Distributed Computing Environment using DCOM

Gaurav Pancholi, Jyoti Jacob, Keshavaprasad Malavalli, Saishankar Madanogolapan
Computer Science and Engineering
The University of Texas at Arlington
{pancholi, jacob, malavall, madanogo}@cse.uta.edu

April 30, 2002

Abstract

Distributed Computing Environment (DCE) is a widely accepted set of tools for developing and deploying multi-platform, secure, enterprise-wide distributed computing applications. Distributed Component Object Model (DCOM) technology introduced by Microsoft provides a framework for developing distributed components. This paper describes an efficient implementation of a Distributed Computing Environment using the DCOM technology. The application covers major issues concerning a distributed environment like fault tolerance, load balancing and security. The paper also has a brief comparison of the DCOM with CORBA, a popular alternative for distributed component development.

Keywords: Distributed Computing Environment application, DCOM.

1 Introduction

The high volume of networked computers, workstations, LANs has prompted users to move from a simple end user computing to a complex distributed computing environment. This transition is not just networking the computers, but also involves the issues of scalability, security etc. A *Distributed Computing Environment* herein referred to, as DCE is essentially an integration of all the services necessary to develop, support and manage a distributed computing environment [1]. A distributed application is an application whose components reside on more than on computer on a network, with the network typically composed on diverse computers and operating systems. *Middleware* is a software layer that simplifies the construction of distributed applications by providing standardized mechanisms that distributed components can use to communicate. Many of the innovative features of DCE have been incorporated into two most popular middleware technologies, *Distributed Component Object Model* (DCOM) by Microsoft and *Common Object Request Broker Architecture* (CORBA) by the Object Management Group.

The paper describes an application based on DCE specification, build using the Microsoft DCOM technology. Microsoft first came up with a document structuring technology known as *Object Linking and Embedding* (OLE), which was later, transformed into a more generic object-oriented technology called *Component Object Model* (COM). COM is used to define common

interfaces between software components residing on same computer. To create a true middleware product Microsoft extended COM to support component residing on different computers and called it *Distributed COM* or DCOM. DCOM is shipped with Windows NT 4.0 onwards and is available as an add-on for Win 95\98. DCOM provides a clear separation between the interface on an object and the implementation of an object [5]. This allows flexibility during development, since objects can be implemented in different ways or even in different languages for different components of the network.

Section 2 provides a very brief overview of DCE, Section 3 deals with DCOM and DCOM architecture in brief. Project description is provided in Section 4, Section 5 deals with project design and the implementation prototype is discussed in Section 6. Section 7 has a technology comparison between DCOM and CORBA highlighting the features of the two popular middleware technologies in the market, Section 8 had discussion and conclusions drawn.

2 Distributed Computing Environment (DCE)

DCE is an integration of all the services necessary to develop, support and manage a distributed computing environment. The high volume of networked computers, workstations, LANs has prompted users to move from a simple end user computing to a complex distributed computing environment. The present day computing industry depends on the efficient usage of resources. So instead of duplicating the resources at every node of computing, a remote method of accessing the resources is more efficient and saves costs. A *DCE* Provides a Global Computing Environment, which can interoperate with other services like DNS and X.500 [1]. This sort of global interoperability provides the much-needed interface for *Write Once Run Anywhere* applications. Also the suite of components is completely integrated and interoperable, which facilitates the networking of two systems for processing even though they have different hardware and software configurations.

The DCE cloud refers to the distributed computing environment tools that facilitate distributed computing. The DCE cloud consists of the following components [2]:
a) Distributed File Service
b) Distributed Time Service
c) Security Service
d) Cell Directory Service
e) Threads Service

All these services are achieved by the use of Remote Procedure calls (RPC) [3].

3 DCOM – An Overview

DCOM is a Microsoft technology for distributed software component development. DCOM is an abbreviation for *Distributed Component Object Model*. Microsoft introduced the technology in 1996, shortly after COM (*Component Object Model*) was introduced [4]. Using COM,

