2

COMMUNICATION IN DISTRIBUTED SYSTEM

2.1 COMMUNICATION IN DISTRIBUTED SYSTEMS

The most important difference between a distributed system and a uniprocessor system is the **interprocess communication.**

Interprocess communication (IPC) is a set of programming* interfaces *that allows a programmer to coordinate activities among different program* processes *that can run concurrently in an operating system.

- ✓ In a uniprocessor system, interprocess communication assumes the existence of shared memory whereas in a distributed system, there‟s no shared memory, so the entire nature of interprocess communication must be completely reframed from scratch.
- ✓ All communication in distributed system is based on **message passing**.
- ✓ The processes run on different machines and they exchange information through message passing.
- ✓ Successful distributed systems depend on communication models that hide or simplify message passing.

2.2 SYSTEM MODELS

System models describe common properties and design choices for distributed system in a single descriptive model. The system models of distributed systems are classified into the following types:

- **Physical models:** It is the most explicit way in which to describe a system in terms of hardware composition.
- **Architectural models**: They describe a system in terms of the computational and communication tasks performed by its computational elements.
- **Fundamental models**: They examine individual aspects of a distributed system. They are again classified based on some parameters as follows:
  - ✓ **Interaction models:** This deals with the structure and sequencing of the communication between the elements of the system
  - ✓ **Failure models:** This deals with the ways in which a system may fail to operate correctly
  - ✓ **Security models:** This deals with the security measures implemented in the system against attempts to interfere with its correct operation or to steal its data.

2.2.1 Physical Models

A physical model is a representation of the underlying hardware elements of a distributed system hides the details of the computer and networking technologies employed.

There are many physical models available from the primitive baseline model to the complex models that could handle cloud environments.

- **Baseline physical model:**
  - ✓ This is a primitive model that describes the hardware or software components located at networked computers.
  - ✓ The communication and coordination of their activities is done by passing messages.
  - ✓ This is a minimal physical model of a distributed system.
  - ✓ This model could be extended to a set of computer nodes interconnected by a computer network for the required passing of messages.
  - ✓ This model helps us to categorize the three generations of distributed systems.
- **Early distributed systems**:
  - ✓ The period is in the late 1970s and early 1980s.
  - ✓ This was developed after the emergence LAN.

- ✓ These systems could support 10 to 100 nodes interconnected by a LAN.
- ✓ It has very limited internet connectivity and can support a small range of services such as shared local printers and file servers.
- ✓ Individual systems were homogeneous, with poor quality of service.

➢ **Internet-scale distributed systems:**

- ✓ The period is from the 1990s after the growth of internet.
- ✓ The physical set up of distributed system is described as an extensible set of nodes interconnected by a network of networks (the Internet).
- ✓ They incorporate large numbers of nodes and provide distributed system services.
- ✓ The systems were heterogeneous which could adopt open standards.
- ✓ They had good QOS.

➢ **Contemporary distributed systems:**

- ✓ The nodes were desktop computers and therefore relatively static, discrete and independent.
- ✓ Mobile computing has led to physical models with movable nodes. Service discovery is of primary concern in these systems.
- ✓ The cloud computing and cluster architectures hassled to pools of nodes that collectively provide a given service. This resulted in enormous number of systems given the service.

➢ **Distributed systems of systems:**

The ultra large scale (ULS) distributed systems is defined as a complex system consisting of a series of subsystems that are systems in their own right and that come together to perform a particular task or tasks. Some important characteristics of these systems include their

- very large size
- global geographical distribution
- operational and managerial independence of their member systems.

The main function of these systems arises from the interoperability between their components.

Example: Flood monitoring systems.

Parameter	**Early Systems**	**Internet Scale**	**ULS System**
Scale	Small	Large	Ultra Large
Heterogeneity	Homogeneous	Platform independent software and technologies.	Supports many types of architectures
Openness	Systems are mostly closed	Many open standards are available	The already existing standards are not able to cope with complex systems.
QOS	Poor	Significant	QOS could still be improved

2.2.2 Architectural Models

The architecture abstracts the functions of the individual components of the distributed system.

This describes the components and their interrelationships to ensure the system is reliable, manageable, adaptable and cost-effective. The different models are evaluated based on the following criteria:

- architectural elements
- architectural patterns
- middleware platforms

2.2.2.1 Architectural elements

The following must be decided to build a distributed system:

- Communicating entities (objects, components and web services);
- Communication paradigms (inter process communication, remote invocation and indirect communication);
- Roles responsibilities and placement

Communicating entities

The components that are communicating and how those entities communicate together are dealt here. The following are some of the problem oriented entities:

- **Objects:** They are used to implement object oriented approaches in distributed systems. Objects are accessed through interfaces.
- **Components:** Components resemble objects and they offer problem-oriented abstractions for building distributed systems. They are also accessed through interfaces. The key difference between component and object is that a components holds all the assumptions specified to other components and their interfaces in the system. Components and objects are used to develop tightly coupled applications.
- **Web services:** Web services are closely related to objects and components. Web services are integrated into the World Wide Web. They are partially defined by the web-based technologies they adopt. Web services are generally viewed as complete services that can be combined to achieve value-added services across organizational boundaries.

Object	**Components**	**Web services**
Object oriented solution	Problem oriented solution	Problem oriented solution
The dependencies and assumptions of the interfaces holds only on the objects that act upon them.	The dependencies and assumptions of the all interfaces holds on all the components in the system.	They are integrated into the WWW.
Tightly coupled services	Tightly coupled services	Complete service that could also be made as a value added service

Communication paradigms

There are three major modes of communication in distributed systems:

- **Interprocess communication**:

  This is a low-level support for communication between processes in distributed systems, including message-passing primitives. They have direct access to the API offered by Internet protocols and support multicast communication.

- **Remote invocation:**

  It is used in a distributed system and it is the calling of a remote operation, procedure or method.

a) **Request-reply protocols:**

This is a message exchange pattern in which a requestor sends a request message to a replier system which receives and processes the request, ultimately returning a message in response. This is a simple, but powerful messaging pattern which allows two applications to have a two-way conversation with one another over a channel. This pattern is especially common in client–server architectures.

b) **Remote procedure calls:** Remote Procedure Call (**RPC**) is a protocol that one program can use to request a service from a program located in another computer in a network without having to understand network details.

c) **Remote method invocation**: RMI (Remote Method Invocation) is a way that a programmer can write object-oriented programming in which objects on different computers can interact in a distributed network. This has the following two key features:

- ✓ Space uncoupling: Senders do not need to know who they are sending to
- ✓ Time uncoupling: Senders and receivers do not need to exist at the same time

➢ **Indirect communication**

Key techniques for indirect communication include:

a) **Group communication:** Group communication is concerned with the delivery of messages to a set of recipients and hence is a multiparty communication paradigm supporting one-to-many communication. A group identifier uniquely identifies the group. The recipients join the group and receive the messages. Senders send messages to the group based on the group identifier and hence do not need to know the recipients of the message.

b) **Publish-subscribe systems: This** is a messaging pattern where senders of messages, called publishers, do not program the messages to be sent directly to specific receivers, called subscribers. Instead, published messages are characterized into classes, without knowledge of what, subscribers there may be. Similarly, subscribers express interest in one or more classes, and only receive messages that are of interest, without knowledge of what, if any, publishers there are. They offer one-to-many style of communication.

c) **Message queues:** They offer a point-to-point service whereby producer processes can send messages to a specified queue and consumer processes can receive messages from the queue or be notified of the arrival of new messages in the queue. Queues therefore offer an indirection between the producer and consumer processes.

d) **Tuple spaces:** The processes can place arbitrary items of structured data, called tuples, in a persistent tuple space and other processes can either read or remove such tuples from the tuple space by specifying patterns of interest. This style of programming is known as **generative communication**.

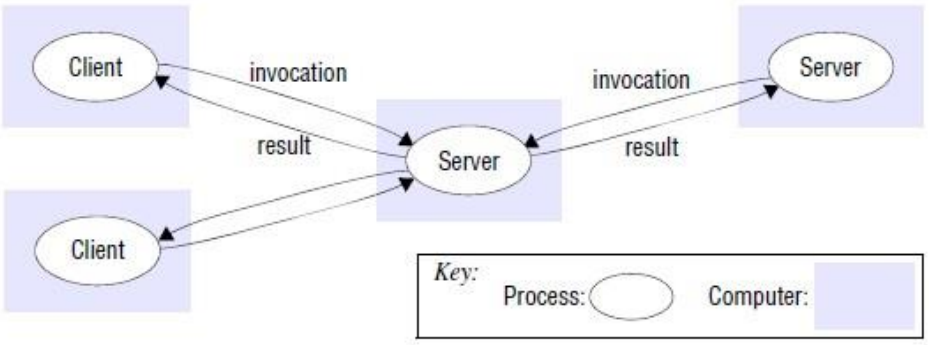
e) **Distributed shared memory:** In computer architecture, distributed shared memory (DSM) is a form of memory architecture where the (physically separate) memories can be addressed as one (logically shared) address space. Here, the term shared does not mean that there is a single centralized memory but shared essentially means that the address space is shared (same physical address on two processors refers to the same location in memory

Roles and responsibilities

There are two types of architectures based on roles and responsibilities:

1) **Client-server:**

- ✓ The system is structured as a set of processes, called servers, that offer services to the users, called clients.
- ✓ The client-server model is usually based on a simple request/reply protocol, implemented with send/receive primitives or using remote procedure calls (RPC) or remote method invocation (RMI).



for some
requested)

this new

ng in poor
ne server.

Figure 2.1: Client server style

2) **Peer-peer**

- ✓ All processes (objects) play similar role.
- ✓ Processes (objects) interact without particular distinction between clients and servers.
- ✓ The pattern of communication depends on the particular application.
- ✓ A large number of data objects are shared; any individual computer holds only a small part of the application database.

- ✓ Processing ts are distributed across mar
- ✓ This is the

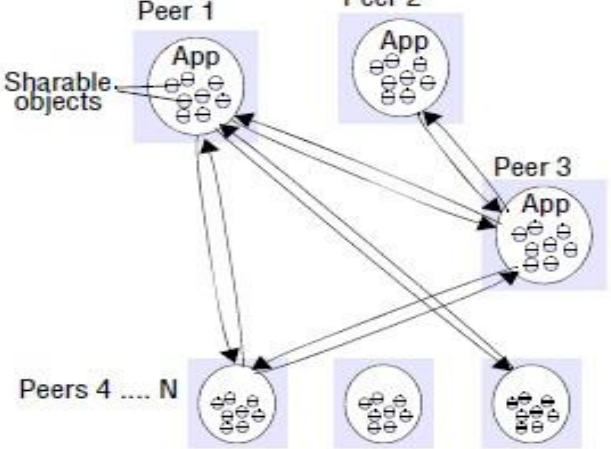


Figure 2.2: Peer-peer style

- ✓ Peer-to-Peer model distributes shared resources widely share computing and communication loads.
- ✓ Problems with peer-to-peer are high complexity due to cleverly place individual objects and there are large number of replicas.

Placement

Mapping of objects or services to the underlying physical distributed infrastructure is considered here. The following points must be taken care while placing the objects: reliability, communication pattern, load, QOS etc. The following are the placement strategies:

- ➢ **Mapping of services to multiple servers**
  - ✓ The servers may partition the set of objects on which the service is based and distribute those objects between themselves, or they may maintain replicated copies of them on several hosts.
  - ✓ The Web provides a common example of partitioned data in which each web server manages its own set of resources. A user can employ a browser to access a resource at any one of the servers.
- ➢ **Caching**
  - ✓ A cache is a local store of recently used data objects that is closer to one client or a particular set of clients than the objects themselves.
  - ✓ When a new object is received from a server it is added to the local cache store, replacing some existing objects if necessary.

- ✓ When an object is needed by a client process, the caching service first checks the cache and supplies the object from there if an up-to-date copy is available. If not, an up-to-date copy is fetched.
- ✓ Caches may be co-located with each client or they may be located in a proxy server that can be shared by several clients.

➢ **Mobile code:**

- ✓ Applets are a well-known and widely used example of mobile code.
- ✓ The user running a browser selects a link to an applet whose code is stored on a web server; the code is downloaded to the browser and runs.
- ✓ They provide interactive response.

