IN

Intelligent Network

&

TMN

Telecom Management Network

Comparing DCE and CORBA

(c) Iabc
Information GmbH
Kommunikation und Beratung

abc Information GmbH
Phone:+49 30 383983-0
Fax: +49 30 383983-115
eMail: info@Berlin.Broker.de

Iabc
nformation
GmbH Kommunikation
und Beratung

Comparing DCE and CORBA

D.Heide / dhd@Berlin.Broker.de
C.Lang / cla@Berlin.Broker.de

Abstract

Many people perceive DCE and CORBA as competing technologies. Indeed, both support the construction and integration of client-server applications in heterogeneous distributed environments. Comparisons typically focus on differences between individual capabilities or on differences between the maturity of specifications and products that conform to them. There is a fundamental difference between DCE and CORBA, however, that we feel far overshadows either of these criteria as a basis for selecting a distributed computing platform. This document summarizes the main features of DCE and CORBA, presents what we feel is the most important difference between them, discusses differences between individual capabilities and the maturity of both specifications and products, and concludes with our view of how an organization should select the technology most appropriate to its distributed computing goals.

Table of Contents

List of Figures

Comparing DCE and CORBA

1. Introduction

Many people perceive the Open Software Foundation (OSF) Distributed Computing Environment (DCE) and Object Management Group (OMG) Common Object Request Broker Architecture (CORBA) as competing technologies. Indeed, both support the construction and integration of client-server applications in heterogeneous distributed environments, and both do so in very similar ways with very similar capabilities.

Figure 1 depicts at a very high level the manner in which service requests are handled by DCE and CORBA. Both define an Interface Definition Language (IDL).

DCE IDL is based on the C programming language; CORBA IDL is based on C++. IDL is used to define the interface that a server implements, that is, the set of

services that clients may request of it. Both DCE IDL and CORBA IDL compile into client and server stubs. A client application calls a client stub to request a service. The client stub interfaces to the runtime system, which eventually invokes server code that implements the requested service through the appropriate server stub.

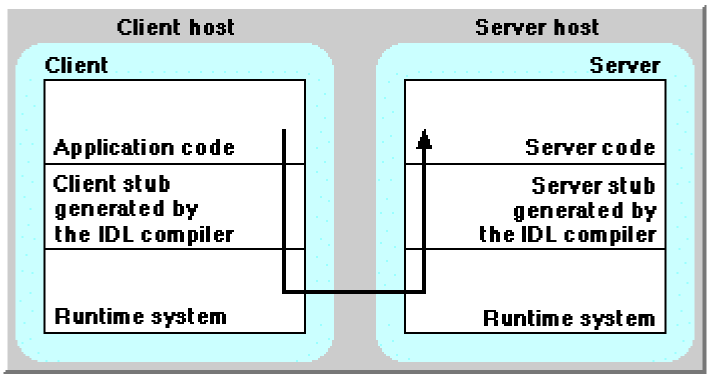

Figure 1. Requesting Services Using DCE or CORBA

The transmission of service requests and responses between clients and servers is handled by both DCE and CORBA so that applications need not deal with concerns like: where clients and servers are located on the network; differences between hardware platforms, operating systems, and implementation languages (for example, data formats or calling conventions); networking protocols; and others.

There are many other similarities between DCE and CORBA, as one would expect. However, our purpose here is not to present a litany of the similarities between these two technologies. It is our intention to examine how they differ from one another.

Comparisons of DCE and CORBA are commonplace; however, they typically focus either on differences between individual capabilities or on differences between the relative maturity of specifications and of products that conform to them. There is a fundamental difference between DCE and CORBA, however, that we feel far overshadows either of these criteria as a basis for selecting a platform for distributed computing.

The purpose of this document is to discuss the differences between DCE and CORBA on all of these levels. Before doing so, we summarize the features that DCE and CORBA offer. After presenting what we feel is the most important difference between the two technologies, we discuss the differences between the individual capabilities they provide and the relative maturity of both the specifications and the products that conform to them. Then we present our view of how an organization should select the technology most appropriate for its distributed computing goals.

