ANALYSIS OF DISTRIBUTED OBJECT SYSTEMS (DCOM VS. CORBA)

Dissertation Submitted to
JAWAHARLAL NEHRU UNIVERSITY
in partial fulfilment of requirements
for the award of the degree of
Master of Technology
in
Computer Science & Technology

by
RAJESH

Jawaharlal Nehru University
SCHOOL OF COMPUTER & SYSTEM SCIENCES
JAWAHARLAL NEHRU UNIVERSITY
NEW DELHI - 110 067
January 1998

CERTIFICATE

This is to certify that the dissertation entitled **"Analysis of Distributed Object System (DCOM vs. CORBA) "** being submitted by **'RAJESH'** to School of Computer and System Sciences, Jawaharlal Nehru University, New Delhi.in partial fulfillment of the degree of Master of Technology in Computer Science, is a bonafide work carried by him under the guidance and supervision of Prof. R.C.Phoha. This work has not been submitted elsewhere for any other purpose.

[illegible] 19/3

Dean, SC&SS
(Prof. P.C. Saxena)
Sc&SS,J.N.U.,
New Delhi-110067

[illegible]

Supervisor
Dr. R.C.Phoha
SC&SS
New Delhi-110067

ACKNOWLEDGEMENT

I express my profound sense of gratitude to Prof. R.C.Phoha, SC&SS, JNU, under whose invaluable guidance and incessant encouragement, my work has taken its present shape.

I am also indebted to my parents,brothers,sister and friends for their invaluable support and constant encouragement. I am also thankful to all those who cooperated me during the whole course work.

[illegible]

(RAJESH)

CONTENT

ABSTRACT

Distributed computing has made significant advances in the past few years. Distributed systems are becoming very popular because of the potential benefits of distributed processing. Distributed systems are intended to form the backbone of emerging next generation communication systems, including electronic commerce, PCS , satellite surveillance systems, distributed medical imaging, real-time data feeds and flight reservation systems. Distributed technology is also applied to multimedia and distributed transaction processing systems. There are several emerging product-level implementations of distributed transactions that conform to new standards. X/Open consortium has defined the X/Open Distributed transaction processing (DTP) application programming interface named XA.

Distributed processing improves functionality, performance, economics, reliability and scalability. In order to exploit this capacity appropriate support is needed that enables the development and execution of distributed applications. The supporting infrastructure should make the inherent complexity of distributed processing transparent as much as possible.

An important characteristic of large computer networks such as the internet, the world wide web and corporate intranets is that they are heterogeneous. For example, a corporate intranet might be made up of mainframes, UNIX workstations and servers, PC system running various flavours of Microsoft Windows, IBM OS/2 or Apple Macintosh and perhaps even devices such as telephone switches, robotic arms or manufacturing testbeds.

The networks and protocols underlying and connecting these systems might be just as diverse: Ethernet, fibre distributed data interface (FDDI), asynchronous transfer mode (ATM), transmission control protocol/Internet protocol (TCP/IP), Novell netware and various remote procedure calls(RPC) for example. Fundamentally .the rapidly

increasing extents of these networks are due to the need to share information and resources within and across diverse computing enterprise.

Distributed processing coupled with object oriented technique facilitates the integration of seperately developed component with application objects are inherent distributed and concurrent. Distributed object oriented system can enhance software reuse and speedup computation simultaneously. In pure object oriented programming, clients can not access the concrete state of an object directly but via the objects methods. This methodology applies beautifully to distributed computing, since the method calls are a convenient place to insert the communication required by the distributed system.

There are many such infrastructure :DCE, DCOM, CORBA, Emerald, SOS, ORCA and Amber for multiprocessor, Argus and Arjuna and many more keep arriving everyday. My study is focussed to DCOM and CORBA. I have discussed their architecture and underlying technology. Finally the capabilities of one respect to another is discussed.

Chapter - 1.

INTRODUCTION .

CORBA (Common object request broker architecture) and DCOM (distributed component object model) are popular for distributed computing. In OMG CORBA objects interact over networks using ORBs. DCOM is built on active X and OLE technologies. COM and CORBA use almost identical IDLS, both of which are derived from OSF DCE. Programming tools then compile these into proxies, stubs and type libraries that a developer can access from the actual application development languages such as C++, Visual Basic, Java or Smalltalk.

There are a number of projects that are currently employing object - oriented distributed technology and are Planned to become operational shortly. For example, the Iridium system, which is being designed and manufactured by Motorola and associated companies, intends to provide global personal satellite based communications via handhelds terminals by the end of Year 1998. Motorola using CORBA to implement portions of the Iridium system control software. Several other firms are using CORBA based technology successfully including Federal Express, Boeing, chevron Petroleum. One of the major strength of DCOM is OLE. New enhancement to OLE has been completed which gives three tier architecture. OLE is the object technology that dominates most desktop -development efforts today . As Microsoft extends OLE reach into a distributable environment, the combination of the client stranglehold with strong network services could creates an insurmountable mountain for CORBA.

The whole work is organized into 8 chapters starting chapters 2,3, and 4, discuss various concepts and techniques required in distributed system. Chapter 2 details objects oriented paradigm and ingredient factors : objects classes, Polymorphism and and overloading, Reusability, Inheritance and how these techniques are incorporated into the system. Chapter 3 gives detail of about Distributed system design issues : communication, transparency, Garbage collection, Interprocess communication, Marshalling and concurrency. Chapter 4 describes a hierarchical architecture of a typical distributed system.

Chapter 5 provides the architecture of DCOM and various issues : COM functions, Marshalling, structured storage, monikers, Uniform Data Transfer. Version management, OLE automation OLE control, OLE documents and Object services. In chapter 6 architecture of CORBA and model (OMG) in which it is based in described. Other elements of Object model are also described in brief.

Finally chapter 7 provides analysis of both DCOM and CORBA w.r.t. certain parameters : Interface definition language, object Oriented. Approach, object reference and Interpretable object reference, memory management technique, remote procedure call, Platform neutrality, Implementation, Enterprise level support, Transaction processing Security Service, Exception and error conditions. Chapter 8 is dedicated to conclusion.

CHAPTER - 2

OBJECT ORIENTED PARADIGM

OOP is a programming and design methodology in which the system to be constructed is modeled by a set of cooperating objects, which interact by message passing. Object oriented programming techniques gives more natural view of real life problems. User can not access the concrete state of an object directly, but only via the objects method(member functions).The data is hidden, so it is safe from accidental alteration. Data and its functions are said to be encapsulated into a single entity. Objects communicate with each other by calling one another's member functions. Calling an object's member function is referred to as sending a message to the object.

ANALOGY

A whole company can be modeled by set of cooperating departments each treated as an object-such as sales, accounting, personnel, and so on. Departments provide an important approach to corporate organization. Each department has its own personnel, with clearly assigned duties. It also has its own data: payroll, sales figures, personnel records, inventory, or whatever, depending on the department. The people in each department control and operate on that department's data. Dividing the company into departments makes it easier to comprehend and control the company's activities, and helps maintain the integrity of the information used by the company. The payroll department, for instance, is responsible for the payroll data. If user is from the sales department, and he wants to know the total of all the salaries paid in the southern

region in July. He should send a memo to the appropriate person in the department, and then wait for that person to access the data and send him reply with the information. This ensures that the data is accessed accurately and that it is not corrupted by outsiders. In the same way, objects provide and approach to program organization, while helping to maintain the integrity of the program's data.

A few major elements of object-oriented languages are discussed below:

2.1.OBJECTS,CLASSES :-

An object is a self-contained module. It has local state and a set of operations which can modify or return its state. Objects of the same category form a class. A class acts as a template. Objects of the same class thus have common operations and uniform behaviour. A class can inherit operations from its superclasses.

2.2. POLYMORPHISM AND OVERLOADING

Polymorphism allows the same message to be sent to different objects, which then in turn can decide how to handle each one. This is very useful when users are viewing the same basic data in a different ways a new object can be displayed differently for one user in a graphical window versus another in text window. Using operators or functions in different ways, depending on what they are operating on, is called polymorphism(one thing with several distinct forms). When a function is given the capability to operate on a new data type, it is said to be overloaded. Overloading is a kind of polymorphism.

2.3. REUSABILITY

Once a class has been written, created, and debugged, it can be distributed to other programmers for use in their own programs. This is called reusability. It is similar to the way a library of functions in a procedural language can be incorporated into different programs.

In OOP, the concept of inheritance provides an important extension to the idea of reusability. A programmer can take an existing class, and without modifying it, add additional features and capabilities to it. This is done by deriving a new class from the existing one. The new class will inherit the capabilities of the old one, but is free to add new features of its own.

2.4. INHERITANCE

Inheritance builds the hierarchical relationship between classes. A class can be divided into subclasses. Original class is called base class and the latter is called derived class. Derived class inherit some characteristics from their base class, but add new ones of their own. However, this sharing may cause the following problems :-

2.4.a) More Complexity in Dynamic Binding

If an inheritance hierarchy spans multiple nodes, the implementation of dynamic binding and execution of an operation become more complicated e.g at the handling a message, the entire inheritance hierarchy of its receiving object may be searched for the method. Therefore the search may be done in several nodes. Besides, the method may be in different address space from where the object stays. There are in general two approaches to solve the problem caused by dynamic binding:-

2.4.a.i) To restrict an inheritance hierarchy in a node and let an instance and its class be inside the same node

2.4.a.ii) Allowing multi-node inheritance hierarchy and using replication method to alleviate the dynamic-binding problem.

