The Open Group – Discussion Paper

Distributed Computing for the Extended Enterprise – Challenges and Directions, July 1998

Executive Summary

This discussion paper is concerned with distributed computing for the extended enterprise. It addresses integration and interworking of a variety of software deployed to handle distributed computing in a heterogeneous network environment. It compares and contrasts four different technologies: CORBA, DCE, DCOM, and Java. It also looks at the development of high-level services such as Business Objects. The paper is not exhaustive nor is it complete in all areas; its purpose is to stimulate discussion. It will be reviewed and improved in the light of feedback.

The document is written based on the notion that enterprises will need to employ multiple middleware technologies. There is no intent to provide advice that would inform the selection or purchase of a particular middleware. The technical annex is there to show the differences between the technologies using our layered model to highlight the problems of interworking. We would seek coallescence rather than divergence in the future.

We should stress The Open Group position in relation to Distributed Computing Services. The reality is that there are different technologies deployed. Most enterprises will have to live with this diversity. We therefore have to be concerned with the questions of co-existence and interoperability, working to facilitate co-operation between the various technologies and encouraging standardization. An effective program in this area will allow customers to get the maximum business value from distributed computing.

We use a simple layered model to compare the different technologies. The paper identifies challenges in working with multiple technologies. Where similar services exist they tend to be incompatible, which leads to security, naming, and management problems. Interoperability problems are also identified at the transport, infrastructure, and language levels.

The paper also identifies other problems related to performance and the maturity of the technologies. Finally the paper highlights the business need for Business Objects or their equivalent to aid enterprises in achieving their goals of reduced cost, increased flexibility, and adaptability to change in a world involving distributed computing within and beyond the enterprise.

Contents

Introduction

This discussion paper is concerned with distributed computing for the extended enterprise. It addresses integration and interworking of a variety of software deployed to handle distributed computing in a heterogeneous network environment. It compares and contrasts four different technologies: CORBA, DCE, DCOM, and Java. It also looks at the development of high-level services such as Business Objects. The paper is not exhaustive nor is it complete in all areas; its purpose is to stimulate discussion. It will be reviewed and improved in the light of feedback.

We should stress The Open Group position in relation to Distributed Computing Services. The reality is that there are different technologies deployed. Most enterprises will have to live with this diversity. We therefore have to be concerned with the questions of co-existence and interoperability, working to facilitate co-operation between the various technologies and encouraging standardization. An effective program in this area will allow customers to get the maximum business value from distributed computing.

Businesses are moving to embrace Network Computing (the subject of a recent Open Group Discussion Paper [1]). They are looking to integrate business systems across the enterprise that extends from the in-house network (or Intranet), across the Internet to the Extranet and beyond! The enterprise needs to employ diverse IT resources extending from the mainframe to palm top and must therefore harness multiple technologies such as CORBA, DCE, DCOM, and Java, and others as appropriate.

It is unlikely that any one system or technology can handle all the user requirements, each has its strengths and weaknesses. In any case, mergers and acquisitions bring about the need to integrate different systems, including legacy systems. It is just not practical to rebuild from scratch, neither is it practical to dictate the choice of technologies or solutions to business or trading partners, and customers and suppliers, who need to communicate for effective business operations.

Some of the material for this paper is drawn from key reports and it is appropriate to acknowledge these here. The first report comes from the Open Business Object Environment (OBOE) project.[2] The Mitre Corporation [3] is responsible for the second report. Although a little dated, the Mitre Report helped in structuring this document prior to in-house review and update.

This paper has been prepared to assist in identifying the issues; the challenges and the opportunities in the area of distributed computing. The discussion at The Open Group Member Conference, especially the meeting in Miami this July, will help us fill out the detail. You will find some questions for consideration at the end of each chapter. We hope you will participate in the discussion by emailing ogpubs@opengroup.org.

---

[1] The Open Group Discussion Paper on *Network Computing – A New Business Paradigm* was published in April 1998 and is available in hardcopy or from the Web at http://www.opengroup.org/pubs/catalog/w801.htm.

[2] OBOE is a consortium of partner companies brought together under the European Union's ESPRIT program to investigate and develop business systems that take advantage of the principles of Object-Oriented Technology. The partners are Geco, LogOn, Prism, SSA, Sintef, and The Open Group and the report *D5.2.1-1 Business Object Facilities: A Comparative Analysis (Version 1.0 (Draft))* was published in April 1998.

[3] *Comparing DCE and CORBA. Mitre Document MP 95B-93* (March 1995) available at http://www.mitre.org/research/domis/reports/DCEvCORBA.html.

The Need for Distributed Computing

Before examining the technical solutions, we should be clear as to the general requirements for Distributed Computing. What are the business requirements and the underlying problems enterprises are looking to solve?

Where customers need to deploy multiple middleware technologies their first requirement is for them to interoperate. Basic services (naming, security, time, distributed file, etc.) need to be compatible across the technologies.

Customers also need to have a wide range of services, including higher-level services for business operations, enabling them to reduce the cost of software as well as to work more efficiently

Beyond this there is a clear movement toward the development of higher-level services such as Business Objects. These have characteristics that go well beyond the usual services, such as a security service. They offer a solution to a whole business function and integrate the required components of that function. This is illustrated, for example, in a business object for handling the acquisition of stocks and shares. The user requirement is for a high volume automated service which involves the verification of customer details, details of the availability and price of the stock, interaction with the banks concerned to check credit and initiate a bank transfer, notification of the acquisition to the company, etc. etc. A business object would integrate these tasks to ensure a secure and reliable transaction. It would also handle failures and exceptions to preserve the integrity. There are no shortages of examples.