2. Overview of DCE

DCE supports the construction and integration of C-based client/server applications in heterogeneous distributed environments. Figure 2 shows the various elements that comprise the DCE architecture.

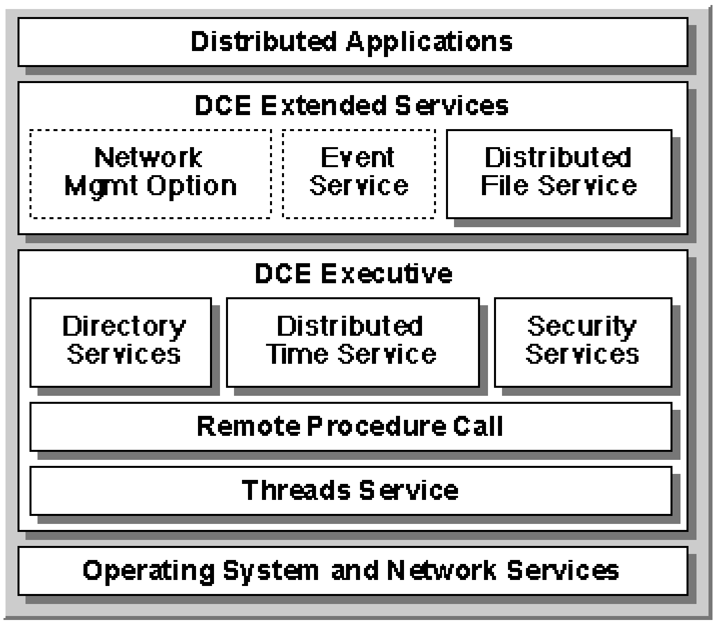

Figure 2. OSF DCE Architecture

DCE Executive

The DCE Executive consists of the following components:

Security Services that support authentication (using Kerberos V5, clients and servers can prove who they are), authorization (servers can use access control lists to determine whether a client is authorized to obtain a given service), integrity (checksums guarantee that information is received as transmitted), and privacy (DES encryption protects sensitive information from disclosure during transmission between a client and server).

Directory Services that support local DCE administration domains called cells and inter-cell name resolution. These services consist of a Cell Directory Service (CDS), Global Directory Service (GDS, which uses X.500), Domain Name Service (DNS, not supplied, but used by DCE), and a Global Directory Agent (GDA).

A Distributed Time Service (DTS) that synchronizes clocks on all hosts in a DCE cell, as well as between cells. DTS uses the UTC standard and is interoperable with NTP.

A Remote Procedure Call (RPC) mechanism by which clients invoke procedures in servers. A client may use directory services to bind to a particular server of interest at run time, and the client and server may use security services to guarantee desired levels of authentication, authorization, integrity, and privacy.

The RPC mechanism insulates clients from details of where servers are located on the network, the types of hardware and operating system platforms on which they execute, differences in data representations between client and server platforms, and the particular network transports in use.

A Threads package based on POSIX 1003.4a (draft 4) that supports the creation and management of multiple threads of control within a client or server. A multi-threaded client may perform additional work (perhaps invoke additional RPCs) while one RPC is pending. The dispatcher that receives RPCs at a server and invokes the appropriate RPC handlers is already multi-threaded, automatically permitting DCE servers to handle multiple RPCs concurrently. The maximum number of concurrent RPCs at a server is easily configured by the developer, who is also responsible for ensuring the thread-safe behavior of all RPC handlers.

DCE Extended Services

DCE Extended Services currently consist of the Distributed File Service (DFS) alone. The DFS is a DCE application that implements a single logical filesystem that is available (through Directory Services) throughout an entire cell and across cell boundaries. DFS supports replication of files for availability and fault-tolerance and log-based recovery from hardware failures.

