UNIT I
Evolution and Emergence of Web Services: Evolution of distributed computing, Core distributed computing technologies, client/server, CORBA, JAVA RMI, Micro Soft DCOM, MOM, Challenges in Distributed Computing, role of J2EE and XML in distributed computing, emergence of Web Services and Service Oriented Architecture (SOA).

Evolution of distributed computing:
The Internet has revolutionized our business by providing an information highway, which acts as a new form of communication backbone. To achieve this goal of obtaining an Internet business presence, organizations are exposing and distributing their business applications over the Internet by going through a series of technological innovations. The key phenomenon of enabling business applications over the Internet is based on a fundamental technology called distributed computing. Distributed computing has been popular within local area networks for many years, and it took a major step forward by adopting the Internet as its base platform and by supporting its open standard-based technologies.

We discuss the the evolution of Internet-enabled technologies by focusing on the following:

- The definition of distributed computing
- The importance of distributed computing
- Core distributed computing technologies such as the following:
  - Client/server
  - CORBA
  - Java RMI
  - Microsoft DCOM
  - Message-Oriented Middleware
  - Common challenges in distributed computing
  - The role of J2EE and XML in distributed computing
  - Emergence of Web services and service-oriented architectures

Definition of Distributed Computing:

History:
In the early years of computing, mainframe-based applications were considered to be the best-fit solution for executing large-scale data processing applications. With the advent of personal computers (PCs), the concept of software programs running on standalone machines became much more popular in terms of the cost of ownership and the ease of application use. With the number of PC-based application programs running on independent machines growing, the communications between such application programs became extremely complex and added a growing challenge in the aspect of application-to-application interaction. Lately, network computing gained importance, and enabling remote procedure calls (RPCs) over a network protocol called Transmission Control Protocol/Internet Protocol (TCP/IP) turned out to be a widely accepted way for application software communication.
1Q. What is Distributed Computing? What are its advantages

As a definition, “Distributing Computing is a type of computing in which different components and objects comprising an application can be located on different computers connected to a network”.

Figure 1.1 shows a distributed computing model that provides an infrastructure enabling invocations of object functions located anywhere on the network. The objects are transparent to the application and provide processing power as if they were local to the application calling them.

Today, Sun Java RMI (Remote Method Invocation), OMG CORBA (Common Object Request Broker Architecture), Microsoft DCOM (Distributed Component Object Model), and Message-Oriented Middleware (MOM) have emerged as the most common distributed computing technologies. These technologies, although different in their basic architectural design and implementation, address specific problems in their target environments.

The importance of Distributed Computing:

The distributed computing environment provides many significant advantages compared to a traditional standalone application. The following are some of those key advantages:

**Higher performance**: Applications can execute in parallel and distribute the load across multiple servers.

**Collaboration**: Multiple applications can be connected through standard distributed computing mechanisms.

**Higher reliability and availability**: Applications or servers can be clustered in multiple machines.

**Scalability**: This can be achieved by deploying these reusable distributed components on powerful servers.

**Extensibility**: This can be achieved through dynamic (re)configuration of applications that are distributed across the network.
Higher productivity and lower development cycle time: By breaking up large problems into smaller ones, these individual components can be developed by smaller development teams in isolation.

Reuse: The distributed components may perform various services that can potentially be used by multiple client applications. It saves repetitive development effort and improves interoperability between components.

Reduced cost: Because this model provides a lot of reuse of once developed components that are accessible over the network, significant cost reductions can be achieved.

Core distributed computing technologies such as the following:

- Client/server
- CORBA
- Java RMI
- Microsoft DCOM
- Message-Oriented Middleware

2Q. Explain a Client/Server application model with an example. What are its limitations?

Client Server Applications:

The early years of distributed application architecture were dominated by two-tier business applications. In a two-tier architecture model, the first (upper) tier handles the presentation and business logic of the user application (client), and the second/lower tier handles the application organization and its data storage (server). This approach is commonly called client-server applications architecture.

Server: the server in a client/server application model is a database server that is mainly responsible for the organization and retrieval of data.

Client: The application client in this model handles most of the business processing and provides the graphical user interface of the application.

It is a very popular design in business applications where the user interface and business logic are tightly coupled with a database server for handling data retrieval and processing. For example, the client-server model has been widely used in enterprise resource planning (ERP), billing, and inventory application systems where a number of client business applications residing in multiple desktop systems interact with a central database server.