➢ **Mobile agents:**

- ✓ A mobile agent is a running program (including both code and data) that travels from one computer to another in a network carrying out a task on someone‟s behalf, such as collecting information, and eventually returning with the results.
- ✓ A mobile agent is a complete program, code + data, that can work (relatively) independently.
- ✓ The mobile agent can invoke local resources/data.
- ✓ A mobile agent may make many invocations to local resources at each site it visit.
- ✓ Typical tasks of mobile agent includes: collect information , install/maintain software on computers, compare prices from various vendors bay visiting their sites etc.
- ✓ Mobile agents (like mobile code) are a potential security threat to the resources in computers that they visit.

2.2.2.2 Architectural patterns

Architectural patterns are reusable solution to a commonly occurring problem in software architecture within a given context. They are not complete solutions but rather offer partial insights. The following are some of the common architectural patterns:

➢ **Layering:**

- ✓ In a layered approach, a complex system is partitioned into a number of layers, with a given layer making use of the services offered by the layer below.

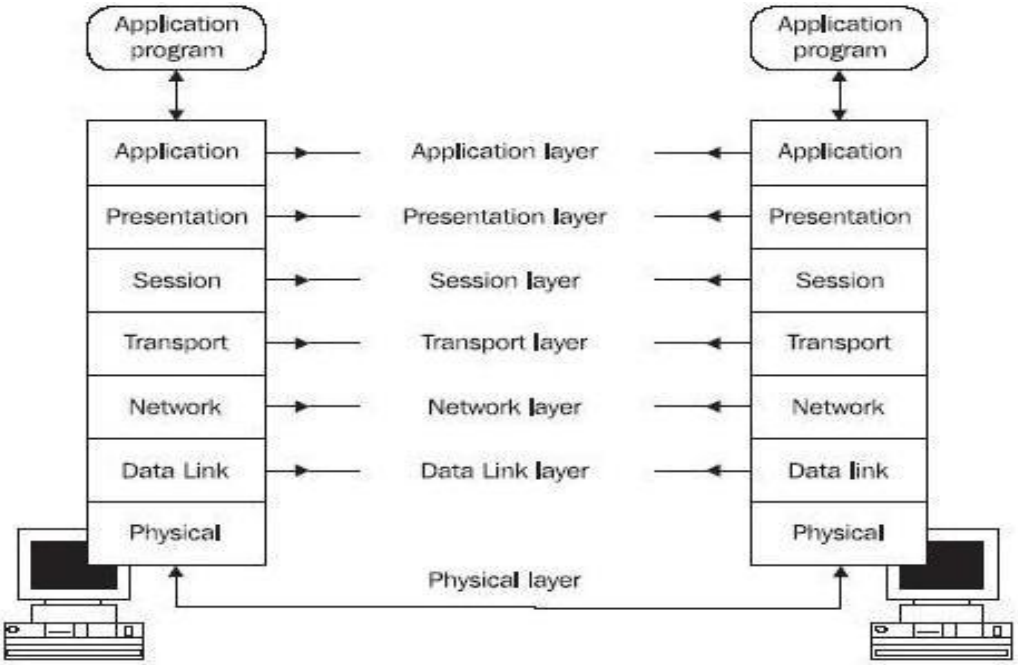


Figure 2.3: Layered Architecture

- ✓ A given layer is unaware of the implementation details, of other layers beneath them.
- ✓ In terms of distributed systems, this equates to a vertical organization of services into service layers.
- ✓ A distributed service can be provided by one or more server processes, interacting with each other and with client processes in order to maintain a consistent system-wide view of the service‟s resources.

➢ **Tiered architecture**

- ✓ Tiering is a technique to organize functionality of a given layer and place this functionality into appropriate servers and, as a secondary consideration, on to physical nodes.
- ✓ The two tiered architecture refers to client/server architectures in which the user interface (**presentation layer)** runs on the client and the database (**data layer**) is stored on the server. The actual application logic can run on either the client or the server.

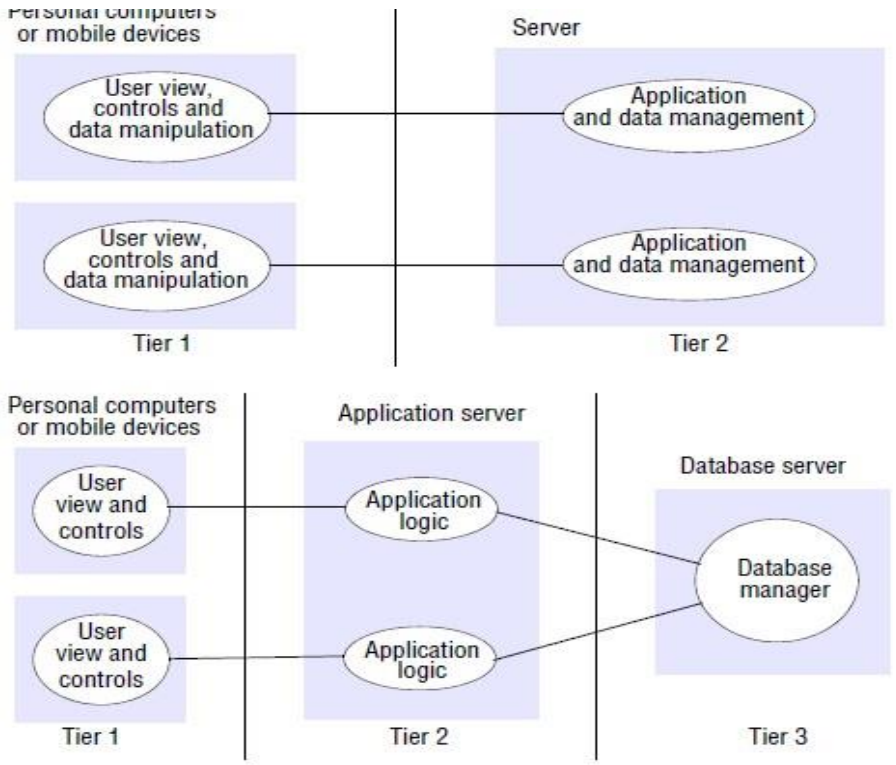


Fig 2.4: Two and three tiered architectures

- ✓ The three tiered architecture has:

  a) **Presentation tier:** This is the topmost level of the application. is concerned with handling user interaction and updating the view of the application as presented to the user;

  b) **Application tier (business logic, logic tier, or middle tier):** The logical tier is pulled out from the presentation tier and, as its own layer, it controls an application‟s functionality by performing detailed processing.

  c) **Data tier:** The data tier includes the data persistence mechanisms (database servers, file shares, etc.) and the data access layer that encapsulates the persistence mechanisms and exposes the data.

- ➢ **Thin clients**

  - ✓ In a thin-client model, all of the application processing and data management is carried out on the server.
  - ✓ The client is simply responsible for running the responsible for running the presentation software.
  - ✓ This is used when legacy systems are migrated to client server architectures.

- ✓ The legacy system acts as a server in its own right with a graphical interface implemented on a client.
- ✓ The major disadvantages of thin clients is that it places a heavy processing load on both the server and the network and the delay due to operating system latencies.
- ✓ The **virtual network computing**(VNC) has emerged to overcome the disadvantages og thin clients.
- ✓ VNC is a type of remote-control software that makes it possible to control another computer over a network connection.
- ✓ Keystrokes and mouseclicks are transmitted from one computer to another, allowing technical support staff to manage a desktop, server, or other networked device without being in the same physical location.
- ✓ Since all the application data and code is stored by a file server, the users may migrate from one network computer to another. So VNC is of big use in distributed systems.

➢ **Other patterns**

a) Proxy:

- This facilitates location transparency in remote procedure calls or remote method invocation.
- A proxy is created in the local address space to represent the remote object with same interface as the remote object.
- The programmer makes calls on this proxy object and he need not be aware of the distributed nature of the interaction.

b) Brokerage:

- It is used to bring interoperability in potentially complex distributed infrastructures.
- The service broker is meant to be a registry of services, and stores information about what services are available and who may use them.

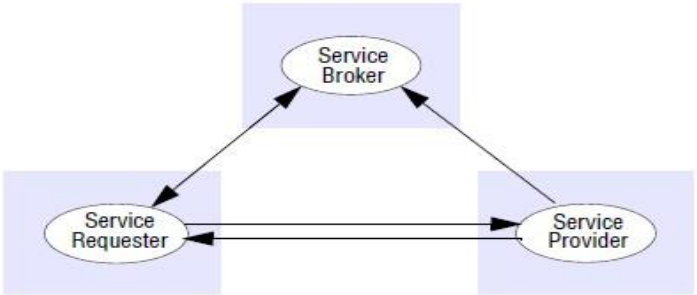


Fig 2.5: Web service with brokers

c) Reflection:

- The Reflection architectural pattern provides a mechanism for changing structure and behavior of software systems dynamically.
- It supports the modification of fundamental aspects, such as type structures and function call mechanisms.
- In this pattern, an application is split into two parts:

  1) **Meta Level:** This provides information about selected system properties and makes the software self-aware.
  2) **Base level:** This includes the application logic. Its implementation builds on the meta level. Changes to information kept in the meta level affect subsequent base-level behavior.

2 2.2.3 Associated middleware solutions

Middleware is a general term for software that serves to "glue together" separate, often complex and already existing, programs.

Middleware provides a higher-level programming abstraction for the development of distributed systems. Middleware makes it easier for software developers to perform communication and input/output, so they can focus on the specific purpose of their application. Middleware is the software that connects software components or enterprise applications and it lies between the operating system and the applications on each side of a distributed computer network. Middleware includes Web servers, application servers, content management systems, and similar tools that support application development and delivery.

Categories of middleware

The following are some of the categories of middleware:

- **Distributed Objects**

  The term distributed objects usually refers to software modules that are designed to work together, but reside either in multiple computers connected via a network or in different processes inside the same computer. One object sends a message to another object in a remote machine or process to perform some task. The results are sent back to the calling object.

- **Distributed Components**

  A component is a reusable program building block that can be combined with other components in the same or other computers in a distributed network to form an application. **Examples**: a single button in a graphical user interface, a small interest calculator, an interface to a database manager. Components can be

deployed on different servers in a network and communicate with each other for needed services. A component runs within a context called a container . **Examples:** pages on a Web site, Web browsers, and word processors.

- **Publish subscriber model**

  Publish–subscribe is a messaging pattern where senders of messages, called publishers, do not program the messages to be sent directly to specific receivers, called subscribers. Instead, published messages are characterized into classes, without knowledge of what, if any, subscribers there may be. Similarly, subscribers express interest in one or more classes, and only receive messages that are of interest, without knowledge of what publishers are there.

- **Message Queues**

  Message queues provide an asynchronous communications protocol, meaning that the sender and receiver of the message do not need to interact with the message queue at the same time. Messages placed onto the queue are stored until the recipient retrieves them. Message queues have implicit or explicit limits on the size of data that may be transmitted in a single message and the number of messages that may remain outstanding on the queue.

- **Web services**

  A web service is a collection of open protocols and standards used for exchanging data between applications or systems. Software applications written in various programming languages and running on various platforms can use web services to exchange data over computer networks like the Internet in a manner similar to inter-process communication on a single computer. This interoperability (e.g., between Java and Python, or Windows and Linux applications) is due to the use of open standards.

- **Peer to peer**

  Peer-to-peer (P2P) computing or networking is a distributed application architecture that partitions tasks or work load between peers. Peers are equally privileged, equipotent participants in the application. They are said to form a peer-to-peer network of nodes.

Advantages of middleware

- ✓ Real time information access among systems.
- ✓ Streamlines business processes and helps raise organizational efficiency.
- ✓ Maintains information integrity across multiple systems.
- ✓ It covers a wide range of software systems, including distributed Objects and components, message-oriented communication, and mobile application support.
- ✓ Middleware is anything that helps developers create networked applications.

Disadvantages of middleware

- ✓ Prohibitively high development costs.
- ✓ There are only few people with experience in the market place to develop and use a middleware.
- ✓ There are only few satisfying standards.
- ✓ The tools are not good enough.
- ✓ Too many platforms to be covered.
- ✓ Middleware often threatens the real-time performance of a system.
- ✓ Middleware products are not very mature.

