCOM and has already announced new initiatives that should make it more powerful and useful on both its native Windows systems and on other operating systems. The most important event for DCOM will be the full commercial release of Windows NT 5.0 in late 1998 or 1999. NT 5.0 will include an advanced set of middleware-oriented services (called Active Directory) that should make DCOM much more powerful and easier to use. Overall, it is still too early to tell how such future versions of DCOM will compete with the growing CORBA/Internet/Java collection of multi-vendor middleware efforts.

Expect to see more use of Java in distributed applications. Java provides features that make applications more portable and more scalable, particularly when it is accessed though the IIOP. Java is currently the best overall candidate for “mobile objects” that can be flexibly reassigned to new platforms to increase efficiency during everyday use of a distributed application. Due to the many market, technical, and legal factors that are affecting Java, it is difficult to predict exactly how large a role Java will play in the future of middleware, although a fairly major role looks fairly well assured as of early 1998.

Expect to see continued expansion in the use of scripting languages. Contrary to the minimalist implications of the term “scripting,” scripting languages such as Perl, Visual Basic, Javascript, Python, and tcl/Tk are actually among the most powerful programming languages in existence. They are particularly useful for integrating legacy software, since much of their power comes from an open and highly accommodating programming style that does not require legacy components to meet the internal programming conventions of new software. In terms of their relationship to middleware technologies such as CORBA, scripting languages are best understood as complementary technologies that can be used to bring legacy software into a middleware framework rapidly and easily. For example, Perl or Python can be used to convert the inputs and outputs of a legacy system into IIOP messages, so that the resulting combination looks like a CORBA-compliant server.

Consider using XML to simplify sharing of middleware data. XML is a new World Wide Web Consortium (W3C) standard for representing complex data with human-readable labels and structuring. XML complements the closed, encapsulated data structures approach of CORBA and object-oriented programming languages by allowing complex data to be “publicized” with both context and structure preserved, but without requiring that users of the data have any special knowledge of the internals of the original object. XML thus is in many ways the data equivalent of a scripting language, since it provides an open style of “data scripting” to represent complex data, without requiring that either side of the exchange know much about the internals of the other. In terms of its relationship to a middleware technology such as CORBA, XML provides a standardized way to represent complex, object-oriented data without forcing all software components to use the same underlying object-oriented or relational database. This kind of database independence translates into greater flexibility when using a middleware technology such as CORBA to integrate diverse types of components into a single distributed application.

Expect change. The Internet has drastically increased the rate at which new technologies impact the market place. Approaches that emphasize open, well-standardized message protocols (e.g., the CORBA IIOP and the recent XML standard) are generally the best choices in such situations, as opposed to selections of specific tools or functions.

Appendix A. Orbix CORBA Products

Orbix is one of the leading CORBA vendors, and it will be included in the 4.0 release of the DII COE. Orbix is one of the first CORBA vendors, and they are also one of the first CORBA vendors to provide significant security services.

Categories of Orbix products include:

- Orbix for various operating systems (Unix, MVS, Windows, others)
- Orbix for real-time applications
- Security
- Internet
- System Management
- Transactions
- Messaging

A full listing of Orbix products can be found at:

Orbix Products (IONA, April 1998), http://www.orbix.com/products/fulllist.html

Appendix B. VisiBroker CORBA in Netscape Products

B.1 VisiBroker Components in Netscape Products

Borland VisiBroker is an easy-to-use, Java-compatible CORBA product that provides flexible object-oriented communications between Internet software components. From a middleware design perspective, VisiBroker is notable both for its leading position in integrating Java with CORBA[1], and for its emphasis on dynamic architectures that provide improved scalability and availability of the resulting applications.[2] One indicator of the viability of the VisiBroker design and product is the recent (March 1998) abandonment of by Sun Microsystems of its own NEO CORBA product in favor of using VisiBroker.[3]

In June of 1997, parts of two VisiBroker products were bundled into two Netscape products.[4] The VisiBroker products used in this bundling were VisiBroker for Java and VisiBroker for C++, and the Netscape products were the popular (and free[5]) Netscape Communicator 4.X browser and the Netscape Enterprise Server 3.X. As a result of this bundling and the resulting free distribution of VisiBroker clients with Netscape Communicator, as of March 1998 roughly 30 million systems[6] (mostly Windows PCs) contain working installations of Java-capable CORBA middleware from Borland. This huge base of installed VisiBroker clients provides a ready interoperability target for CORBA-compliant servers, and for now makes CORBA middleware a *de facto* standard for rapid development of new and legacy distributed applications that are Internet-based and PC-hosted. An example of a distributed application that was developed rapidly by using this installed base of VisiBroker clients in Netscape is Tracker97, [7] a U.S. Department of State application that enables the U.S. and its partner countries to track international movement of hazardous and dual-use materials.