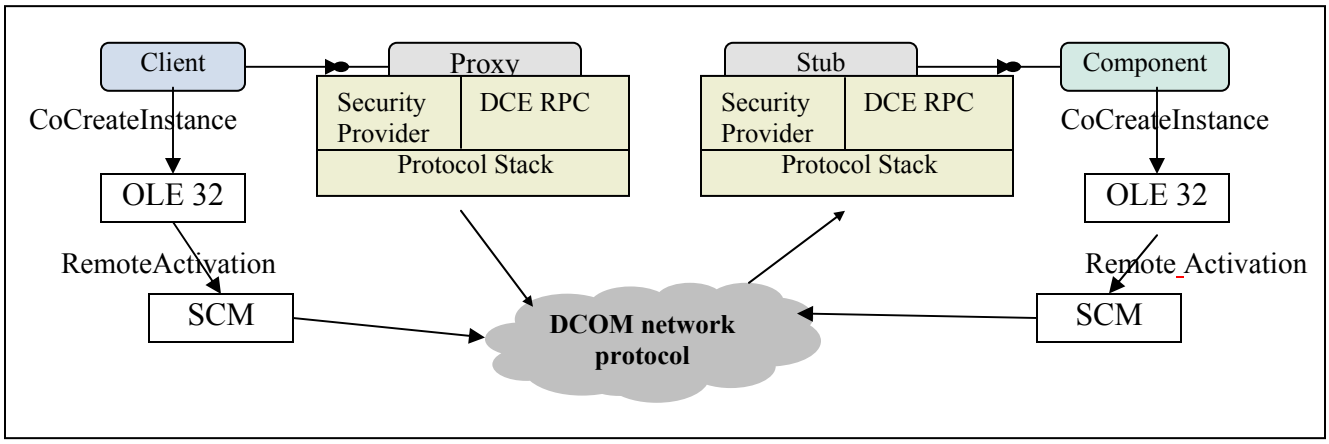
applications can be built using many binary objects. DCOM is an extension to COM, wherein applications can be build using many COM objects that may reside on different machines.

3.1 DCOM Architecture

Since DCOM is an extension of COM, the mechanisms used for client-component interaction is similar to that in COM. COM distinguishes between three different *interactions* [6].

1. First type of interaction is when the client and the component reside in the same process. In this case the client can directly call the methods in the component without any overhead.
2. The second case in when the client and the component are in different processes. In this case the client has to use some form of inter-process communication. COM provides this mechanism; it takes the client calls and forwards them to the component in another process.
3. In the third case the client and the component are on different machines. Here DCOM uses a network protocol for communication, neither the client nor the component may be aware of the physical distance.

This section describes the underlying architecture of DCOM as shown below.



DCOM Architecture

In order for components to communicate, they are assigned a 128 bit globally unique identifier (GUID). To create a COM object several functions like “CoCreateInstance” provided in the COM library can be used [7]. The COM libraries search the registry for appropriate binary code, create the instance and return the interface pointer. In case of DCOM the components may reside on a remote machine. In this case, the client needs to know location of the machine where the component is. Once the server address is known the Service Control Manager (SCM) on the client contacts the SCM on the server and creates the object.

A component can be either in the form of a library file (DLL file) or can be an executable. A DLL component cannot run by itself, a surrogate process is created on the component side within which the component can run. Each interface is given a 128 bit globally unique identifier (GUID) [6].

3.2 Date transfer between the clients and components

To pass parameters between distributed clients and servers, DCOM uses a technique knows as *marshalling\unmarshalling*. COM provides sophisticated mechanisms for marshaling and unmarshaling method parameters that are build on the *remote procedure call* (RPC) infrastructure defined as part of the DCE standard. DCE RPC defines a standard data representation for all relevant data types, the Network Data Representation (NDR). In order for COM to be able to marshal and unmarshal parameters correctly, it needs to know the exact method signature, including all data types, types of structure members, and sizes of any arrays in the parameter list. This description is provided using *Interface Definition Language* (IDL), which is also built on the DCE RPC standard IDL [7]. IDL files are compiled using a special IDL compiler (typically the Microsoft IDL compiler, or MIDL, which is part of the Win32 SDK). The IDL compiler generates C source files that contain the code for performing the marshaling and unmarshaling for the interface described in the IDL file. The client-side code is called the *proxy,* while the code running on the object's side is called the *stub.* The MIDL generated proxies and stubs are COM objects that are loaded by the COM libraries as needed. When COM needs to find the proxy/stub combination for a particular interface, it simply looks up the Interface ID in the system registry [5].