2.2.3 Fundamental Models

Fundamental Models are concerned with a formal description of the properties that are common in all of the architectural models.

Models addressing time synchronization, message delays, failures, security issues are addressed as:

- **Interaction Model** – deals with performance and the difficulty of setting of time limits in a distributed system.
- **Failure Model** – specification of the faults that can be exhibited by processes
- **Secure Model** – discusses possible threats to processes and communication channels.

The purpose of the fundamental model is to:

- make explicit all the relevant assumptions about the systems we are modelling.
- make generalizations (general purpose algorithms) concerning what is possible or impossible with given assumptions.

2.2.3.1 Interaction model

- ✓ In this model the computations occurs within processes.
- ✓ The processes interact by passing messages to achieve communication (information flow) and coordination (synchronization and ordering of activities) between processes.

- ✓ The two significant factors affecting interacting processes in a distributed system are:

  1) Communication performance

  2) Global notion of time.

Communication Performance

- The communication channel in can be implemented in a variety of ways in distributed systems: Streams or through simple message passing over a network.

- The performance characteristics of a network are:

  a) **Latency:** A delay between the start of a message‟s transmission from one process to the beginning of reception by another.

  b) **Bandwidth:** The total amount of information that can be transmitted over in a given time. The communication channels using the same network, have to share the available bandwidth.

  c) Jitter: The variation in the time taken to deliver a series of messages. It is very relevant to multimedia data.

Global Notion of time

- Each computer in a distributed system has its own internal clock, which can be used by local processes to obtain the value of the current time.

- Any two processes running on different computers can associate timestamp with their events.

- However, even if two processes read their clocks at the same time, their local clocks may supply different time because the computer clock drifts from perfect time and their drift rates differ from one another.

- Even if the clocks on all the computers in a distributed system are set to the same time initially, their clocks would eventually vary quite significantly unless corrections are applied.

- There are several techniques to correct time on computer clocks.

- One technique is to use radio receivers to get readings from GPS (Global Positioning System) with accuracy about 1 microsecond**.**

Variants of Interaction model

- ✓ In a DS it is hard to set time limits on the time taken for process execution, message delivery or clock drift. There are two models based on time stamp:

- **Synchronous DS**
  - This is practically hard to achieve in real life.
  - In synchronous DS the time taken to execute a step of a process has known lower and upper bounds.
  - Each message transmitted over a channel is received within a known bounded time.
  - Each process has a local clock whose drift rate from real time has known bound.
- **Asynchronous DS**
  - There are no bounds on: process execution speeds, message transmission delays and clock drift rates.

Event Modeling

- ✓ Event ordering is of major concern in DS.
- ✓ The concept of one event happening before another in a distributed system is examined, and is shown to define a partial ordering of the events.
- ✓ The execution of a system can be described in terms of events and their ordering despite the lack of accurate clocks.
- ✓ Consider a mailing list with users X, Y, Z, and A.
- ✓ Due to independent delivery in message delivery, message may be delivered in different order.
- ✓ If messages m1, m2, m3 carry their time t1, t2, t3, then they can be displayed to users accordingly to their time ordering.
- ✓ The mail box is:

Item	**From**	**Subject**
23	Z	Re: Meeting
24	X	Meeting
26	Y	Re: Meeting

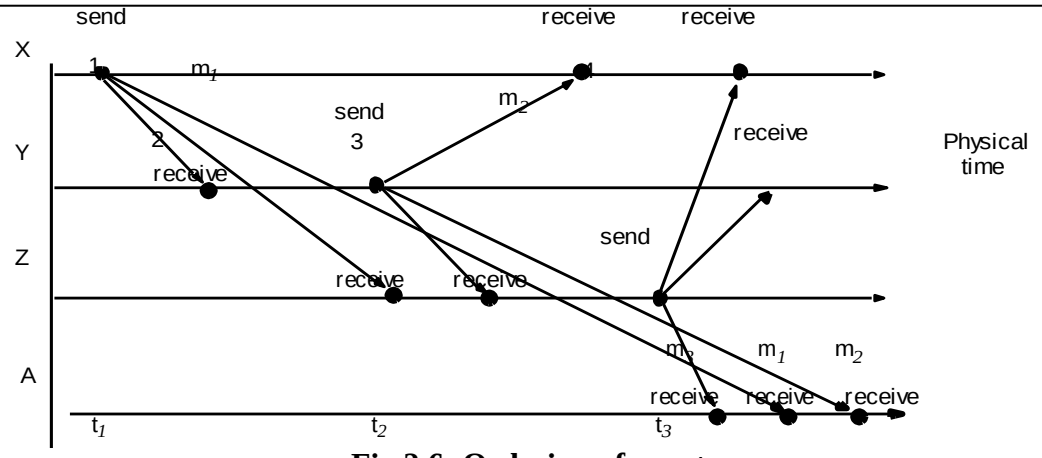


Fig 2.6: Ordering of events

2.2.3.2 Failure Model

In a DS, both processes and communication channels may fail – i.e., they may depart from what is considered to be correct or desirable behavior. The following are the common types of failures:

- Omission Failure
- Arbitrary Failure
- Timing Failure

a) Omission failure

Omission failures occur due to communication link failures. They are detected through timeouts. They are classified as:

- **Process omission failures**
  - ✓ Omission failures that occur due to process crash (i.e.) the execution of the process could not continue.
  - ✓ This is detected using timeouts.
  - ✓ A fixed period of time is fixed for all the methods to complete its execution. If the method takes time longer than the allowed time, a time out has occurred.
  - ✓ In an asynchronous system a timeout can indicate only that a process is not responding.
  - ✓ A process crash is called **fail-stop**if other processes can detect certainly that the process has crashed.

- ✓ Fail-stop behavior can be produced in a synchronous system if the processes use timeouts to detect when other processes fail to respond and messages are guaranteed to be delivered.

➢ **Communication omission failures**

- ✓ This occurs when the messages are dropped between sender and the receiver.
- ✓ The message can be lost in sender buffer, receiver buffer or even in the communication channel.
- ✓ The loss of messages between the sending process and the outgoing message buffer is known as **send omission failures.**
- ✓ The loss of messages between the incoming message buffer and the receiving process as **receive-omission failures.**
- ✓ The loss of messages in a channel is **channel omission failure.**

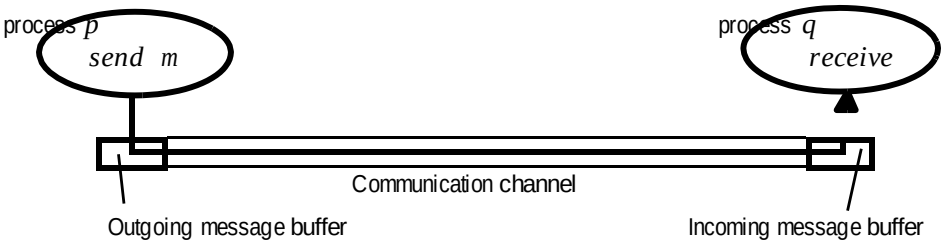


Fig 2.7: Communication between processes in a channel

b) **Arbitrary failures (Byzantine failure):**

- ✓ This includes all possible errors that could cause failure.
- ✓ Communication channels often suffer from arbitrary failures
- ✓ The following table gives an overview of the different failures and its categories:

Class	**Affects**	**Comments**
Fail stop	Process	Process halts and remains halted. Other processes may detect that this process has halted.
Crash	Process	Process halts and remains halted. Other processes may not detect that this process has halted.

Omission	Channel	A message inserted in an outgoing message buffer never arrives at the other end‟s incoming messagebuffer.
Send-omission	Process	A process completes a send operation but the message is not put in its outgoing message buffer.
Receiveomission	Process	A message is put in a process‟s incoming messagebuffer, but that process does not receive it.
Arbitrary	Process or channel	Process/channel exhibits arbitrary behaviour: it may send/transmit arbitrary messages at arbitrary times or commit omissions; a process may stop or take an incorrect step.

c) **Timing failures**

- ✓ Timing failures refer to a situation where the environment in which a system operates does not behave as expected regarding the timing assumptions, that is, the timing constraints are not met.
- ✓ Timing failures are applicable in synchronous distributed system where time limits are set on process execution time, message delivery time and clock drift rate.
- ✓ The following are the common failures withnrespect to timings:

Class	**Affects**	**Comments**
Clock	Process	Process‟s local clock exceeds the bounds on its rate of drift from real time
Performance	Process	Process exceeds the bounds on the interval between two steps
Performance	Channel	A message‟s transmission takes longer than the stated bound.

Masking failures

- ✓ Each component in a distributed system is generally constructed from a collection of other components. It is always possible to construct reliable services from components that exhibit failures.
- ✓ A knowledge of failure characteristics of a component can enable a new service to be designed to mask the failure of the components on which it depends.

Reliability of one-to-one communication

The term **reliable communication** is defined in terms of validity and integrity.

- **Validity:** Any message in the outgoing message buffer is eventually delivered to the incoming message buffer.
- **Integrity:** The message received is identical to one sent, and no messages are delivered twice.

The threats to integrity come from two independent sources:

- Any protocol that retransmits messages but does not reject a message that arrives twice.
- Malicious users that may inject spurious messages, replay old messages or tamper with messages.

2.2.3.2 Security model

The security of a DS can be achieved by securing the processes and the channels used in their interactions and by protecting the objects that they encapsulate against unauthorized access.

Protection is described in terms of objects, although the concepts apply equally well to resources of all types.

Protecting objects

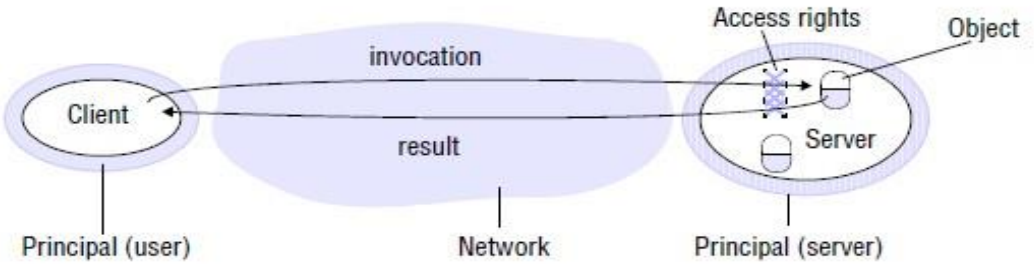


Fig 2.8: Objects and Principals

- ✓ The server manages a collection of objects on behalf of some users.
- ✓ The users can run client programs that send invocations to the server to perform operations on the objects.
- ✓ The server carries out the operation specified in each invocation and sends the result to the client.

- ✓ This uses use "access rights" that define who is allowed to perform operation on a object.
- ✓ The server should verify the identity of the principal (user) behind each operation and checking that they have sufficient access rights to perform the requested operation on the particular object, rejecting those who do not.
- ✓ A principal is an authority that manages the access rights. The principal may be a user or a process.

Securing processes and their interactions

- ✓ The messages send by the processes are exposed to attack because the network and the co
- ✓ To m nding any proce
- ✓ Threa hreats to comm

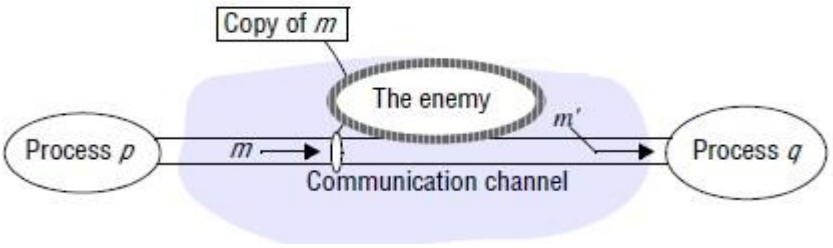


Fig: 2.9: Enemy

- ➢ **Threats to processes:**
  - ♦ A process that is designed to handle incoming requests may receive a message from any other process in the distributed system
  - ♦ It cannot necessarily determine the identity of the sender.
  - ♦ This lack of reliable knowledge of the source of a message is a threat to the correct functioning of both servers and clients.
- ➢ **Servers**:
  - ♦ Servers cannot necessarily determine the identity of the principal behind any particular invocation.
  - ♦ Without reliable knowledge of the sender"s identity, a server cannot tell whether to perform the operation or to reject it.