Because of the unusually large Netscape-based deployment of VisiBroker and its overall strong orientation towards the Internet, this appendix provides more detailed information on both the

---

[1] “Visigenic's ORB fills the gaps and helps developers write CORBA apps in Java” (January 1998), http://www.advisor.com/wArticle.nsf/wPages/IA9801.Micks07

[2] “VisiBroker: A Better ORB by Design” (Borland white paper) (October 1997), http://www.visigenic.com/prod/vbrok/wp.html#BOBD (Note: Withdrawn by Borland in March 1997)

[3] “Sun Picks Visigenic As ORB Supplier” (January 1998), http://techweb.cmp.com/internetwk/news/news0109-4.htm

[4] Borland press release on bundling of VisiBroker into Netscape products (June 1997) , http://www.visigenic.com/news/ns697.html

[5] New Netscape policy of free distribution of Netscape Communicator source code (February 1998) , http://home.netscape.com/free.html

[6] Borland press release on VisiBroker (March 1998) , http://www.borland.com/visibroker/press/1998/visi30m.html

[7] NDF Tracker97 Project (February 1998), http://www.corba.org/gov.htm#usdsfg

features of the Netscape versions of VisiBroker and of the full VisiBroker product line. The information in the rest of this appendix was provided in a draft white paper from Visigenic (now part of Borland International), and was current as of January 15, 1995. The original draft white paper has been partially re-arranged, edited, and heavily reformatted, and so should be taken only as non-binding guidance information. For this reason, it is recommended that this appendix be used only as a guide for understanding features and constraints. If you have a specific application in which you wish to use features listed in this appendix, it is therefore recommended that you verify those features directly with Borland International.[1] (Verification is also a good idea due of the current rapid pace of new product releases in the middleware industry.) New data on the VisiBroker product may be obtained from the Borland web site, or from the older Visigenic site.[2]

Developers interested in using Borland VisiBroker should note that VisiBroker 2.X components used in Netscape products are based on an older and less flexible connection and thread model that is no longer used in the most recent (3.X) full releases of VisiBroker products. Also, there is other incompatibility for VisiBroker 2.X and 3.X. The differences between the Netscape (VisiBroker 2.X) and current full commercial release (VisiBroker 3.X) are addressed in the feature comparison table at the end of this appendix.

B.2 VisiBroker Components in Netscape Communicator 4.X

Netscape Communicator 4.X releases can be downloaded free of charge from Netscape, and includes the following VisiBroker component:

- VisiBroker for Java 2.5, Runtime Client

VisiBroker for Java is the first CORBA 2.0 Object Request Broker (ORB) written completely in Java, and it can best be described as a development tool for building, managing, and deploying distributed Java applications that are open, flexible, and interoperable across multiple platforms. VisiBroker for Java enables true interoperable distributed applications for the Internet, Intranets, and enterprise computing environments through a native implementation of CORBA's Internet Inter-ORB protocol (IIOP).

B.3 VisiBroker Components in Netscape Enterprise Server 3.1

The Netscape Enterprise Server 3.1 (which must be purchased from Netscape) includes the following four VisiBroker components:

- VisiBroker for Java 2.5, Runtime Client
- VisiBroker for Java 2.5, Development Kit
- VisiBroker for C++ 2.1, Runtime

[1] Borland International home page (March 1998) , http://www.borland.com/

[2] Original VisiGenic home page, now owned by Borland (March 1998) , http://www.visigenic.com/

- VisiBroker for C++ 2.1, Development Kit

In addition to development kits, the Netscape Enterprise Server also includes a second major VisiBroker product, VisiBroker for C++. VisiBroker for C++ is a complete CORBA 2.0 Object Request Broker (ORB) implementation written in the high-performance C++ language. VisiBroker for C++ serves as a development tool for building, managing, and deploying distributed C++ applications that are open, flexible, and interoperable across multiple platforms. Objects built with VisiBroker for C++ can easily be accessed by Web-based applications, such as applications built with VisiBroker for Java that communicate using the CORBA Internet Inter Orb protocol (IIOP).