The remaining components that appear in Figure 2 as DCE Extended Services are vestiges of OSF’s Distributed Management Environment (DME), which failed to obtain vendor buy-in and has ceased to exist as an entity in its own right. These components are shown in dashed boxes because it is not clear at this time whether they will be incorporated into future releases of DCE. The Network Management Option would provide a means for management applications to access management information using standard network management protocols (CMIP and SNMP). The Event Service would provide a common way for system and user applications to generate, forward, filter, and log events.

The Bibliography lists sources of additional information about DCE.

3. Overview of CORBA

CORBA supports the construction and integration of object-oriented software components in heterogeneous distributed environments. Figure 3 shows the various elements that comprise the CORBA architecture.

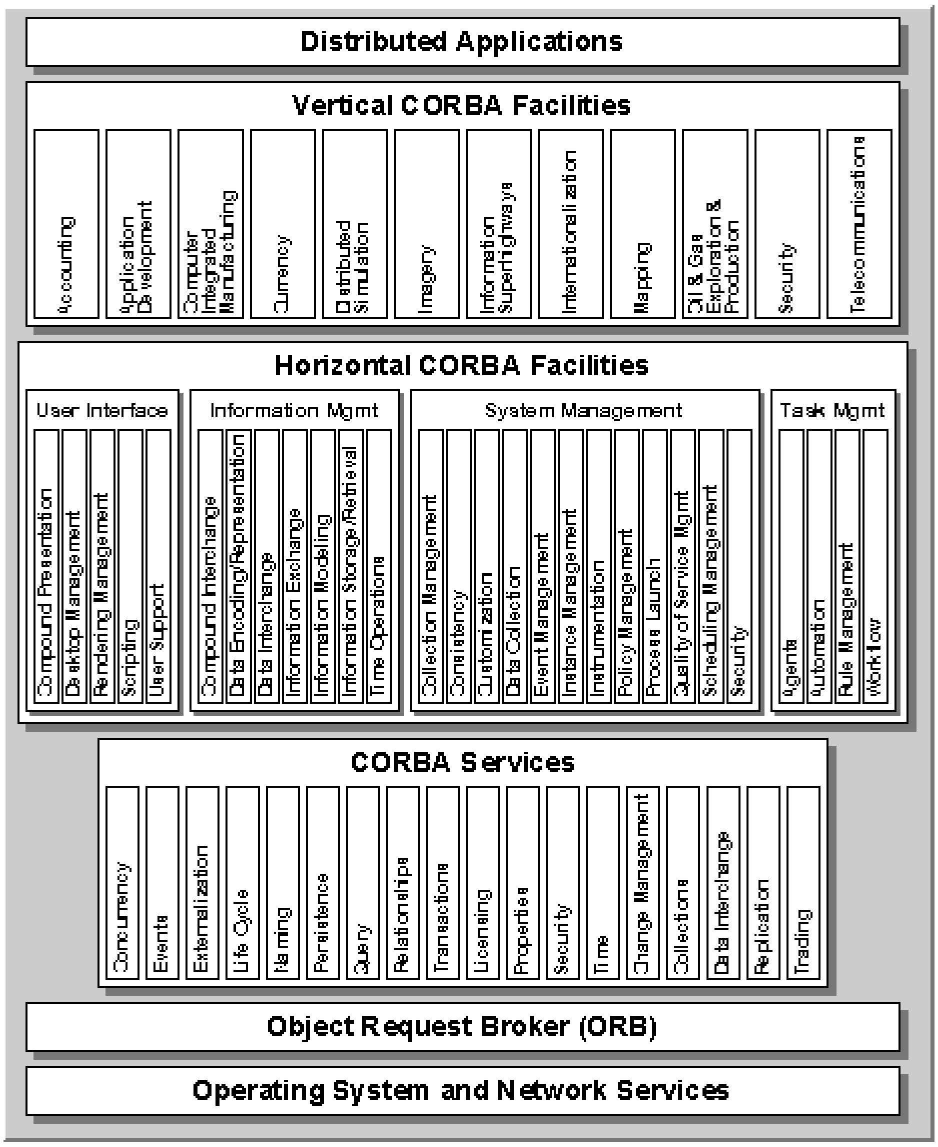

Figure 3. OMG CORBA Architecture

Object Request Broker

An Object Request Broker (ORB) provides a communication infrastructure for invoking operations on objects transparently with respect to where they are located on the network, the types of hardware and operating system platforms on which they execute, differences in data representations between platforms, the languages in which objects are implemented, and network transports used to communicate with them. CORBA specifies all of the functions that must be provided by an ORB and a set of standard interfaces to those functions.

