

Middleware Technologies - 2 Marks

UNIT I

1. What is Client/Server?

Client : The client hardware is the desktop machine that runs client software. It could be a micro or a workstation. The client software formulates data request and passes the request to the network software. This software sends the request to the server, accepts the result from the server and passes the request back to the client software.

Server : A server is the machine that runs data management software that has been designed for server functionality. A server has operating system software, data management software and a portion of the network software.

2. Define File server.

File Server : File servers manage a work group's applications and data files, so that they may be shared by the group. File servers are very I/O oriented. They pull large amounts of data off their storage subsystems and pass the data over the network. When the data from the file is requested, a file server transmit all records from the file.

3. What are the Service Specific middleware available?

1. ODBC : Open database connectivity is a windows technology that lets a database client application connect to a external database. To use ODBC, the database vendor must provide an ODBC driver for data access.
2. HTTP : Hyper Text Transfer Protocol is the underlying protocol used by the world wide web. HTTP defines how messages are formatted and transmitted and what action web servers and browsers should take in response to various commands. HTTP only supports transmission of text.

4. Define the services provided by Web server.

Web server : A webservice is a computer system that delivers web pages. Every web server has an IP address and possibly a domain name. for example an URL: <http://www.niceindia.com/index.html> in our browser. This sends a request to the server whose domain name is niceindia.com. the server fetches the page named index.html and sends back to our browsers.

5. Define TCP/IP and IPx/SPx.

TCP/IP : Transfer control Protocol and Internet Protocol.

6. What is Remote Procedure Call?

Remote Procedure Call (RPC): One of the earliest facilities that was created to help programmers write client/server software is known generically as a Remote Procedure Call mechanism. When implementing a program, the programmer uses procedures to keep the code manageable. Instead of defining a single, large procedure that performs many tasks, the programmer divides the tasks into sets and uses shorter procedure to handle each set. A client procedure process calls a function on a remote server and suspends itself until it gets back the result. This process that issues the calls and waits until it gets the result is called remote procedure call.

7. What are the different categories of client/Server applications?

Classes of client/server applications: Client/Server applications can be categorized by class, based on where most of the processing is done. Each class requires different hardware and software capabilities on the client, the server and the network.

1. Host-Based Processing
2. Client-Based Processing
3. Cooperative Processing

8. What is meant by Client, Server & Network?

Client : The client hardware is the desktop machine that runs client software. It could be a micro or a workstation. The client software formulates data request and passes the request to the network software. This software sends the request to the server, accepts the result from the server and passes the request back to the client software.

Server : A server is the machine that runs data management software that has been designed for server functionality. A server has operating system software, data management software and a portion of the network software.

Network : The network hardware is the cabling, the communication cards and the devices that link the servers and the clients.

9. Define Application server.

Application Server : Application server, which provides specific application services to an application. Application servers are the type of middleware, which occupy a large chunk of computing territory between database servers and end users.

10. What is meant by HTTP and ODBC?

1. ODBC : Open database connectivity is a windows technology that lets a database client application connect to a external database. To use ODBC, the database vendor must provide an ODBC driver for data access.
2. HTTP : Hyper Text Transfer Protocol is the underlying protocol used by the world wide web. HTTP defines how messages are formatted and transmitted and what action web servers and browsers should take in response to various commands. HTTP only supports transmission of text.

11. Define Group Server.

Groupware Server : A Groupware server is a software designed to enable users to collaborate, regardless of location through the internet or a corporate intranet and to work together in a virtual atmosphere.

12. Define TCP/IP and IPx/SPx.

TCP/IP : Transfer control Protocol and Internet Protocol.

13. What is Peer-to-Peer communication?

Peer-to-Peer : Most early client/server applications were implemented using low level conversational peer-to-peer products. The term peer-to-peer indicates that the two sides of a communication link use the same protocol interface to conduct a networked conversation. Any computer can initiate a conversation with any other computer. A peer-to-peer network relationship defines one in which computer on the network communicate with each other as equals. Each computer is responsible for making its own resources available.

14. What are the different categories of client/Server applications?

- a) Host-Based Processing
- b) Client-Based Processing
- c) Cooperative Processing

15. What is Server & Network?

Server : A server is the machine that runs data management software that has been designed for server functionality. A server has operating system software, data management software and a portion of the network software.

Network : The network hardware is the cabling, the communication cards and the devices that link the servers and the clients.

16. What is meant by Client & Network?

Client : The client hardware is the desktop machine that runs client software. It could be a micro or a workstation. The client software formulates data request and passes the request to the network software. This software sends the request to the server, accepts the result from the server and passes the request back to the client software.

Network : The network hardware is the cabling, the communication cards and the devices that link the servers and the clients.

17. What is Application server ?

Application servers, which provide specific application services to an application. An example is a server that runs a database that a distributed application users.

Application servers are the type of middleware, which occupy a large chunk of computing territory between database servers and end users, and they often connect the two.

18. What is Database servers ?

Database servers, which provide database storage and data sharing with other computers in the network. Database servers more and store data records or databases over corporate networks and across the internet.

19. What is Group Server ?

A Groupware server is a software designated to enable users to collaborate, regardless of location through the internet or a corporate intranet and to work together in a virtual atmosphere.

20. What is Web Server ?

A Web Server is a computer system that delivers web pages. Every web server has an IP address and possibly a domain name.

21. What is FTP Servers ?

FTP servers one of the Oldest Internet services. It makes it possible to move one or more files securely between computers while providing file security and organization as well as transfer control.

22. What is Mail Server?

Mail Servers move and store mail over corporate networks(Via LAN's and WAN's) and across the Internet.

23. What is News Servers?

News Servers act as a distribution and delivery source for the thousands of public news groups currently accessible over the USENET news network.

24. What is ODBC?

Open Database Connectivity is a windows technology that lets a database client application connect to a external database. To use ODBC, the Database vendor must provide an ODBC driver for data access.

25. What is HTTP?

Hyper Text Transfer Protocol is the underlying protocol used by the world wide web. HTTP defines how messages are formatted and transmitted and what action web servers and browsers should take in response to various commands.