4 Project Description [8]

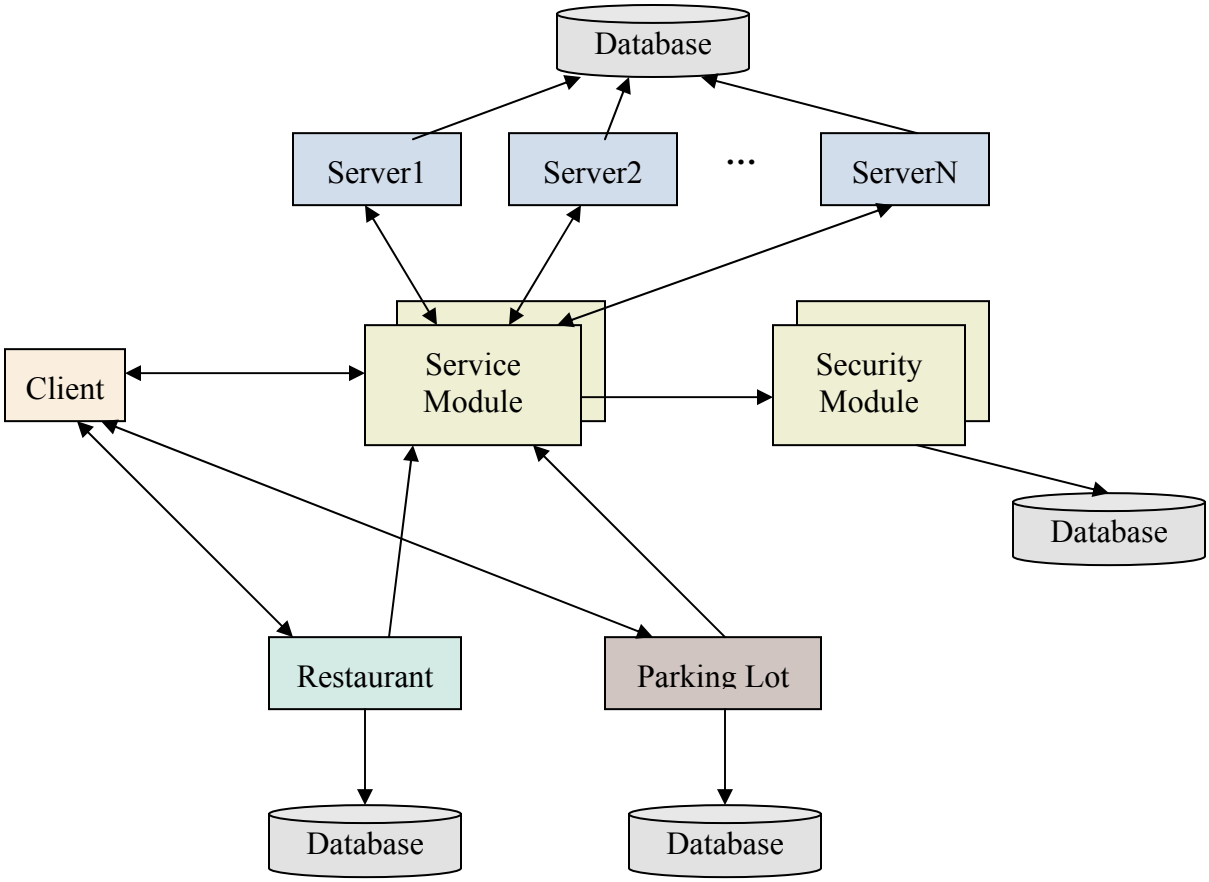
"Consider the following Scenario: The central business area of a city has several restaurants and parking places distributed geographically. It's very hard to get a parking slot and a table during lunchtime. It's even harder to get a parking slot close to a restaurant with available tables.

The aim of the project is to develop a 'service' for the *distributed computing environment* that keeps track of available parking slots and tables in restaurants in a dynamic fashion. A user can contact this 'service' through his palm device (or cell phone). Basically the user tells the service where she/he is, what kind of cuisine she/he likes and a time limit. The 'service' processes the request and finds a parking slot – restaurant match so that the distances (driving and walking) and waiting time are minimized".

The project should be developed using the Microsoft DCOM technology.

5 Project Design

The project design is depicted in the domain model shown below.



Domain Model

The DCE constitutes of a *Client*, a *Service Module* that is replicated, and number of *Servers* that serve the client request, and *Restaurants* and *Parking Lots* that subscribe with the Server. *Security Module* is responsible for authenticating the client before he\she can use the service. Servers, Restaurant, Parking Lots and Security Module maintain their individual database.