- **Clients:**
  - When a client receives the result of an invocation from a server, it cannot necessarily tell whether the source of the result message is from the intended server or from an enemy, perhaps „spoofing" the mail server.
  - Thus the client could receive a result that was unrelated to the original invocation, such as a false mail item (one that is not in the user"s mailbox).
- **Threats to communication channels**:
  - An enemy can copy, alter or inject messages as they travel across the network and its intervening gateways.
  - Such attacks present a threat to the privacy and integrity of information as it travels over the network and to the integrity of the system.
- **Defeating security threats**
  - Encryption and authentication are use to build secure channels.
  - Each of the processes knows the identity of the principal on whose behalf the other process is executing and can check their access rights before performing an operation.

Other possible threats from an enemy

- **Denial of service:**

  Denial of service (DoS) attack is an incident in which a user or organization is deprived of the services of a resource they would normally expect to have. In a distributed denial-of-service, large numbers of compromised systems (sometimes called a botnet) attack a single target.

- **Mobile code:**

  Mobile code is software transferred between systems, e.g. transferred across a network, and executed on a local system without explicit installation by the recipient. Mobile code raises new and interesting security problems for anyprocess that receives and executes program code from elsewhere.

The uses of security models

The use of security techniques incurs processing and management costs. The distributed systems faces threats from various points. The threat analysis demands the construction of security models.

2.3 INTERPROCESS COMMUNICATION

Interprocess communication (IPC) is a set of programming interfaces that allow a programmer to coordinate activities among different program processes that can run concurrently in an operating system.

- IPC refers to exchange of data between two or more separate, independent processes/threads.
- The operating systems provide facilities/resources for inter-process communications (IPC), such as message queues, semaphores, and shared memory.
- Distributed computing systems make use of these facilities/resources to provide application programming interface (API) which allows IPC to be programmed at a higher level of abstraction. (e.g., send and receive)
- Distributed computing requires information to be exchanged among independent processes.

2.3.1 The API's for the Internet Protocols

Characteristics of IPC:

- **Synchronous and asynchronous communication**
  - ✓ In the synchronous form of communication, the sending and receiving processes synchronize at every message.
  - ✓ In this case, both send and receive are blocking operations.
  - ✓ In the asynchronous form of communication, the use of the send operation is non blocking in that the sending process is allowed to proceed as soon as the message has been copied to a local buffer, and the transmission of the message proceeds in parallel with the sending process.
  - ✓ The receive operation can have blocking and non-blocking variants.
- **Message destinations**
  - The messages in the Internet protocols, messages are sent to (Internet address, local port) pairs.
  - A local port is a message destination with in a computer, specified as an integer.
  - A port has exactly one receiver but can have many senders.
  - Processes may use multiple ports to receive messages.
  - Any process that knows the number of a port can send a message to it.
  - Client programs refer to services by name and use a name server or binder to translate their names into server locations at runtime.
  - This allows services to be relocated but not to migrate – that is, to be moved while the system is running.

- **Reliability**
  - A point-to-point message service can be described as reliable if messages are guaranteed to be delivered.
  - A point-to-point message service can be described as unreliable if messages are not guaranteed to be delivered in the face of even a single packet dropped or lost.
- **Ordering**
  - Some applications require that messages be delivered in the order in which they were transmitted by the sender.
  - The delivery of messages outof sender order is regarded as a failure by such applications.

Sockets

A socket is one endpoint of a two-way communication link between two programs running on the network.

Interprocess communication consists of transmitting a message between a socket in one process and a socket in another process. The java API for interprocess communication in the internet provides both datagram and stream communication. Both forms of communication, UDP and TCP, use the socket abstraction, which provides and endpoint for communication between processes. Interprocess communication consists of transmitting a message between a socket in one process and a socket in another process. The two communication patterns that are most commonly used in distributed programs:

- Client-Server communication
  - The request and reply messages provide the basis for remote method invocation (RMI) or remote procedure call (RPC).
- Group communication
  - The same message is sent to several processes.

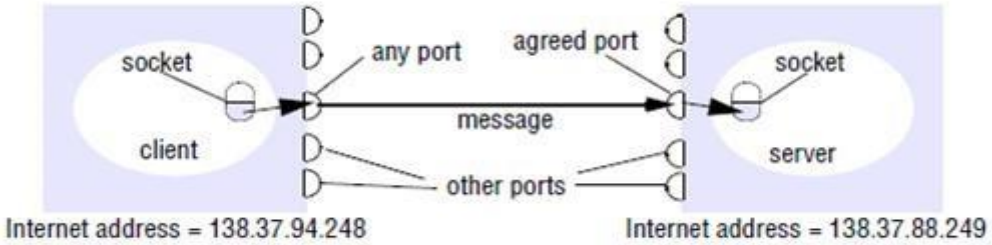


Fig 2.6: Sockets and ports

For a process to receive messages, its socket must be bound to a local port and one of the Internet addresses of the computer on which it runs. Messages sent to a particular Internet address and port number can be received only by a process whose socket is associated with that Internet address and port number.

Java API for Internet addresses

- ✓ Java provides a class, InetAddress, that represents Internetaddresses.
- ✓ Users of this class refer to computers by Domain Name System (DNS) hostnames. InetAddressaComputer = InetAddress.getByName("abc.ac.in");

UDP Datagram Communication

- The application program interface to UDP provides a message passing abstraction.
- Message passing is the simplest form of interprocess communication.
- API enables a sending process to transmit a single message to a receiving process.
- The independent packets containing theses messages are called **datagrams.**
- In the Java and UNIX APIs, the sender specifies the destination using a socket.
- A datagram is transmitted between processes when one process sends it and another receives it.
- The following are the steps in socket communication:
    - Creating a socket
    - Binding a socket to a port and local Internet address
        - A client binds to any free local port
        - A server binds to a server port
- Receive() returns Internet address and port of sender, along with the message.

Issues in datagram communication:

- **Message size:**
    - ✓ The receiving process needs to specify an fixed size array to receive a message.
    - ✓ If the message is too big for the array, it is truncated.
    - ✓ Any application requiring messages larger than the maximum must fragment them.

- **Blocking:**
  - ✓ Sockets normally provide non-blocking sends and blocking receives for datagram communication.
  - ✓ The send() returns after delivering the message to theUDP and IP protocols. These protocols are now responsible for transmitting the message to its destination
  - ✓ The message is placed in a queue for the socket that is bound to the destination port.
  - ✓ Messages are discarded atthe destination if no process has a socket bound to the destination port.
  - ✓ The receive()blocks until a datagram is received, unless a timeout has been set on the socket.
- **Timeouts:**
  - ✓ The receive() cannot wait indefinitely. This situation occurs when the sending process may have crashed or the expected message may have been lost.
  - ✓ To avoid this timeouts can be set on sockets.
  - ✓ The timeouts must be much larger than the time required to transmit a message.
- **Receive from any:**
  - ✓ The receive() does not specify an origin for messages. This can get datagrams addressed to its socket from any origin.
  - ✓ The receive() returns the Internet address and local port of the sender, allowing the recipient to check where the message came from.
  - ✓ It is possible to connect a datagram socket to a particular remote port and Internet address, in which case the socket is only able to send messages to and receive messages from that address.

Failure model for UDP datagrams

UDP datagrams suffer from following failures:

- Omission failure: Messages may be dropped occasionally
- Ordering: Messages can be delivered out of order.

Java API for UDP datagrams

The Java API provides datagram communication by two classes:

1. DatagramPacket
   - It provides a constructor to make an array of bytes comprising:
     - Message content
     - Length of message
     - Internet address
     - Local port number
   - It provides another similar constructor for receiving a message.
2. DatagramSocket
   - This class supports sockets for sending and receiving UDP datagram.
   - It provides a constructor with port number as argument.
   - No-argument constructor is used to choose a free local port.
   - DatagramSocket methods are:
     1. **send and receive:**These methods are used for transmitting datagrams between a pair of sockets.

        Syntax:

        send(packet);

        receive();

     2. **setSoTimeout:** This method allows a timeout to be set. With a timeout set, the receive method will block for the time specified and then throw an InterruptedIOException.

        Syntax:

        setSotimeout(time in seconds);

     3. **connect:** This method is used for connecting to a particular remote port and Internet address, in which case the socket is only able to send messages to and receive messages from that address.

        Syntax:

        int connect(in sockfd, const structsockaddr *addr, socklen_taddrlen);

Arrays of bytes containing message	**Message length**	**Internet address**	**Port number**

Fig 2.7: Datagram Packet

UDP Client server Communication

Client Program

```
import java.net.*;
import java.io.*;
public class UDPClient
{
public static void main(String args[])
{
DatagramSocketaSocket = null;
        try {
aSocket = new DatagramSocket(); // Create an object for the datagram class
                byte [] m = args[0].getBytes(); //Buffer to get data
InetAddressaHost = InetAddress.getByName(args[1]);  // Get Inetaddress
intserverPort = 6789;
DatagramPacket request =
                    new DatagramPacket(m, m.length(), aHost, serverPort);
aSocket.send(request); //Sent the packet
   byte[] buffer = new byte[1000];
DatagramPacket reply = new DatagramPacket(buffer, buffer.length);
aSocket.receive(reply);
System.out.println("Reply: " + new String(reply.getData()));
              }
       catch (SocketException e){System.out.println("Socket: " + e.getMessage());
    }
 catch (IOException e){System.out.println("IO: " + e.getMessage());
}
```

```
 finally
  {
       if(aSocket != null) aSocket.close();}
   }
}
```

Server program

```
import java.net.*;
import java.io.*;
public class UDPServer
{
public static void main(String args[])
{
DatagramSocketaSocket = null;
    try
   {
aSocket = new DatagramSocket(6789);
           byte[] buffer = new byte[1000];
           while(true)
            {
DatagramPacket request = new DatagramPacket(buffer, buffer.length);
aSocket.receive(request);
DatagramPacket reply = new DatagramPacket(request.getData(),
request.getLength(), request.getAddress(), request.getPort());
aSocket.send(reply);
            }
    }
catch (SocketException e)
{
System.out.println("Socket: " + e.getMessage());
```

```
 }
 catch (IOException e)
{
System.out.println("IO: " + e.getMessage());
}
finally
{
if (aSocket != null) aSocket.close();}
}
}
```

TCP stream Communication

The API to the TCP protocol provides the abstraction of a stream of bytes to be written to or read from. The following features are hidden stream abstraction:

- Message sizes
- Lost messages
- Flow control
- Message destinations

Working of stream communication

- ✓ The pair of sockets in the client and server are connected by a pair of streams, one in each direction. Thus each socket has an input stream and an output stream.
- ✓ One of the pair of processes can send information to the other by writing to its output stream, and the other process obtains the information by reading from its input stream.
- ✓ When an application closes a socket, this indicates that it will not write any more data to its output stream. Any data in the output buffer is flushed to the other end of the stream and put in the queue at the destination socket, with an indication that the socket is closed.
- ✓ The process at the destination can read the data in the queue, but any further reads after the queue is empty will result in an indication of end of stream.
- ✓ When a process exits or fails, all of its sockets are eventually closed and any process attempting to communicate with it will discover that its connection has been broken.

This form of stream communication has the following issues:

- **Matching of data items:** Two communicating processes need to agree as to the contents of the data transmitted over a stream.
- **Blocking:** The process that writes data to a stream may be blocked by the TCP flow-control mechanism if the socket at the other end is queuing as much data as the protocol allows.
- **Threads:** When a server accepts a connection, it generally creates a new thread in which to communicate with the new client. In an environment in which threads are not provided, an alternative is to test whether input is available from a stream before attempting to read it;

Java API for TCP streams

The Java interface to TCP streams is provided in the classes:

- **ServerSocket:** It is used by a server to create a socket at server port to listen for connect requests from clients.

  Syntax:

  publicServerSocket(int port, InetAddress address) ;

- **Socket:** It is used by a pair of processes with a connection. The client uses a constructor to create a socket and connect it to the remote host and port of a server.It provides methods for accessing input and output streams associated with a socket.