26. What is MOM?

Message Oriented Middleware allows general-purpose message to be exchanged in a client/server systems using message queues. MOM messaging and queuing allow clients and servers to communicate across a network without being linked by a private, dedicated, logical connection.

27. What is File server ?

File server manage a work groups applications and data files, so that they may be shared by the group. File servers are very I/O oriented, they pull large amount of data off their storage subsystems and pass the data over the network. When data from the file is requested, a file server transmit all records of a file and the entire index to the client.

UNIT II & III

28. What is EJB?

Enterprise Java Bean is a server-side component architecture that simplifies the process of building enterprise-class distributed component application in Java.

29. What is the role of system administrator in EJB Server?

System administrators are responsible for the day-to-day operations of the EJB Server. Their responsibilities include keeping security information up-to-date and monitoring the performance of the server.

30. Define Bean with Bean-Managed Persistent?

An Entity Bean with Bean-Managed Persistence contains code that updates the underlying database. Each Entity bean instance has an associated primary key. Logically, a primary key is a value or combination of values that allows to uniquely specify a row of data.

31. What is Entity Bean?

Entity beans are long-lived, they exist across client session, one shared by multiple clients and remain alive even after a restart of the server or other failures. An entity bean must implement the entity bean interface.

32. What is Session Bean?

Session beans are generally tied to the lifetime of a given client session. They are relatively short-lived, state full session object are created in response to a single clients request.

33. What is Enterprise Java Bean?

Enterprise Java Bean : EJBs are write-once, run-any-where middle-tire components. Enterprise JavaBeans is a server side component architecture that simplifies the process of building enterprise-class distributed component applications in java.

34. What is session Bean?

Session beans are generally tied to the lifetime of a given client session. They are relatively short-lived, statefull session object are created in response to a single clients request.

35. What is Statefull Session?

Stateful Session is a bean that changes state during a conversation, ie a bean that has instance variables.

36. What is meant by Deployment?

Deployment is the process of installing an object in server side.

37. What is Interface?

An *interface* defines the protocol of communication between two separate components of a system. The interface describes what services are provide by a component and the protocol for using those services.

38. Write short notes on Transaction Monitor?

A *transaction monitor* service oversees transactions on behalf of other objects. A transaction, in turn, is an operation or set of operation that must be performed automatically; that is, either all objects involved in the transaction must commit the transaction or all objects involved must abort the transaction.

39. Define Directory Services.

Directory services refers to a set of services that enable objects, which can be servers, businesses, or even people, to be located by other objects.

40. What is Marshaling and unmarshaling?

Marhsaling refers to the process of translating input parameters to a format that can be transmitted across a network. *Unmarshaling* is the reverse of marshaling; this process converts data from the network to output parameters.

UNIT IV

41. Define CORBA.

The *Common Object Request Broker Architecture* (CORBA) from the Object Management Group (OMG) provides a platform-independent, language-independent architecture for writing distributed, object-oriented applications. CORBA objects can reside in the same process, on the same machine, down the hall, or across the planet. The Java language is an excellent language for writing CORBA programs. Some of the features that account for this popularity include the clear mapping from OMG IDL to the Java programming language, and the Java runtime environment's built-in garbage collection.

42. What is meant by ORB?

A fundamental part of the Common Object Request Broker architecture is the *Object Request Broker* (ORB). The concept of an ORB is this: When an application component wants to use a service provided by another component, it first must obtain an object reference for the object providing that service. After an object reference is obtained, the component can call methods on that object, thus accessing the desired services provided by that object. The primary responsibility of the ORB is to resolve requests for object references, enabling application components to establish connectivity with each other.

43. What does IIOP stand for and what is its significance?

The *Internet Inter-ORB Protocol* (IIOP) is a specialization of the GIOP. IIOP is the standard protocol for communication between ORBs on TCP/IP based networks. An ORB must support IIOP (but can support other additional protocols) in order to be considered CORBA 2.0-compliant.

44. What is an Object reference?

When an object is *passed by reference*, the object itself remains "in place" while an object reference for that object is passed. Operations on the object through the object reference are actually processed by the object itself.

45. What is IDL and why is it useful?

The *Interface Definition Language* (IDL) is a standard language used to define the interfaces used by CORBA objects. The IDL specification is responsible for ensuring that data is properly exchanged between dissimilar languages. For example, the IDL `long` type is a 32-bit signed integer quantity, which can map to a C++ `long` (depending on the platform) or to a Java `int`. It is the responsibility of the IDL specification--and the IDL compilers that implement it--to define such data types in a language-independent way.

46. Write short notes on Deployment Descriptor?

Deployment is the process of specifying middleware requirements to the container. The deployment descriptor class is the base class used both Session Descriptor and Entity Descriptor.

47. Define Language Mapping.

A *language mapping* is a specification that maps IDL language constructs to the constructs of a particular programming language. For example, in the C++ language mapping, the IDL `interface` maps to a C++ `class`.

48. Define IDL Comments ?

Comments in IDL follow the same conventions as Java and C++. Both C-style and C++-style comments are allowed, as illustrated in Listing 3.1. (Note that the second comment in the listing contains embedded comment characters; these are for description purposes only and are not actually allowed by IDL.)

```
// This is a C++-style comment. Anything following the "//"
// characters, to the end of the line, is treated as part of the
// comment.
/* This is a C-style comment. Anything between the beginning
   "/*" characters and the trailing "*/" characters is treated
   as part of the comment. */
```

49. What is Partitioning ?

The grouping together of similar interfaces, constant values, and the like is commonly referred to as *partitioning* and is a typical step in the system design process (particularly in more complex systems). Partitions are also often referred to as modules (which should be no surprise) or as packages (in fact, the IDL module concept closely resembles the Java package concept--or the other way around, because IDL came first).

50. Define GIOP.

The *General Inter-ORB Protocol (GIOP)* is a high-level standard protocol for communication between ORBs. Because GIOP is a generalized protocol, it is not used directly; instead, it is specialized by a particular protocol that would then be used directly.