One of the main drivers for this kind of solution is the evolution of business processes. Business managers look to integrated business solutions taking account the life cycle management, physical distribution of activities, speed of response to changing circumstances (agility), and of coping with mergers and acquisitions. They want applications that are easier to maintain and evolve. They need a greater degree of compatibility between their systems and those of their partners, suppliers, and customers. Hence the concept of the Business Objects covering all the participants/needs for the activity.

A Business Object is a representation of a business concept (physical or procedural) active in a business domain, including at least its business name and definition, attributes, behavior, relationship, rules, policies, and constraints. A Business Object may represent, for example, a person, place, and event or a business process such as an employee personnel file, an invoice, a personnel timing procedure,or payment procedure.

An object-oriented approach clearly fits the business environment of the future and ties in with the interest in object-oriented applications and *vice versa*. There can be direct mapping between the definition of a Business Object and its implementation. The methodologies for this are already in widespread use.

We must recognize that messaging, transactions, and workflow are also prevalent in distributed computing systems. In highlighting the need to handle business objects we must preserve the qualities which make distributed, necessary to the business environment, including portability across systems, scalability, security and location independence.

The IT mission needs to match the business requirements of the enterprise. This must be borne in mind as we approach discussion of distributed computing requirements. In parallel with this is the constant demand for de-skilling, both in the enterprise and in its computing environment. The flexibility and value of business objects and their re-use can make a big

impact in this area. Why write and maintain custom-made applications when one could deploy a common business object? Why set up communication channels with one to one interfaces involving specific hand holding, security, and other risks when one could deploy a standard business object?

Finally, we should accept the need to cope with legacy environments. Enterprises have to evolve business operations and require systems that interact with legacy systems and the data embedded in them.

Role of Middleware

Middleware products are infrastructures needed for the integration of different applications and legacy components in heterogeneous and distributed environments. Middleware products hide much of the complexity involved in resolving this heterogeneity and distribution. Distributed middleware provides features that can't be provided on a single system: naming and location services, location-independent remote execution, data transfer, synchronization, and security capabilities.

Common Business Objects and Business Object Facility

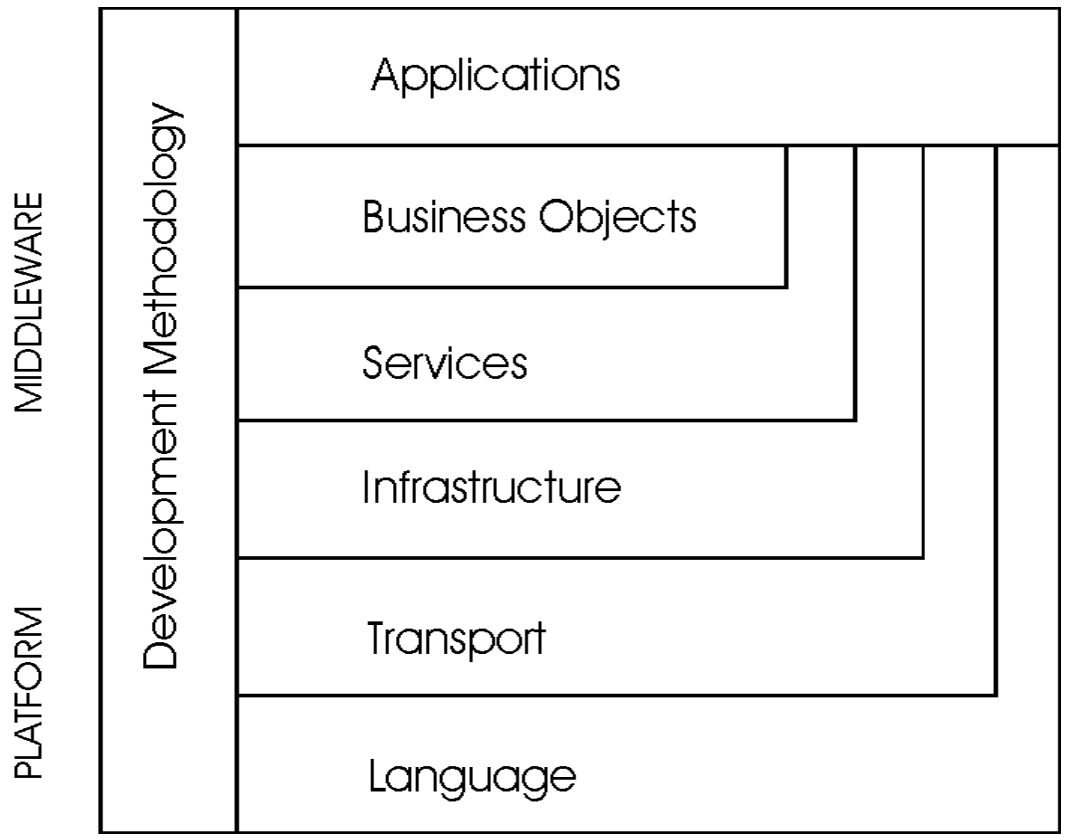
Further information on Business Objects can be found on the OMG web site at http://www.omg.org/library/schedule/Business_Objects_RFP.htm.