2.4.b) The uncertainty of behaviour of an object

If a class or its superclass(es) is redefined, the behaviour of its instances is changed accordingly. This problem is even more serious when an inheritance hierarchy is shared by multiple users (nodes).

The easiest way to solve this problem is to let a class unchangeable after it is released. While a class is being created, it is tagged with developing and during its development other users except creator should not use this class. After the development is complete it is tagged with released. There is another solution which keep multiple versions of a class. Each time a class is released, a new version is added to the class. The user of an object needs to know its version exactly.

2.5 IN THE SYSTEM

There are different ways of adopting the object oriented paradigm in distributed systems. Some systems, e.g. CORBA, provide object model for development. The object model defines a set of requirements that must be supported by that system. There are two different ways of defining the object: global identifiers, object reference. The object reference can be passed as a parameter in a operation, or stringified and stored into a file and database. Later the string can be retrieved from persistent storage and turned back into an object reference. Object references can have standardized formats, such as those for the OMG standard Internet Inter-ORB Protocol and DCE Common Inter-ORB Protocol. Naming scheme should better be node independent.

There are two design alternatives :-

a- node-wide vs. system-wide global variables .

b- unique vs. non-unique global variables :- Microsoft's COM -uses global unique identifier.

Object granularity varies from system to system . In CORBA whole application could be an object. This flexibility allows CORBA support a non-object oriented application. CORBA and DCOM does not support polymorphism. CORBA supports multiple interface inheritance i.e a class can inherit operations from multiple superclasses. CORBA and DCOM are designed to support many languages. It is achieved by mapping the interface to desired language. Every system is provided with some language which may not be full-fledged. It may not provide features like control constructs and may not be used for implementing distributed application. However language mappings determine how IDL features are mapped to the facilities of a given programming language. OMG has standardized language mappings for C, C++, Smalltalk, Ada.

CHAPTER - 3

DISTRIBUTED SYSTEM DESIGN ISSUES

Distributed systems are usually very complex. In addition to the system part which deals with the application properly a large part of the system is concerned with the communication between distributed components, exchanging data over great distances, and controlling the synchronization and consistency of the operations performed at different locations.

There are several design issues like transparency, communication, concurrency, interprocess communication, marshalling e.t.c which are discussed one by one below :-

3.1. COMMUNICATION

The communication system itself is usually built as a hierarchical layered system. **Fig 3.2 and 3.3** shows an additional level of detail. Two system components, building the process-to-process communication service out of a more primitive communication service are shown, located with each of the communicating processes respectively. They may be considered service processes which communicate with one another via the more primitive communication service, according to a particular protocol.

3.2 TRANSPARENCY

We assume that, instead of directly interacting, two processes in a distributed system communicate via some subsystem providing a communication service. Interaction should be as much transparent as possible. Possible transparency may be due to :-

a) throughput limitations
b) delay
c) limitations of the available interaction primitives
d) transmission errors

e) loss or duplication of messages

f) loss of the message sequencing

g) complicated interfaces to the communication subsystem

3.3.GARBAGE COLLECTION

Garbage collection mechanism is used for freeing the inaccessible objects. To design a distributed garbage collector, two problems must be solved. The first one is that an object is reclaimed only when it is neither locally nor remotely referenced. The second one is cyclic garbages (local and distributed). There are three Techniques for garbage collection :reference counting, marking and generation scavenging.

3.3.i REFERENCE-COUNTING

In Reference-counting technique a reference count for each object is maintained. Whenever an additional reference is created to the object the reference count is incremented and when reference is dropped reference count is decremented. When the reference count of an object reaches zero, no other objects can reference that object thus the memory occupied by that object can be reclaimed. In distributed system ,the reference to an object may be spanning several nodes. A network communication is needed to update the count. This may cause excessive system load. This scheme will not garbage-collect cycles that span address spaces. To avoid this storage leak, programmers are responsible for explicitly breaking cycles that span address spaces.

3.3.ii MARKING TECHNIQUE

In marking technique the whole object space is searched and marked. In the second phase all the objects in object space is scanned again for identifying the unmarked(inaccessible) objects and then reclaimed. Cyclic garbage problem can be can be solved. The whole normal system activities are stopped until the marking job is completed.

3.3.iii GENERATION SCAVANGING

In generation-scavenging based garbage collectors, objects are seperated into two or more generations by their ages and each generation has its own object space. Objects are initially allocated in the youngest generation. Its space is filled up quickly and then scavenged.

Most objects are dead ,only a few living objects are copied, the cost of garbage collection is low and most space can be reclaimed. This technique is based on the observation that all the objects do not have the same lifetime. It works well in local node but for using it in distributed system some modifications are required.

3.4. INTERPROCESS COMMUNICATION

Interprocess communication (IPC) provides a way of communication amongst the processes. There are various methods for IPC: shared memory/variables, pipes, message queue, semaphore. Two important issues dealt in interprocess communication are synchronization and communication. There are various synchronization primitives available with distributed systems. Some of them are :semaphore, critical region, conditional critical region, monitor and path expression. As a form of IPC, they are not used for exchanging large amounts of data but are intended to let multiple processes synchronize their operations. Main use of semaphores is to synchronize the access to shared memory segments.

Inter process communication(IPC) can be classified into three categories shared variables, message passing and message sharing.

3.4.a) SHARED VARIABLES

Two or more processes can communicate by using shared variables. Synchronization is needed for shared variables because the effects of a process on a shared variables are implicit and immediate to other process. Access of the shared variable should be mutually exclusive and

process should access the shared variable only when its state is appropriate i.e the access should be conditionally synchronized.

3.4.b) MESSAGE PASSING

Message passing is more complex than shared variables, because it involves the interaction of two processes. Message passing requires the proper synchronization between the sending and receiving processes, for a message can be received only after it has been sent. There are following issues related with message passing :-

1) naming : direct or indirect
2) synchronization : synchronous or asynchronous message passing
3) transport medium
4) argument format :Is there a format for each type of message
5) one-to-one or one-to-many communication

There are basically six models for message passing : point-to-point messages, one-to-many messages, remote procedure call, asynchronous methods call, network stream. Pipe can be implemented using Message passing technique. A pipe provides a one way flow of data. For bulk data transfer, logical pipes can be established between client and server by passing pipe references as RPC parameters. A server can then request large chunks of data via pipe dynamically, and can also send bulk data back to the client this way. This facility is available in DCE.

3.4.c) MESSAGE SHARING

Message passing is natural for distributed systems which have no direct-shared memory. Message queue(MQ) is an important technique for IPC. It requires storage medium for sharing the message. A process S that wants to reliably send message M to process R submits the message to its local MQ handler. The handler writes the content of the message on nonvolatile storage to avoid message loss if a crash occurs. After submitting the message, the process is releived of any activity necessary to deliver M. The MQ handler consists of an independent process that is responsible for storing and delivering messages on behalf of application processes. S's handler attempts to transfer the message to R's handler. If the destination handler happens to be unavailable because of downtime, a site crash, or a network partition, S's handler will attempt to deliver the message periodically until R's handler becomes available. Message queue facility can be used in communication and transaction processing .

3.5. MARSHALLING

Assembling a collection of typed data into a form suitable for being sent across a network is called marshalling. As in any distributed programming system, argument values and results are communicated by marshalling them into a sequence of bytes, transmitting the bytes from one program to the other, and then unmarshalling them into values in the receiving program. Thus It is the basic functionalities of distributed system. There are two techniques for marshalling : compiled, interpreted.

3.5.1 COMPILED STUB TECHNIQUE

Marshalling code is generated from the object type at compilation time. Pieces of code(stubs) are tailored to each type and linked to each process that needs them. A process that wants to build a message containing a given data structure invokes the stub tailored to that structure. The stub linearizes these data, e.g, it copies them on a contiguous area of memory, in particular, by dereferencing all pointers encountered, and usually converts them into machine-independent representation. The reconstruction of a data structure enclosed in a message is also performed by a stub. It follows that different message structures require different stubs ,and a process must be linked with a seperate stub for each message structure it may use. Often, an executable includes all stubs it will possibly need at run-time.

3.5.2 INTERPRETED APPROACH

A single module of code(the interpreter itself) is able to marshal any message exchanged in the distributed system, provided that it is given proper structural description(MM program). MM-programs are written in a simple language that resembles an assembly language specialized for marshalling data. The problem of generating MM-programs is essentially identical to the problem of generating marshalling and unmarshalling stubs: they may be written by hand or generated by a dedicated tool, starting from a description of the desired message structure expressed in a specialized 'Interface Definition Language'. The same MM program can be used for marshalling as well as unmarshalling the message depending upon the direction.

MM-programs are ordinary data that can be freely exchanged across heterogeneous machines. Therefore, entire machinery for building a distributed application may become simplified.