CORBA Services

CORBA Services are services that are essential for implementing objects. The CORBA Services that have been specified thus far by OMG include: A Concurrency Control Service that protects the integrity of an object’s data when multiple requests to the object are processed concurrently. An Event Service that supports the notification of interested parties when program-defined events occur.

An Externalization Service that supports the conversion of object state to a form that can be transmitted between systems by a means other than a request broker.

Life Cycle Services that support creation, copying, moving, and destruction of objects.

A Naming Service that permits object references to be retrieved through associations between names and objects, and for those associations to be created and destroyed.

A Persistent Object Service that supports the persistence of an object’s state when the object is not active in memory and between application executions.

A Query Service that supports operations on sets and collections of objects that have a predicate-based, declarative specification and may result in sets or collections of objects.

A Relationship Service that provides for creating, deleting, navigating, and managing relationships between objects (for example, a containment relationship between a „folder“ object and the „document“ objects that are considered to be „in“ that folder).

A Transaction Service that provides support for ensuring that a computation consisting of one or more operations on one or more objects satisfies the requirements of atomicity (if a transaction is interrupted by a failure, any partially completed results are undone), isolation (transactions are allowed to execute concurrently, but the results are the same as if they executed serially), and durability (if a transaction completes successfully, the results of its operations are never lost, except in the event of catastrophe).

CORBA Services that are in the process of being specified and are expected to be completed in 1995 include:

A Licensing Service that will control and manage remuneration of suppliers for services rendered.

A Property Service that will support the association of arbitrary named values (the dynamic equivalent of attributes) with an object.

A Security Service that will support authentication, authorization, integrity, and privacy to degrees, and using mechanisms, that are yet to be determined.

A Time Service that will provide synchronized clocks to all objects, regardless of their locations.

CORBA Services that are not yet in the process of being specified include:

A Change Management Service that would support the identification and consistent evolution of configurations of objects.

A Collection Service that would support the creation and manipulation of collections of objects.

A Data Interchange Service that would support the exchange of data between objects.

A Replication Service that would provide for the explicit replication of objects in a distributed environment (for the purpose of availability or fault tolerance) and for the management of consistency of replicated copies.

A Trader Service that would provide a matchmaking service between clients seeking services and objects offering services.

CORBA Facilities

CORBA Facilities are useful for constructing applications across a wide range of application domains. They are divided into Horizontal CORBA Facilities, which are typically more user-oriented, and Vertical CORBA Facilities, which support specific application domains. The Horizontal CORBA Facilities currently identified by OMG are grouped into four areas:

User Interface Facilities, which include Compound Presentation, Desktop Management, Rendering Management, Scripting, and User Support Facilities.

Information Management Facilities, which include Compound Interchange, Data Encoding and Representation, Data Interchange, Information Exchange, Information Modeling, Information Storage and Retrieval, and Time Operations Facilities.

System Management Facilities, which include Collection Management, Consistency, Customization, Data Collection, Event Management, Instance Management, Instrumentation, Policy Management, Process Launch, Quality of Service Management, Scheduling Management, and Security Facilities.

Task Management Facilities, which include Agent, Automation, Rule Management, and Workflow Facilities.

No Horizontal CORBA Facilities have been specified as yet. The first CORBA Facilities RFP was issued in October of 1994 for Compound Document Facilities (Compound Presentation and Compound Interchange). Further RFPs will be forthcoming in 1995.