Some questions: *Are the business requirements as stated above? What are the relative priorities? How far can one generalize?*

Distributed Computing – The Choice of Technologies

It is possible to construct a long list of technologies that can be employed to enable and manage a distributed computing environment. Any list must include CORBA, DCE, DCOM, and Java. As explained in the introduction we have concentrated on these four technologies at this time. Other technologies can be brought into the equation. These include MQ-Series, MS-Messaging, and Pipes. We recognize that other relationships concerned with messaging and transactions, such as that between CICS and DCE, might be drawn into the equation. We have concentrated on just four technologies since these are the predominant technologies in the market today and a comparison of these highlights the differences sufficiently for the purposes of this paper.

We have developed a greatly simplified diagram as the basis of comparison. A layered model shows the middleware sitting between the platform and the application. The language, transport, and some proportion of the infrastructure layer can be regarded as part of the platform. Above this we have a number of services and a relatively new class of business objects.

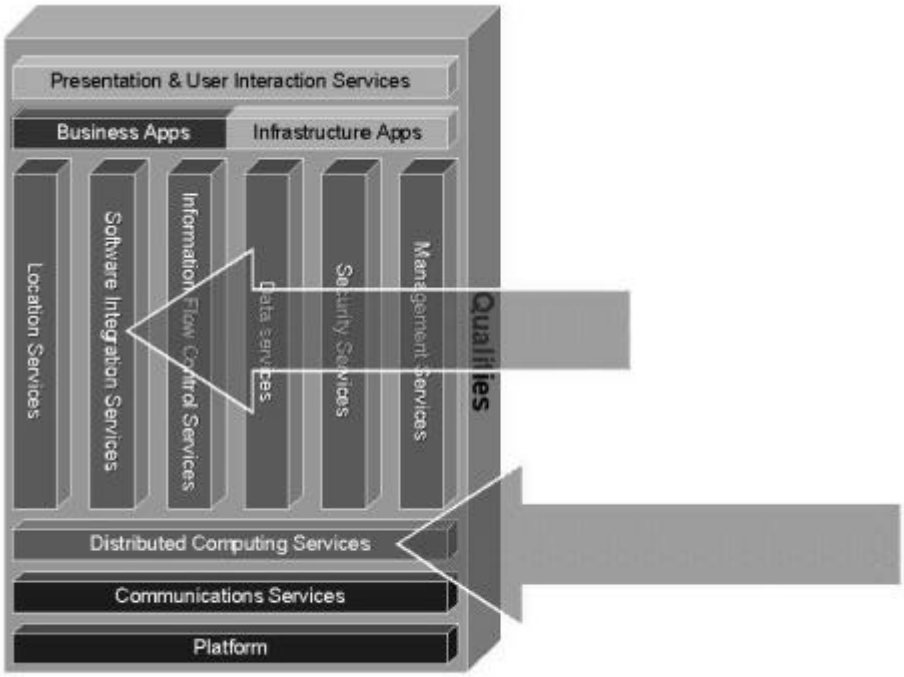


This diagram allows us to examine the various technology solutions for distributed computing. We can show how each technology caters for the requirements at each level. Again, a simplified matrix brings out the different approaches for each technology. Note the matrix could be extended for other technologies. Further details are provided in the technical annex to this paper that includes a more complete listing of the services. The labels refer to the names of well-known acronyms or products.

Technologies

	CORBA	DCOM	DCE	Java
Business Objects	BOF, UML	N/A	N/A	EJB
Services (incomplete)	e.g., name, trader	e.g., ODBC?	e.g., name, security, time	e.g., JDBC, JMAP
Infrastructure	ORB, IDL	Active X Components	N/A	Java Beans, Applets, RMI
Transport (above TCP-IP)	IIOP	MS RPC	DCE RPC	HTTP
Language	Java, C, C++, etc.	Java, C, C++, VB, etc.	Java, C, C++, etc.	Java

These diagrams provide a different viewpoint from that of the IT DialTone Architecture Reference Model. This reference model is shown below with two arrows superimposed. The arrow show the overall position of distributed computing services above communication services and the position of the various services. The technologies which are the basis of discussion in this paper can obviously be mapped onto the IT DialTone Architecture. We have chosen the simple diagram above as the best way of comparing the technologies and stimulating discussion at this stage.



Challenges

Our matrix with the four technologies illustrates clearly that we have divergent technologies based on different paradigms and approaches. We could add to the mix by adding the other technologies. At this stage we can begin to assess the challenges presented for those who need to work in a distributed computing environment with multiple technologies.

Most customers do not have the luxury of simply selecting a single distributed computing technology. If they did, then some of the material presented below might help. More likely customers need applications that collectively depend on more than one technology approach. In this case, an understanding of the differences between the technologies and an ability to deal with diversity is the greater challenge.

We have looked at the challenges in two ways: by references to the qualities referred to in the IT DialTone Architecture and by reference to the layer diagram in this paper.

Qualities

One way of looking at the challenges for distributed computing is to look at the "..ilities" presented in the IT DialTone Architecture.

These qualities include manageability, interoperability, scalability, and usability. We discuss only a few of these below.

Adaptability

We mentioned the need for adaptability in relation to changing business circumstances. The distributed computing systems must also cope with change.

Availability

Customers should be concerned about the availability and reliability of systems and services. This will be impacted by the decision to employ multiple technologies where problems of availability may be compounded.