6 Prototype Description

6.1 Client

The client has a graphical user interface. Before using the service, the user has to authenticate himself with the server, via a user-name and password. The request goes to the service module, which forwards it to the security module for authentication. On entering valid user-name and password user gets the tab-screen where he can select the preferences. The user selects the type of restaurant he wants, the number of seats, maximum driving time to the restaurant, maximum waiting time in the restaurant and number of parking spaces. The user can also indicate the priority order amongst the parameters. After filling the various parameters, user sends the request

to the server. The server gets the request, does the processing required to find the restaurant-parking lot match and sends the list of restaurant that satisfy the client request. The client now gets a list of restaurant that meets his\her selection criteria. The client selects the restaurant from the list for which it wants to confirm the booking. The selected restaurant is contacted to get the seating and parking lot information. The restaurant sends it current status to the client. The client if it is interested confirms the booking. The restaurant then sends its menu to the client for browsing. Similarly it can contact the parking lot to confirm a parking space.

6.2 Restaurant

The restaurant has to register with the active servers once in a day at start-up. It contacts the service module and gets the list of active servers. Next the restaurant contacts each server and registers with them indicating the type, location and whether it has parking space available within the restaurant. Each server stores this information in its database. A client contacts the restaurant to know its current seating information. If there are no seats available, the restaurant indicates its expected waiting time. The client can confirm the seating with the restaurant.

6.3 Parking Lots

Parking lots are similar to restaurants. They also register with the server once a day at start up similar to what the restaurant does, with their location. The client can contact the parking lot for its current parking information and can confirm a parking slot.

6.4 Service Module

The service module is a routing entity. The servers on startup register with the service module. Hence the service module has a list of servers currently active. It uses this list for load balancing and fault tolerance. With each server is a load measurement. If the load on a server is over the threshold the module forwards the client request to another server. Similarly if the one server fails, the module can route the request to another active server. The service module collaborates with the security module for client authentication. When the restaurant or parking lots contacts the service module at start-up, it returns the list of active servers. The restaurant and parking lots use this list to register with each active server.

6.5 Server

The server gets the client request and finds the restaurant-parking lot match based on client preference. As the restaurants and parking lots register with the server, it has information about the restaurants and parking lots and their location. The server uses this information and a simple address matching mechanism to find the restaurant-parking lot match to send back to the client.

6.6 Caching

A Push Caching mechanism has been implemented in the project. The client has the cached data from all the previous requests. The data is stored in form of a file with a header. Header has a field, which indicates if the data is valid, or not. The client sends a request and gets the response

from the service module in form of list of restaurants and parking lots. This information is stored in form of a file. The header of the file has a field indicating that the data is valid. When any restaurant shuts down for the day, it indicates this to the server. The server then goes to the clients and set the field in the header file to invalid. If the client sends the same request again, and if the cached data is valid, the request is not send to the server but the cached data is used. Only if the data is invalid the request goes to the service module. Similar caching mechanism is also used for data received from restaurants and parking lots.

6.7 Load Balancing

The service module handles most of the loads balancing issues. When a server starts up, it registers itself with the service module. Hence the service module knows the number of servers in the environment. When it gets the request from the client the service module iterates through the list of servers. If the first server is over loaded it finds the next server with less load.

6.8 Fault Tolerance

The service module can be a single point of failure in the system. To ensure fault tolerance of service module, inbuilt services in DCOM are used. DCOM Config utility allows the user to specify the alternate location of the component. Hence the service module can be placed at more than one location. There is a primary service module and many replicated secondary service modules. All the requests from the clients go to the primary service module. If the primary service module fails, the DCOM Config utility routes the request to one of the secondary service module. At regular intervals, the state of the primary service module is replicated into all the secondary modules. As of now, the above mechanism is not implemented in the system.

6.9 Location Transparency

DCOM provides complete location transparency amongst the components. None of the components in the system know the actual physical location of any other component. They just have a handle to the component. The underlying DCOM architecture takes care of communicating with the components.

6.10 Scalability

The service module routes the request of the client to the different server depending upon the current load. Once the new servers are registered with the service module, the client request will be routed to them. The service module only handles the load balancing issue and so is not overloaded with much computation. Hence it can support increased number of servers. The number of restaurants and parking lots can be increased as required. Once the new components are registered with the server and has the DCOM application loaded, then they can communicate with the server and the clients. As the load balancing issue has been considered, scalability can be easily achieved.

Our current design is designed for one geographic zone. In future, if the application scales to more than one zone, then the current module can be replicated easily.