Figure 1.2 shows an architectural model of a typical client server system in which multiple desktop-based business client applications access a central database server.

![Figure 1.2 An example of a client-server application.](image)
limitations of the client-server application model:
- Complex business processing at the client side demands robust client systems.
- Security is more difficult to implement because the algorithms and logic reside on the client side making it more vulnerable to hacking.
- Increased network bandwidth is needed to accommodate many calls to the server, which can impose scalability restrictions.
- Maintenance and upgrades of client applications are extremely difficult because each client has to be maintained separately.
- Client-server architecture suits mostly database-oriented standalone applications and does not target robust reusable component oriented applications.

3Q. Explain the CORBA Architectural Model

CORBA:

The Common Object Request Broker Architecture (CORBA) is an industry wide, open standard initiative, developed by the Object Management Group (OMG) for enabling distributed computing that supports a wide range of application environments. OMG is a nonprofit consortium responsible for the production and maintenance of framework specifications for distributed and interoperable object-oriented systems.

CORBA differs from the traditional client/server model because it provides an object-oriented solution that does not enforce any proprietary protocols or any particular programming language, operating system, or hardware platform. By adopting CORBA, the applications can reside and run on any hardware platform located anywhere on the network, and can be written in any language that has mappings to a neutral interface definition called the Interface Definition Language (IDL).

CORBA also defines a collection of system-level services for handling low-level application services like life-cycle, persistence, transaction, naming, security, and so forth.

In a CORBA-based solution, the Object Request Broker (ORB) is an object bus that provides a transparent mechanism for sending requests and receiving responses to and from objects, regardless of the environment and its location. The ORB intercepts the client’s call and is responsible for finding its server object that implements the request, passes its parameters, invokes its method, and returns its results to the client. The ORB, as part of its implementation, provides interfaces to the CORBA services, which allows it to build custom-distributed application environments.

Figure 1.3 illustrates the architectural model of CORBA with an example representation of applications written in C, C++, and Java providing IDL bindings.
The CORBA architecture is composed of the following components:

**IDL.** CORBA uses IDL contracts to specify the application boundaries and to establish interfaces with its clients. The IDL provides a mechanism by which the distributed application component’s interfaces, inherited classes, events, attributes, and exceptions can be specified in a standard definition language supported by the CORBA ORB.

**ORB.** It acts as the object bus or the bridge, providing the communication infrastructure to send and receive request/responses from the client and server. It establishes the foundation for the distributed application objects, achieving interoperability in a heterogeneous environment.

**4Q. What are the advantages of CORBA over a traditional client/server application model?**

**Advantages of CORBA** over a traditional client/server application model are as follows:

**OS and programming-language independence.** Interfaces between clients and servers are defined in OMG IDL, thus providing the following advantages to Internet programming: Multi-language and multi-platform application environments, which provide a logical separation between interfaces and implementation.

**Legacy and custom application integration.** Using CORBA IDL, developers can encapsulate existing and custom applications as callable client applications and use them as objects on the ORB.

**Rich distributed object infrastructure.** CORBA offers developers a rich set of distributed object services, such as the Lifecycle, Events, Naming, Transactions, and Security services.

**Location transparency.** CORBA provides location transparency: An object reference is independent of the physical location and application level location. This allows developers to create CORBA-based systems where objects can be moved without modifying the underlying applications.

**Network transparency.** By using the IIOP protocol, an ORB can interconnect with any ORB located elsewhere on the network.

**Remote callback support.** CORBA allows objects to receive asynchronous event notification from other objects.

**Dynamic invocation interface.** CORBA clients can both use static and dynamic methods invocations. They either statically define their method invocations through stubs at compile time, or have the opportunity to discover objects’ methods at runtime. With those advantages, some key factors, which affected the success of CORBA evident while implementing CORBA-based distributed applications, are as follows:

**High initial investment.** CORBA-based applications require huge investments in regard to new training and the deployment of architecture, even for small-scale applications.
Availability of CORBA services. The Object services specified by the OMG are still lacking as implementation products.

Scalability. Due to the tightly coupled nature of the connection oriented CORBA architecture, very high scalability expected in enterprise applications may not be achieved.