  Syntax:

  public Socket(InetAddress host, int port, InetAddresslocalAddress, intlocalPort);

TCP Client Server Communication

TCP Client

```
import java.net.*;
import java.io.*;
public class TCPClient
{
public static void main (String args[])
{
    Socket s = null;
```

```
    try
    {
intserverPort = 7896;
            s = new Socket(args[1], serverPort);
DataInputStream in = new DataInputStream( s.getInputStream());
DataOutputStream out = new DataOutputStream( s.getOutputStream());
out.writeUTF(args[0]); // UTF is a string encoding;
           String data = in.readUTF();
System.out.println("Received: "+ data) ;
      }
     catch (UnknownHostException e)
   {
System.out.println("Sock:"+e.getMessage());
   }
catch (EOFException e)
{       System.out.println("EOF:"+e.getMessage());
}
catch (IOException e)
{         System.out.println("IO:"+e.getMessage());
}
finally
{       if(s!=null)
try
{
s.close();
}
catch (IOException e)
{
```

```
/*close failed*/
}
}
}
}
```

TCP server

```
import java.net.*;
import java.io.*;
public class TCPServer
{
public static void main (String args[])
{
try{
intserverPort = 7896;
ServerSocketlistenSocket = new ServerSocket(serverPort);
while(true) {
Socket clientSocket = listenSocket.accept();
Connection c = new Connection(clientSocket);
}
} catch(IOException e) {System.out.println("Listen :"+e.getMessage());}
}
}
class Connection extends Thread
{
DataInputStream in;
DataOutputStream out;
Socket clientSocket;
```

```
public Connection (Socket aClientSocket)
{
try {
clientSocket = aClientSocket;
in = new DataInputStream( clientSocket.getInputStream());
out =new DataOutputStream( clientSocket.getOutputStream());
this.start();
} catch(IOException e) {System.out.println("Connection:"+e.getMessage());}
}
public void run(){
try { // an echo server
String data = in.readUTF();
out.writeUTF(data);
} catch(EOFException e) {System.out.println("EOF:"+e.getMessage());
} catch(IOException e) {System.out.println("IO:"+e.getMessage());
} finally { try {clientSocket.close();}catch (IOException e){/*close failed*/}}
}
}
```

2.4 EXTERNAL DATA REPRESENTATION AND MARCHALLING

The information stored in running programs is represented as data structures, whereas the information in messages is in the form of sequences of bytes. Irrespective of the form of communication used, the data structure must be converted to a sequence of bytes before transmission and must be rebuilt on arrival.

External Data Representation is an agreed standard for the representation of data structures and primitive values.

But there are some problems in data representation. They are:

- Two conversions are necessary (i.e.) conversion of bytes to data structures and vice versa in case of using an agreed format at both ends.
- Using sender‟s or receiver‟s format and convert at the other end.

Marshalling is the process of taking a collection of data items and assembling them into a form suitable for transmission in a message. Unmarshalling is the process of disassembling a collection of data on arrival to produce an equivalent collection of data items at the destination.

The following are the three common approaches to external data representation and marshalling:

- **CORBA:** The data representation is concerned with an external representation for the structured and primitive types that can be passed as the arguments and results of remote method invocations in CORBA. It can be used by a variety of programming languages. Marshalling and unmarshalling is done by the middleware in binary form.
- **Java's object serialization:** This is concerned with the flattening and external data representation of any single object or tree of objects that may need to be transmitted in a message or stored on a disk. It is for use only by Java. Marshalling and unmarshalling is done by the middleware in binary form.
- **XML**: This defines a textual fomat for representing structured data. It was originally intended for documents containing textual self-describing structured data. Marshalling and unmarshalling is in texual form.

2.4.1 CORBA'a Common Data Representation (CDR)

CORBA CDR is the external data representation defined with CORBA 2.0.It consists 15 primitive types:

- ✓ Short (16 bit)
- ✓ Long (32 bit)
- ✓ Unsigned short
- ✓ Unsigned long
- ✓ Float(32 bit)
- ✓ Double(64 bit)
- ✓ Char
- ✓ Boolean(TRUE,FALSE)
- ✓ Octet(8 bit)
- ✓ Any(can represent any basic or constructed type)

There are two types of CDR in CORBA:

- **Primitive types:** CDR defines a representation for both big-endian and little-endian orderings. Characters are represented by a code set agreed between client and server.
- **Constructed types**: The primitive values that comprise each constructed type are added to a sequence of bytes in a particular order. The following are the constructed types:
  - ✓ sequence length (unsigned long): followed by elements in order
  - ✓ string length (unsigned long): followed by characters in order (can also have wide characters)
  - ✓ array: array elements in order (no length specified because it is fixed)
  - ✓ struct: in the order of declaration of the components
  - ✓ enumerated unsigned long: the values are specified by the order declared
  - ✓ union: type tag followed by the selected member

Example:

struct with value {„Sona", „Chennai", 1986}

0-3	4	Length of the string
4-7	"Sona"	String
8-11	"@_____"	
12-15	7	Length of the string
16-19	"Chen"	Chennai
20-23	"on____"	
24-27	1986	Unsigned long

Marshalling in CORBA

The type of a data item not given in case of CORBA. CORBA assumes that both the sender and recipient have common knowledge of the order and types of data items. The types of data structures and types of basic data items are described in CORBA IDL. This provides a notation for describing the types of arguments and results of RMI methods

Example:

```
Struct student
 {
string name;
stringrollnumber;
 }
```

2.4.2 Java Object Serilaization

In Java RMI, both object and primitive data values may be passed as arguments and results of method invocation. An object is an instance of a Java class. The Java equivalent for the above mentioned CORBA construct is :

```
Public class student implements Serializable
{        Private String name;
          Private String rollnumber;
         Public student(String aName ,String  arollnumber)
        {
       name = aName;
       rollnumber= arollnumber;
       }
}
```

Student	8 bit version number	h0	Class name, version
2	Java.lang.string.name	Java.lang.string.rollnumber	Number, type and name of instance variables
5Sona	8CS50	h1	Values of variables

To serialize an object: ‹

(1) its class info is written out: name, version number ‹

(2) types and names of instance variables ➥ If an instance variable belong to a new class, then new class info must be written out, recursively. Each class is given a handle ‹

(3) values of instance variables ‹

Example: student s = new student("5Sona", "8CS50");

‹ Thefollowinga re the steps to serialize objects in Java:

- ✓ create an instance of ObjectOutputStream ‹
- ✓ Invoke writeObject method passing Person object as argument ‹

To deserialize

- ✓ create an instance of ObjectInputStream ‹
- ✓ Invoke readObject method to reconstruct the original object

```
ObjectOutputStream out = new ObjectOutputStream(… );
out.writeObject(originalStudent);
ObjectInputStream in = new ObjectInputStream(…);
Student theStudent = in.readObject();
```

Reflection:

Reflection is inquiring about class properties, e.g., names, types of methods and variables, of objects. This allows to perform serialization and deserialization in a generic manner, unlike in CORBA, which needs IDL specifications. For serialization, use reflection to find out

(1) class name of the object to be serialized

(2) the names, types

(3) values of its instance variables „

For deserialization,

(1) class name in the serialized form is used to create a class

(2) it is then used to create a constructor with arguments types corresponding to those specified in the serialized form.

(3) the new constructor is used to create a new object with instance variables whose values are read from the serialized form

2.4.3 XML (Extensible Markup Language)

- ✓ XML is a markup language that was defined by the World Wide Web Consortium (W3C) for general use on the Web.
- ✓ Markup language refers to a textual encoding that represents both a text and details as to its structure or its appearance.

- ✓ In XML, the data items are tagged with „markup‟ strings.
- ✓ The tags are used to describe the logical structure of the data and to associate attribute-value pairs with logical structures.
- ✓ XML is used to enable clients to communicate with web services and for defining the interfaces and other properties of web services.
- ✓ XML is extensible in the sense that users can define their own tags, in contrast to HTML, which uses a fixed set of tags.
- ✓ XML could be used by multiple applications for different purposes.

Example:

```
<student id="123456789">
<name>Sona</name>
<rollnumber>CS50</rollnumber>
<year>1984</year>
```

XML elements and attributes

- **Elements:** An element in XML consists of a portion of character data surrounded by matching start and end tags.
- **Attributes:** A start tag may optionally include pairs of associated attribute names and values such as id="123456789".
- **Names:** The names of tags and attributes in XML generally start with a letter, but can also start with an underline or a colon. The names continue with letters, digits, hyphens, underscores, colons or full stops. Letters are case-sensitive. Names that start with xml are reserved.
- **Binary data:** All of the information in XML elements must be expressed as character data. The encrypted elements or secure hashes are represented as base64 notation.
- **CDATA:** XML parsers normally parse the contents of elements because they may contain further nested structures. If text needs to contain an angle bracket or a quote, it may be represented in a special way.
- **Namespace:** A namespace applies within the context of the enclosing pair of start and end tags unless overridden by an enclosed namespace declaration.

2.4.3 Remote Object Reference

- ✓ Remote object references are used when a client invokes an object that is located on a remote server.

- ✓ A remote object reference is passed in the invocation message to specify which object is to be invoked.
- ✓ Remote object references must be unique over space and time.

32 bits	*32 bits*	*32 bits*	*32 bits*	
Internet address	port number	time	object number	interface of remote object

Fig 2.8: Representation of remote object references

The following are mandatory for remote object reference

- ➢ internet address/port number: process which created object
- ➢ time: creation time
- ➢ object number: local counter, incremented each time an object is created in the creating process
- ➢ interface: how to access the remote object (if object reference is passed from one client to another)

2.5 MULTICAST COMMUNICATION

Multicast (one-to-many or many-to-many distribution) is group communication where information is addressed to a group of destination computers simultaneously.

Multicast operation is an operation that sends a single message from one process to each of the members of a group of processes. The simplest way of multicasting, provides no guarantees about message delivery or ordering.

The following are the characteristics of multicasting:

- ➢ Fault tolerance based on replicated services:
  - ✓ A replicated service consists of a group of servers.
  - ✓ Client requests are multicast to all the members of the group, each of which performs an identical operation.
- ➢ Finding the discovery servers in spontaneous networking
  - ✓ Multicast messages can be used by servers and clients to locate available discovery services in order to register their interfaces or to look up the interfaces of other services in the distributed system.

- Better performance through replicated data
  - ✓ Data are replicated to increase the performance of a service.
- Propagation of event notifications
  - ✓ Multicast to a group may be used to notify processes when something happens.

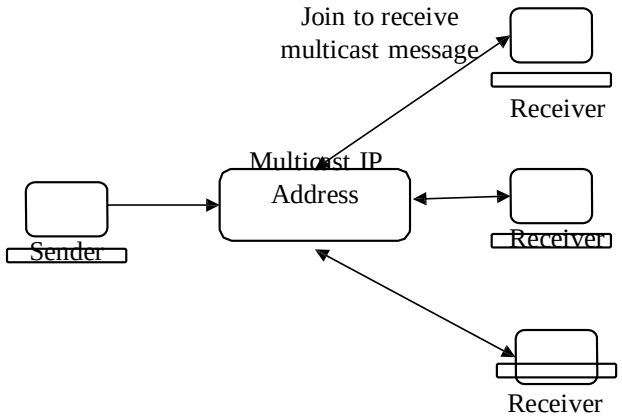


Fig 2.9: Multicasting

2.5.1 IP Multicast

- IP multicast is built on top of the Internet protocol.
- IP multicast allows the sender to transmit a single IP packet to a multicast group.
- A multicast group is specified by class D IP address with1110 as it starting byte.
- The membership of a multicast group is dynamic.
- A computer is said to be in a multicast group if one or more processes have sockets that belong to the multicast group.

The following details are important when using IPV4 IP multicasting:

- **Multicast IP routers**
  - IP packets can be multicast both on local network and on the Internet.
  - Local multicast uses local network such as Ethernet.
  - To limit the distance of propagation of a multicast datagram, the sender can specify the number of routers it is allowed to pass- called the time to live (TTL).

- **Multicast address allocation**
  - Multicast addressing may be permanent or temporary.
  - Permanent groups exist even when there are no members.
  - Multicast addressing by temporary groups must be created before use and will stop to exit when all members have left the particular multicast group.
  - To start or to join a particular multicast group, a **session directory (sd)** program is used.
  - This is a tool with an interactive interface that allows users to browse advertised multicast sessions and to advertise their own session, specifying the time and duration that it wishes to be in the group.