The Vertical CORBA Facilities currently identified by OMG (as a result of responses from interested vertical market segments to an OMG RFI) are: Accounting, Application Development, Computer Integrated Manufacturing, Currency, Distributed Simulation, Imagery, Information Superhighways, Internationalization, Mapping, Oil and Gas Exploration and Production, Security, and Telecommunication. No Vertical CORBA Facilities have been specified as yet.

The Bibliography lists sources of additional information about CORBA.

4. The Fundamental Difference Between DCE and CORBA

The fundamental difference between DCE and CORBA is that DCE was designed to support procedural programming, while CORBA was designed to support object-oriented programming. Object-oriented programming environments are usually characterized by their support for:

- Encapsulation of data and the functions that manipulate the data into objects. This enforces data hiding, since the only way to access an object's data is through the operations in the object's public interface.
- Abstraction of common features shared by objects into classes. A class definition describes the data associated with each instance of the class, defines the set of operations that can be invoked on an instance of the class, and prescribes the functions that are executed in response to requests for those operations. Inheritance of interfaces and implementations. This is the mechanism that supports the specialization or refinement of classes into subclasses. It is also one example of reuse in object-oriented programming.
- Polymorphism, which is the ability for a request for a specific operation to be handled differently depending on the type of object on which it is invoked. For example, subclasses of a common superclass may override functions defined by the superclass to differentiate how instances of the subclasses and the superclass behave.

In addition to these common characteristics, object-oriented programming environments usually support a style of programming in which:

- Not only new objects, but new classes may be created at runtime.
- Late binding of operation invocations to function calls allow programs to be written without regard for the types of objects they will manipulate.
- Object references are passed among objects freely, which can lead to dynamic patterns of request invocations among objects of arbitrary types (by virtue of late binding).
- Once defined, objects and classes may be reused or refined in subsequent applications, extending the usefulness of object implementations across multiple applications.

CORBA supports all of the common characteristics and programming styles described above, with the possible exception of creating new classes at runtime. We believe the creation of new classes at runtime may be enabled by recently adopted CORBA 2.0 specifications. In particular, we refer to additional Interface Repository operations for adding information to a repository at runtime and a Dynamic Skeleton Interface, which supports the implementation of servers capable of handling requests for objects whose types are unknown at compile time.

Distributed procedural programming environments such as DCE support a different set of capabilities than those described above. The basic approach to distributing a procedural program is to:

1.Partition the program's data and the functions that manipulate the data into servers;

2.Distribute those servers across multiple hosts; and

3.Change function calls to RPCs, as appropriate.

This style of programming does encapsulate data and functions in servers, because the only way to access the data is through the server's RPC interface. It does not protect any of the data within a server from access by any of the functions in the server, however. Nor does it support abstraction, inheritance, polymorphism, or the dynamic style of programming described above.

Comparing DCE and CORBA

DCE does have additional capabilities that begin to overlap with traditional capabilities of object-oriented systems:

- A DCE client can determine at runtime the specific servers to which it will bind and make RPCs (although the interfaces supported by those servers must be fixed at compile time).
- A DCE server may generate what are called object UUIDs (universal unique identifiers) to denote different resources managed by the server. A client that does an RPC to the server can use an object UUID to identify a specific resource. For example, a print server might generate object UUIDs for the different printers it controls, and a client submitting a print request would specify the desired printer.
- A DCE server may also generate what are called object type UUIDs, associate each object UUID with an object type UUID, and register a separate set of RPC handlers for each object type UUID. When a client does an RPC to the server and specifies an object UUID, the specific function that is invoked in the server depends on the object type with which the object UUID is associated. For example, our print server might associate one object type UUID with RPC handlers that support line printers and another object type UUID with a corresponding set of RPC handlers that support PostScript printers.
- One can argue that these features support some of the characteristics of object-oriented systems described above. The object type UUID does indeed support some form of abstraction and polymorphism. The important distinction to be made here is that procedural programming is not object-oriented programming, although it can be used to implement an object-oriented programming environment, just as C is often used to implement C++ (that is to say, C++ is often pre-processed into C before compilation).
- The analogy between C and C++ is a good one. Many CORBA-conformant ORB vendors (DEC, HP, and IBM, for example) are implementing ORBs on top of DCE. The desirability of the object-oriented approach even within the DCE community is evidenced by several efforts to provide C++ interfaces to DCE [Dilley, Leddy, Mock, Viveney].