51. What is Distributed Computing Environment ?

The *Distributed Computing Environment* is a set of standard pioneered by the open software foundation (OSF), includes a standard for RPC. Although the DCE standard has been around for some time, and was probably a good idea, it has never gained wide acceptance and exists today as little more than an historical curiosity.

52. What is Java RMI?

Java Remote Method Invocation helps to invoke remote object from remote server.

53. What is Client Stub?

A *client stub*, which is generated by the IDL compiler, is a small piece of code that makes a particular CORBA server interface available to a client.

54. What is Server Skeleton?

A *server skeleton*, also generated by the IDL compiler, is a piece of code that provides the "framework" on which the server implementation code for a particular interface is built.

55. What is Language mapping?

A *language mapping* is a specification that maps IDL language constructs to the constructs of a particular programming language. For example, in the C++ language mapping, the IDL `interface` maps to a C++ `class`.

56. Define serialization.

Serialization refers to the encoding of an object's state into a stream, such as a disk file or network connection. When an object is serialized, it can be written to such a stream and subsequently read and *deserialized*, a process that converts the serialized data containing the object's state back into an instance of the object.

57. Define activation policy.

A server *activation policy* indicates how that particular server is intended to be accessed; for example, if there is a single server used by all clients, or a new instance of the server should be started for each client, and so on.

58. What is Callback method?

Client callback method, or simply *callback*, is a generic term given to a method that is implemented by a client and called by a server. Callbacks essentially make a client

59. Define the capabilities of OMA?

OMA capabilities include event management, licensing, object persistence, naming, security, transactions, user interface management, data interchange, and much more. The interfaces for using these capabilities are standardized by the OMG, meaning that their usage is consistent across platforms and products.

60. Why are language mappings a necessary part of CORBA?

Because CORBA object interfaces are specified in IDL, which is independent of any implementation language, it is necessary to specify a methodology for converting IDL data types to data types of the implementation language(s) chosen. The language mapping for a particular implementation language describes this methodology. Furthermore, language mappings for many common languages are standardized, meaning that an application written to use one CORBA product can be made to work with a different product with little or no modification.

61. What is Naming Service?

Naming defines how CORBA objects can have friendly symbolic names

UNIT V

62. What is meant by COM?

Component Object Model (COM) specifies an architecture, a binary standard, and a supporting infrastructure for building, using, and evolving component-based applications. It extends the benefits of object-oriented programming such as encapsulation, polymorphism, and software reuse to a dynamic and cross-process setting. *Distributed COM (DCOM)* is the distributed extension of COM. It specifies the additional infrastructure that is required to further extend the benefits to networked environments.

63. Define the term Interface ?

An *interface* is a collection of functionally related *abstract* methods, and is identified by a 128-bit globally unique identifier (GUID) called the *interface ID (IID)*.

64. What is "Comman Language Runtime" (CLR)?

CLR is .NET equivalent of Java Virtual Machine (JVM). It is the runtime that converts a MSIL code into the host machine language code, which is then executed appropriately.

65. What is Microsoft Intermediate Language (MSIL)?

A .NET programming language (C#, VB.NET, J# etc.) does not compile into executable code; instead it compiles into an intermediate code called Microsoft Intermediate Language (MSIL). As a programmer one need not worry about the syntax of MSIL - since our source code is automatically converted to MSIL

66. What is Common Type System (CTS) ?

Achieved through introduction of Common Type System (CTS). CTS, much like Java, defines every data type as a Class. Every .NET compliant language must stick to this definition. Since CTS defines every data type as a class; this means that only Object-Oriented (or Object-Based) languages can achieve .NET compliance.

67. What is Web Services?

Web services is an extension of ActiveX. Those of you who have used ASP and JSP both, know the apparent shortcomings of ASP. JSP has been enriched with the concepts of Beans and Tags. ASP equivalent for Beans and Tags was ActiveX Controls and ActiveX automation servers. Let me take a minute to explain this point a bit further. *Web Services is NOT a Microsoft proprietary standard. It is a W3Consortium standard, and has been developed by Microsoft, IBM and many other big names of the industry.*

68. What is mean by Web forms ?

Just as the Win Forms provide a unified way of developing GUI for desktop application, the Web Forms provide similar tool for web applications. Web Forms has been introduced in .NET as a part of ASP.NET. Web Forms are a forms engine, that provide a browser-based user interface.

69. What is Windows forms?

Windows forms (also called Win Forms) are used to create GUI for Windows desktop applications. The idea of Win Form has been borrowed from Windows Foundation Classes (WFC) which were used for Visual J++. Win Form provide an integrated and unified way of developing GUI. It has a rich variety of Windows controls and user interface support.

70. What is Binary standard ?

COM specifies a *binary standard* for interfaces to ensure dynamic interoperability of binary objects possibly built using different programming languages. Specifically, any COM interface must satisfy two requirements. First, its instantiation must follow a standard memory layout, which is the same as the C++ virtual function table . In other words, a COM interface pointer is a pointer to a pointer that points to an array of virtual function pointers. Second, any COM interface must inherit from the Unknown interface so that its first three methods are (1) QueryInterface() for navigating between interfaces of the same object instance, (2) AddRef() for incrementing reference counts, and (3) Release() for decrementing reference counts.

71. What is programming model?

A typical client/server interaction in COM goes like this: client starts the activation phase by calling `CoCreateInstance()` with the CLSID of the requested object and the IID of the requested interface. It gets back an interface pointer from the call. Upon returning the interface pointer, the object calls `AddRef()` on itself. In the method invocation phase, the client invokes methods of the interface through the pointer as if the object resides in its own address space. When the client needs to call methods of another interface of the same object, it calls `QueryInterface()` on the current interface and specifies the IID of the second interface. Once it gets back a pointer to the second interface, it can invoke methods as usual. When the client finishes using either interface pointer, it calls `Release()` on the pointer.