B.4 Restrictions on the Use of VisiBroker in Netscape

In their agreement to allow Netscape to use components of their VisiBroker 2.5 and 2.1 products, Borland licensed the distribution of both the development system (that is, the IDL compiler) and the ORB within the Netscape SuiteSpot Server. However, Borland limits use of that development system to one developer on the same system on which the Netscape server was shipped. Also, Borland does not permit the runtime ORB to be used "stand-alone" outside of its Netscape application. The runtime ORB may be used by third party applications that run on the Netscape server box only to invoke remote methods defined by the Netscape server or other Netscape product. Finally, Borland does not allow the runtime ORB to be used by "independent" third party applications or on any on boxes that do not contain the Netscape server.

B.5 VisiBroker Components Not Included With Netscape

The following VisiBroker components are currently available by purchase from Borland, but are not included in any Netscape products:

- VisiBroker for C++ 3.1, Runtime
- VisiBroker for C++ 3.1, Development Kit
- VisiBroker for Java 3.1, Runtime
- VisiBroker for Java 3.1, Development Kit
- Visigenic Gatekeeper
- Object Request Debugger for Java
- Object Request Debugger for C++
- SSL Pack for VB for Java
- SSL Pack for VB for C++
- Naming Service (Java and C++ Versions)

- Event Service (Java and C++ Versions)
- VisiBroker Manager Tool Set
- VisiBroker Integrated Transaction Service
- Advanced Thread Pooling and Connection Management
- Borland Technical Support

B.6 Comparison of Netscape VisiBroker and VisiBroker 3.X

Table 1 compares VisiBroker in Netscape products to the full VisiBroker 3.X release.

Table 1. VisiBroker Features Supported in Netscape and in VisiBroker 3.X

Legend: ● = supported; **-** = not supported; **blank** = not applicable	**In Netscape Browser and Server (VBJ 2.5)**	**In Netscape Server (only) (VBC 2.1)**	**VisiBroker for Java 3.X (full release) (VBJ 3.X)**	**VisiBroker for C++ 3.X (full release) (VBC 3.X)**
CORBA Standards Compliance				
New OMG IDL to Java Mapping 1.0	●		●	
Revised OMG IDL to C++ Mapping 1.1		-		●
CORBA 2.0 Core	●	●	●	●
CORBA 2.0 Interoperability	●	●	●	●
IIOP 1.0	●	●		
IIOP 1.1			●	●
CORBA 2.0 Security: IIOP over SSL			●	●

Legend: ● = supported; **-** = not supported; **blank** = not applicable	**In Netscape Browser and Server (VBJ 2.5)**	**In Netscape Server (only) (VBC 2.1)**	**VisiBroker for Java 3.X (full release) (VBJ 3.X)**	**VisiBroker for C++ 3.X (full release) (VBC 3.X)**
Advanced CORBA Implementation				
IDL to Java compiler	●		●	
Built-in preprocessing	●		●	
IDL to C++ compiler		●		●
Built-in preprocessing	●	-	●	●
Transient and persistent object references: local/global objects	●	●	●	●
Exception handling	●	●	●	●
Extended IDL data types (Notes 1,2)	●	-	●	●
IDL Extensible Structs	●	-	●	●
Basic Object Adapter (BOA)	●	●	●	●
Dynamic object impl. activation (Object Activation Daemon)	-	●	●	●
Dynamic object instance activation (Activators)	-	●	●	●
Implementation Repository	●	●	●	●
Implementation Repository API (IDL interface to OAD)	-	●	●	●
Interface Repository (IR)	●		●	●
Dynamic Invocation Interface (DII)	●	●	●	●
Support for typecodes and anys	●	●	●	●
In-process DII	-	-	●	●
Dynamic Skeleton Interface (DSI)	●	●	●	●
ORB Interface	●	●	●	●
Native IIOP (Internet Inter-ORB Protocol)	●	●	●	●
Caffeine: Java Ease-of-Use				
java2iiop compiler (a.k.a. caffeine compiler)	●		●	
Pass-by-value	●		●	
java2idl compiler	●		●	
URL Naming Service	●		●	●
Gatekeeper: Web Server gateway				
Request forwarding to servers (object servers)	●		●	
Request forwarding to applets (callback objects)	●		●	
Automatic HTTP tunneling (without callbacks)	●		●	
Access to Smart Agents	●		●	
Support for IIOP over SSL (with callbacks)	-		●	
Enhanced integration into firewall environments	-		●	
Multi-home support, configurable ports, proxy object refs	-		●	
Ease of Development				
Easy binding with available object servers (via Smart Agents)	●	●	●	●
TIE programming option	●	●	●	●
Enhanced user documentation	-	-	●	●
Object Request Debugger	-	-	●	●
Object Database Activator (ODA)	-	-	●	●
Caffeine Java tools and services	●		●	