5Q. Explain the Architectural model of JAVA RMI

JAVA RMI

Java RMI was developed by Sun Microsystems as the standard mechanism to enable distributed Java objects-based application development using the Java environment. RMI provides a distributed Java application environment by calling remote Java objects and passing them as arguments or return values. It uses Java object serialization—a lightweight object persistence technique that allows the conversion of objects into streams.

Before RMI, the only way to do inter-process communications in the Java platform was to use the standard Java network libraries. Though the java.net APIs provided sophisticated support for network functionalities, they were not intended to support or solve the distributed computing challenges.

Java RMI uses Java Remote Method Protocol (JRMP) as the interprocess communication protocol, enabling Java objects living in different Java Virtual Machines (VMs) to transparently invoke one another’s methods. Because these VMs can be running on different computers anywhere on the network, RMI enables object-oriented distributed computing. RMI also uses a reference-counting garbage collection mechanism that keeps track of external live object references to remote objects (live connections) using the virtual machine. When an object is found unreferenced, it is considered to be a weak reference and it will be garbage collected.

In RMI-based application architectures, a registry (rmiregistry)-oriented mechanism provides a simple non-persistent naming lookup service that is used to store the remote object references and to enable lookups from client applications.

The RMI infrastructure based on the JRMP acts as the medium between the RMI clients and remote objects. It intercepts client requests, passes invocation arguments, delegates invocation requests to the RMI skeleton, and finally passes the return values of the method execution to the client stub. It also enables callbacks from server objects to client applications so that the asynchronous notifications can be achieved.

Figure 1.4 depicts the architectural model of a Java RMI-based application solution.
The Java RMI architecture is composed of the following components:

**RMI client.** The RMI client, which can be a Java applet or a standalone application, performs the remote method invocations on a server object. It can pass arguments that are primitive data types or serializable objects.

**RMI stub.** The RMI stub is the client proxy generated by the *rmi compiler* (*rmic* provided along with Java developer kit—JDK) that encapsulates the network information of the server and performs the delegation of the method invocation to the server. The stub also marshals the method arguments and unmarshals the return values from the method execution.

**RMI infrastructure.** The RMI infrastructure consists of two layers: the remote reference layer and the transport layer. The remote reference layer separates out the specific remote reference behavior from the client stub. It handles certain reference semantics like connection retries, which are unicast/multicast of the invocation requests. The transport layer actually provides the networking infrastructure, which facilitates the actual data transfer during method invocations, the passing of formal arguments, and the return of back execution results.

**RMI skeleton.** The RMI skeleton, which also is generated using the RMI compiler (*rmic*) receives the invocation requests from the stub and processes the arguments (unmarshalling) and delegates them to the RMI server. Upon successful method execution, it marshals the return values and then passes them back to the RMI stub via the RMI infrastructure.

**RMI server.** The server is the Java remote object that implements the exposed interfaces and executes the client requests. It receives incoming remote method invocations from the respective skeleton, which passes the parameters after unmarshalling. Upon successful method execution, return values are sent back to the skeleton, which passes them back to the client via the RMI infrastructure.

---

**6Q. What are the Advantages and Limitations/Disadvantages of Java RMI**

**Advantages of JAVA RMI:**

Developing distributed applications in RMI is simpler than developing with Java sockets because there is no need to design a protocol, which is a very complex task by itself. RMI is built over TCP/IP sockets, but the added advantage is that it provides an object-oriented approach for interprocess communications. Java RMI provides the Java programmers with an efficient, transparent communication mechanism that frees them of all the application-level protocols necessary to encode and decode messages for data exchange. RMI enables distributed resource management, best processing power usage, and load balancing in a Java application model. RMI-IIOP (RMI over IIOP) is a protocol that has been developed for enabling RMI applications to interoperate with CORBA components.

**Limitations of JAVA RMI:**

- RMI is limited only to the Java platform. It does not provide language independence in its distributed model as targeted by CORBA.
- RMI-based application architectures are tightly coupled because of the connection-oriented nature. Hence, achieving high scalability in such an application model becomes a challenge.
- RMI does not provide any specific session management support.
Microsoft DCOM

The Microsoft Component Object Model (COM) provides a way for Windows-based software components to communicate with each other by defining a binary and network standard in a Windows operating environment. COM evolved from OLE (Object Linking and Embedding), which employed a Windows registry-based object organization mechanism.