Integrity

The technologies, when working together in a multiple environment, must ensure message and data integrity. For example, differences in naming services can result in loss of resources. See the discussion on naming services below.

Interoperability

Given technical diversity and the need to employ more than one distributed computing technology, interoperability is the key issue.

Manageability

Management of distributed computer systems is costly. Systems management ranks amongst the most important benefits deriving from network computing and the use of network computers since it takes responsibility away from the user. However, there must be a question

mark over the ability to manage systems in a distributed computing environment where multiple technologies are deployed. This is highlighted in the discussion of services below.

Maturity

Customers are rightly concerned about the maturity of the technologies that they acquire. Many will prefer not to be pioneers. They will also wish to see a range of suppliers, the availability of skills, and they will welcome the opportunity to learn from the experience of other customers. The future development path for the technology will also be a factor.

Portability

Portability is less of a problem than in the past. See the discussion on languages below.

Productivity

Enterprises are rightly concerned to improve productivity. This applies to the business processes that need the right IT solutions. It also applies to the cost and maintenance of software development.

Scalability

Customers, especially those from large enterprises or concerned with high volume traffic, perhaps across global networks, will be concerned with the performance of the technologies especially when present and interoperating in a multiple technology environment.

Security

Absolute security is probably unobtainable, however the highest level of security is needed in today's business environment and solutions must be found for handling security across systems that employ multiple technologies. Security services differ for each technology. See the discussion on services below.

Layers

We have identified some of the challenges by reference to our layered diagram and the matrix.

Services

Each technology offers a range of services differing from one another. Some provide or have the potential to provide a rich variety of services.

The key question is whether in a distributed environment where one or more of the technologies are employed one can have satisfactory interworking of services. Three areas come immediately to mind: security, naming, and system management.

Where two or more middleware technologies are employed each has its own security service. They are not compatible. This means that system administrators and users have to log into each system. This gives rise to a multiplicity of passwords, etc., and makes single sign-on virtually impossible. One has to ask whether one can really secure the system where more than one security service is in use. What is the solution?

A second area of difficulty arises with naming services. Can one provide a satisfactory solution where there is more than one naming service in place? Different naming services across the systems mean that resources may not be identifiable; they may even get lost.

System management is that much more difficult with multiple technologies. The complexity resulting from more than one environment translates into increased costs and the need for a greater level of knowledge and skill.

Some Questions: *What are the limitations caused by the multiplicity of middleware technologies? Our problem is that services of the same type differ from one technology to another. Can they be harmonized?*

Transport

The specifications for Java RMI and CORBA IIOP need to be converged to meet the goal of interoperation. Most systems employ TCP/IP, the concern is how, for example, they implement RPC. These differences can probably be managed in the server, but in the client these differences ensure that multiple stacks have to be present and this causes an overhead.

Question: *Can we avoid different solutions at the Transport level?*

Language

Nowadays language differences do not seem to be much of a problem in relation to distributed computer systems. Those deployed, with the exception of Visual Basic, are platform independent. Compilers are readily available.

The choice of language relates more to the strengths and weakness of each. Java has the virtue of portability, ease of coding, and automatic storage management. There are questions of performance. C++ provides good performance without recourse to an interpretive runtime compiler but it requires a high skill level from the programmer.

Languages are in general standardized, and implementations adhere to these standards and are widely available on key platforms. However, there are often a great number of proprietary extensions.

For example, Visual Basic is easy to use, is graphically based, and analogous to scripting languages that many users relate to. However, Visual Basic is not platform-independent.

Java has gained widespread market acceptance since 1997 for the following reasons:

- Java source code is highly portable and will progress rapidly to a formal standard.
- The same byte code can be executed by different virtual machines on a wide variety of platforms.
- Its automatic storage management largely eliminates a primary cause of design errors in C++.
- It fully supports object-oriented design techniques.
- Its object model is claimed to be cleaner and simpler than that of C++.
- It comes with powerful APIs for user interface construction, network access, database access, etc.
- It supports multi-threading.
- It supports distribution.
- Its security model allows for safe loading of programs across a public network.

Whilst there are aspects of language choice that make distributed computing easier to implement, it is not generally seen as a constraint on the choice of distribution technology.

Some questions: *To what extent is the choice of language a factor in a distributed computing environment based on multiple technologies? Are there significant constraints?*

Infrastructure

CORBA is language and platform-independent and by now well established. ORB implementations are not necessarily platform-independent, vendors implement extensions to the basic standard and can lock customers into a particular environment. COM is extremely well established and is language independent. It is available for Window 95, NT, UNIX and Mac.

The main problem at this level is interoperability between the technologies. Some form of bridge can provide a solution at this level although the key concern would be the incompatibility of the services (see above). Bridging solutions do not provide a complete answer to the problem of deploying multiple technologies. The convergence of Java/RMI and CORBA through the development of an open standard for interoperability is to be welcomed and shows a determination to solve the problems.

Question: *How far is it worth pursuing bridging solutions at the infrastructural level without compatible services to run on top of them?*

Conclusions and Recommended Actions

Enterprises need to work with multiple technologies and require that they interoperate with one another.

We need a careful analysis of the technologies to establish how they can be made to interoperate effectively and we need to see coalescence rather than divergence. In particular we need to see like services working in the same way.

Enterprises are looking for high-level component orientation such as is promised by Business Objects. They need technologies that can support these working across multiple technologies. Of course, business objects themselves need to be portable across the underlying implementations to allow independent evolution of the customer's application software and the hardware, OS,and middleware that support it.