RPC can be implemented using MM-based run-time system. Moreover, applications that interact with services whose interface is not completely known at compile-time may be accomodated neatly. This feature is key for simplifying the implementation of applications that have to cope with evolving or/and unfamiliar environments. Future applications of mobile computing are going to become more and more important from this point of view. For instance, equipping a hand-held computer with an MM-based run-time system would simplify the support of scenarios in which a user that happens to be at a certain place discovers on-the-fly the services that are available at that place, and which offer themselves across wireless links. The processes implementing these services could even employ compiled stubs. Other examples may be found in those applications that must adapt to change at run-time and rely on 'self-describing objects' or in the Dynamic Invocation Interface of the CORBA architecture. For instance, a properly enriched MM could constitute a simple support for implementing this aspect of CORBA.

```
struct marks{                    ;;;;;;struct temp (number 0);;;
```

```
    int  i;                  0 intf   2
    char sub[15];            1 longf   1
 }                           2 shortf  20
                             3        callptrf 1 5 1;call procedure at offset 5 once
                             4        return
                             ;;;;;;struct marks (number 1);;;;
struct temp{                 5 intf 1
    int i;                   6 char 15
    int j;                   7 return
   long k;
   short i[20];
struct  marks *head; }
```

FIGURE :- A data structure defined in C(left) and an MM-program_that describes it (right)

3.6. CONCURRENCY

This service defines how an object mediates simultaneous access by one or more clients such that objects it access remain consistent and coherent. It is a very important tool for transaction processing. It ensures that transactional and non-transactional clients are serialized with respect to one another. This is implemented as object service in most of the Distributed object systems.

CHAPTER 4

DISTRIBUTED SYSTEM ARCHITECTURE

A four layered approach to communication system design is discussed below :-

4.1 COMMUNICATION OVER A DEDICATED CIRCUIT

A dedicated circuit is a means of transmitting data between two fixed locations. It can be considered the lowest layer in the hierarchical structure of the communication system,and provides as service the transmission of bit sequences as described below :-

4.1.a) Transmission of bit sequences

Transmission of bit sequences, simultaneously or alternatively between two locations in both direction is considered a basic communication service. Such a service is provided by analogue (e.g telephone) circuits with modems and digital circuits.

The service is characterized by :-

1- nominal transmission speed (in bits per second)

2- end-to-end delay

3- transmission error characteristics

4- possible limitations of code transparency

5- reliability and availability e.t.c

4.1.b) Framing and bit sequence transparency

The service provided by this layer is the transmission of data blocks consisting of arbitrary bit sequences (i.e there is bit sequence transparency) usually limited to a maximum length.

The service may be characterized by :

1- fixed or variable data block length, and possibly a maximum data block length.

2- the probability of a transmitted data block being lost

3- the overhead induced e.t.c

4.1.c) Transmission error detection

The service provided by this layer is the transmission of data blocks and detection of possible transmission errors. Some redundancy coding scheme is used to detect transmission errors. The service is characterized by :-

1- the probability of undetected transmission errors

2- the introduced overhead

4.1.d) Link initialization and data transfer

The link initialization layer is concerned with establishing agreement, between the communicating subsystem, on the status of the communication subsystem, its initialization, and recovery from major faults of the layer below.

The data transfer layer provides reliable data transmission by using retransmission techniques to recover from(detected) transmission errors and loss of data blocks. Following facilities should be provided

1- flow control

2- fragmentation

4.2. COMMUNICATION OVER A NETWORK

In network a given subsystem may exchange information not only with one, but with a large number of different subsystem located at different places. The different subsystems connected to a network, or several different interconnected networks, are usually distinguished by network subscriber address. The desired subsystem is selected in one of the following way :-

4..2.i- LONG TERM SELECTION

Network administration establishes "permanent" or "dedicated" circuits between subscriber addresses.

4..2.ii- MEDIUM TERM SELECTION

Real or virtual (Packet switched) circuits are established between subscriber addresses and cleared dynamically.

4.2.iii- SHORT TERM SELECTION

The address of the destination subsystem is indicated in each data packet sent through the network. This selection mode is adopted for datagrams.

4.3. A UNIFORM TRANSPORT SERVICE

The transport service provides the facility needed for communication between (logical) processes, such as application programs terminals, host computer log-in processes, data base access procedures, e.t.c. The communication system components are identified by network subscriber address and port. The communication facilities provided by the transport service may include :-

1- Process addressing, via ports

2- establishment and clearing of port-to-port associations,

3- transport of "messages" and "interrupts", directly between ports or through established associations.

4- protection against transmission errors

5- sequencing of messages (this includes against message loss and duplication)

6- flow control of messages

7- delivery confirmation e.t.c

The transport protocol layer should be designed such that :-

a- It may be implemented in many different environments in order to allow for the interworking of different computer systems

b- The same transport service can be provided using different network transmission services, such as dedicated or switched circuits, packet switched circuits or datagrams.

4.4. HIGHER LEVEL PROTOCOLS

Usually, the term "higher level protocols" denotes these layers of a distributed system (from the transport layer up) which provide functions that are general to be used by a variety of different applications. These protocols are also called "function-oriented" protocols, since each of them provides a particular set of functions used for obtaining access ,from a distance, to a given kind of resource, such as terminals, files, data bases, e.t.c. Typical examples of higher-level protocols are the following :-

a- Terminal access protocols specify the interaction between an application program and a terminal, or between two terminals. There are different access protocols :-

- line and/or page-oriented interactive characters terminals
- data entry terminals, handling forms which are structured into fields of characters
- graphics terminals
- batch terminals for remote job entry.

b- File transfer protocols specify how complete data files may be transferred from one computer system to another. It may be used for remote entry of batch processing jobs, and for many distributed processing applications.

c- File access protocols specify how an application program may selectively access certain elements of a file at a different location. Different classes of file access protocols are :-

- file transfer i.e obtaining a complete copy of distant file.
- record oriented file access ,i.e selective access.
- structure oriented file access, i.e retrieval and update access to structured databases.

There are many distributed processing systems such as : Distributed communication systems, Distributed multimedia systems, Distributed transaction systems, and various plateforms for distributed computing.

CHAPTER - 5

DISTRIBUTED COMPONENT OBJECT MODEL

Microsoft's Distributed Component object model allows component(object) wise computation. A COM object lets a client access its methods through interfaces, each of which contains one or more methods. Client software using this object can acquire individual pointers to each interface and invoke that interface's method. COM itself lets a client remain unaware of whether the object it's using is implemented in a dynamically linked library or in another process on the same machine. How a COM object calls a methods in another COM object depends on where they are running. If they are in the same process, they can call each other via pointers. Objects running in different processes interact via proxy objects and stubs that pack and unpack the called parameters into a standard format for transmission. Communication between components running on different machines takes place via remote procedure calls(RPCs)-the core technology inside distributed COM. In all these cases, however, the client object's method doesn't need to know the details of how the communication is done(location transparency). Proxys and stubs provide a static link between components, but COM also enables components to discover and call new interfaces at runtime. When a client invokes a method on a remote object, DCOM locates the object on the network and issues an RPC to the destination system. The remote object's location can be supplied by the client, stored in the client registry. or, in NT 5.0,looked up using Active Directory. Both the client and the remote object can behave just as in the local case.

COM IDL is used to define a language-independent binary interface for objects that allows them to behave in consistent ways. COM also handles all the communications between components. While COM can be used by itself for custom development, it is more commonly the basis of an integrated OLE solution that uses a variety of OLE services. **Fig 5.1** shows architecture of DCOM.

In addition to the binary object specification itself, COM includes the following features :

5.1 COM FUNCTIONs:

The COM function library provides a number of useful routines for software developers. In general, these functions begin with "Co" and have names like CoInitialize and CoCreateInstance.

5.2 MARSHALLING:

COM handles the process of packaging, sending, and unpackaging interface parameters across process, machine and network boundaries. Marshalling and unmarshalling are basically synonyms for packaging and unpackaging. The actual transport mechanism is provided by the operating system itself and is not considered part of COM. Locally, COM uses a process called "lightweight" remote procedure calls(LRPCs); remotely, it uses the industry standard Distributed Computing Environment (DCE)RPC.

5.3 STRUCTURED STORAGE:

COM provides a full-featured system for handling storage and stream objects in a robust, persistent, hierarchical manner. In general, a single structured storage object is like an entire disk volume : It has something that maps out the contents(like a file allocation table),one or more storage objects(analogous to root directories and subdirectories), and one or more stream objects(similar to files in directories). Structured storage objects can be aggregated and nested, and they can exist inside a disk file, in memory, or even as database records.

In addition to these file system-like features, structured storage also provides complete transaction processing that you can use, for example, to implement Undo operations. OLE also provides a default implementation of structured storage called compound files, from which OLE compound documents are derived.

The flexibility of COM structured storage is helpful in enabling legacy applications with OLE; they often have propietry storage models that can be difficult to reimplement. Structured storage also offers a major improvement over dealing with file systems directly, particularly for multiplateform solutions are required. While COM structured storage is fundamental when implementing servers,it can also handle custom storage needs.

5.4 MONIKERS :

As the word implies, a moniker is a name for a specific COM objects. Like a fully quantified filename, which includes drive and path information, a moniker contains information about an object as well as the instruction for connecting to it. Monikers can be serialized into

stream objects. This consistent access mechanism allows applications to automate connections to objects. COM provides built in implementations for file, item and composite monikers and allows developers to easily create their own implementations. One example is the new URL moniker, which holds a uniform resource locator that allows the client applications to access server resources on the internet using a variety of protocols.

TH-6851

5.5 UNIFORM DATA TRANSFER:

Uniform data transfer(UDT) is an important mechanism in any component based software. COM insures that OLE services using the clipboard, performing drag-and-drop operations, and doing OLE automations all use compatible data formats.