Legend: ● = supported; **-** = not supported; **blank** = not applicable	**In Netscape Browser and Server (VBJ 2.5)**	**In Netscape Server (only) (VBC 2.1)**	**VisiBroker for Java 3.X (full release) (VBJ 3.X)**	**VisiBroker for C++ 3.X (full release) (VBC 3.X)**
Ease of Deployment				
Zero administration object server registry (Smart Agents)	●	●	●	●
Enhanced ease of installation	-	-	●	●
Integration with Windows registry and NT services	-	-	●	●
Security: IIOP over SSL				
IIOP over SSL v3 (Secure Socket Layer)	-	-	●	●
SSL BOA	-	-	●	●
Optional mutual (X509 certificate) client/server authentication	-	-	●	●
Data encryption using RSA BSAFE cryptographic library	-	-	●	●
Scalability and Performance				
Efficient multithreading: use of native OS thread support	●	●	●	●
Optimized client-to-object communication (Smart Binding)	●	●	●	●
Multiple, distributed Smart Agents	●	●	●	●
Dynamic partitioning of object server information	●	●	●	●
Automatic federation of Smart Agents on same LAN	●	●	●	●
Object server load balancing (via Smart Agents)	●	●	●	●
Optimized thread mgmt: thread-per-session/thread pooling	-	-	●	●
Optimized connection management: client and server-side	-	-	●	●
Optimized interprocess invocation (e.g. shared memory)		-		●
High Availability				
Support for replicated object servers	●	●	●	●
Transparent binding of clients with currently available servers	●	●	●	●
Support for client-server resynch after server/network failure	●	●	●	●
Robust system architecture: ORB self-use of fault tolerance	●	●	●	●
Automatic restart of object servers (via OAD)	-	●	●	●
Automatic Smart Agent fail-over	-	●	●	●
Gatekeeper support for applet-server resynchronization	-		●	

Legend: ● = supported; - = not supported; **blank** = not applicable	**In Netscape Browser and Server (VBJ 2.5)**	**In Netscape Server (only) (VBC 2.1)**	**VisiBroker for Java 3.X (full release) (VBJ 3.X)**	**VisiBroker for C++ 3.X (full release) (VBC 3.X)**
Customizable ORB System				
Smart stubs	-	-	●	●
Interceptors	-	-	●	●
In-process interceptors	-	-	●	●
Location Service API (IDL interface to Smart Agents)	-	-	●	●
Dynamic ORB module upgrades	-	-	●	
Plug-in custom communication transports (Note 3)	-	-	●	●
Plug-in custom object adapters (Note 3)	-	-	●	●
Get and set policy and system parameters via:				
Bind management API	●	●	●	●
Buffer management API	-	●	●	●
Thread management API	-	-	●	●
Connection management API	-	-	●	●
Optional use of Gatekeeper	●	-	●	
Optional use of Smart Agents	●	-	●	●

Note 1: The wchar/wstring is only supported on platforms that support UNICODE, such as Windows 95/NT or Solaris 2.6.

Note 2: Due to a change in their implementation, VBJ 2.5 and VBJ 3.0 extended data types are not interoperable. Also, long double is not available on Windows or in Java. Fixed-point decimal is also not available, but is planned.

Note 3: Available in a separate support package.