There is a case for greater awareness and training for those concerned with deploying multiple technologies. Sharing of experiences and assistance in weighing up and making judgements concerning mixed environments is called for.

The subject needs to be looked at in relation to the IT DialTone Architecture and Vision and also set against developments in Network Computing.

Once again there is a case for greater standardization at the protocol and API level.

Some questions: *How can we prepare and present information to aid customers in solving the problems arising from the deployment of multiple technologies? How do*

we encourage the vendors to co-operate and adopt common practices and standards? How do we get convergence rather than divergence?

TECHNICAL ANNEX

Four Technologies

As an introduction to each of technologies we quote from the vendor web sites giving a high-level introduction to their offering. These pieces reveal some of the underlying rationale as well as the original purpose.

What is CORBA?

"The Common Object Request Broker Architecture (CORBA), is the Object Management Group's answer to the need for interoperability among the rapidly proliferating number of hardware and software products today. Simply stated, CORBA allows applications to communicate with one another no matter where they are located or who has designed them. CORBA 1.1 was introduced in 1991 by the Object Management Group (OMG) and defined the Interface Definition Language (IDL) and the Application Program Interface (API) that enable client/server object interaction within a specific implementation of an Object Request Broker (ORB). CORBA 2.0 adopted in December 1994 and CORBA 2.2 adopted in February 1998 defines true interoperability by specifying how ORBs from different vendors can interoperate." Further information is available at http://www.omg.org/about/.

IIOP (Internet Interoperability ORB Protocol), the CORBA interoperability protocol, has been demonstrated to work and is actively used to achieve interoperability of applications that work on different ORBs.

What is DCE?

"The OSF Distributed Computing Environment (DCE) is an industry standard, vendor-neutral set of distributed computing technologies. It provides security services to protect and control access to data, name services that make it easy to find distributed resources, and a highly scalable model for organizing widely scattered users, services, and data. DCE runs on all major computing platforms and is designed to support distributed applications in heterogeneous hardware and software environments. DCE is a key technology in three of today's most important areas of computing: security, the World Wide Web, and distributed objects." Further information is available at http://www.camb.opengroup.org/dce/.

What are COM and DCOM?

"The Component Object Model (COM) is a software architecture that allows applications to be built from binary software components. COM is the underlying architecture that forms the foundation for higher-level software services, like those provided by OLE. OLE services span various aspects of commonly needed system functionality, including compound documents, custom controls, inter-application scripting, data transfer, and other software interactions.

The Distributed Component Object Model (DCOM) is a protocol that enables software components to communicate directly over a network in a reliable, secure, and efficient manner. Previously called "Network OLE," DCOM is designed for use across multiple network transports, including Internet protocols such as HTTP. DCOM is based on The Open Group's DCE RPC specification and will work with both Java applets and ActiveX components through its use of the Component Object Model (COM). For example, a developer could use Java to build a Web

browser applet that calculates the value of a portfolio of securities, using DCOM to communicate stock values to the applet in real time over the Internet."

What is the Java™ Platform and Java/RMI?

"The Java platform is a fundamentally new way of computing, based on the power of networks and the idea that the same software should run on many different kinds of computers, consumer gadgets, and other devices. With Java technology, you can use the same application from any kind of machine – a PC, a Macintosh computer, a network computer, or even new technologies like Internet screen phones. The idea is simple: Java software works just about everywhere – from the smallest devices to supercomputers. Java technology components don't care what kind of computer, TV, or operating system they run on. They just work, on any kind of compatible device that supports the Java platform." Further information is available at http://www.javasoft.com/nav/.

Java Remote Method Invocation (Java/RMI) was introduced with Java 1.1. It supports the invocation of methods from non-local Java objects. Hence Java has distributed computing capabilities that are comparable to those of CORBA.

The Marketplace

The Object Technology market [4] has been boosted during the last two years by the boom of the Internet. A joint survey of Cutter Information Corporation and the OMG has revealed that Java has brought object-oriented programming to the Web. An increasing number of Inter- and Intranet projects are being developed in pure Java. Enterprise JavaBeans are about to replace previously developed component technologies such as OpenDoc. Support for Java persistency has increased demand for object databases, object relational databases, or object-persistency frameworks for relational databases. Execution of Java applications has created a demand for new operating system products such as Web servers and browsers. The need for integrating new Java components with legacy components that exist in almost every enterprise has doubled the size of the market for object-oriented middleware.

There has also been a large increase in business components written in visual basic. There are large numbers of VB components available both on the Web and through third-party suppliers. VB may be the most common form of business object programming in the financial community where "VB rules".

Object-Oriented Middleware products are designed to integrate different legacy applications and legacy components in heterogeneous and distributed environments, hiding the complexity

[4] Market Segmentation: (a) Object-Oriented Middleware – products that facilitate the integration of heterogeneous and distributed applications, (b) Object-Oriented Application Development – products that achieve the rapid development of applications from high-level descriptions, (c) Object-Oriented Programming Languages – products that support the creation, compilation and interpretation of object-oriented programs, (d) Object-Oriented Operating Systems – products, such as Web servers and Web browsers that facilitate execution of object-oriented programs, (e) Object-Oriented Case Tools – products that support the analysis and design of object-oriented programs, (f) Object-Oriented and Object-Relational DBMSs – products that achieve persistent storage, reliable updates and queries of object states, and (g) Object-Oriented Component Technology – products that support the development of applications by integrating existing components. It follows that categories (a), (g), and (f) are the most directly related to distributed computing.

from the user. They treat applications that need to be integrated as objects that have defined interfaces. An Interface Definition Language (IDL) is used to define the services that an application exports to other applications.