5.6 VERSION MANAGEMENT:

Using a COM interface creates a contract between the object provider and consumer.It's important that this contract not be broken as objects evolve.COM intereface version management allows adding services to objects without breaking existing applications.

5.7 OLE AUTOMATION

Controlling applications work with objects and with associated commands that are exposed by server applications. With automation, OLE's original linking-and-embedding paradigm starts to get lost : while a controller may obtain a pointer to an object in a server, it does so merely to get and set the server's properties and methods and not to create a persistent

storage objects. It's possible to serve any object that can be created in code :result sets return from database queries, real-time data, or -perhaps more powerfully-internally developed business objects for things like orders and invoices.

5.8 OLE CONTROLS :

OLE controls are mix of OLE automation server and OLE in-process server. The former allows one OLE control to expose its class modules to other OLE controls and latter is a server which a server implements as a DLL. OLE controls support embedding; OLE automation; event notification; and capability to connect objects which establishes two-way communications between object an object and an application. This link lets an object notify an application when there's a change in its data or when a user has fired an event, such as executing a mouse click. In addition ,OLE controls register themselves in the Windows registry (through the DllregisterServer function),provide licensing feature and proper editing) .

OLE controls transform user-generated events (such as mouse clicks) into messages that communicate with the application (the container in OLE parlance). OLE controls use these events to trigger event handlers that carry out the bidding of the OLE control.

There are two major steps in the process of creating an OLE Control. First we must design the control which means creating, writing, and compiling the code that draws the control

and sets up all the methods and data encapsulated inside it. The code eventually becomes a DLL with an .OCX or .DLL extension. Second, we need to design the interface that allow

Microsoft's Visual Basic, Borland's Delphi,or other appropriate development environment to use the OLE Control. OLE controls are DLLs and are not linked to a single application.

The communication between applications and OLE Controls is through a messaging interface. The host application tells the control what to do through this interface, and the control carries out the operation. Every OLE object has a receptor, known as a sink, to receive an application's instruction. In some cases, however, a user may generate an event within OLE Control, and the controls needs to communicate this to the application. OLE Controls set up these two-way communication links dynamically. The control first tells the application what language it can speak. The application then sets up the proper sink to accept this language, and the two sides make a connection.

To ease development, OLE provides standard events(called stock events) for each OLE control. These are the base events that developers build on to create their OLE controls. The OLE control parent class, Microsoft Foundation Classes'COleControl, manages stock events by default. To make an OLE Control interactive, developers must add interface methods and properties. Methods provide the OLE Control with basic behaviours: properties generally include color and fonts for use in the OLE control. Methods and properties together make up the basic mechanism that allows the appearence and values in the control to change as application processing takes place.

Internally, OLE controls are compound document objects that are controlled via OLE automation objects. They combine the features of both services .Support for OLE controls is

growing development packages such as Borland's Delphi,Microsoft's Foxpro and Access,and Visual Basic all support OLE controls. Moreover OLE is an integral part of Windows 95 and other Microsoft operating system.

5.9 OLE DOCUMENTS:

OLE documents (sometimes called OLE compound documents) are a form of compound document that incorporate data created in any OLE enabled applications. The most common example is probably an Excel Spreadsheet object embedded in a word document, but a virtually unlimited number of scenerios is possible. Several OLE subservices are at work here : The object linking and embedding itself (from which OLE originally got its name but which is now a historical footnote); use of property sets within the compound documents; and the ability to edit the objects in-place. Yet another service, drag-and-drop, originated with OLE documents but has recently been extended to places like the new Windows shell, so it's better to think of drag-and-drop as a seperate OLE service. Application programs that create compound documents are called OLE containers and applications that furnish objects are called OLE servers. It is possible for an application to be both an OLE container and an OLE server. which is the case with both Microsoft Word and Excell.

5.9.1.i LINKING & EMBEDDING :

In addition to static information like the worksheet mentioned earlier, OLE document can also incorporate live elements such as multimedia an external services (stock market and sports information feeds, e.g). If an object is linked, it still resides outside the compound document, typically on a server where multiple users have access to a single version. What's more ,when you update source object, documents that include links back to the source are automatically updated, too.

Embedded objects are contained within and actually travel with the compound document. The data in such objects becomes the part of the container program's data file. The original data file becomes irrelevant.

Linking and embedding also lets you convert the same object to different types. This means multiple applications can work with the same object, so it is not necessary for all users to have the same OLE server.

5.9.1.ii PROPERTY SETS :

OLE documents define an extension to structured storage that provides a method for storing information about objects; this information can be distinct from the objects themselves. Property sets are extensible but in general have a defined data structure, a common format, a defined header, and built in support for localized dictionaries. The only predefined property set is Document Summary Information, which contains relatively static attributes like author, subject, and date of creation, as well as dynamic attribute like word and page count. All major Microsoft applications of the past several years have provided this information. You can access it from the summary information selection on the file menu in Windows 3.1 or from the summary

tab of the document's properties page in the new Window shell. If you use this summary information, you may have also noticed that support for the Document Summary Information property set is integrated into the new Windows shell: from the shell, select a document and choose properties from the context menu; you will see additional summary and statistics tabs that are not provided for other file types such as .TXT files.

5.9.1.iii VISUAL EDITING :

Also called in-place activation, visual editing is the name for the process of editing a server object inside an OLE container. It includes support for what amounts to bringing up the server application inside the container. To do this, it's necessary to merge the menus of the two applications, display the OLE server's docked are floating toolbars ,handle keyboard integration for hot keys and accelerators, and provide for frame adornments-e.g, the top and left rulers used in most drawing applications-where applicable. This lets you remain in a familiar host application without having to activate and switch to another application.

5.10 OBJECT SERVICES

OLE is a set of object services built on top of COM. The first service distributed by Microsoft was OLE documents. Microsoft heavily marketed this service to end users, and it is what most people still think of when they hear the term OLE. The next OLE technology was OLE automation, initially useful only from Visual Basic. Next was OLE controls-Internally, a

hybrid of OLE documents and OLE automation. Now we have general purpose, industry-specific, and even internet-related services discussed below :-

5.10.1 OLE DRAG-AND-DROP:

Available in both OLE documents and OLE controls, drag-and-drop is now also a key function in the Windows95 user interface Essentially, it is another OLE service. So far, Microsoft has defined three types :**Inter-window ::-** Lets you drag objects from one application window and drop them into another-one way of embedding an object using OLE documents.

- **Inter-object::-** Lets you drag objects and drop them inside other objects.

- **Dropping into icons** ::-Lets you drag objects in the Win95 desktop and drop them onto resource icons such as printers and mailboxes. Some new OLE controls, like the Rich Text control that ships with 32bit versions of Windows, support drag-and-drop operations on the desktop.

5.10.2 INDUSTRY SOLUTIONS :

Microsoft sells vertical-market OLE services. These have been co developed with leading companies in specific industries. These industry standard make it possible to create reusable Line-Of-Business objects(LOBjects). So far, Microsoft has released specifications

for the following industries:

- WOSA/XRT (extensions for real time market data)
- OLE for Health care
- OLE for Insurance
- OLE for Retail/POS
- OLE for design and modelling

5.10.3 TRANSACTION SERVICES

Microsoft has provided specifications for more general-purpose OLE transactions. Microsoft's own products are starting to incorporate these specifications. SQL servers 6.0,e.g, uses OLE database technology, which is sometimes called SQL OLE. Similarly, OLE messaging is a key component in Microsoft's new Exchange mail server.

CHAPTER - 6

COMMON OBJECT REQUEST BROKER ARCHITECTURE (CORBA)

The OMG has developed a conceptual model, known as the core object model, and a reference architecture, called the Object Management Architecture(OMA) . In the Object model, an object is an encapsulated entity with a distinct unchanging identity whose services can be accessed only through well-defined interfaces. The implementation and location of object are hidden from the client. Object management architecture (OMA) consists of four components : Object request Broker(ORB), Object services(OS), Common facilities(CF), and Application objects(AO). These components define the composition of objects and their interfaces. Objects are categorized into Object Services, Common Facilities, and Application objects to establish the standardization for the OMG.

6.1. OBJECT REQUEST BROKER (ORB) :-

The core of the OMA is the Object Request Broker(ORB). ORB is a communication mechanism between interacting objects. It includes all functions required to support distributed computing, such as a location of objects, marshalling of request parameters and results, and object referencing. The technology adopted for ORBs is known as the Common Object Request Broker Architecture (CORBA),which specifies a framework for transparent communication between application objects. CORBA is the first specification adopted by the OMG.

The latest version is CORBA2 adopted at the end of 1994.The main features of CORBA 2.0 are described one by one (below/next page).

6.1.a CORBA INTERFACE ARCHITECTURE-

The CORBA specification defines an architecture of interfaces consisting of three specific components: client-side interface, object implementation side interfaces, and ORB Core ***a.1 Client-side interface***

a.1.1) IDL stubs.

The IDL stub presents interfaces comprised of functions generated from IDL interface definitions and linked into the client program. The stub helps to convert the request from its representation in the programming language to one suitable for transmission over the connection to the target object. Stubs are also sometimes called proxies or surrogates.

a.1.2) Dynamic Invocation Interface(DII)

It supports dynamic client request invocation. It is used for specifying and building a request at run time, rather than calling linked-in stubs. The operations that support the DII include: create_request, invoke, send, get_response requests. It can be invoked in one of three ways :

◊ **SYNCHRONOUS INVOCATION**:- The client invokes the request, and then blocks waiting for the response. It is very much like RPC in behaviour.