Failure model for IP multicast datagram

They suffer from omission failures. The messages are not guaranteed to be delivered to the group member in case of omission failures.

2.5.2 Java API to IP multicast

- ✓ The Java API provides a datagram interface to IP multicast through the class MulticastSocket.
- ✓ MulticastSocket is a subset of DatagramSocket.
- ✓ It has the capability of being able to join multicast groups.
- ✓ The multicast datagram socket class is useful for sending and receiving IP multicast packets.
- ✓ A MulticastSocket is a (UDP) DatagramSocket, with additional capabilities for joining "groups" of other multicast hosts on the internet.
- ✓ The class MulticastSocket provides two constructors:
  - MulticastSocket(): Create a multicast socket.
  - MulticastSocket(int): Create a multicast socket and bind it to a specific port.

MulticastSender.java

```
import java.io.*;

import java.net.*;

public class MulticastSender {
```

```
  public static void main(String[] args) {
    DatagramSocket socket = null;
    DatagramPacketoutPacket = null;
    byte[] outBuf;
    final int PORT = 8888;
     try {
      socket = new DatagramSocket();
      long counter = 0;
      String msg;
       while (true) {
        msg = " multicast! " + counter;
        counter++;
        outBuf = msg.getBytes();
         //Send to multicast IP address and port
        InetAddress address = InetAddress.getByName("224.2.2.3");
        outPacket = new DatagramPacket(outBuf, outBuf.length, address, PORT);
         socket.send(outPacket);
        System.out.println("Server sends : " + msg);
        try {
          Thread.sleep(500);
        } catch (InterruptedExceptionie) {
        }
      }
    } catch (IOExceptionioe) {
      System.out.println(ioe);
    }
  }
}
```

Multicastreceiver.java

```
import java.io.*;
import java.net.*;
public class MulticastReceiver {
  public static void main(String[] args) {
    MulticastSocket socket = null;
    DatagramPacketinPacket = null;
    byte[] inBuf = new byte[256];
    try {
      //Prepare to join multicast group
      socket = new MulticastSocket(8888);
      InetAddress address = InetAddress.getByName("224.2.2.3");
      socket.joinGroup(address);
       while (true) {
        inPacket = new DatagramPacket(inBuf, inBuf.length);
        socket.receive(inPacket);
        String msg = new String(inBuf, 0, inPacket.getLength());
        System.out.println("From " + inPacket.getAddress() + " Msg : " + msg);
      }
    } catch (IOExceptionioe) {
      System.out.println(ioe);
    }
  }
}
```

The following methods are available in MulticastSocket class:

- **getInterface**():Retrieve the address of the network interface used for multicast packets.
- **getTTL**():Get the default time-to-live for multicast packets sent out on the socket.

- **joinGroup**(InetAddress): Joins a multicast group.
- **leaveGroup**(InetAddress):Leave a multicast group.
- **send**(DatagramPacket, byte):Sends a datagram packet to the destination, with a TTL (time- to-live) other than the default for the socket.
- **setInterface**(InetAddress): Set the outgoing network interface for multicast packets on this socket, to other than the system default.
- **setTTL**(byte): Set the default time-to-live for multicast packets sent out on this socket.

2.5.2 Reliability and ordering in multicasting

There are many factors that puts the reliability of multicasting under question:

- ✓ A datagram sent from one multicast router to another may be lost.
- ✓ There are chances for a single datagram to arrive at multiple recipients, any one of those recipients may drop the message because its buffer is full.
- ✓ If a multicast router fails, the group members beyond that router will not receive the multicast message.

Ordering issue:

IP packets do not necessarily arrive in the order in which they were sent. Messages sent by two different processes will not necessarily arrive in the same order at all the members of the group. The problem of reliability and ordering is more common in:

- Fault tolerance based on replicated services
- Discovering services in spontaneous networking
- Better performance through replicated data
- Propagation of event notifications

2.6 NETWORK VIRTULIZATION: OVERLAY NETWORKS

Network virtualization is the construction of many different virtual networks over an existing network such as the Internet.

Network virtualization refers to the management and monitoring of an entire computer network as a single administrative entity from a single software-based administrator's console.

NV is implemented by installing software and services to manage the sharing of storage, computing cycles and applications. All servers and services in the network are treated as a single pool of resources that can be accessed without regard for its physical components.

2.6.1 Overlay Networks

An overlay network is a virtual network of nodes and logical links that is built on top of an existing network with the purpose to implement a network service that is not available in the existing network.

The overlay network may sometimes change the properties of the underlying network. The overlay network provides:

- a service that is tailored towards the needs of a class of application or a particular higher-level service
- more efficient operation in a given networked environment
- additional feature – for example, multicast or secure communication.

Advantages:

- ✓ They enable new network services to be defined without requiring changes to the underlying network.
- ✓ They encourage experimentation with network services and the customization of services to particular classes of application.
- ✓ Multiple overlays can be defined and can coexist, with the end result being a more open and extensible network architecture.

Disadvantages:

- ✓ They introduce an extra level of indirection.
- ✓ They add to the complexity of network services.

Types of Overlay Networks

The overlay networks are classified based on the following criteria:

1. **Classification based on application:**

   - **Distributed hash tables:**

     A method for storing hash tables in geographically distributed locations in order to provide a failsafe lookupmechanism for distributed computing. The

overlay network offers a service that manages a mapping from keys to values across a potentially large number of nodes in a completely decentralized manner (similar to a standard hash table but in a networked environment).

- **Peer-to-peer file sharing:**

  Overlay structures that focus on constructing tailored addressing and routing mechanisms to support the cooperative discovery and use of files.

- **Content distribution networks:**

  Overlays that subsume a range of replication, caching and placement strategies to provide improved performance in terms of content delivery to web users; used for web acceleration and to offer the required real-time performance for video streaming.

2. **Classification based on network style:**

   - **Wireless ad hoc networks:** Network overlays that provide customized routing protocols for wireless ad hoc networks.

   - **Disruption-tolerant networks:** Overlays designed to operate in hostile environments that suffer significant node or link failure and potentially high delays

3. **Classification based on additional features:**

   - **Multicast:** Overlay networks provide access to multicast services where multicast routers are not available.

   - **Security:** Overlay networks that offer enhanced security over the underling IP network, including virtual private networks.

   - **Resilience:** Overlay networks that seek an order of magnitude improvement in robustness and availability of Internet paths

2.6.2 Skype –An overlay Network

Skype employs a partially decentralized architecture – a mix of the peer-to-peer and client-server architectures. The client-server system is used for authentication while the peer-to-peer system is used for IP telephony, relaying, indexing peers and file transfers. On top of this is the Skype overlay network which users interact with directly, an overlay network is a virtual network that is formed above and independent of the underlying Internet protocol network. Skype uses overlay networks to achieve the following:

1. It enables them to design and use their own proprietary protocols and application over the Internet. This enables Skype to encrypt all packet transmissions.

nd-

the

om

usly

ling

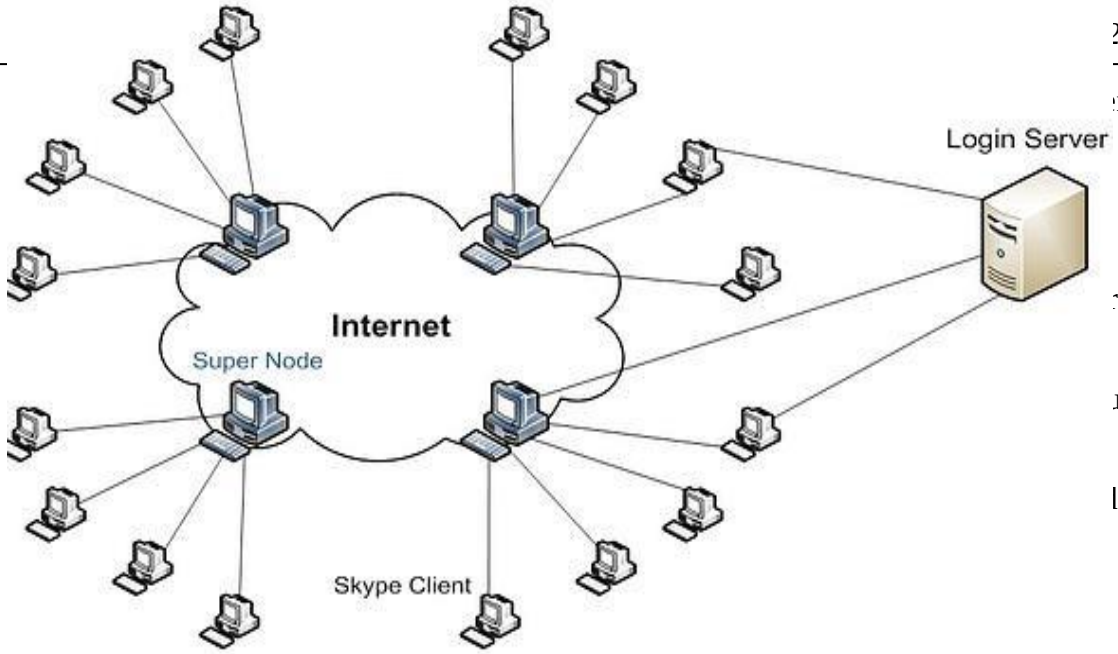


Fig 2.10: Skype Overlay Network

Skype architecture

- ✓ Skype is based on a peer-to-peer infrastructure consisting of ordinary users" machines called hosts and super nodes.
- ✓ Super nodes are ordinary Skype hosts that happen to have sufficient capabilities to carry out their enhanced role.
- ✓ Super nodes are selected on demand based a range of criteria including bandwidth available, reach ability and availability.

User connection

- ✓ Skype users are authenticated through a login server.
- ✓ They then make contact with a selected super node. To achieve this, each client maintains a cache of super node identities.
- ✓ At first login this cache is filled with the addresses of around seven super nodes, and over time the client builds and maintains a much larger set (perhaps several hundred).

Search for users

- ✓ Super nodes search the global index of users, which is distributed across the super nodes.
- ✓ The search is orchestrated by the client‟s chosen super node and involves an expanding search of other super nodes until the specified user is found.
- ✓ A user search typically takes between three and four seconds to complete for hosts that have a global IP address.

Voice connection

- ✓ Skype establishes a voice connection between the two parties using TCP for signaling all requests and terminations and either UDP or TCP for the streaming audio.

2.7 MESSAGE PASSING INTERFACE (MPI)

Message Passing Interface (MPI) is a standardized and portable message-passing system to function on a wide variety of parallel computers.

- MPI primarily addresses the message-passing parallel programming model: data is moved from the address space of one process to that of another process through cooperative operations on each process.
- The primary goal of the Message Passing Interface is to provide a widely used standard for writing message passing programs.
- The interface attempts to be: practical, portable, efficient and flexible.
- The basic principles of exchanging messages between two processes using send and receive operations is covered in MPI.

- There are two types of message passing:
  - **Synchronous message passing:** This is implemented by blocking send and receive calls.
  - **Asynchronous message passing:** This is implemented by a non-blocking form of send.

Originally, MPI was designed for distributed memory architectures.As architecture trends changed, shared memory SMPs were combined over networks creating hybrid distributed memory / shared memory systems. MPI runs on virtually any hardware platform like distributed memory, shared memory and in hybrid memory.

Send Operations	**Blocking Mode**	**Non Blocking Mode**
Generic	**MPI_Send**: the sender blocks until it is safe to return (i.e.) until the message is in transit or delivered and the sender‟s application buffer can therefore be reused.	**MPI_Isend:** the call returns immediately and the programmer is given a communication request handle, which can then be used to check the progress of the call via MPI_WaitorMPI_Test.
Synchronous	**MPI_Ssend:** the sender and receiver synchronize and the call only returns when the message has been delivered at the receiving end.	**MPI_Issend:** as with MPI_Isend,but with MPI_WaitandMPI_Testindicating whether the message has been delivered at the receive end.
Buffered	**MPI_Bsend:** the sender explicitly allocates an MPI buffer library and the call returns when the data is successfully copied into this buffer.	**MPI_Ibsend:** as with MPI_I send but with MPI_WaitandMPI_Testindicating whether the message has been copied into the sender‟s MPI buffer and hence is in transit.
Ready	**MPI_Rsend:** the call returns when the sender‟s application buffer can be reused but the programmer is also indicating to the library that the receiver is ready to receive the message, resulting in potential optimization of the underlying implementation.	**MPI_Irsend:** the effect is as with MPI_Isend, but as with MPI_Rsend, the programmer is indicating to the underlying implementation that the receiver is guaranteed to be ready to receive.