72. What is Distributed COM ?

The DCOM wire protocol extends the remoting architecture across different machines. Currently, it is specified as a set of extensions layered on top of the DCE RPC specification . It adopts DCE RPC's Network Data Representation (NDR) format for marshaling data to be transmitted across heterogeneous network. It also leverages DCE RPC's security capabilities for authentication, authorization, and message integrity. In addition, DCOM specifies the RPC interfaces for remote server activation, ID-to-endpoint resolution, remote Unknown method invocation, and pinging for robust reference counting . It also defines the data structure of object references and the DCOM-specific portion of RPC packets.

73. What is Threading Model ?

If an application allows multiple clients to concurrently invoke methods of the same COM object, some synchronization mechanisms need to be provided to protect the data. COM introduces the concept of *apartments* to allow objects with different concurrency constraints to live in the same process. An apartment is a logical grouping of objects that share the same concurrency constraints. Before a thread can use COM, it must first enter an apartment by calling `CoInitializeEx()`. Every COM process can have at most one *multithreaded apartment (MTA)*, but it can contain multiple *single-threaded apartments (STAs)*. Multiple threads can execute in an MTA concurrently, so object data in an MTA need to be properly protected. In contrast, only one thread can execute in an STA and so concurrent accesses to objects in an STA are automatically serialized.

74. What is Versioning ?

COM's approach to versioning is based on the following three requirements: first, any interface (identified by an IID) must be immutable. Second, a new implementation of the same CLSID must support existing interfaces. Finally, any client must start interacting with a server by querying an interface with an IID. Such a combination allows independent evolution of client and server software. Suppose, on a particular machine, the server software is upgraded before the client is. Since the new server supports all old interfaces, the old client can still obtain all the interface pointers that it needs and run smoothly. When the client software is also upgraded, the new client will query the new interfaces to enjoy the new features. In contrast, suppose the client software is upgraded first on another machine. The new client will try querying the new interfaces on the old server and fail. This procedure forces the new client to handle the failure by, for example, providing only old features. But it will not cause the new client to crash or unknowingly execute incorrectly. Admittedly, there are still problems in practice that remain to be solved. For example, bug fixes of an existing interface implementation may change the behavior; new implementations of the same CLSID may not be willing to carry all old implementations.

75. What is Marshalling and Remoting ? In computer programming, marshalling is the process of gathering [data](#) from one or more [applications](#) or non-contiguous sources in computer storage, putting the data pieces into a message [buffer](#), and organizing or converting the data into a format that is prescribed for a particular receiver or programming interface. Marshalling is usually required when passing the output parameters of a program written in one language as input to a program written in another language. The process of gathering data and transforming it into a standard format before it is transmitted over a [network](#) so that the data can transcend network boundaries. In order for an [object](#) to be moved around a network, it must be converted into a data stream that corresponds with the [packet](#) structure of the network transfer [protocol](#). This conversion is known as *data marshalling*. Data pieces are collected in a message [buffer](#) before they are marshaled. When the data is transmitted, the receiving computer converts the marshaled data back into an object.

76. What is Object Creation ?

To create a COM object and get an interface from the object, you call the COM library API `CoCreateInstance()`.

The parameters are:

`reclsid`

The CLSID of the coclass. For example, you can pass `CLSID_ShellLink` to create a COM object used to create shortcuts.

`pUnkOuter`

This is only used when aggregating COM objects, which is a way of taking an existing coclass and adding new methods to it. For our purposes, we can just pass NULL to indicate we're not using aggregation.

`dwClsContext`

Indicates what kind of COM servers we want to use. For this article, we will always be using the simplest kind of server, an in-process DLL, so we'll pass `CLSCTX_INPROC_SERVER`. One caveat: you should not use `CLSCTX_ALL` (which is the default in ATL) because it will fail on Windows 95 systems that do not have DCOM installed.

`riid`

The IID of the interface you want returned. For example, you can pass `IID_IShellLink` to get a pointer to an `IShellLink` interface.

`ppv`

Address of an interface pointer. The COM library returns the requested interface through this parameter.

When you call `CoCreateInstance()`, it handles looking up the CLSID in the registry, reading the location of the server, loading the server into memory, and creating an instance of the coclass you requested.

77. What is Object Destruction ?

We don't free COM objects, you just tell them that you're done using them. The **IUnknown** interface, which every COM object implements, has a method **Release()**. You call this method to tell the COM object that you no longer need it. Once you call **Release()**, you must not use the interface pointer any more, since the COM object may disappear from memory at any time.

If our app uses a lot of different COM objects, it's vitally important to call **Release()** whenever you're done using an interface. If you don't release interfaces, the COM objects (and the DLLs that contain the code) will remain in memory, and will needlessly add to your app's working set. If your app will be running for a long time, you should call the **CoFreeUnusedLibraries()** API during your idle processing. This API unloads any COM servers that have no outstanding references, so this also reduces your app's memory usage.

78. Explain COM Interfaces ?

Every COM interface is derived from **IUnknown**. The name is a bit misleading, in that it's not an unknown interface. The name signifies that if you have an **IUnknown** pointer to a COM object, you don't know what the underlying object is, since *every* COM object implements **IUnknown**. **IUnknown** has three methods:

1. **AddRef()** - Tells the COM object to increment its reference count. You would use this method if you made a copy of an interface pointer, and both the original and the copy would still be used. We won't need to use **AddRef()** for our purposes in this article.
2. **Release()** - Tells the COM object to decrement its reference count. See the previous example for a code snippet demonstrating **Release()**.

QueryInterface() - Requests an interface pointer from a COM object. You use this when a coclass implements more than one interface.

79. What is a component ?

Component is an independent piece of code that may be shared with different programs.

80. What is an Object ?

An Object is a set of code that is designed to be reusable with a well-defined interface.

81. What is DCOM ?

Distributed Component Object Model is the combination of COM plus the network protocols that allow to run a COM Object on a remote Computer.

82. What is Class Module ?

The class module in Visual Basic is where we define an object's template. Within the class we setup the properties, methods and events that the class will offer to outside users.

83. What are the elements of Class Module ?

It's merely a collection of subroutines, functions, variables and option declarations.