◊ **DEFFERED SYNCHRONOUS INVOCATION**:- Caller may choose whether or not to wait for response. It allows asynchronous interaction between the caller and callee.

◊ **ONE-WAY INVOCATION**:- The client invokes the request and then continues processing; there is no response.

a.1.3) ORB Interface

It allows functions of the ORB to be accessed directly by the client code. It is shared by both client-side and implementation-side architecture.

a.2 Implementation-side Interface

The implementation-side interfaces consist of the following up-call interfaces, allowing calls from the ORB up to object implementation :

a.2.1 IDL skeleton

This is the server-side counterpart of the IDL stub interface. ORB and the skeleton cooperate to unmarshal the request(convert it from its transmissible form to a programming language form) arrived at the target object and dispatch it to the object. Once the object completes the request, any response is sent back through the client ORB and stub, before finally being returned to the client application.

a.2.2 Dynamic skeleton Interface(DSI)

Analogous to the DII is the server-side DSI. It allows servers to be written without having skeletons for the objects being invoked compiled statically into the program.

a.2.3 Object Adaptor

The Object Adapter is the means by which object implementations access most ORB services. Though CORBA states that multiple object adapters are allowed, it currently only provides one standard called the Basic Object Adapter(BOA). OMG recently issued a Portability Enhancement RFP(14) that will result in the adoption of specifications for standard portable object adapters. Responsibilities of object adaptors include :

1.Object registration

2.object reference generation

3.Server process activation

4.Object activation

5.Request demultiplexing

6.Object upcalls-OAs dispatch requests to registered objects.

7.Security-related request(e.g authentication)

a.3 ORB CORE

The ORB Core provides the basic representation of objects and communication requests. It moves a request from a client to an appropriate adaptor for the target object.ORB hides the following :

a .Object Location: Where the object is located is unknown to the client. It may be within the same process, in a different process, across the network, on the same machine but in a different process.

b. Object Implementation: Client need not bother about how the target object implemented.

c. Object Execution State: Client does not need to know whether the object is currently in a executing process and ready to accept requests.

d. Object Communication mechanisms: The client does not need to know about the communication mechanisms(e.g. TCP/IP, shared memory, Novell netware).

6.1.b. INTERFACE DEFINITION LANGUAGE

OMG IDL is used to statically define the interfaces to objects, to allow invocation of operations on objects. An object's interface specifies the operations and types that the object supports. It is programming language-neutral and network neutral declarative language. Since OMG IDL is a declarative language, not programming language, it forces interfaces to be defined seperately from object implementation.

```
// OMG IDL specification example :myBank.idl

module BANK {

    interface BankAccount {

    // types

    enum account_kind {checking,saving};

    //exceptions

    exception account_not_available{string reason;};

    exception incorrect_pin{};

    //attributes

    readonly attribute float balance;

    attribute account_kind what_kind_of_account;

    //operations

    void access (in string account,in string pin)

raises(account_not_available,incorrect_pin);

void deposit(in float f,out float new_balance)

raises(account_not_available);

void withdraw(in float f,out float new_balance)
raises (account_not_available);

}; // end of interface Bankaccount

}; // end of module BANK
```

Interfaces are similar to classes in C++ and interfaces in Java. From the IDL definitions, it is possible to map CORBA objects into particular programming language or object systems. The OMG IDL type system is described below.

b.1 BUILT-IN TYPES- The CORBA specification precisely defines the sizes of any OMG IDL type. OMG IDL supports the following built-in types:

- long(signed and unsigned)-arithmetic types
- long long(signed and unsigned)-64-bit arithmetic types
- short(signed and unsigned)-16 bit arithmetic types
- float, double and long double-IEEE 754-1985 floating point types
- char and wchar-character and wide character types.
- boolean
- enum: enumerated type
- any: it can hold value of any OMG IDL type, including built-in types and user-defined types.

b.2 CONSTRUCTED TYPES- OMG IDL supports constructed types :

- sruct: data aggregation construct(similar to structs in C/C++)
- discriminated union-OMG IDL unions are similar unions in C/C++, with the addition of the discriminator that keeps track of which alternative is currently valid.

b.3 TEMPLATE TYPES- Characteristics of these data-types are defined at declaration time :

- string and wstring-Bounded string has length limit and unbounded string has no length limit e.g a string with maximum length of

10 characters requires angle brackets to specify the bound.

- sequence-a dynamic-length linear container whose maximum length and

  element type can be specified in angle brackets e.g sequence<string,10> and sequence<factory>

b.4 OBJECT REFERENCE TYPES- OMG IDL object reference types can be declared by naming the desired interface type. e.g

```
// OMG IDL
interface FactoryFinder {  // define a sequence of Factory
// object references
typedef sequence<Facory> FactorySeq;
FactorySeq find_factories{
in string interface_name
        }
}
```

This OMG IDL specification defines an interface named FactoryFinder that contains the definition of a type named FactorySeq. The FactorySeq type is defined as an unbounded sequence of Factory object references. The find_factories operation takes an unbounded string type as an argument and returns an unbounded sequence of Factory object references as its result.

6.1.c INTERFACE INHERITANCE- Interface inheritance makes it possible to reuse existing interfaces when defining new services. The OMG IDL specification given below shows this characteristic.

```
 interface Factory{
 object create(); };
 interface spreadsheet;
// SpreadsheetFactory derives from Factory
interface SpreadsheetFactory : Factory {
     Spreadsheet create_spreadsheet();
};
```

A derived interface inherits all operations defined in all its base interfaces.Here object supporting the SpreadsheetFactory interface provides two operations:-

1. The create operation inherited from factory
2. The create_spreadsheet operation defined directly in the SpreadsheetFactory interface.

This allows object references for derived interfaces to be substituted anywhere object references for base interfaces are allowed. Spreadsheetfactory object reference can be used anywhere that a Factory object reference is expected.

6.1.d LANGUAGE MAPPING-

OMG IDL language mappings are where the abstractions and concepts specified in the CORBA meet the "real world" of implementation. language mappings determine how OMG IDL features are mapped to the facilities of a given programming language.

To understand what a language mapping contains, consider the mapping for the C++ language. Not surprisingly, OMG IDL interfaces map to C++ classes. with operations mapping to member functions of those classes. Object references map to objects that support the_operator->function (i.e either a normal C++ pointer to an interface class, or an object instance overloaded operator->)Modules map to C++ namespaces(or to nested classes for C++). Mappings for IDL types are shown in the table below :-

OMG IDL TYPE	C++ MAPPING TYPE
long, short	long, short
float, double	float, double
enum	enum
char	char

boolean	bool
octet	unsigned char
any	any class
struct	struct
union	class
string	char *
wstring	wchar_t*
sequence	class
fixed	fixed template class
object reference	pointer or object
interface	class

TABLE 1. C++ mappings for OMG IDL types (Ref. No. 11)

6.1.e INTERFACE REPOSITRY-Using an interface repository, a client should be able to locate an object unknown at compile time, enquire about its interface, and then build a request to be forwarded through the ORB.OMG IDL type specification is required by application at execution time because :

1.application must know the types of values to be passed as request arguments

2.application must know the types of interfaces supported by the object

Usually type system is fixed at compile time but it may sometimes be required at runtime e.g, if a client application depends on the Factory interface, and the name of the create operation in the Factory interface is changed to create_object, the client application will have to be rebuilt before it can make requests on any Factory objects.

There are two ways of doing this-

⇒ .An application starts at the top-level scope of the IR and iterate over all of the module definitions defined there. When the desired module is found, it can open it and iterate in a similar manner over all the definitions inside it. This hierarchical traversal approach can be used to examine all the information stored within an IR.

⇒ .Obtain an InterfaceDef object reference from the get_interface_ operation defined in CORBA.

6.1.f ORB INTEROPERABILITY

ORB-interoperability specifies a comprehensive, flexible approach to supporting networks of objects that are distributed across and managed by multiple, heterogeneous CORBA-compliant ORBs.

The elements of interoperability as specified in CORBA2 include:

- ORB interoperability architecture
- Inter-ORB bridge support
- General and Internet Inter-ORB Protocols(GIOPs and IIOPs).

The ORB Interoperability Architecture provides a conceptual framework for defining the elements of interoperability. The architecture clearly identifies the roles of different domains of ORB-specific information. Domains are joined by bridges. Bridges map concepts in one domain to the equivalent in another. Full interoperability requires that all concepts used in one domain be translatable into concepts in the other. A bridge that provides one-to-one protocol translation is called full bridge. Every translatable protocol needs bridge. Half bridges and mediate bridges are used to avoid the explosion in number of full bridges.CORBA2 has designed a very basic inter-ORB protocol, called general Inter-ORB Protocol(GIOP), which serves as a common backbone protocol so that the number of different combinations of "half bridges" needed between domains is minimized.

General Inter-ORB Protocol specifies transfer syntax and a standard set of message formats for ORB interoperation over any connection-oriented transport. The IIOP specifies how GIOP is built over TCP/IP transports. All variants of the GIOP, such as the Internet IOP(IIOP),share common specifications for the following:

- common data representation(CDR) including marshalling conventions
- .interoperable object reference
- .interoperable typecodes
- .IOP message content, format and semantics(independent of the method of message conveyance)

In addition the architecture accommodates Environment Specific inter-ORB Protocols(ESIOPs) that are optimized for particular environments such as DCE. The architecture clearly identifies the roles of different domains of ORB-specific information. Domains are joined by bridges, which map concepts in one domain to the equivalent in another.