2.8 REMOTE INVOCATION

In a distributed object system, an object's data should be accessible only via its methods. The objects can be accessed via object references. The following terminologies are important in remote invocations:

- An **interface** provides a definition of the signatures of a set of methods without specifying their implementation.
- An **action** is initiated by an object invoking a method in another object.
- The **state** of an object consists of the values of its instance variables.

Remote invocations are method invocations for objects in different processes.

- **Remote objects** are objects that can receive remote invocation.

The core part of the distributed object model is:

- remote object reference;
- remote interface: specifies which methods can be invoked remotely;

2.8.1 Request reply protocols

The overview of request reply protocol is:

- The client sends a query to the server, and reads from the socket, usually in a blocking way, for the server response.
- The server processes the command and sends the response back to the client.

The Remote Procedure Call (RPC) is a common model of request reply protocol. There are two types of communication:

- **Synchronous:** The client process blocks until the reply arrives from the server. It is termed as reliable form of communication since the reply from the server is actually an acknowledgement to the client.
- **Asynchronous**: The clients can retrieve the replies from the server later.

UDP is preferred over TCP for the following reasons:

- Acknowledgements are redundant, since requests are followed by replies.
- Establishing a connection involves two extra pairs of messages in addition to the pair required for a request and a reply.
- Flow control is redundant for the majority of invocations, which pass only small arguments and results.

Request Reply protocol:

The following are the primitives in request reply protocols:

- **public byte[] doOperation (RemoteObjectRef o, intmethodId, byte[] arguments):**

  This method sends a request message to the remote object and returns the reply. The arguments specify the remote object, the method to be invoked and the arguments of that method.

- **public byte[] getRequest ():**

  This method acquires a client request via the server port.

- **public void sendReply (byte[] reply, InetAddressclientHost, intclientPort):**

  This method sends the reply message reply to the client at its Internet address and port.

Message identifiers

Each message have a unique message identifier for referencing it. A message identifier consists of two parts:

- request Id: Increasing sequence of integers assigned by the sending process. This makes the message unique to the sender.
- an identifier: for the sender process. This makes the message unique in the distributed system.

Failure model of the request-reply protocol

The following are the common failures in request reply protocol:

- Omission failures.
- Messages are not guaranteed to be delivered in sender order.
- Failure of processes.

Discarding duplicate request messages

The server may receive the same request message more than once. To avoid this, the protocol is designed to recognize successive messages from the same client with the same request identifier and to filter out duplicates.

Lost reply messages

If the server has already sent the reply to which it receives a duplicate request it will need to execute the operation again to obtain the result. This could be avoided if the server has stored the result of the original execution.

History

History is used to refer to a structure that contains a record of reply messages that have been transmitted by the server.

Each entry in the history has

- ✓ a request identifier
- ✓ a message
- ✓ client identifier

The main goal of history is to allow the server to retransmit replymessages when client processes request them. The main issue faced by the history is the size. As more communication takes place, more messages need to be saved in the history which incurs more cost.

Styles of exchange protocols

There are three exchange protocols:

- **request (R) protocol:** a single request message is sent by the client to the server
- **request-reply (RR) protocol:** used in client-server exchanges because it is based on the request-reply protocol. Special acknowledgement messages are not required.
- **request-reply-acknowledge reply (RRA) protocol:** It is based on request-reply-acknowledge reply. The Acknowledge reply message contains the requested from the reply message being acknowledged. This will enable the server to discard entries from its history. The arrival of a requested in an acknowledgement message will be interpreted as acknowledging the receipt of all reply messages with lower request Ids.

TCP streams to implement the request-reply protocol

It is difficult to determine the buffer size for the UDP datagrams. Also the fixed size of the datagrams imposes a limit on the message size which is not favorable. So TCP streams are preferred since they allow messages of any size to be transmitted. If the TCP protocol is used, it ensures that request and reply messages are delivered reliably thereby avoiding acknowledgments. The overhead due to TCP acknowledgement messages is reduced when a reply message follows soon after a request message.

HTTP: An Example Request Reply protocol

Hyper Text Transfer Protocol (HTTP) is a stateless request-response based communication protocol. It's used to send and receive data on the Web i.e., over the Internet. This protocol uses reliable TCP connections either for the transfer of data to and from clients (Web Browsers).

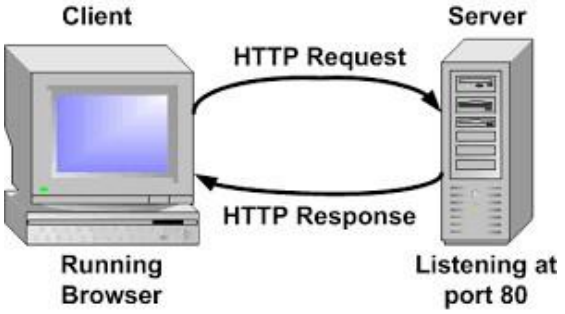


Fig 2.11: HTTP Request-Response protocol

HTTP specifies the messages involved in a request-reply exchange, the methods, arguments and results, and the rules for representing (marshalling) them in the messages. This protocol allows for content negotiation and password-style authentication

- **Content negotiation:** Clients" requests can include information as to what data representations they can accept.
- **Authentication:** Credentials and challenges are used to support password-style authentication.

The following are the connection establishment and closing steps:

- The client requests and the server accepts a connection at the default server port or at a port specified in the URL.
- The client sends a request message to the server.
- The server sends a reply message to the client.
- The connection is closed after the communication is over.

Connections in HTTP

There are two types of connections in HTTP: Persistent and non persistent

Differences between persistent and non persistent connections

Persistent Connection	Non persistent Connection
On the same TCP connection the server, parses request, responds and waits for new requests.	The server parses request, responds, and closes TCP connection.
It takes fewer RTTs to fetch the objects.	It takes 2 RTTs to fetch each object.
It has less slow start.	Each object transfer suffers from slow start

HTTP Request Message

Method	URL	HTTP version	Headers	Message body
GET	http://www.abc.ac.ind/data.html	HTTP/1.1		

Fig 2.12: HTTP Request Message

The following are the parts of HTTP request frame :

- **Request Method:** The request **method** indicates the method to be performed on the resource identified by the given **Request-URL**. The method is case-sensitive and should always be mentioned in uppercase.
- **Request-URL:** This Uniform Resource Identifier and identifies the resource upon which to apply the request.
- **Request Header Fields:** The request-header fields allow the client to pass additional information about the request, and about the client itself, to the server. These fields act as request modifiers.
- **HTTP Version:** This specifies the version of HTTP used.

HTTP Methods

The following are the HTTP methods:

- **GET:** The GET method is used to retrieve information from the given server using a given URI. Requests using GET should only retrieve data and should have no other effect on the data.
- **HEAD:** Same as GET, but it transfers the status line and the header section only.
- **POST:** A POST request is used to send data to the server, for example, customer information, file upload, etc. using HTML forms.
- **PUT:** Replaces all the current representations of the target resource with the uploaded content.
- **DELETE:** Removes all the current representations of the target resource given by URI.
- **CONNECT:** Establishes a tunnel to the server identified by a given URI.
- **OPTIONS:** Describe the communication options for the target resource.
- **TRACE:** Performs a message loop back test along with the path to the target resource.

HTTP Response Message

HTTP Version	Status Code	Reason	Headers	Message body
HTTP/1.1	2xx	OK		Data

Fig 2.13: HTTP Response Message

A response message consists of the protocol version followed by a numeric status code and its associated textual phrase.

- HTTP Version: A server supporting HTTP version 1.1 will return the following version information: HTTP-Version = HTTP/1.1
- Status Code: The Status-Code element is a 3-digit integer where first digit of the Status-Code defines the class of response and the last two digits do not have any categorization role. There are 5 values for the first digit:
  - ✓ 1xx- Informational :It means the request was received and the process is continuing.
  - ✓ 2xx- Success: It means the action was successfully received, understood, and accepted.
  - ✓ 3xx-Redirection: It means further action must be taken in order to complete the request.
  - ✓ 4xx- Client Error: It means the request contains incorrect syntax or cannot be fulfilled.
  - ✓ 5xx- Server error: It means the server failed to fulfill an apparently valid request.

HTTP status codes are extensible and HTTP applications are not required to understand the meaning of all registered status codes.

- **Response Header Fields:** The response-header fields allow the server to pass additional information about the response which cannot be placed in the Status-Line. These header fields give information about the server and about further access to the resource identified by the Request-URI.

2.8.2 Remote Procedure Call (RPC)

RPC is a powerful technique for constructing distributed, client-server based applications. In RPC, the procedure need not exist in the same address space as the calling procedure. The two processes may be on the same system, or they may be on different systems with a network connecting them. By using RPC, programmers of distributed applications avoid the details of the interface with the network. RPC makes the client/server model of computing more powerful and easier to program.

Remote Procedure Call (RPC) is a protocol that one program can use to request a service from a program located in another computer in a network without having to understand network details.

Design issues in RPC

The following are the design issues in RPC:

- **Programming with interfaces**
  - Communication between modules in a program can be done by procedure calls between modules or by direct access to the variables in another module.
  - To control the interactions between modules, an explicit interface is defined for each module that specifies the procedures and the variables that can be accessed from other modules.
  - The interface hides the programming details of one module from the other.
  - The interaction between the modules can happen even after the modification of the modules but without modifying the interfaces.
- **Interfaces in distributed systems**
  - Each server in the client server model provides a set of procedures that are available for use by clients.
  - The separation between interface and implementation offers the following advantages:
    - ✓ The programmers need not be aware of implementation details of each modules.
    - ✓ In the perspective of distributed systems, the programmers need to know the programming language or underlying platform used to implement the service.
    - ✓ It provides support for software evolution in which implementations can change as long as long as the interface remains the same. But the interface can also change as long as it remains compatible with the original.
- **Interface definition languages:**
  - An RPC mechanism can be integrated with a particular programming language if it contains notations for defining interfaces, allowing input and output parameters to be mapped onto the language‟s normal use of parameters.

- But programs can also be written in a variety of languages to access them remotely.
- In such cases Interface definition languages (IDL) are used which contains procedures implemented indifferent languages to invoke one another.

❖ **RPC call semantics**

- In request-reply protocols doOperation can be implemented in different ways to provide different delivery guarantees:
  - ✓ Retry request message: Controls whether to retransmit the request message until either a reply is received or the server is assumed to have failed.
  - ✓ Duplicate filtering : Controls when retransmissions are used and whether to filter out duplicate requests at the server.
  - ✓ Retransmission of results: Controls whether to keep a history of result messages to enable lost results to be retransmitted without re-executing the operations at the server.
- The above choices lead to a variety of possible semantics as given below:

a) **Maybe semantics:** Here, the remote procedure call may be executed once or not at all. Maybe semantics arises when no fault-tolerance measures are applied and can suffer from the following types of failure:

- ✓ omission failures if the request or result message is lost
- ✓ crash failures.

May be semantics is useful only for applications in which occasional failed calls are acceptable.

b) **At-least-once semantics:** Here the invoker receives either a result, in which case the invoker knows that the procedure was executed at least once, or an exception informing it that no result was received. At-least-once semantics can suffer from the following types of failure:

- ✓ crash failures when the server containing the remote procedure fails
- ✓ arbitrary failures

c) **At-most-once semantics:** Here**,** the caller receives either a result, in which case the caller knows that the procedure was executed exactly once, or an exception informing it that no result was received, in which case the procedure will have been executed either once or not at all.

❖ **Transparency**

- The remote procedure calls are much like local procedure calls as possible, with no distinction in syntax between a local and a remote procedure call.

- All the necessary calls to marshalling and message-passing procedures were hidden from the programmer making the call.
- Although request messages are retransmitted after a timeout, this is transparent to the caller to make the semantics of remote procedure calls like that of local procedure calls
- The usage of middleware can also offer additional levels of transparency to RPC.
- But RPC"s are more vulnerable to failure than local calls since they involve a network, another computer and another process.
- So the clients making remote calls must be able to recover from such situations.
- The latency of a remote procedure call is several orders of magnitude greater than that of a local one.