6.11 Security

The client authenticates using a login-password mechanism before using the service. DCOM security features will be discussed in section 7.

6.12 Services

The present system allows the client to book seats in the restaurant on-line based on his preference of cuisine, shortest distance, parking lot availability and shortest waiting period. The server sends a list of restaurants based on cuisine and shortest distance. The client selects the restaurant of preference, contacts the restaurant and reserves seats based on the availability. In future, this can be easily extended to allow the client to place the order online as the current design opens a communication channel between the client and restaurant and displays the menu.

Multi-threading is inherently supported by DCOM. Our design makes use of the security and life cycle management services provided by DCOM. In future, the event services of DCOM can also be added.

7 Technology Comparison

CORBA is an alternative middleware solution for distributed computing environment (DCE). It is a specification for distributed objects from Object Management Group (OMG). Though the methodology of implementation is different in both DCOM and CORBA, the applications for which both are used are same.

7.1 Programming

DCOM supports objects with multiple interfaces unlike CORBA. This allows for greater flexibility in programming. DCOM does efficient garbage collection. It uses pinging mechanism to collect garbage references. CORBA doesn't attempt to perform general-purpose distributed garbage collection [9].

Developing applications in DCOM is easy as most of the services are hidden from the user. In CORBA, many individual services have to be invoked by the user. The user has to be aware of many commands to invoke the ORB's [10], making usage of CORBA difficult.

The specification of ORB's varies from vendor to vendor, each having their own standards. There is a lack of inter-operability between the different ORB's.

7.2 Feature Comparison

Comparing the features of both the tools, the following differences come out [9][10][11][12]:

7.2.1 Language Independence

In CORBA, language independence is provided by the use of a common Interface Definition Language (IDL). IDL interfaces are translated to standard languages through mapping. IDL has been mapped to C, C++, Smalltalk, Ada, Java and Objective C.

DCOM achieves language independence based on binary standard. Each language is converted into its machine binary code. DCOM currently supports Microsoft products, Java, PowerBuilder, Delphi and Micro Focus COBOL.

In case of CORBA, the translation is done at the interface level and is left to the ORB vendors to create the translation. Hence the interface specification varies based on different ORB. Though CORBA offers language independence, it is still dependent on the vendor specification of ORB.

7.2.2 Network Communication

CORBA relies for cross-ORB communication on Internet Inter-ORB Protocol (IIOP), which uses TCP. CORBA only supports the TCP protocol.

DCOM utilizes UDP/DCOM, a connectionless protocol and also TCP. It also allows for IPX/SPX and Net Bios protocol. DCOM does not depend on only one protocol for communication unlike CORBA. DCOM is protocol independent.

7.2.3 Reliability

CORBA's Object Transaction Service (OTS) offers support for both flat and nested transaction. A single IDL supports both transactional and non-transactional implementations. Developers use an interface that inherits from an abstract OTS class to make an object transactional.

DCOM uses MTS, which supports transactions implicitly. This frees the developer from the complexity of dealing with the transaction services directly unlike CORBA where the user has to be involved.

7.2.4 Platform Independence

CORBA offers platform independence. DCOM works on Windows platform. Software AG has come up with products that allow DCOM to work on UNIX, Solaris and Linux.

7.2.5 Fault Tolerance

CORBA specification doesn't directly support fault tolerant services [9]. The ORB vendor is responsible for providing these services. CORBA object directory sit on a single node and does not allow object addresses to be replicated [10]. This allows a single point of failure. Performance is also degraded, as every user on the system has to access this remote node.

Basic fault tolerance is provided at the protocol level in DCOM. DCOM uses Active Directory, which creates a single directory that can be replicated around the network. Hence, there is no single point of failure. DCOM utilizes reference counts, 'keep alive' messages and pinging of

essential component of the DCOM object. To reduce network traffic generated because of this approach, positive steps like piggybacking, grouped pings and delta pinging is used.

7.2.6 Security

Both CORBA and DCOM provide good security options. DCOM has inbuilt security features and can be easily incorporated in the application design. Access, launch, security identity and connection policies are defined in DCOM [section 3]. Access security is provided in DCOM by allowing only authorized users to connect to an object. Allowing only authorized users to create instances of the object provides launch security. The capability of the object is limited by the caller's privileges, which are set in the security identity. DCOM also offers security on data communication between the caller and object by allowing the application to dynamically choose the level of security required in the connection policy.