84. What is Module Variable ?

Variables declared outside the routines in a module are called variables.

85. What is EXE files ?

EXE files contain object code that is run under control of windows. EXE files use the portable Executable format, which is common across all windows systems, including those that run on different processors.

86. What is Dynamic Link Library?

Dynamic Link Library file contain collection of functions and subroutines that can be shared among multiple programs, including EXE programs and other DLL's.

87. What is Method Routines ?

Method Routines are nothing but functions and subroutines that are buried in a class module.

88. What is Event Declaration ?

Event Declaration allow our class objects to call a subroutine in a client program.

89. What is Indirection ?

Many software problems can be solved by one more level of indirection. Supporting indirection is a special form of providing extensibility. In most traditional programming paradigms, offering one more level of indirection often involves tricky programming hacks that may impose certain limitations. In contrast, COM builds into its architecture the support for indirection. As demonstrated in the following discussion, activation indirection can be used for on-line software update and load balancing, while call indirection can facilitate fault tolerance and object migration.

90. What is Coupling ?

Coupling means that components in separate modules are not tightly integrated with each other; an application using components in one module generally need not know about components in another module. (Of course, there is often some overlap between modules for various reasons, such as the need to share data between modules or to facilitate common functionality between modules.) When there is little or no dependency between components, they are said to be loosely coupled.

91. What is Cohesion ?

Cohesion means that interfaces within the module are tightly integrated with each other. For example, a module called InternalCombustionEngine might contain interfaces such as CylinderHead, TimingChain, Crankshaft, Piston, and many others. It is difficult to describe the purpose of one of these components without referring to the others; hence, one might say that the components are tightly cohesive. By way of comparison, you would probably find very little in common between the components of InternalCombustionEngine and, for instance, AudioSystem; InternalCombustionEngine components such as OilFilter and SparkPlug are loosely coupled to AudioSystem components such as CompactDiscPlayer and Subwoofer.

92. Explain Constructed Type in Data types?

Constructed types, which combine other types, enable the creation of user-defined types. Perhaps the most useful of these constructs is the interface, which defines the services provided by your application objects. Because IDL is, after all, the Interface Definition Language, it seems fitting that interfaces should comprise the bulk of IDL source code.

93. Explain Enumerated Data types?

The enumerated type, `enum`, allows the creation of types that can hold one of a set of predefined values specified by the `enum`. Although the identifiers in the enumeration comprise an ordered list, IDL does not specify the ordinal numbering for the identifiers. Therefore, comparing `enum` values to integral values might not be safe, and would almost certainly not be portable across languages. C and C++ also have an enumerated type that works similarly.

94. Explain the Structure data types?

IDL provides a structure type--`struct`--that contains, as in C and C++, any number of member values of disparate types (even other structs). structs are especially useful in IDL because, unlike CORBA objects (which are represented by interfaces), structs are passed by value rather than by reference. In other words, when a struct is passed to a remote object, a copy of that struct's values is created and marshaled to the remote object.

95. Explain Union Data types?

The IDL union type, like a struct, represents values of different types. The IDL union type will appear somewhat odd to C and C++ programmers, resembling something of a cross between a C/C++ union and a case statement, but Pascal programmers should recognize the format.

96. What is Discriminator?

A *discriminator*, as used in an IDL union, is a parameter that determines the value used by the union. In the example in Listing 3.5, a long was used for the discriminator; other types can be used also, including long, long long, short, unsigned long, unsigned long long, unsigned short, char, boolean, or enum. The constant values in the case statements must match the discriminator's type.

97. What is Method Signature ?

A *method signature*, often simply called a *signature*, describes what a method does (ideally, the method name should specify, at least in general terms, what the method does), what parameters (and their types) the method takes as input, and what parameters (and their types) it returns as output. `in`, `out`, and `inout` Parameters As already mentioned, parameters in a method can be declared as `in`, `out`, or `inout`. These names are fairly self-explanatory: An `in` parameter serves as input to the method; an `out` parameter is an output from the method; and an `inout` parameter serves as an input to and an output from the method.

98. What is Blocking on a remote method call ?

The term *blocking* refers to any point at which a process or thread is waiting for a particular resource or another process/thread. Within the context of CORBA, if a client invokes a remote method and must wait for the result to be returned, the client is said to *block*.

A *request* is simply another name for a remote method invocation. The term is commonly used when referring to the operation of a distributed system. In fact, when you study CORBA's Dynamic Invocation Interface (DII), you'll see that remote methods can be invoked through a Request object.

99. What is .NET?

The .NET Framework introduces a completely new model for the programming and deployment of applications. .NET is Microsoft's vision of "software as a service", a development environment in which you can build, create, and deploy your applications and the next generation of components, the ability to use the Web rather than your own computer for various services.

100. What are the major Components (Layers) of the .NET framework:

The top layer includes user and program interfaces. Windows Forms are a new way to create standard Win32 desktop applications, based on the Windows Foundation Classes (WFC) produced for J++. Web Forms provide a powerful, forms-based UI for the web. Web Services, which are perhaps the most revolutionary, provide a mechanism for programs to communicate over the Internet using SOAP. Web Services provide an analog of COM and DCOM for object brokering and interfacing, but based on Internet technologies so that allowance is made for integration even with non-Microsoft platforms. Web Forms and Web Services, comprise the Internet interface portion of .NET, and are implemented through a section of the .NET Framework referred to as ASP.NET. The middle layer includes the next generation of standard system services such as ADO.NET and XML. These services are brought under the control of the framework, making them universally available and standardizing their usage across languages.

101. What is Remoting Architecture?

We use the term *remoting architecture* to refer to the entire infrastructure that connects COM clients to out-of-process server objects. (See Figure 1.) The standard remoting architecture includes, among other things, (1) *object proxies* that act as the client-side representatives of server objects and connect directly to the client; (2) *interface proxies* that perform client-side data marshaling and are aggregated into object proxies; (3) client-side *channel objects* that use remote procedure calls (RPCs) to forward marshaled calls; (4) server-side endpoints that receive RPC requests; (5) server-side *stub manager* that dispatches calls to appropriate interface stubs; (6) *interface stubs* that perform server-side data marshaling and make actual calls on the objects; and (7) *standard marshaler* that marshals interface pointers into *object references* on the server side and unmarshals the object references on the client side. Note that interface proxies and stubs are application-specific and are generated by running an Interface Definition Language (IDL) compiler on application-supplied IDL files. The other objects are application-independent and are provided by COM.