6.2 OMA OBJECT SERVICES

Object services provide fundamental(infrastructure-level) object interfaces necessary for building object-oriented distributed applications. RFP1 and RFP2 led to the adoption of the OMG of a set of specifications known as Common Object Service Specification, Volumes 1 and 2 (COSS1 and COSS2). Services in COSS1 and COSS2 are discussed below:-

6.2.a COSS1

i. **Object naming service:-** A name binding is always defined relative to a naming context. Different names can be bound to an object in the same or different contexts at the same time. This service addresses many design points identified for a name service :

- naming standards
- federation of namespaces
- scope of names: The name context defines the name scope
- operations: It supports bind, unbind, lookup, and sequence operations on a name context. Rename operation is not supported.

ii Object Event Notification Service:- This service supports notification of events to interested objects. Objects perform one of the two roles: suppler(produces data) and consumer role(which processes event data). Event data are communicated between suppliers and consumers by issuing standard CORBA request. The service defines two approaches to initiating event communication: push model and pull model. Multiple suppliers can communicate with multiple consumers asynchronously using event channel.

iii. Object Lifecycle service :- This service represents a framework for creating, deleting and moving objects based on location. Any piece of code that initiates a lifecycle operation is a client. The client's model of creation is defined in terms of factory objects. Factories are objects specialized in creating objects of a specific class. They provide a uniform model for creating objects in a distributed environment.

iv. Persistent Object service :- Persistent Object Service provides common interfaces to the mechanisms used for retaining and managing the persistent state of objects in a data store independent manner.

6.2.b COSS 2

The services provided in this section are:

1. Concurrency Control Service

This service defines how an object mediates simultaneous access by one or more clients such that objects it accesses remain coherent and consistent. It can be used with Object Transaction service to coordinate the activities of concurrent transactions.

2. Externalization Service

Externalizing and internalising an object is similar to copying the object-the copy operation creates a new object which is initialized from the state of an existing object.

3. Object relationship service

This service provides for creating, deleting, navigating, and managing relationships between objects. It defines three levels of service:

a. The basic level defines relationships and roles

b. The graph level extends the basic level service with nodes

c. Specific relationships are defined by third level.

4. Object transaction service

The service provides a transactional infrastructure and facilities to develop transactional classes of objects. There are many other object services which are being debated. RFP have been issued, and some cases submissions have been presented and mergers are in progress. Some of them are : Object security services, Object time service, Object licensing service, Object properties service, Object query service, object collection service, Trading service, startup service, Object change management service.

6.3 OMA COMMON FACILITIES :

It is categorized into two : horizontal common facilities and vertical common facilities. Horizontal set of common facilities includes functions covering many or more systems regardless of application content like user interface, information management, system management, task management. Vertical set of common facilities represent technology which supports various vertical market segments, such as financial systems or CAD systems.

Many more common facilities have been proposed in CF RFP1,CF RFP2,and CF RFP3 for technologies in the horizontal common facilities: compound presentation facility, compound interchange facility, internationalization facility, time operation facility, data interchange facility, mobile agent facility.

6.4 DOMAIN INTERFACES :

These interfaces are oriented toward specific application domains. An example of domain interface is Product Data management Enabler for manufacturing domain. OMG RFPs have already been issued or will be issued in the telecommunications, medical and financial domains.

6.5 APPLICATION INTERFACES :

These are interfaces developed specifically for a given application.

CHAPTER - 7

ANALYSIS

Similarity and differences between DCOM and CORBA are discussed below w.r.t to different parameters :-

1. Interface definition language :-

Both COM and CORBA use almost identical IDL's both of which are derived from OSF's DCE. IDL is used for defining the interface of objects(an objects interface specifies the operations and types that the object supports and thus defines the requests that can be made on the object). Programming tools then compile these interfaces into proxies, stubs, and type libraries that a developer can access from the actual application development language, such as C++, Visual Basic, Java or Smalltalk. For Java RMI, Java itself is the IDL(which works because the language is itself plateform neutral).

2.Object Oriented approach :-

CORBA is well suited for use by object-oriented languages. DCOM does not provide management classes for the method arguments or a way to link error conditions to exception mechanism. CORBA also has superior mechanism for handling arrays and sequences and provides an "any" data type for marshaling arguments whose type is not known in advance. For object-oriented languages, the DCOM interface is cumbersome and requires more low-level code than necessary. On the other hand DCOM can be used without any special gateway software-directly from popular, non object-oriented languages such as Visual Basic.

3. Object Reference and Interoperable object reference :-

Whereas COM uses globally unique identifiers(128 bit integer), objects in OMG model are identified by object references-an implementation defined type guranteed to identify the same object each time the reference is used in a request. While CORBA does not specify how references are to be implemented, it explicitly states that references are not guranteed to be unique. It has some concerns about implementation efficiency .management and interaction with legacy applications that have different approaches to objects ID. Every ORB may implement it in whatever way is convenient. This gives rise to an interoperability problem. For an object to make a request to an object within another ORB domain, ORB must understand the object reference passed over from the foreign ORB. This interoperability problem is solved in CORBA2 where a data structure, Interoperable Object Reference(IOR), has been specified. An interoperable object reference is a sequence of object-specific protocol profiles, plus a type ID. It is only used when crossing object reference domain(ORB) boundaries. IORs need not be used internally by any given ORB, and are not intended to be visible to application level ORB programmers. In COM developer provides universal identifier UUID that uniquely identifies the interface and class definition. The UUID identifies classes instead of a class name so that there may be multiple classes with same name but different functionality. CORBA uses naming system that includes the class name and optional module name.

4. Memory Management Technique :-

DCOM uses reference counting technique whereas CORBA uses distributed garbage collection technique. DCOM - When an object is created, its reference count is 1.When additional proxy connects to that object it must invoke the addref method to record reference. As references are dropped the client must call the release. When the reference count goes to 0,the objects can delete itself. CORBA does not attempt to track the number of clients communicating with a particular object. Transaction manager is integrated into distributed system.

5. Remote procedure call :-

When a client invokes a method on a remote object, DCOM locates the object on the network and issues an RPC to the destination system. Microsoft refers to the DCOM variation as object RPC(ORPC) but the packets on the wire conform almost exactly to the original DCE specification. Object management group has defined an alternative called IIOP (internet inter ORB protocol). IIOP is a little more than an RPC protocol-it also provides a redirection facility to let clients learn about objects that have moved -but it addresses the same basic problem as DCOM's ORPC.

6. Plateform Independence :-

As for as plateform support is concerned CORBA is better than DCOM. ORB is available for nearly every popular operating system. There are some OS'es for which more than one ORBs are available e.g MAC OS, AIX, MVS, OS/2 Warp, OS/400, Digital UNIX, Open VMS, HP-UX, Windows 3.x, Windows 95, Windows NT. Many vendors are working on Java versions of their ORBs, so expect to see CORBA on any plateform that has Java Virtual Machine. Currently released for Windows 95 and NT,DCOM is making its way to other plateforms.

7. Implementation :-

CORBA implementations work from a set of written standards. DCOM implementations work from source code licensed either from Microsoft or from the Open group. If there is a difference between the written DCOM standard and the source code, the source code stands as correct. OMG specifications have always been publicly available and were designed from the start to be plateform-neutral, programming-language neutral, and to support distribution, rather than having these features added piecemeal.

8. Enterprise level support :-

Enterprise-level application needs object services such a naming, event notification, transactions, concurrency control and life cycle control. Some of these facilities are provided with CORBA. For DCOM these services are on the way.

9. Transaction processing :-

Both DCOM and CORBA handles the transaction processing in different way. In CORBA it is implemented as object service. DCOM integrated with MTS(microsoft transaction server) and MTS applications are written as COM objects(Microsoft calls them Active X component). Independent software vendors can create applications that conform to MTS standards and let users combine them to build complete solutions.

10. Security Service :-

Security services are required to prevent unauthorized access to object. It is in the process of adoption in OMG. DCOM can use the service provided with Windows NT.

11. Exception and error condition:-

With COM, all methods return an HRESULT integer value that indicates the success or failure of the call. This integer value in fact is split up into a number of bit fields that allow the programmer to specify context, facility, severity, and error codes. CORBA implementations, on the other hand, provide an exception mechanism that returns errors as a structure embedded within another object called the Environment. A standard System Exception structure is defined for system-level and communications errors that can occur during a remote method call. As for as this facility is concerned CORBA is better than DCOM.

CHAPTER - 7

CONCLUSION

Both CORBA and DCOM are used for same purpose. Both of them have strengths and weaknesses. It is therefore quite difficult to decide which one will be suitable for an application. Efforts are being done to bring the both systems under one umbrella. There are two ways of achieving interoperability: **mapping and interworking**.

- Mapping solution makes objects in one system available to another .It allows one way interoperability. Mapping faces asymmetry because COM and CORBA have different techniques for accompolishing similar goals.