Microsoft developed the Distributed Common Object Model (DCOM) as its answer to the distributed computing problem in the Microsoft Windows platform. DCOM enables COM applications to communicate with each other using an RPC mechanism, which employs a DCOM protocol on the wire.

Figure 1.5 shows an architectural model of DCOM.

DCOM applies a skeleton and stub approach whereby a defined interface that exposes the methods of a COM object can be invoked remotely over a network. The client application will invoke methods on such a remote COM object in the same fashion that it would with a local COM object. The stub encapsulates the network location information of the COM server object and acts as a proxy on the client side. The servers can potentially host multiple COM objects, and when they register themselves against a registry, they become available for all the clients, who then discover them using a lookup mechanism.

DCOM is quite successful in providing distributed computing support on the Windows platform. But, it is limited to Microsoft application environments. The following are some of the common limitations of DCOM:

- Platform lock-in
- State management
- Scalability
- Complex session management issues

Message Oriented Middleware (MOM):

Message-Oriented Middleware (MOM) is based upon a loosely coupled asynchronous communication model where the application client does not need to know its application
Web Services - UNIT I

SREC-NDL

S.MD.FAROOQ

recipients or its method arguments. MOM enables applications to communicate indirectly using a messaging provider queue. The application client sends messages to the message queue (a message holding area), and the receiving application picks up the message from the queue. In this operation model, the application sending messages to another application continues to operate without waiting for the response from that application.

Figure 1.6 illustrates a high-level MOM architecture showing application-to-application connectivity.

In MOM-based architecture, applications interacting with its messaging infrastructure use custom adapters. Client applications communicate with the underlying messaging infrastructure using these adapters for sending and receiving messages. For reliable message delivery, messages can be persisted in a database/file system as well.

Some of the widely known MOM-based technologies are SunONE Message Queue, IBM MQSeries, TIBCO, SonicMQ, and Microsoft Messaging Queue (MSMQ). The Java Platform provides a Java-based messaging API (JMS-Java Message Service), which is developed as part of the Sun Java Community Process (JCP) and also is currently part of the J2EE 1.3 specifications. All the leading MOM vendors like SunONE, TIBCO, IBM, BEA, Talarian, Sonic, Fiorano, and Spiritwave support the JMS specifications.

JMS provides Point-to-Point and Publish/Subscribe messaging models with the following features:

- Complete transactional capabilities
- Reliable message delivery
- Security

Some of the common challenges while implementing a MOM-based application environment have been the following:

- Most of the standard MOM implementations have provided native APIs for communication with their core infrastructure. This has affected the portability of applications across such implementations and has led to a specific vendor lock-in.
- The MOM messages used for integrating applications are usually based upon a proprietary message format without any standard compliance.
9Q. What are the Common challenges in Distributed Computing

**Common Challenges in Distributed Computing:**

some of the common challenges noticed in the CORBA-, RMI-, and DCOM-based distributed computing solutions:

- Maintenance of various versions of stubs/skeletons in the client and server environments is extremely complex in a heterogeneous network environment.
- Quality of Service (QoS) goals like Scalability, Performance, and Availability in a distributed environment consume a major portion of the application’s development time.
- Interoperability of applications implementing different protocols on heterogeneous platforms almost becomes impossible. For example, a DCOM client communicating to an RMI server or an RMI client communicating to a DCOM server.
- Most of these protocols are designed to work well within local networks. They are not very firewall friendly or able to be accessed over the Internet.
- The biggest problem with application integration with this tightly coupled approach spearheaded by CORBA, RMI, and DCOM was that it influenced separate sections of the developer community who were already tied to specific platforms. Microsoft Windows platform developers used DCOM, while UNIX developers used CORBA or RMI. There was no big effort in the community to come up with common standards that focused on the interoperability between these diverse protocols, thus ignoring the importance, and hence, the real power of distributed computing.

10Q. Explain the Role of J2EE and XML in Distributed Computing

**The Role of J2EE and XML in Distributed Computing:**

The emergence of the Internet has helped enterprise applications to be easily accessible over the Web without having specific client-side software installations. In the Internet-based enterprise application model, the focus was to move the complex business processing toward centralized servers in the back end.

The first generation of Internet servers was based upon Web servers that hosted static Web pages and provided content to the clients via HTTP (HyperText Transfer Protocol). HTTP is a stateless protocol that connects Web browsers to Web servers, enabling the transportation of HTML content to the user.