16 Marks

1. Explain the categories of Client/Server applications.

- Client/Server (8)
- Architecture Diagram (4)
- Client/Server Application (4)

2. Explain about Remote Procedure Call (RPC) and Java RMI.

- Introduction to RPC (4)
- Diagram (2)
- Mechanics of RPC (2)
- Java RMI (8)

3. Explain the Enterprise Java Bean Architecture.

- Introduction to EJB (8)
- EJB Architecture (Physical) (4)
- EJB Architecture (Software) (4)

4. Discuss the overview of EJBs Software Architecture.

- Introduction to EJB (8)
- EJB Architecture (Software) (8)

5. Explain EJB Session Bean and EJB Entity Bean.

- EJB Session Bean (8)
- EJB Entity Bean (8)

6. Explain about EJB Conversations.

- Finding the Bean (4)
- Getting Access to the Bean (4)
- Calling the Bean (4)
- Getting Rid of the Bean (4)

7. Overview of the CORBA architecture.

- Introduction to CORBA (4)
- CORBA Architecture (12)
 - ORB, IDL, Network Model & Object Model

8. Discuss the various Roles in EJB.

- Enterprise Bean provider
- Deployer
- Application Assembler
- EJB Server provider
- EJB Container provider
- System Administrator

9. Discuss the process of Deployment.

- Deployment
- Deployment Descriptors
- Process of Deployment

10. Explain Briefly Distributed Applications.

- Distributed Applications
- Distributed System Architecture

11. Explain CORBA and Networking Model

- CORBA Communication Model (8)
- CORBA Network Diagram (4)
- Various Protocols (4)

12. Explain CORBA and Object Model

- Object Distribution
- Object references
- Object Adaptors

13. Explain the various steps involved in Building and Deploying EJB applications.

- Steps taken to Develop an Applications
- Deployment Process

14. Exploring CORBA Alternatives.

- Socket programming
- Remote procedure Call
- Distributed Computing Environment
- DCOM

15. Explain the Following

- Object Request Broker , Interface Definition Language, Object Model

16. Discuss the various CORBA Services and CORBA Facilities.

- Event Management, Licensing, Object persistence, Naming, Security, Transactions, User Interface Management, Data Interchange

17. Discuss briefly the .Net Architecture

- .NET Framework Elements
- CLR Architecture
- System Class Library
- Web Services

18. Discuss Briefly the Basics of COM?

1. Object model

The separation of interface and implementation is at the core of COM. An *interface* is a collection of functionally related *abstract* methods, and is identified by a 128-bit globally unique identifier (GUID) called the *interface ID (IID)*. In contrast, an *object class* is a *concrete implementation* of one or more interfaces, and is also identified by a GUID called the *class ID (CLSID)*. The use of GUIDs allows programmers to independently generate unique IDs without requiring central authority. An *object instance* (or *object*) is an instantiation of some object class. An *object server* is a dynamic link library (DLL) or an executable (EXE) capable of creating object instances of potentially multiple classes. A *client* is a process that invokes methods of an object.

2. Binary interface standard

COM specifies a *binary standard* for interfaces to ensure dynamic interoperability of binary objects possibly built using different programming languages. Specifically, any COM interface must satisfy two requirements. First, its instantiation must follow a standard memory layout, which is the same as the C++ virtual function table. In other words, a COM interface pointer is a pointer to a pointer that points to an array of virtual function pointers. Second, any COM interface must inherit from the Unknown interface so that its first three methods are (1) QueryInterface() for navigating between interfaces of the same object instance, (2) AddRef() for incrementing reference counts, and (3) Release() for decrementing reference counts.

3. Programming model

A typical client/server interaction in COM goes like this: client starts the activation phase by calling CoCreateInstance() with the CLSID of the requested object and the IID of the requested interface. It gets back an interface pointer from the call. Upon returning the interface pointer, the object calls AddRef() on itself. In the method invocation phase, the client invokes methods of the interface through the pointer as if the object resides in its own address space. When the client needs to call methods of another interface of the same object, it calls QueryInterface() on the current interface and specifies the IID of the second interface. Once it gets back a pointer to the second interface, it can invoke methods as usual. When the client finishes using either interface pointer, it calls Release() on the pointer.

4. Remoting architecture

We use the term *remoting architecture* to refer to the entire infrastructure that connects COM clients to out-of-process server objects. (See Figure 1.) The standard remoting architecture includes, among other things, (1) *object proxies* that act as the client-side representatives of server objects and connect directly to the client; (2) *interface proxies* that perform client-side data marshaling and are aggregated into object proxies; (3) client-side *channel objects* that use remote

procedure calls (RPCs) to forward marshaled calls; (4) server-side endpoints that receive RPC requests; (5) server-side *stub manager* that dispatches calls to appropriate interface stubs; (6) *interface stubs* that perform server-side data marshaling and make actual calls on the objects; and (7) *standard marshaler* that marshals interface pointers into *object references* on the server side and unmarshals the object references on the client side. Note that interface proxies and stubs are application-specific and are generated by running an Interface Definition Language (IDL) compiler on application-supplied IDL files. The other objects are application-independent and are provided by COM.