Key players in the object-oriented middleware market [5] are Microsoft with DCOM and around a dozen vendors with implementations of CORBA and Java.

Convergence

In the near future, a CORBA specification will provide for interworking between RMI and CORBA's IIOP protocol. This will enable CORBA clients to invoke methods from Java objects via RMI or *vice versa*. Moreover, several CORBA vendors integrate their implementations with transaction monitors. [6] In addition, middleware products will be integrated into message-oriented middleware systems. [7] DCE RPC is the basis of the Encina Transaction monitor used in the portable CICS implementation supported by several suppliers.

Microsoft integrates DCOM with MTS, the Microsoft Transaction Server product. In the future, it will be integrated with MSMQ, Microsoft's Message Queue Server, which will allow direct access to MTS applications.

These developments open new application areas for object-oriented middleware applications in which reliability, availability, and performance requirements are predominant.

The OMG has also adopted an Environment-Specific Interoperability Protocol (ESIOP) for interoperability between ORBs based on OSF/RFCs, and it has defined an RPC interoperability specification that allows interoperability between CORBA objects and OSF/RPC programs.

CORBA Implementations

Vendor	Product	URL
BEA	ObjectBroker	http://www.europe.digital.com/info/objectbroker
Expersoft	PowerBroker	http://www.expersoft.com/prod_ser/
HP	ORBPlus	http://www.dmo.external.hp.com/gsy/orbplus.html
IBM	SOM/DSOM	http://www.software.ibm.com/sw-guide/enu/ad/ad129.htm
ICL	DAIS	http://www.icl.co.uk/products/dais
IONA	Orbix, OrbixWeb	http://www.iona.com/Orbix/.
Inprise	Visigenic/C++, Visigenic/Java	http://www.visigenic.com/prod

[5] The OBOE report (see footnote on page 3) suggests that since RMI has been re-focused as a pure Java implementation rather than a CORBA implementation, it does not solve the heterogeneity problem and, therefore, is not a real middleware product.
[6] The Iceberg project at BEA integrates ObjectBroker, which BEA acquired from Digital, with Tuxedo.
[7] Examples include IBM's integration of MQSeries with DSOM.

ParcPlace	Dist. Smalltalk	http://www.parcplace.com/products/dst

DCE Implementations

Vendor	Product	URL
Compaq	DCE Product Family	http://www.digital.com/dce
Dascom	Interverse and DCE Cell Manager	http://www.dascom.com
Gradient Technologies	NetCrusader and PC-DCE Product Families	http://www.gradient.com
Hewlett Packard	DCE Product Family	http://www.hp.com/go/dce
IBM	DCE Product Family	http://www.software.ibm.com/enetwork/dce
Siemens Nixdorf	Open Enterprise Computing	http://www.siemensnixdorf.com/servers/dce
The Open Group	DCE (licensing)	http://www.opengroup.org/dce

DCOM Implementations

DCOM is ubiquitous.

Vendor	Product	URL
Microsoft	COM/DCOM	http://www.microsoft.com/com
Software AG	EntireX DCOM	http://www.sagus.com
The Open Group	DCOM (licensing)	http://www.opengroup.org

Java/RMI Implementations

Java/RMI has been adopted by a number of companies as a strategic development language. These companies include Oracle, Netscape, IBM, Lotus and Microsoft. [8]

Some questions: *How important is it to consider maturity, range of product, and support in deciding which technology to employ?*

[8] Microsoft have not implemented all the facilities of Java/RMI and are in a legal dispute with the licensor Sun Microsystems. They are using the name J++ for their implementation.

Comparison

Common Features

These are all middleware products that are designed to facilitate the integration of heterogeneous and distributed applications. They employ an Interface Definition Language (IDL) to define services that an application exports to other applications.

Procedural Programming versus Object-Oriented Programming.

The majority of technologies discussed in this paper employ object-oriented programming techniques. DCE is the exception since it was designed to support procedural programming. However, DCE RPC is designed to be object-neutral. It supports several object programming models, including CORBA, C++, and DCOM. The Mitre Report from 1995 considers that such a fundamental difference ultimately influences the choice of technology for distributed computing. However, nothing is so "black and white".

Object-oriented programming environments, supported by CORBA, are usually characterized by their support for:

- Encapsulation of data and 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 described 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 superclass behave. [9]

Distributed procedural programming environments, supported by DCE, support a different set of capabilities than those of object-oriented programming environments. The basic approach to 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.

[9] In addition to these common characteristics, object-oriented programming environments usually support a style of programming in which (a) not only new objects, but new classes may be created at runtime, (b) late binding of operation invocations to function calls allow programs to be written without regard for the types of objects they will manipulate, (c) 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), and (d) once defined, objects and classes may be reused or refined in subsequent applications, extending the usefulness of object implementations across multiple applications.

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. Nor does it support abstraction, inheritance polymorphism, or the dynamic style of programming.[10],[11]

Having said this, DCE does have comparable capabilities:

- 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 a remote procedure call 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 a remote procedure call to the server and specifies an object UUID, the specific function that is invoked in the server might associate one object type UUID with RPC handlers that support line printers and another object type UUISD with a corresponding set of RPC handlers that support Postscript printers.