The OODCE RFC [Dilley] relates DCE and CORBA as follows:

- „...our work focuses on integrating C++ within the existing DCE system infrastructure, simplifying the use of the DCE object model.
- „OMG CORBA will address creating distributed object systems in C++; some implementations will run on top of DCE. CORBA IDL provides for interface inheritance, which DCE IDL is lacking, and provides a more C++-like syntax for interface specification. The CORBA runtime environment provides a richer set of object invocation and passing than the DCE environment.
- „We suggest that our work may assist in the migration from DCE to CORBA by providing an intermediate C++-based distributed object system until CORBA implementations are widely available.“

5. Differences Between Individual Capabilities

Although we feel that the most significant difference between DCE and CORBA is in the style of programming each is intended to support, there are inevitable differences between the individual capabilities they provide. For example:

DCE supports several useful datatypes that CORBA does not support:

A varying array in DCE is an array of fixed size, of which only part is passed between client and server; however, the entire array is allocated at the server, which may return more array elements than were passed to it. CORBA has no equivalent datatype; however, equivalent behavior may be obtained using a CORBA sequence.

DCE pipes permit very large parameter values to be passed in a series of smaller blocks so that data transmission and processing may be pipelined.

CORBA supports no corresponding mechanism for dealing with very large parameter values; the programmer can implement pipelining only by breaking up what might be a single operation conceptually into a series of operations.

DCE supports contexts, which are a mechanism for maintaining server state during a series of logically related requests from a single client. (For example, the client might be issuing a series of RPCs to retrieve a set of records from a database one at a time.) The server state is passed to the client as an opaque context structure; the client includes the context in subsequent RPCs, and the server uses and modifies the state information in the context, as appropriate. DCE provides support for contexts directly in client and server stubs. CORBA has no corresponding mechanism; the programmer is responsible for managing contextual information explicitly. (It should be noted that DCE contexts have no relationship to CORBA contexts, which are used to carry user preferences along with a request to an object.)

DCE fully supports the use of pointers as, and within, operation parameters. An operation that is defined in DCE IDL may take a pointer as a parameter. Of course, such a parameter is not actually passed beween a DCE client and server as an address, which is generally not meaningful in a distributed environment. The DCE run-time system packages up the value the address points to as part of the process of marshalling the request in preparation for transmission to the server. Furthermore, if a parameter is a complex structure that contains pointers, the DCE run-time system will package up all of the values addressed by those pointers during marshalling, transmit them to the server, and reconstitute the complex parameter with pointers in the server.

CORBA does not support the use of pointers as, or within, operation parameters. The set of CORBA IDL basic datatypes and constructs for building complex datatypes do not include pointers (although they may be implemented using pointers). This means that a programmer may pass complex structures that contain pointers as operation parameters in two ways. The programmer may write additional code to marshall the actual values addressed by pointers into pointer-free datatypes in the client and then reconstitute the complex parameter with pointers in the server. Alternatively, the programmer may redefine the complex data structure as a collection of one or more objects, since CORBA does support complex structures composed of objects.

CORBA supports an „any“ data type that DCE does not support. This permits a value of an arbitrary type to be passed between a client and server. The value carries with it a code that indicates its type.

DCE IDL does not support interface inheritance and defines a flat namespace. CORBA IDL supports multiple inheritance and defines a hierarchical namespace.

CORBA defines an Interface Repository that contains information equivalent to that in IDL files and can be queried at runtime. DCE defines no such repository.

In addition to a static (stub) interface, CORBA defines a dynamic invocation interface that can be used by a client to invoke an arbitrary operation on an arbitrary object type at runtime (supporting the late binding of operation invocations to function calls). DCE defines no such invocation mechanism - the appropriate RPC stubs must be linked into a DCE client.