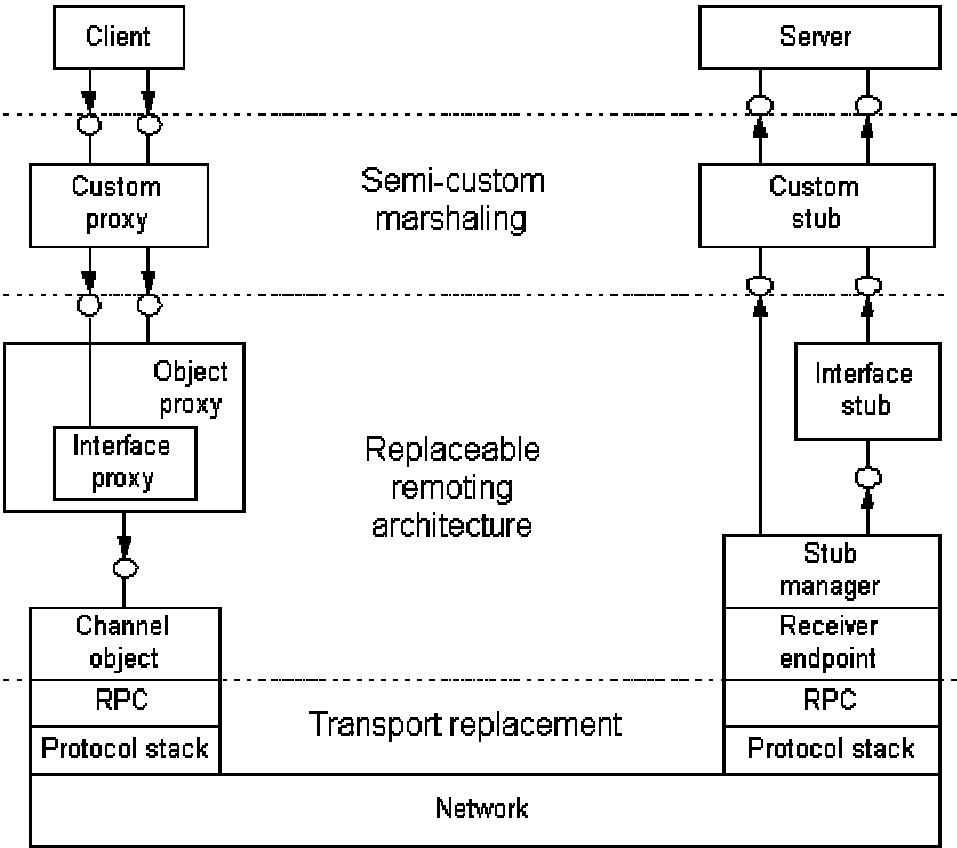


Figure 1. COM remoting architecture and extensibility.

5. DCOM

The DCOM wire protocol extends the remoting architecture across different machines. Currently, it is specified as a set of extensions layered on top of the DCE RPC specification . It adopts DCE RPC’s Network Data Representation (NDR) format for marshaling data to be transmitted across heterogeneous network. It also leverages DCE RPC’s security capabilities for authentication, authorization, and message integrity. In addition, DCOM specifies the RPC interfaces for remote server activation, ID-to-endpoint resolution, remote Unknown method invocation, and pinging for robust reference counting . It also defines the data structure of object references and the DCOM-specific portion of RPC packets.

19. Explain briefly the Features of COM ?

1. Transparency
2. Extensibility
3. Indirection
4. Versioning

1. Transparency

In the activation phase, COM supports both non-transparent and transparent modes. In the non-transparent mode, a client can explicitly specify whether the server component resides in a DLL or an EXE, and the remote machine name if the component is to be run remotely. Alternatively, the client can select the transparent mode and let COM consult the registry to determine such attributes. Once the machine name is determined, COM will try to attach to a running server instance, hosting the requested object class, on that machine. If none exists, COM will automatically locate the server implementation file, start a server process, and create an object instance.

To provide call transparency in the method invocation phase, the server returns an *object reference*, instead of a physical connection, as the result of an activation request. An object reference encapsulates all the necessary connection information for the client to reach the server object. It typically includes *machine IP address*, *port number*, and *object ID*. Although a server usually returns an object reference representing an object instance that it is hosting, it can also return an object reference that it has obtained from another machine. When the object reference is shipped to the client side, the client-side COM infrastructure *unmarshals* it by extracting the connection information, making the connection, and returning an interface pointer to the client. When the client makes a call through the pointer, the call will be transparently routed to the object identified by the object reference, without passing through the server that was initially contacted.

A good example of exploiting COM's location transparency is the auto-distribution work in the Coign project [Hunt97]. Programmers write distributed component-based applications on a single machine without having to worry about how to deploy the components on their networks. Coign then instruments the binary code to perform intercommunication analysis in scenario-based profiling. Based on the analysis results, Coign can automatically partition and distribute the components to nodes on the given network to achieve efficient execution.

6. Extensibility

Usually, COM applications use standard marshaling and rely on COM to provide standard marshaling and transport. However, some applications may need to customize the client-server connection for a number of reasons. For example, a client process may wish to cache read-only data to speed up access. An application may need to run RPC on a new transport, or may require multicast or asynchronous transport that does not fit the RPC paradigm. Some applications

need to integrate DCOM with third-party compression or encryption packages. Distributed shared object systems may wish to hide data consistency logic in the proxies. All these applications demand extensibility in the remoting architecture.

Extensibility provided by COM can be divided into three categories: *below*, *above*, and *within*. The first category extends COM at the RPC layer and below, as shown in Figure 1. The main advantage is that it is totally transparent to the standard remoting architecture. A disadvantage is that it is only applicable to transport replacement applications. To achieve the other two types of extensibility, COM supports a *custom marshaling* mechanism. By implementing an IMarshal interface, a server object indicates that it wants to bypass the standard remoting architecture and supply its own custom connection. By implementing the method calls of the IMarshal interface, the object has the flexibility of creating any number of objects and connecting them in any way to serve as the custom remoting architecture. In particular, the object can construct a custom object reference and specify the CLSID of the client-side custom unmarshaller, which will be automatically instantiated by COM to receive and interpret the custom object reference and to make the custom connection.

The second type of extensibility allows inserting a handler layer above the standard remoting architecture and below the user application. It is often called semi-custom marshaling because most of the tasks are eventually delegated to the standard remoting architecture, as shown in Figure 1. Specifically, the server object's IMarshal implementation delegates the task of marshaling to the standard marshaler, and performs additional processing on top of that. Similarly, the custom unmarshaller's IMarshal implementation also delegates the task of unmarshaling to the standard marshaler and builds additional logic on top of that. As part of the marshaling/unmarshaling process, a custom proxy and a custom stub are inserted to allow additional processing of each method invocation.