Appendix C. References

The following list of non-linked (plain text) Internet references summarizes the ones provided in context by the footnotes of this document. The references are in alphabetical order by Universal Resource Locator (URL).

Internet Address (URL)	**Title or Description**
"http://www.quoininc.com/quoininc/COMCORBA.html#COM versus CORBA: A Decision Framework"	COM versus CORBA: A Decision Framework (April 1998)
http://corbanet.dstc.edu.au/	The ORB Interoperability Showcase (OMG, June 1997)
http://home.netscape.com/free.html	New Netscape policy of free distribution of Netscape Communicator source code (February 1998)
http://premium.microsoft.com/msdn/library/techart/html/cpptocom.htm	"From CPP to COM" (Microsoft, October 1995)
http://stsc.hill.af.mil/crosstalk/1997/feb/corba.html	An Introduction to CORBA (MITRE, February 1997)
http://techweb.cmp.com/internetwk/news/news0109-4.htm	"Sun Picks Visigenic As ORB Supplier" (January 1998)
http://www.advisor.com/wArticle.nsf/wPages/IA9801.Micks07	"Visigenic's ORB fills the gaps and helps developers write CORBA apps in Java" (January 1998)
http://www.beasys.fr/products/objectbroker.htm	"BEA ObjectBroker presentation" (March 1998)
http://www.beasys.fr/products/quotes.htm	Press quotes on BEA products: TUXEDO, Jolt, ObjectBroker, MessageQ (March 1998)
http://www.borland.com/	Borland International home page (March 1998)
http://www.borland.com/visibroker/press/1998/visi30m.html	"Borland's VisiBroker ORB Surpasses 30 Million Licenses Deployed Worldwide" (March 1998)

Internet Address (URL)	**Title or Description**
http://www.borland.com/visibroker/press/1998/visi30m.html	Borland press release on VisiBroker (March 1998)
http://www.camb.opengroup.org/tech/dce/info/faq-mauney.html	DCE Frequently Asked Questions (August 1997)
http://www.camb.opengroup.org/tech/dce/info/faq-mauney.html#Q1_01	DCE FAQ Question 1.01: What is DCE? (August 1997)
http://www.camb.opengroup.org/tech/dce/info/faq-mauney.html#Q1_03	DCE FAQ Question 1.03: What platforms support DCE? (August 1997)
http://www.camb.opengroup.org/tech/dce/info/faq-mauney.html#Q1_08	DCE FAQ Question 1.08: What is the relationship between DCE and CORBA? (August 1997)
http://www.camb.opengroup.org/tech/dce/info/faq-mauney.html#Q2_19	DCE FAQ Question 2c-01: Will Windows NT communicate with DCE?(August 1997)
http://www.camb.opengroup.org/tech/dce/info/faq-mauney.html#Q2_20	DCE FAQ Question 2c-02: Can I use DCE from C++? (August 1997)
http://www.camb.opengroup.org/tech/dce/info/papers/osf-dce-ds-1195.htm	OSF DCE 1.2.1 New Features (November 1995)
http://www.cariplo.it/HomeBanking.htm	Home Banking web site (Cariplo Bank in Italy) implemented using DCOM technology (October 1997)
http://www.corba.org/gov.htm#usdsfg	NDF Tracker97 Project (February 1998)
http://www.corba.org/index.html	“CORBA Success Stories” (OMG, November 1997)
http://www.cs.wustl.edu/~schmidt/submit/Paper.html	“DCOM and CORBA Side by Side, Step by Step, and Layer by Layer” (September 1997)
http://www.iona.com/news/pressroom/msoft.html	IONA CORBA-COM Bridge announcement: OrbixCOMet™ Desktop (January 1998)