7.2.7 Messaging

CORBA addresses messaging from a primitive standpoint [9]. Either the event service primitive or a propriety mechanism is used. Different vendors have included messaging services in par with the message-oriented middleware (MOM) but inter-service integrations is lacking.

DCOM offers messaging through Microsoft Message Queue Server (MSQM). MSQM supports all the important features of reliable messaging.

7.2.8 Market

According to Ovum report [10], Iona Technologies has the largest market share of CORBA and it has only about 2000 customers with a share of five licenses each. DCOM, because of the ease of use and as it is built on COM, has a huge market, though it came in the market after CORBA.

8 Discussion

A distributed system should have the capability to support the distributed computing environment and must have the features like fault tolerance, load balancing, location transparency and security. The advantage of using a distributed system can be realized only if the computing is distributed among the various components.

A good design of a distributed system should take into considerations the above said features. Our design incorporates most of the features . The workload is distributed between all the components. As seen from our design, the server takes the request for the clients, matches it based on cuisine and shortest distance and opens communication with the client and the restaurant/parking lot. The rest of communication is between these components. Hence the server is not overloaded with the entire computation.

Fault tolerance is an important consideration in a distributed system. In our design, if one of the servers crashes then the service module would route the request to the other server. DCOM

allows for backup of importance components. In future, the service module can be replicated and kept as a backup. . If the service module crashes, then the request can be routed immediately to the replicated service module using DCOM Config. DCOM offers inbuilt fault tolerance in the application implementation [section 7.2.5].

Distributed system should have capability to accommodate increased load and scalability. Load balancing can provide this capability. DCOM offers inbuilt features to incorporate load balance options in the design. The service module takes care of the load balancing issues in our system. Usage of distributed system is more in large centric network like the Internet, which is not a trusted network. Hence, security is a very important issue. DCOM provides inbuilt security [section 7.2.6], which can be implemented by the user easily in the system. Security issues have been incorporated in our design by authenticating the clients.

DCOM provides location transparency in the application. The clients need not know the location of the server. Application development is easy in DCOM as the user can incorporate most of the features easily. The software implicitly incorporates many of the features without the user being involved. DCOM can be used to develop reliable distributed applications as it provides features like load balancing, fault tolerance and security.

In future, the design can be extended to order food online using the service. The present design allows the display of menu; hence the online food-ordering feature can be easily incorporated. The distributed system can be expanded to different geographic zones. The client, depending upon his zone, can contact the respective service module.

9. Conclusion

DCOM, though in the market after CORBA, is rapidly gaining the ground in the DCE market. As most of the existing applications use COM objects, shifting to distributed environment using DCOM is easy as DCOM is built on COM. Hence, instead of designing from the scratch, only the additional distributed features need to be added.

DCOM is the future of DCE and slowly becoming the leader for distributed software component development because of its ease of use, language and protocol independence and its ability to provide a fault tolerant and reliable system.

10. References

[1] *An OSF White Paper*: The OSF Distributed Computing Environment: Building on International Standards.

[2] *IBM AIX Website* available at http://www-3.ibm.com/software/network/dce/library/publications/dceaix.html

[3] *IBM White Paper on Remote Procedure calls* available at http://www.transarc.ibm.com/Library/whitepapers/OSF_Whitepapers/rpc.html

[4] *Learning DCOM* – Thuan L. Thai – O'Reilly Publications

[5] *COM\DCOM Blue Book* – Nathan Wallace – CORIOLIS publications

[6] *COM\DCOM Unleashed* – Randy Abernethy – SAMS Publications

[7] *DCOM Architecture White Paper* by Microsoft Corporation available at http://www.microsoft.com/ntserver/techresources/appserv/com/DCOM/2_DCOMarchitectur.asp

[8] *Slides by Dr. Mohan Kumar from the class CSE 6306* – Spring 2002 at University of Texas at Arlington.

[9] *CORBA Vs DCOM: Solutions for the Enterprise*, Meta Group Consulting, May 1998.

[10] *CORBA no match for Microsoft's DCOM, Ovum report,* Ron Condom, Computerworld. August 1997 available at www.computerworld.com/cwi/story/0,1199,NAV47_STO21627,00.html

[11] *A Detailed Comparision of CORBA, DCOM and Java RMI*, Gopalan Suresh Raj, available at http://www.execpc.com/~gopalan/misc/compare.html

[12] *DCOM and CORBA Side by Side, Step-by-Step and Layer-by-Layer*, P Emerald Chung, et al., September 1997.