The third type of extensibility is the most general one. Very often, applications want to supply a few custom objects, while reusing most of the standard marshaling objects. For example, algorithms that manipulate marshaled data streams instead of individual call parameters may want to replace channel-level objects and reuse marshaling objects. This is hard to accomplish in current COM architecture. A new componentized architecture called *COMERA (COM Extensible Remoting Architecture)* has been proposed to promote binary software reuse at the infrastructure level [Wang98].

7. Indirection

Many software problems can be solved by one more level of indirection. Supporting indirection is a special form of providing extensibility. In most traditional programming paradigms, offering one more level of indirection often involves tricky programming hacks that may impose certain limitations. In contrast, COM builds into its architecture the support for indirection. As demonstrated in the following discussion, activation indirection can be used for on-line software update and load balancing, while call indirection can facilitate fault tolerance and object migration.

The first indirection occurs when a client requests the activation of a server object by specifying its CLSID. Since the mapping from a CLSID to the server executable filename is determined by registry keys such as TreatAs and LocalServer32, the same client can actually be invoking two different implementations with the same CLSID across two activations if a key value is changed. This provides the basis for *on-line software update*. Another indirection exists after the server receives the activation request and before it returns an interface pointer. As pointed out earlier, the server can potentially return an object reference that it has obtained from another machine. This provides the basis for *load balancing*. (See the broker example [Nelson97].)

As part of the (interface pointer) marshaling and unmarshaling process, interface stubs and proxies responsible for data marshaling are dynamically loaded and created. Since these stubs and proxies are themselves COM objects, this process provides yet another indirection point. By controlling through the registry (1) the mapping from the interface IID to the CLSID of these proxy/stub objects, or (2) the mapping from such CLSID to the implementation filename, applications can choose to either reuse the standard interface proxies and stubs or supply their own custom ones.

In standard marshaling, once an interface pointer is returned to the client, the client is bound to the server object and so it is in general harder to provide indirection for method calls. A limited form of indirection is provided by the RPC layer: if the server's IP address is failed-over to another machine and the original connection is broken, the RPC layer will retry the connection and get redirected to the new machine. The next-generation *COM+* runtime and services [Kirtland97][COM+98] will provide a general mechanism called *interceptors*, which support indirection through receiving events related to object creation as well as method invocation.

Custom marshaling provides the ultimate form of method call indirection. Basically, the entire remoting architecture can be considered as a built-in architecture for indirection. At the higher layer, custom proxies can perform client-side data-dependent routing by examining input parameters. At the lower layer, custom channels can dynamically decide on which physical connection to send a message through. This serves as the basis for *client-transparent object migration and fault tolerance*. The above approach supplies indirection logic *inside* the proxy and channel objects. Alternatively, since these objects are themselves COM objects that are dynamically created during the unmarshaling process, indirection can also be supported by either specifying different CLSIDs or changing the CLSID to filename mapping.

8. Versioning

COM's approach to versioning is based on the following three requirements: first, any interface (identified by an IID) must be immutable. Second, a new implementation of the same CLSID must support existing interfaces. Finally, any client must start interacting with a server by querying an interface with an IID. Such a combination allows independent evolution of client and server software.

Suppose, on a particular machine, the server software is upgraded before the client is. Since the new server supports all old interfaces, the old client can still obtain all the interface pointers that it needs and run smoothly. When the client software is also upgraded, the new client will query the new interfaces to enjoy the new features. In contrast, suppose the client software is upgraded first on another machine. The new client will try querying the new interfaces on the old server and fail. This procedure forces the new client to handle the failure by, for example, providing only old features. But it will not cause the new client to crash or unknowingly execute incorrectly.

Admittedly, there are still problems in practice that remain to be solved. For example, bug fixes of an existing interface implementation may change the behavior; new implementations of the same CLSID may not be willing to carry all old implementations.

9. Server lifetime management

COM supports a rich set of server styles that require different server lifetime management strategies. In the basic style, an activated server creates *class factories* for all supported CLSIDs. Object instances are created when clients make such requests. Reference counting is used to manage server lifetime: the server increments the count upon returning an interface pointer. Each client must follow a set of rules to increment the count (when duplicating an interface pointer, for example) and to decrement the count when finish using a pointer. When the count drops to zero, the server object realizes that it is no longer serving any client and can therefore be destroyed. To solve the problem of an abnormally terminated client holding a reference indefinitely, COM provides an automatic *pinging* mechanism [Brown98]: the client-side ping code starts sending periodic heartbeat messages to the server after an object reference is unmarshaled. It stops sending the heartbeats when the client process terminates. Upon detecting that the number of missing heartbeats has exceeded a threshold, the server-side ping code declares that the client has died and so decrements the server object's reference count on the client's behalf.

Long-running, singleton objects are another style of implementing COM servers. A server process creates a set of object instances upon starting and destroys them upon exiting. Since there is no class factory, clients cannot use the usual CoCreateInstance() call. Instead, they use CoGetClassObject() to connect to these shared server objects directly. Since the server is long-running and the objects' lifetime is determined by the server process' lifetime, reference counting and pinging are turned off.

20. Explain Briefly the following

- a) Object Creation
- b) Object Invocation
- c) Object Destruction

21. Discuss Briefly on IDL Data types.

IDL
Primitive Data Types
Constructed Data Types
Other IDL Constructs

22. Explain Briefly about Message Oriented Middleware (MOM)?

Definition (8)
Diagram (8)

23. Explain the various Service Specific Middleware and General Middleware ?

Service Specific Middleware (8)
General Middleware (8)

24. Explain briefly different categories of client/Server applications?

1. Host-Based Processing
2. Client-Based Processing
3. Cooperative Processing

25. Explain briefly about Distributed System

System Architecture (8)
Diagram (8)