DCE servers must be brought up by means external to DCE. CORBA also supports automatic server activation (in other words, an ORB will bring up a server if it is not up when a request is directed to it). The services that DCE provides to an application are rather limited compared to the components of the CORBA architecture, which covers a much broader spectrum of application support services and provides the developer with a much richer set of capabilities on which to build.

6. Maturity of the Specifications and Conformant Products

DCE Ontogeny

OSF released DCE 1.0, including all the DCE components described above, in 1992. In addition to specifications, OSF delivered a reference implementation to vendors. By 1993, the first DCE implementations were available. A partial list of current DCE vendors includes:

AT&T
Bull
DEC
Gradient Technologies
Hitachi
HP
IBM
Pyramid
SCO
Siemens Nixdorf
SGI
Stratus
Tandem
Transarc

OSF released DCE 1.1 to vendors late in 1994, and vendors are expected to release their implementations by mid-1995. DCE 1.1 provides for enhanced auditing and the X/Open-approved GSSAPI and Extended Registry Attributes (ERA). These will permit enterprises to bring non-DCE domains under the DCE security umbrella for common enterprise-wide security administration.

OSF expects to release DCE 1.2 to vendors in two phases: DCE 1.2.1 in November of 1995 and DCE 1.2.2 in July of 1996. The primary theme for this release will be to remove obstacles to broad end-user deployment of DCE. The focus areas for this release are currently administration, ease of programming, support for legacy systems, scalability, security, and DFS enhancements.

While DCE continues to evolve, a number of large end-user organizations have committed to basing their next-generation, enterprise-wide information systems on DCE.

CORBA Ontogeny

OMG adopted the CORBA 1.0 specification in late 1991. This included no CORBA Services or CORBA Facilities, nor did OMG release a reference implementation. By mid-1993, the first CORBA-conformant ORBs were available; by late 1993, the first CORBA Services were specified by OMG. A partial list of vendors who are currently selling or developing CORBA-conformant ORBs includes:

DEC
Expersoft
HP
IBM
IONA Technologies

Comparing DCE and CORBA

Postmodern Computing
SunSoft

At this time, OMG has specified only nine of the CORBA Services and none of the CORBA Facilities shown in Figure 3. Furthermore, few implementations of the CORBA Services that have been specified are available yet. Nonetheless, several large end-user organizations are prototyping elements of their next-generation, enterprise-wide information systems using CORBA.

Comparison

Clearly, DCE specifications as a whole are closer to completion than CORBA specifications. This is not surprising, given the ambitious task OMG has undertaken;

Figure 3 is impressive in the breadth and depth of services it depicts. It is another matter entirely whether OMG will be completely successful at defining all of the components in Figure 3. As for the maturity of DCE implementations and CORBA-conformant ORBs, there are mature examples of each.

Let us draw the following distinctions:

The completeness of OSF and OMG specifications is one issue. DCE is certainly closer to being fully specified than CORBA (including all of the CORBA Services and CORBA Facilities).

The maturity (that is, stability) of existing specifications is a second issue. Though all of the existing specifications are subject to revision, both OSF and OMG are attempting to do so only when necessary and to maintain backward compatibility whenever possible. In this respect, both the DCE specification and the CORBA specification (the ORB only, not including CORBA Services or CORBA Facilities) are reasonably mature (although the CORBA specification may

require greater revision in view of the large number of services yet to be specified).

The completeness of conformant products as far as implementing all of the existing specifications is a third issue. In this respect, there are reasonably complete implementations of both DCE and CORBA (the ORB only).

The maturity (that is, stability) of conformant products is the fourth and final issue. In this respect, there are mature implementations of both DCE and CORBA-conformant ORBs.

Regarding Interoperability

It is common to hear that DCE achieved true interoperability in 1992 with DCE 1.0, while CORBA interoperability was only recently specified and has yet to be delivered. Typically, each vendor ports DCE to a specific platform, and interoperability with the reference implementation ensures interoperability among vendors' implementations.