The object UUID and object type UUID mechanisms in DCE equate to some of the characteristics of object-oriented systems, such that object type UUIDs provide some form of abstraction and polymorphism.

Although procedural programming is not object-oriented programming, 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). Many CORBA conformant ORB vendors (Digital, HP, and IBM for example) are implementing ORBs on top of DCE.

Characteristics

Characteristics of the underlying (object-oriented) middleware that can be used in comparison of technologies and products:

- Object Model – Whether object-based, if so what are the properties and how rich is the model?

- Interface Definition – Which language constructs are available to determine the properties (of objects)?

- Communication Primitives – What primatives are available for interaction (between objects)?

- Architecture – Which architectural components are involved in an (object) request?

[10] This observation relates more properly to a comparison of the procedural versus object style programming than of DCE RPC. For example, HP's OODCE does support abstraction and inheritance.

[11] At this point one might highlight the importance of portability of objects across implementations and scalability, since what might work for 100 users won't always work for 1 million users.

- Programming Language Bindings – Which programming languages can be used to implement client and server objects?

- Openness/Interoperability – With which other systems can the middleware interoperate?

- Services and facilities available for reuse – What higher-level services and facilities does the middleware support? Here we can examine security, naming, etc.

- Platforms – On which platforms is the middleware available?

	OMG/CORBA	DCE	DCOM	Java/RMI
Object Model	Based on objects, designed to support object-oriented programming. One type of object which determines the operations and attributes that the object exports to other objects. [12]	Object model-neutral. Designed originally to support procedural programming. Supports layered object models.	DCOM's object model is COM. A COM object can have multiple types at the same time; however, COM objects do not support multiple inheritence. [13]	Java and Java/RMI use the same object model. Supports methods, instance,and class variables with different degrees of visibility (public, protected, and private). [14]
Interface Definition	OMG/IDL Interface. Includes language constructs for all concepts of the object model. It is programming language-independent and not computationally complete. OMG/IDL supports multiple inheritance and defines a hierarchical namespace. 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	DCE uses an IDL based on the C programming language. C++ interface support is added for DCE 1.2. As with CORBA, 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. Originally DCE IDL did not support interface inheritance and defines a flat namespace. [15] DCE	COM and DCOM objects are specified using Microsoft's Interface Definition Language (MIDL). MIDL provides language concepts for all concepts of the object model and it reflects the differences between CORBA, and the COM/DCOM object model. [16]	No interface definition (regarded as superfluous since methods can only be invoked by another Java method). Public methods can be invoked remotely, with no restrictions to parameters or result types. Hence objects can be passed and generic servers and mobile agents implemented in Java.

[12] Objects can request the execution of an operation or the value of an attribute of a server object from an ORB. In order to make the request they have to submit a reference to the server object and the name of the attribute or method and possibly method parameters. Operations may raise exceptions and these exceptions are considered an essential part of the object model. There is one root to the object type hierarchy referred to as Object and this defines common properties that all CORBA objects support.

[13] Each type is expressed by an interface that represents a different behavior of the object. An interface consists of a set of methods that are related in functionality. Like CORBA clients, COM clients use an object reference to interact with the DCOM server object.

[14] Mappings between different object models as they occur in CORBA's programming language bindings do not exist. Operations may raise exceptions to indicate failures. Java supports single inheritance. In addition, it supports the concept of interfaces, which can be compared to abstract classes. A Java class may implement many interfaces but can only inherit from one class.

[15] However, DCE servers can export multiple interfaces, with multiple versions of the same interface supported for compatibility. Interfaces can also be constructed out of other interfaces using include statements. Later versions with C++ support language inheritance.

[16] A further difference between OMG/IDL and MIDL is in the support for error handling. OMG/IDL supports the concept of exceptions that are bound to the exception handling mechanism of the respective programming languages used for clients and servers MIDL forces programmers to return a 32-bit error code of type HRESULT.

	object type at runtime.	has no dynamic invocation interface – the appropriate RPC stubs must be linked into the DCE client.		

	OMG/CORBA	DCE	DCOM	Java/RMI
Communication Primitives	CORBA object requests are synchronous invocations of operations that server objects exported. Asynchronous requests are supported by the messaging and the event notification services. [17]	DCE has a variety of communications semantics controlled by the interface definition, including at-most-once, idempotent, maybe, broadcast, and pipe	Remote DCOM operation executions are synchronous calls. Equivalents to deferred synchronous or oneway requests that are available in CORBA do not exit. Asynchronous executions can be implemented in DCOM using multi-threading in the same way as these are implemented in CORBA and Java/RMI.	Similarly to remote procedure calls, Java remote method invocations are strictly synchronous. Asynchronous behaviour can be implemented using Java threads. [18]
Infrastructure	Based on an ORB[19]. Use of either the dynamic invocation interface or by invoking a client stub.	DCE supplies transparent service and object location through the name service, built in security providing authentication, authorization and access control, and dynamic service instantiation through dced. DCE also provides an extensible and scriptable remote management tool.	Distribution between client and server DCOM objects is through an RPC channel. The interface proxy and the interface stub perform the same tasks as the client stubs and server skeletons in CORBA and Java/RMI. [20]	Similar to CORBA, Java includes client stubs and server skeletons that perform the marshaling and unmarshaling of parameters and the results that are returned in reply messages. RMI does not include a dynamic invocation interface.