- There are two main approaches to interworking: system-neutral and system-centric.

a) System-neutral :-

With this approach any existing class or persistent object from CORBA can be installed into OLE and vice versa. Installed class or object appears natural to the foreign system.Each object system-OLE automation, COM and CORBA-has a corresponding Object System Adapter(OSA). When a call passes from one object system to another, the interworking product uses the two corresponding OSA's to perform a single-step conversion. The OSA acts as a compiler front end; it's responsible for reading the native-format description to the just-in-time(JIT) back end. The OSA is also responsible for managing issues such as proxy loop detection, mapping reference counting, and garbage collection across object systems.

b) System-centric :-

In System-centric approach developer begins with CORBA IDL and generates all necessary skeleton and stub code.. The IDL also generates C++ conversion code to map the developer's new OLE Automation class into CORBA. Likewise, the IDL generates C++ conversion code so that a client can use the server as an OLE Automation class. If the developer have existing OLE Automation objects, it can be used by taking the object's type library and generating IDL. From IDL client and server proxy code is generated.

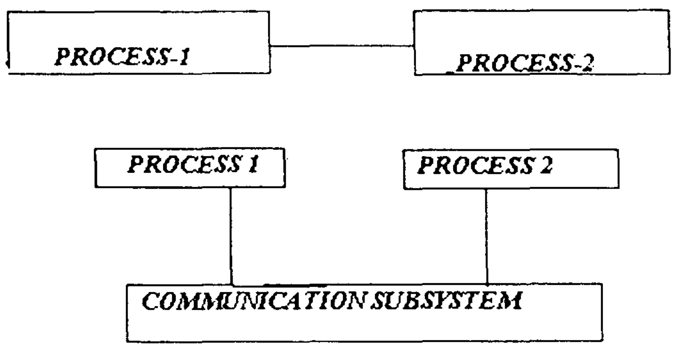


FIG. 3.1 TWO INTERACTING PROCESS

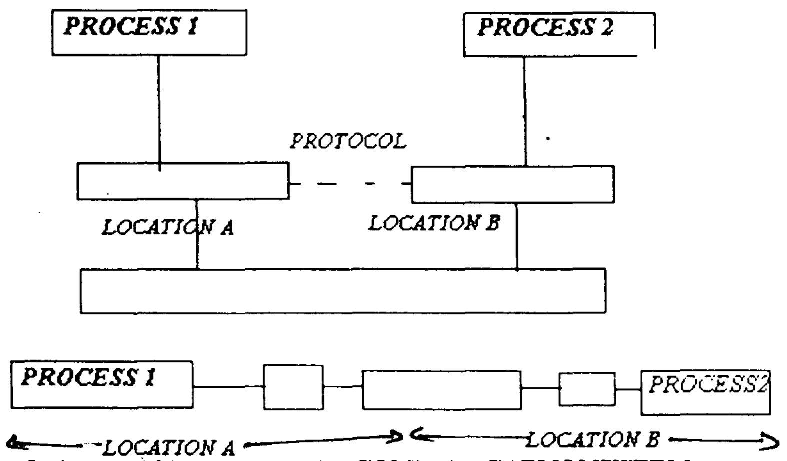


FIG. 3.2 & 3.3
HIERARCHICAL COMMUNICATION SYSTEM

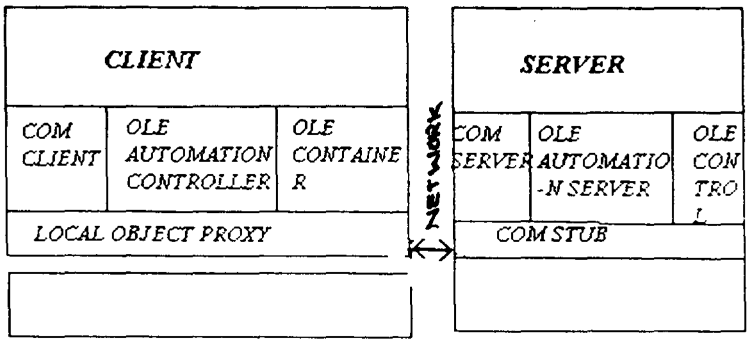
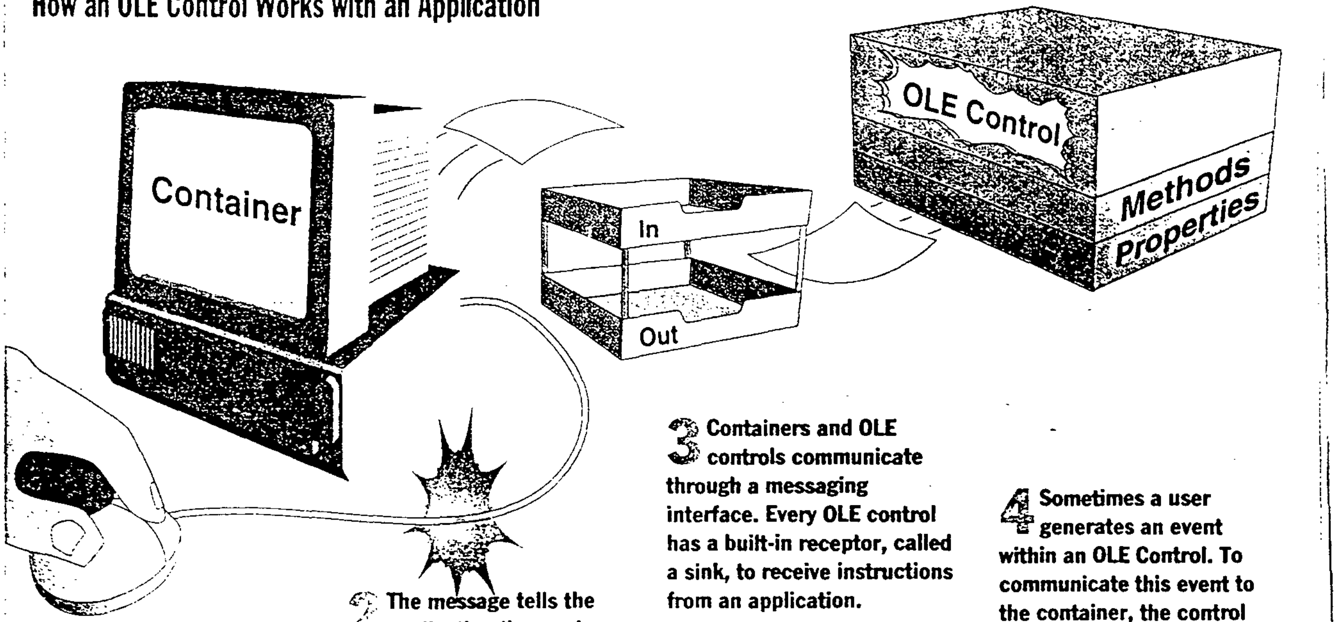


FIG5.1: DCOM ARCHITECTURE

DISTRIBUTING COM CALLS FOR AN OBJECT ON THE CLIENT MACHINE.tHE PROXY PASSES OBJECT CALLS OVER THE NETWORK TO A STUB ON SERVER.

FIGURE 5·2

How an OLE Control Works with an Application



User-generated events, such as a mouse click, send off a message to the OLE application.

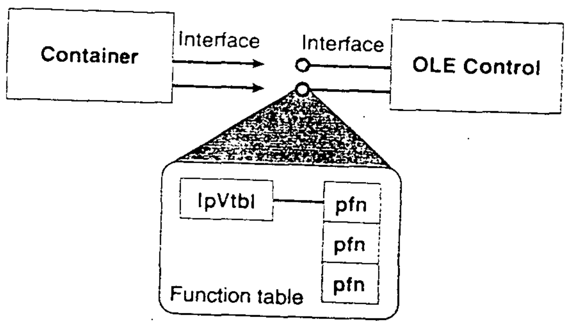
2 The message tells the application (known in OLE parlance as a container) what action it should perform.

3 Containers and OLE controls communicate through a messaging interface. Every OLE control has a built-in receptor, called a sink, to receive instructions from an application.

4 Sometimes a user generates an event within an OLE Control. To communicate this event to the container, the control tells the container to dynamically set up a sink to receive the communication.

FIGURE 5·3

OLE Control Messaging Interface



An OLE Control interface consists of a pointer to a function table that sits between the container (an application) and the OLE Control.

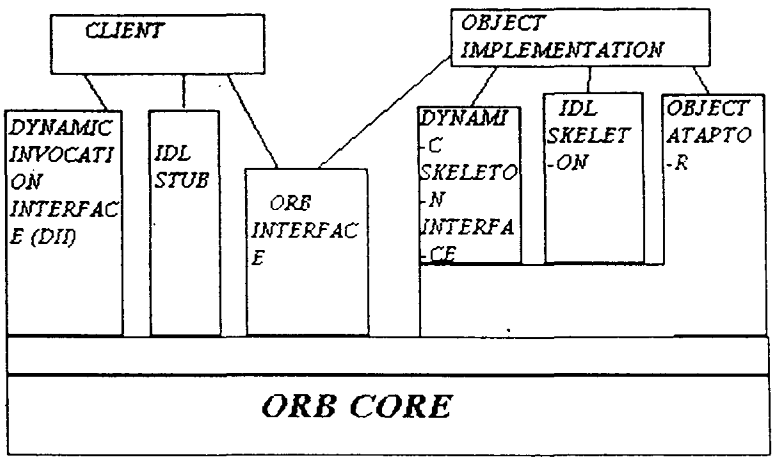


FIGURE (6.1): COMMON OBJECT REQUEST BROKER ARCHITECTURE

IN CORBA,TWO REPOSITARIES HANDLE WHERE OBJECTS ARE AND WHAT THEY DO,AND MATCH CLIENT REQUESTS WITH SERVER OBJECTS

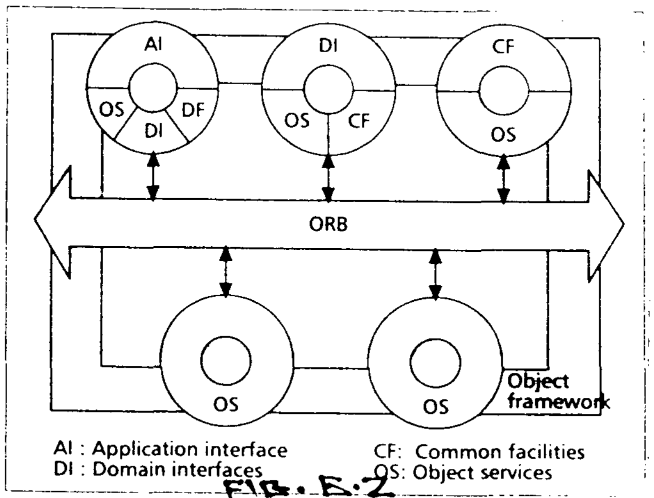


FIG. 6.2

OMA Reference Model interface usage.