OMG does not deliver reference implementations, only specifications. CORBA 1.0 did not address interoperability between ORBs because it was considered premature pending experience implementing basic ORB functions. As a result, many vendors implemented ORBs on a selection of platforms, providing interoperability across all of the platforms supported by a given vendor. In fact, some of the ORBs that are currently available run on over a dozen different platforms and support interoperable clients and servers across all of those platforms at this time.

Interoperability between ORBs (that is, between ORB vendors) will be a reality in 1995. The Internet Inter-ORB Protocol (IIOP) specified by OMG in December of 1994 [CORBA2] may be supported via bridges that can be developed by end-users or third parties without proprietary information about an ORB or modifications to the ORB. In addition, SunSoft recently made available on the OMG server a public domain implementation of the major components needed to implement an IIOP bridge. We therefore expect to see widespread interoperability among ORBs via IIOP by late 1995 or early 1996. We expect this even with ORBs that use DCE for intra-ORB communication, although interoperation between DCE-based ORBs may be provided via a DCE-specific protocol specified by OMG as well.

7. Choosing the Right Technology

We have stated that the most important difference between DCE and CORBA is their programming paradigms: DCE was designed to support distributed procedural programming, while CORBA was designed to support distributed object-oriented programming. Religious convictions must be tempered with more pragmatic concerns, however.

Comparing the individual capabilities that each gives the developer, CORBA provides a much richer and more powerful environment. Most of the CORBA services have yet to be specified, however. Current DCE implementations provide services that are not yet provided by CORBA implementations (in a standard way): CDS (Naming), Security, DTS (Time), and Threads. The CORBA Naming Service has been specified, and partial implementations are available now. The CORBA Security and Time Services will not be specified until late 1995, and we do not expect to see them available in CORBA implementations until late 1996. OMG has no plans at this time to specify a Threads Service.

If these services are essential to a development effort, the alternatives at present are to use CORBA and implement the required services yourself or use DCE. If you implement to OMG-specified interfaces, your service implementations will eventually become superfluous and code that uses those services will port directly to vendor-supplied implementations, when they are available.

If you decide to use DCE, we suggest doing so with an object-oriented package like DEC's DCE++ [Viveney] or HP's OODCE [Dilley]. This will position you to make the transition to CORBA more easily in the future. In addition, it is important to be aware that the use of DCE datatypes that are not supported by CORBA (pointers, for example, as discussed in Section 5) can make the transition to CORBA significantly more painful.

During the next two years, the object technology that OMG is specifying will migrate down into the operating systems that are delivered with workstations from all of the major vendors. As a result of OMG's recent adoption of CORBA 2.0 specifications that include inter-ORB interoperability [CORBA2], we expect to see CORBA gaining widespread acceptance during the next two years, and we expect that its object orientation and rich set of services will make it the distributed computing platform of choice.

Glossary

API - Application Programming Interface
CDS - Cell Directory Service
CMIP - Common Management Information Protocol
CORBA - Common Object Request Broker Architecture
DCE - Distributed Computing Environment
DEC - Digital Equipment Corporation
DES - Data Encryption Standard
DFS - Distributed File Service
DME - Distributed Management Environment
DNS - Domain Name Service
DTS - Distributed Time Service
ERA - Extended Registry Attributes
GDA - Global Directory Agent
GDS - Global Directory Service
GSSAPI - Generic Security Service API
HP - Hewlett-Packard Company
IBM - International Business Machines
IDL - Interface Definition Language
IIOP - Internet Inter-ORB Protocol
NTP - Network Time Protocol
OMG - Object Management Group
OODCE - Object-Oriented Distributed Computing Environment
ORB - Object Request Broker
OSF - Open Software Foundation
POSIX - Portable Operating System Interface for Unix
RFC - Request For Comments
RFI - Request For Information
RFP - Request For Proposals
RPC - Remote Procedure Call
SCO - Santa Cruz Operation
SGI - Silicon Graphics, Inc.
SNMP - Simple Network Management Protocol
UTC - Universal Time Coordinated
UUID - Universal Unique Identifier