With the high popularity and potential of this infrastructure, the push for a more dynamic technology was inevitable. This was the beginning of server-side scripting using technologies like CGI, NSAPI, and ISAPI.

With many organizations moving their businesses to the Internet, a whole new category of business models like business-to-business (B2B) and business-to-consumer (B2C) came into existence.
This evolution lead to the specification of J2EE architecture, which promoted a much more efficient platform for hosting Web-based applications. J2EE provides a programming model based upon Web and business components that are managed by the J2EE application server. The application server consists of many APIs and low-level services available to the components. These low-level services provide security, transactions, connections and instance pooling, and concurrency services, which enable a J2EE developer to focus primarily on business logic rather than plumbing.

The power of Java and its rich collection of APIs provided the perfect solution for developing highly transactional, highly available and scalable enterprise applications. Based on many standardized industry specifications, it provides the interfaces to connect with various back-end legacy and information systems. J2EE also provides excellent client connectivity capabilities, ranging from PDA to Web browsers to Rich Clients (Applets, CORBA applications, and Standard Java Applications).

11Q. Draw and Explain the typical Architecture of J2EE

Figure 1.7 shows various components of the J2EE architecture.
A typical J2EE architecture is physically divided into three logical tiers, which enables clear separation of the various application components with defined roles and responsibilities. The following is a breakdown of functionalities of those logical tiers:

**Presentation tier.** The Presentation tier is composed of Web components, which handle HTTP requests/responses, Session management, Device independent content delivery, and the invocation of business tier components.

**Application tier.** The Application tier (also known as the Business tier) deals with the core business logic processing, which may typically deal with workflow and automation. The business components retrieve data from the information systems with well-defined APIs provided by the application server.

**Integration tier.** The Integration tier deals with connecting and communicating to back-end Enterprise Information Systems (EIS), database applications and legacy applications, or mainframe applications.

With its key functionalities and provisions for partitioning applications into logical tiers, J2EE has been highly adopted as the standard solution for developing and deploying mission critical Web-based applications. The power of J2EE-based applications would be tremendous, if it is enabled to interoperate with other potential J2EE-deployed applications.

It also enables the realization of syndication and collaboration of business processes across the Internet by enabling them to share data and component-level processes in real time. This phenomenon is commonly referred to as business-to-business (B2B) communication.

The emergence of the Extensible Markup Language (XML) for defining portable data in a structured and self-describing format is embraced by the industry as a communication medium for electronic data exchange. Using XML as a data exchange mechanism between applications promotes interoperability between applications and also enhances the scalability of the underlying applications. Combining the potential of a J2EE platform and XML offers a standard framework for B2B and inter-application communication across networks.

With J2EE enabling enterprise applications to the Internet and XML acting as a “glue” bridges these discrete J2EE-based applications by facilitating them to interoperate with each other. XML, with its incredible flexibility, also has been widely adopted and accepted as a standard by major vendors in the IT industry, including Sun, IBM, Microsoft, Oracle, and HP. The combination of these technologies offers more promising possibilities in the technology sector for providing a new way of application-to-application communication on the Internet. It also promotes a new form of the distributed computing technology solution referred to as Web services.
12Q. Briefly write about the Emergence of Web services and SOA

**The Emergence of Web Services:**

Today, the adoption of the Internet and enabling Internet-based applications has created a world of discrete business applications, which co-exist in the same technology space but without interacting with each other. The increasing demands of the industry for enabling B2B, application-to-application (A2A), and inter-process application communication has led to a growing requirement for service-oriented architectures.

Enabling service-oriented applications facilitates the exposure of business applications as service components enable business applications from other organizations to link with these services for application interaction and data sharing without human intervention. By leveraging this architecture, it also enables interoperability between business applications and processes.

By adopting Web technologies, the service-oriented architecture model facilitates the delivery of services over the Internet by leveraging standard technologies such as XML. It uses platform-neutral standards by exposing the underlying application components and making them available to any application, any platform, or any device, and at any location. Today, this phenomenon is well adopted for implementation and is commonly referred to as **Web services.** This promising new technology sets the strategic vision of the next generation of virtual business models and the unlimited potential for organizations doing business collaboration and business process management over the Internet.