Internet Address (URL)	**Title or Description**
http://www.iso.ch/cate/d25486.html	ISO/IEC DIS 14750 (March 1998)
http://www.micromail.com/titles/7666.html	Review of book “Instant CORBA” (Orfali & Harkey, February 1998)
http://www.microsoft.com/	Microsoft home page (March 1998)
http://www.microsoft.com/com/	COM: Component Object Model (Microsoft, February 1998)
http://www.microsoft.com/com/activex.htm	ActiveX™ Controls (January 1998)
http://www.microsoft.com/com/complus.htm	COM+ (November 1997)
http://www.microsoft.com/com/dcom.htm	DCOM (January 1998)
http://www.microsoft.com/com/mts.htm	MTS (March 1998)
http://www.microsoft.com/com/slides/complus.zip	“COM+: Building on the Success of the Component Object Model” (October 1997)
http://www.microsoft.com/com/wpaper/dcomhome.zip	“DCOM Cariplo Home-Banking Case Study” (Microsoft, Nov 1997)
http://www.microsoft.com/com/wpaper/dcomsol.zip	“DCOM Solutions in Action” (November 1997)
http://www.microsoft.com/intdev/com/	COM Technologies home page (March 1998)
http://www.microsoft.com/ntserver/guide/dcom.asp	“DCOM Business Case” (Microsoft, March 1998)
http://www.microsoft.com/ntserver/library/dcomtec.exe	“DCOM Technical Overview” (Microsoft, November 1997)
http://www.netscape.com/newsref/pr/newsrelease538.html	Netscape press release on 68 million users of Communicator Marketplace (December 1997)
http://www.omg.org/	OMG Home Page (February 1998)
http://www.omg.org/about/wicorba.htm	“What IS CORBA????” (OMG, October 1997)

Internet Address (URL)	**Title or Description**
http://www.omg.org/cgi-bin/memlist.pl	OMG Member Company Listings (March 1998)
http://www.omg.org/news/wpjava.htm	Java, RMI and CORBA (OMG, June 1997)
http://www.opengroup.org/	The Open Group home page (March 1998)
http://www.opengroup.org/public/tech/dce/nov96/OOminutes.html#dcom	DCOM-DCE meeting minutes (November 1996)
http://www.opengroup.org/public/vsorb/	CORBA Validation - the VSOrb Test Technology (The Open Group, January 1998)
http://www.orbix.com/products/fulllist.html	Orbix Products (IONA, April 1998)
http://www.qds.com/people/apope/Corba/ap_resources.html	Object Resource Lists (by author of “The CORBA Reference Guide”) (Alan Pope, February 1998)
http://www.realtime-os.com/noteworthy/rt-dcom.html	“Real-Time DCOM's Immediate Future Now Uncertain” (Jensen, November 1997)
http://www.softwareag.com/corporat/default.htm	Software AG home page (March 1998)
http://www.techweb.com/se/directlink.cgi?CRN19970721S0060	“JavaSoft concedes it's not an all-Java world” (Computer Reseller News, July 1997)
http://www.techweb.com/se/directlink.cgi?CRN19970728S0027	“OMG pushes to become standards body, touts CORBA” (Computer Reseller News, July 1997)
http://www.techweb.com/se/directlink.cgi?CRN19971103S0197	“Software AG's Latest DCOM Runs On IBM OS/390” (Computer Reseller News, November 1997)
http://www.techweb.com/se/directlink.cgi?IWK19960603S0076	Traditional middleware players embrace object-request brokers (Information Week, June 1996)

Internet Address (URL)	**Title or Description**
http://www.techweb.com/se/directlink.cgi?IWK19960729S0051	"Microsoft Complies With DCE – For Now" (Information Week, 29 July 1996)
http://www.visigenic.com/	Original Visigenic home page, now owned by Borland (March 1998)
http://www.visigenic.com/news/ns697.html	Borland press release on bundling of VisiBroker into Netscape products (June 1997)
http://www.visigenic.com/prod/vbrok/wp.html (Note: Withdrawn by Borland in March 1998)	"Distributed Object Computing in the Internet Age" (Visigenic, October 1997)
http://www.visigenic.com/prod/vbrok/wp.html#BOBD (Note: Withdrawn by Borland in March 1998)	"VisiBroker: A Better ORB by Design" (Borland white paper) (October 1997)
http://www.w3.org/TR/WD-DOM/level-one-core-971209.html	Document Object Model (Core) Level 1(W3C, December 1997)
http://www-lc.llnl.gov:8080/library/all/SG-2409	"Kerberos User's Guide SG-2409 9.0" (December 1997)