[17] If an operation is declared one way in the interface of the server object, the client regains control as soon as the ORB has accepted the request. For one way requests, operations cannot return output of result types and must not raise specific exceptions.
[18] The main thread is not blocked while the remote method executes. When the remote method is completed, the two threads are synchronized using various methods of synchronization supported by Java's thread model. Note that a lightweight equivalent to CORBA's one way operations and deferred synchronous requests does not exist for Java/RMI.
[19] Transfers requests made by client objects to server objects in a way that location, programming language, and platform of the server object remains transparent for the client.
[20] The Service Control Manager (SCM) performs approximately the same tasks as the object adapter and the request broker in CORBA. The registry is equivalent to CORBA's implementation repository.

	OMG/CORBA	DCE	DCOM	Java/RMI
Programming Language bindings	C,C++, Smalltalk, Ada-95, Cobol, and Java bindings. CORBA facilitates distributed system construction using multiple different programming languages.	C and C++ built in. Other language bindings (Cobol and Ada) available through third parties.	DCOM does not specify programming language bindings. COM objects have to follow a standard memory layout. [21] Hence DCOM does not provide portable source code nor platform independence (see below).	Java remote method invocations can only be done within other Java objects. Using Java Native Invocation (JNI) it is possible to implement a method of a Java server object in C or C++ rather than Java. [22]
Openness	Industry consensus specifications (adopted by TOG)	Open Specifications adopted by The Open Group and ISO. Implementations available from multiple manufacturers. Reference implementations freely available. Has been re-implemented from specification by Microsoft.	COM and DCOM are not based on open standards. However, there is a DCOM implementation based on the DCE RPC Specification available from The Open Group.	Currently not a very open approach. However there is the prospect of an interoperability specification between Java/RMI and CORBA's IIOP. Sun also announced support for embedding Java objects in COM components and *vice versa*
Services and Facilities	CORBA services include naming, trading, transactions, concurrency control, persistence, event notification, externalization, licensing, messages, and life cycle and relationships. CORBA facilities are objects that are useful across many different application domains; examples include printing,	See entry for "Infrastructure" above.	DCOM comes with a naming service (referred to as the registry) that supports DCOM components in a location-transparent way. DCOM integrates with the Microsoft Transaction Server (MTS) which provides concurrency control, distributed transaction processing, and security capabilities. These may not be as	Fewer services than CORBA. Naming is very similar.

[21] If the objects run on the same hardware and operating system architecture, binary compatibility is achieved irrespective of the programming language(s). This contrasts with CORBA where source code compatibility is achieved at the expense of portability.

[22] JNI is provided for the construction of wrappers around legacy components. ILOG have announced a product called TwinPeak that will generate Java wrapper classes for C++ classes fully automatically. The JNI approach is not so general as CORBA's programming language bindings. It might be questioned how many as legacy C++ (business) objects there are that could be wrapped using JNI.

	spooling, and help.		extensive as CORBA. [23],.[24]	

	OMG/CORBA	DCE	DCOM	Java/RMI
Platforms	Windows (3x, 95, & NT), OS2, UNIX, OS400, MVS, OpenVMS, and OpenVME	All major computer systems available including Windows 95, NT, VMS, MVS, OS400, OS/2, and UNIX.	DCOM is shipped with Windows NT (Version 4.0 and upward). It is available on both Intel and Digital Aplha hardware. DCOM implementations are available for various UNIX platforms. It is unclear how effective these are working in isolation from, for example, MTS. Nor is it clear when COM+ might be available on UNIX platforms.	A Java Virtual Machine (JVM) with Java V 1.1 is required to execute Java/RMI. JVM's in the form of byte code interpreters are available for all PCs and UNIX platforms. However, few browsers support Java/RMI directly (Netscape requires a plug-in and Microsoft do not intend to support RMI). [25]

[23] Microsoft is working in a message-oriented middleware (Microsoft's Message Queue Server) that will be integrated with COM+ (an extension of COM/DCOM). This middleware will probably be used to replace RPC-based interaction between clients and servers in order to have a more reliable form of transport that also supports non-synchronous operation and multi-cast operation executions in a more natural way.

[24] One might take into account the number of third-party objects including those implemented in Visual Basic.

[25] The absence of direct browser support for Java/RMI is a limitation. This has often led the choice of CORBA for browser applications with Java used for component implementations.

A comparison of DCE *versus* CORBA reveals some further differences in capability not included in the table above:

	CORBA	DCE
Datatypes	No equivalent datatype, equivalent behavior is obtained using a CORBA sequence.	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.
Pipes	An extension to CORBA is needed to pass large data sets.	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.
Contexts	No corresponding mechanism (CORBA contexts are concerned with user preferences and requests to an object).	Contexts provide a mechanism for maintaining server state during a series of logically, related requests from a single client.
Pointers	CORBA does not support the use of pointers as, or within, operation parameters. The programmer has to write additional code to create pointer free data types with the actual values and then reconstitute them as pointers in the client, or redefine the complex data structure as a collection of one or more objects, since CORBA does not support complex structures composed of objects.	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.
"Any" data type	CORBA supports an "any" data type permitting a value of an arbitrary type to be passed between a client and server.	Not supported.
Interface Repository	Defines an interface repository that contains information equivalent to that in the IDL files and can be queried at runtime.	No such repository.
Server Activation	CORBA supports automatic server activation.	DCE supports dynamic service activation and tear down.