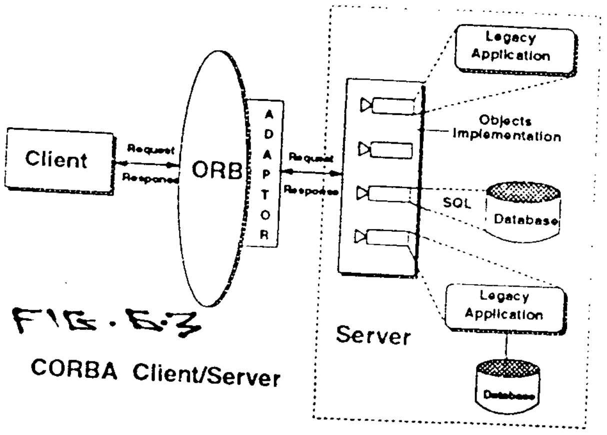
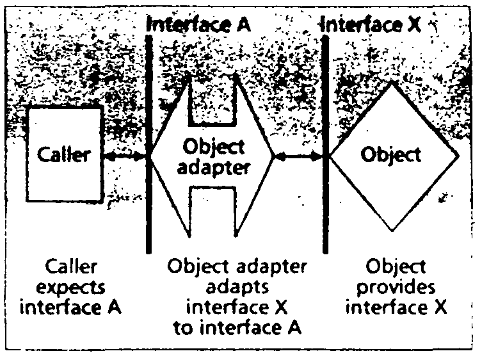


FIG. 6.3

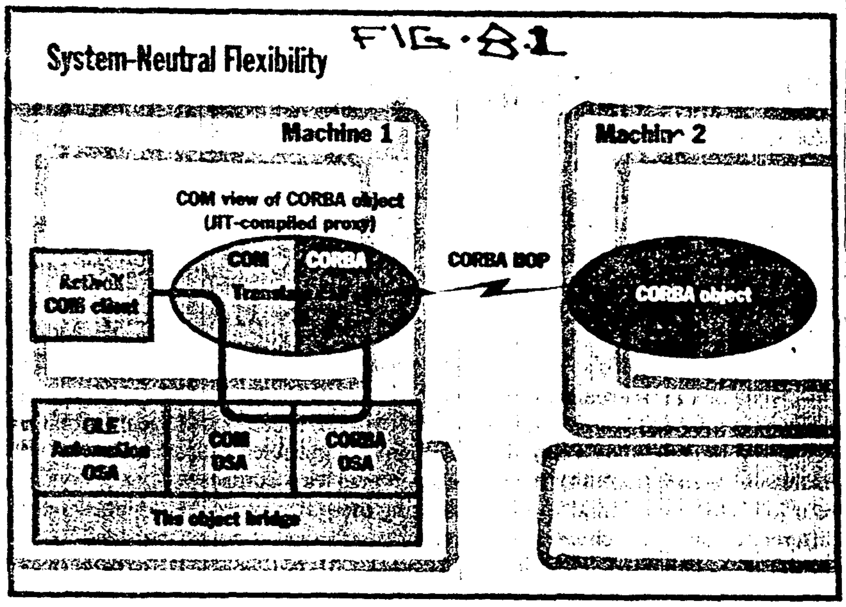
CORBA Client/Server


Interface A
Interface X
Caller
Object adapter
Object
Caller expects interface A
Object adapter adapts interface X to interface A
Object provides interface X

Role of an object adapter.

FIG. 6.4

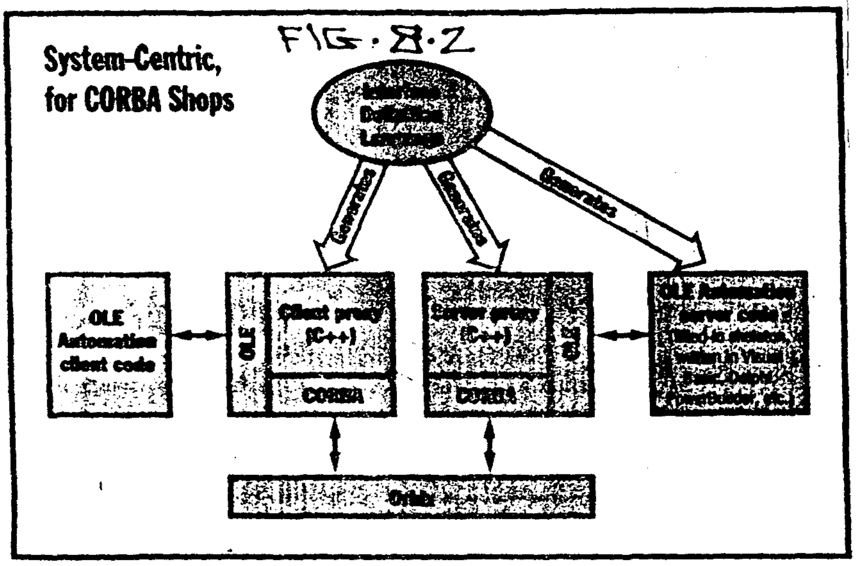
FIG. 8.1



System-neutral interworking passes calls through Object System Adapters (OSAs) to translate them for both CORBA and COM.

The Orbix/Desktop paradigm is fundamentally different from system-neutral examples, starting with the IDL.

FIG. 8.2


System-Centric, for CORBA Shops
Interface Definition Language
Generates
Generates
Generates
OLE Automation client code
OLE
Client proxy (C++)
CORBA
Server proxy (C++)
CORBA
OLE
OLE Automation server code
Orbix

APPENDIX -B: BIBLIOGRAPHY

1. A.D.Birrell and B.J.Nelson. “Implementing Remote Procedure Calls”. ACM Trans.Comp.Sys. ,vol.2, Feb. 1984,pp.39-59

2. A.Birrell, Greg Nelson, Susan Owicki and Edward Wobber, “Network Objects”, ACM Trans. Comp.Sys. ,1993,pp 217-230

3. OMG, “The Common Object Request Broker :Architecture and Specification,” v2.0, july 1995

4. S.Vinoski, “Distributed Object Computing with CORBA,” C++ Rep.,vol. 5, July/Aug. 1993.

5. John Montogomery, “Distributed Components”, Byte April 1997 pp93

6. Aberto Bartoli, “A Novel Approach to Marshalling”, Software Practice and Experience vol 27(1) Jan97 pp63-85

7. Dick Pountain and John Montogomery, “ Web Components”, Byte August97 pp56-68

8. R.M. Soley,Ph.D.,ed.,Object Management Architecture Guide,3rd ed., New York: John Wiley&Sons 1995.

9. A.Gokhale and D.C.scimdt,”Performance of the CORBA Dynamic Invocation Interface and Dynamic skeleton Interface over High-Speed ATM Networks.” Proc. GLOBECOM ‘96,London,U.K.,Nov.1996

10. David S.Linthicum,”Integration, Not Perspiration”, Byte jan96 pp83

11. S.Vinoski, “CORBA:Integrating Diverse Applications Within

Distributed Heterogeneous Environments", IEEE Communications Magazine,feb1997 , pp46

12. Silvano Maffeis,Oslen and Associates,Douglas C. Schimdt, "Constructing Reliable Distributed Systems with CORBA" , IEEE Communications Feb1997, pp56.

13. James Martin, "Computer Networks and Distributed Processing: Software Techniques and Architecture, P.H.I 1981.

14. Andrew Tanenbaum, "Computer Networks", P.H.I 1993.

15. Keith Pleas, "OLE's Missing Links", Byte April 1996 pp99

16. Gregor Von Boochman, " Concepts For Distributed Processing": Springer Verlag Berlin , Heidelberg -New York-1983

17. H.Bever, K.Geihs, L. Heuser, M. Micehlhauser,A.schill , " Distributed Systems, OSF DCE and Beyond ", Lecture Notes in Computer Science pp-1-19.

18. Jong-Hsi Huang, Feng-Jian Wang, " Some Design issues of a Distributed Object-Oriented System",Institute of Computer Science and Information Engineering ,National Chiao Tung University,Taiwan,IEEE-1990 pp -353