Appendix D. Glossary

ACL	Access Control List
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
ATL	ActiveX Template Library
ATM	Asynchronous Transfer Mode (high-speed networking)
BOA	Basic Object Adapter (CORBA)
CDR	Common Data Representation (CORBA)
CDS	Cell Directory Service (DCE)
CF	Common Facilities (CORBA)
CIOP	Common Inter-ORB Protocol (DCE)
CLI	Call-Level Interface
CLSID	Class Identifier
COE	Common Operating Environment
COM	Component Object Model (COM/DCOM)
COM+	Common Object Model Plus (COM/DCOM)
CORBA	Common Object Request Broker Architecture
COSS	Common Object Services Specification (CORBA)
COTS	Commercial Off-The-Shelf
DCE	Distributed Computing Environment
DCE RPC	DCE Remote Procedure Call
DCOM	Distributed Component Object Model
DEC	Digital Equipment Corporation
DII	Defense Information Infrastructure (DoD)
DII	Dynamic Invocation Interface (CORBA)
DLL	Dynamic Link Library
DNS	Domain Name Service
DOC	Distributed Object Computing
DRDA	Distributed Relational Database Architecture
DSI	Dynamic Skeleton Interface (CORBA)
DSM	Distributed System Management
DSOM	Distributed System Object Model
FAQ	Frequently Asked Questions (Internet)
FDDI	Fiber Distributed Data Interface
GDS	Global Directory Service
GIOP	General Inter-ORB Protocol (CORBA)
GUI	Graphical User Interface
GUID	Globally Universal Identifier
GUID	Globally Universal Identifier

HP/UX	Hewlett Packard Unix
HSN	High Speed Networks
HTML	Hypertext Markup Language (Internet)
HTTP	Hypertext Transfer Protocol (Internet)
IBM	International Business Machines
IDL	Interface Definition Language (DCE, DCOM, CORBA)
IID	Interface Identifier
IIOP	Internet Inter-ORB Protocol (CORBA)
IOP	Inter-Orb Protocol
IOR	Interoperable Object Reference
IP	Internet Protocol
IPC	Interprocess Communication
IPX	Internetwork Packet Exchange
IR	Interface Repository
ISAPI	Internet Server API
ISO	International Standards Organization
LAN	Local Area Network
LPC	Local Procedure Call
MFC	Microsoft Foundation Classes (COM/DCOM)
MIDL	Microsoft Interface Definition Language (COM/DCOM)
MS-DOS	Microsoft Disk Operating System
MS-RPC	Microsoft RPC (COM/DCOM)
MVS	Multiple Virtual System
NDR	Network Data Representation (DCOM)
NetBIOS	Network Basic Input/Output System
NOS	Network Operating System
NSID	Name Service Interface Daemon
NTLM	NT LAN Manager
OA	Object Adapter
OCX	OLE Control (COM/DCOM)
ODBC	Open DataBase Connectivity
ODL	Object Description Language (COM/DCOM)
OLE	Object Linking and Embedding (COM/DCOM)
OMA	Object Management Architecture (CORBA)
OMG	Object Management Group (CORBA)
OODCE	Object Orientated Distributed Computing Environment
OOUI	Object Orientated User Interface
ORB	Object Request Broker (CORBA)
ORPC	Object Remote Procedure Call

OS	Object Services (CORBA)
OS	Operating System
OSF	Open Software Foundation (now called The Open Group)
PERL	Pathologically Eclectic Rubbish Lister (no kidding)
RDA	Remote Database Access
RFP	Request For Proposal
RMI	Remote Method Invocation (Java)
ROT	Running Object Table
RPC	Remote Procedure Call (DCE)
SCO	The Santa Cruz Operation
SDK	Software Developer’s Toolkit
SGML	Standard Generalized Markup Language
SII	Static Invocation Interface (CORBA)
SOM	System Object Model
SPX	Sequenced Packet Exchange
SSI	Static Skeleton Interface (CORBA)
SSPI	Security Support Provider Interface
TCL	Tool Command Language
TCP	Transmission Control Protocol (Internet)
TP	Transaction Processing
UDP	User Datagram Protocol
URL	Uniform Resource Locator (Internet)
UTC	Coordinated Universal Time
VINES	Virtual Networking System
VM	Virtual Machine
WAN	Wide Area Network (networks)
WWW	World Wide Web (Internet)
XML	eXtensible Markup Language (Internet)