Implementation of RPC

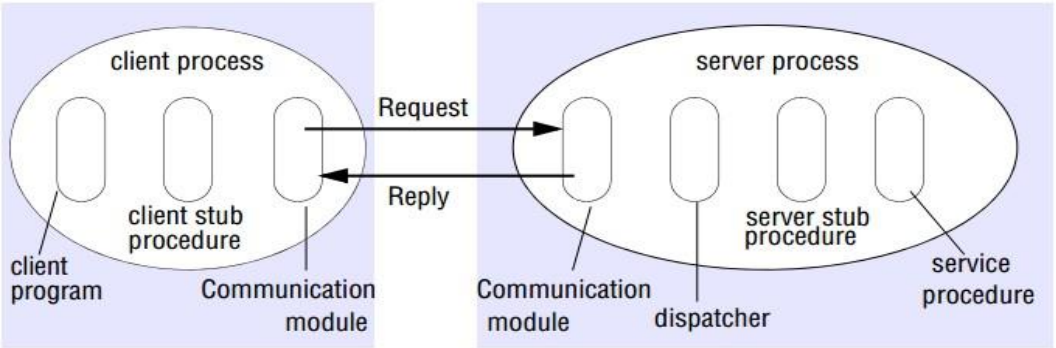


Fig 2.14: Client and Server stub in RPC

- The client that accesses a service includes one stub procedure for each procedure in the service interface.

A stub is a piece of code that is used to convert parameters during a remote procedure call (RPC). An RPC allows a client computer to remotely call procedures on a server computer.

- The parameters used in a function call have to be converted because the client and server computers use different address spaces.
- Stubs perform this conversion so that the remote server computer perceives the RPC as a local function call.
- The client stub routine performs the following functions:

- ✓ **Marshals arguments:** The client stub packages input arguments into a form that can be transmitted to the server.
- ✓ Calls the client run-time library to transmit arguments to the remote address space and invokes the remote procedure in the server address space.
- ✓ **Unmarshals output argument:** The client stub unpacks output arguments and returns to the caller.

- The server stub procedure unmarshals the arguments in the request message, calls the corresponding service procedure and marshals the return values for the reply message.
- The client and server stub procedures and the dispatcher are automatically generated by an interface compiler from the interface definition of the service.

2.8.3 Remote Method Invocation (RMI)

RMI is a set of protocols being developed by Sun's JavaSoft division that enables Java objects to communicate remotely with other Java objects.

RMI is closely related to RPC but extended into the world of distributed objects. RMI is a true distributed computing application interface specially designed for Java, written to provide easy access to objects existing on remote virtual machines

Differences between RMI and RPC

RMI	RPC
RMI uses an object oriented paradigm where the user needs to know the object and the method of the object he needs to invoke.	RPC is not object oriented and does not deal with objects. Rather, it calls specific subroutines that are already established
With RPC looks like a local call. RPC handles the complexities involved with passing the call from the local to the remote computer.	RMI handles the complexities of passing along the invocation from the local to the remote computer. But instead of passing a procedural call, RMI passes a reference to the object and the method that is being called.

The commonalities between RMI and RPC are as follows:

- ✓ They both support programming with interfaces.
- ✓ They are constructed on top of request-reply protocols.
- ✓ They both offer a similar level of transparency.

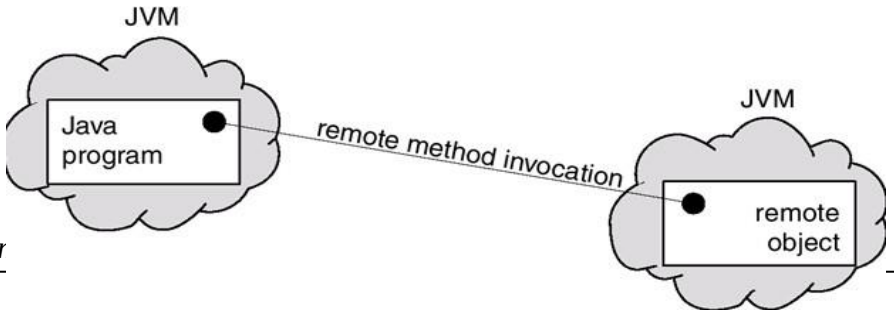


Fig 2.15: RMI in JVM

Features of RMI

- ✓ In RMI, the remote objects are treated similarly to local objects.
- ✓ RMI provide access to objects existing on remote virtual machines.
- ✓ RMI handles marshalling, transportation, and garbage collection of the remote objects.
- ✓ RMI was incorporated as a part of the JDK with version 1.1

Limitations of RMI

- ✓ RMI is not language independent. It is limited only to Java.
- ✓ Interfaces to remote objects are defined using ordinary Java interfaces not with special IDLs.

Design issues of RMI

The RMI evolved from simple object model to distributed objects model.

a) Object Oriented Model

An object communicates with other objects by invoking their methods by arguments and return values. Objects can encapsulate their dataand the code of their methods. In a distributed object system, an object‟s data should be accessible only through its methods.

⇒ **Object references:**

- Objects are accessed through object references.
- Object references could be assigned to variables, passed as arguments and returned as results of methods.

⇒ **Interfaces:**

- An interface provides implementation independent, definition of the signatures of a set of methods.

- An interface is not a class. A class describes the attributes and behaviors of an object. An interface contains behaviors that a class implements.
- An interface can contain any number of methods.
- An interface is written in a file with a .java extension, with the name of the interface matching the name of the file.
- The byte code of an interface appears in a .class file.
- Interfaces do not have constructors.

Differences between Interfaces and Class

Interfaces	**Class**
Interfaces cannot be instantiated.	Classes can be instantiated.
They do not contain constructors.	They can have constructors.
They cannot have instance fields (i.e.) all the fields in an interface must be declared both static and final.	No constraints over the fields.
The interface can contain only abstract method.	They can contain method implementation.
Interfaces are implemented by a class not extended.	Classes are extended by other classes.

⇒ **Actions :**

- Action is initiated by an object invoking a method in another object with or without arguments.
- The receiver executes the appropriate method and thenreturns control to the invoking object.
- The following are the effects of actions:
  - The state of the receiver may be changed.
  - A new object may be instantiated.
  - Further invocations on methods in other objects may take place.

⇒ **Exceptions:**

- Exceptions are run time errors.
- The programmers must write code that could handle all possible unusual or erroneous cases.

⇒ **Garbage collection:**

– This is a method in Java to free the space occupied by obsolete objects.

b) **Distributed objects**

⇒ Distributed object systems may adopt the client-server architecture where the objects are managed by servers and their clients invoke their methods using remote method invocation.

⇒ In RMI, the client‟s request to invoke a method of an object is sent in a message to the server managing the object.

⇒ The invocation is carried out by executing a method of the object at the server and the result is returned to the client in another message.

⇒ Objects with different data formats may be used at different sites.

c) **The distributed object model**

In this model, each process contains a collection of objects, some of which can receive both local and remote invocations, whereas the other objects can receive only local invocations. The core concepts of distributed object model:

- **Remote object references:** Other objects can invoke the methods of a remote object if they have access to its remote object reference.
- **Remote interfaces:** Every remote object has a remote interface that specifies which of its methods can be invoked remotely.
- **Remote object references:** The notion of object reference is extended to allow any object that can receive an RMI to have a remote object reference.

A remote object reference is an identifier that can be used throughout a distributed system to refer to a particular unique remote object.

The remote object to receive a remote method invocation is specified by the invoker as a remote object reference. The remote object references may be passed as arguments and results of remote method invocations.

- **Remote interfaces**: The class of a remote object implements the methods of its remote interface. Objects in other processes can invoke only the methods that belong to its remote interface.

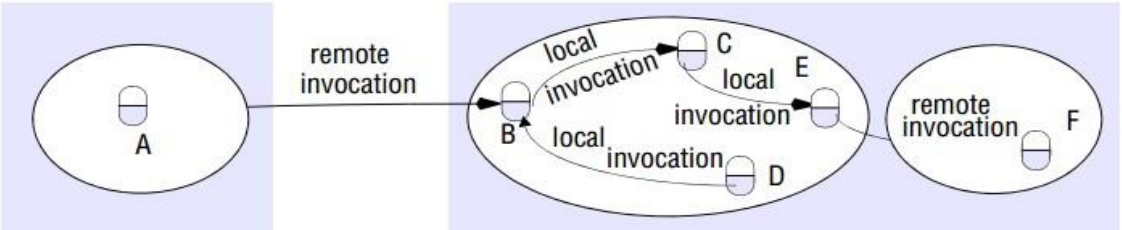


Fig 2.16: Remote and local invocations

Differences between remote and local objects

Remote Objects	**Local Objects**
Creating a remote object requires creating a stub and a skeleton, which will cost more time cycles.	They consume less time cycles.
Marshalling and unmarshalling is required.	Marshalling and unmarshalling is not required.
Longer time is taken by the remote object to return to the calling program.	Comparatively shorter return time.

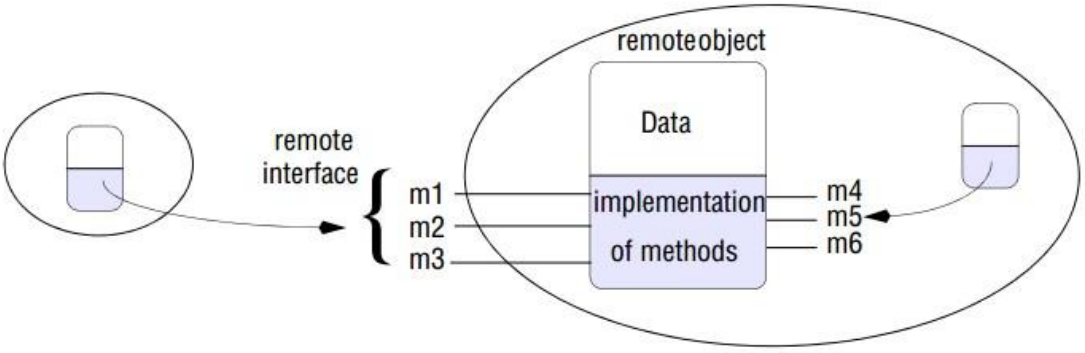


Fig 2.17: Remote Object and remote interfaces

Garbage collection in a distributed-object system:

Distributed garbage collection is usually based on reference counting. The total number of references to the object is maintained by a **reference count field**. The advantage of using reference counts is that garbage is easily identified. When it becomes necessary to reclaim the storage from unused objects, the garbage collector needs only to examine the reference count fields of all the objects that have been created by the program. If the reference count is zero, the object is garbage.

Exceptions:

Any remote invocation may fail because of exceptions. So RMI should include exception handling as a part of their implementation.

Implementation of RMI

Implementation of a RMI requires construction if many modules and the supporting interfaces.

a) Modules:

The two common modules used are communication module and remote reference module.

Communication module:

- ✓ This module implements request-reply protocol.
- ✓ This module specifies the message type, its requestId and the remote reference of the object to be invoked.
- ✓ This specifies only the communication semantics.
- ✓ The communication module in the server selects the dispatcher for the class of the object to be invoked, passing on its local reference, which it gets from the remote reference module in return for the remote object identifier in the request message.

Remote reference module:

- ✓ This module is responsible for translation between local and remote object references and for creating remote object references.
- ✓ This module has a remote object table specific for each process that records the mapping between local object references and remote object references.
- ✓ The entries of the table includes:
  - the remote objects held by the process.
  - local proxy
- ✓ The actions of them remote reference module are as follows:
  - When a remote object is to be passed as an argument for the first time, the remote reference module creates a remote object reference and is added to the table.
  - When a remote object reference arrives in a request or reply message the remote object table is referred for the corresponding local object reference.
  - If the remote object reference is not in the table, the RMI software creates a new proxy and asks the remote reference module to add it to the table.
  - This module is called by components of the RMI software during marshalling and unmarshalling remote object references.

b) Supporting Components

Servants

- ✓ A servant is an instance of a class that provides the body of a remote object.
- ✓ The responsibility of the servant is to handle the remote requests passed on by the corresponding skeleton.
- ✓ Servants live within a server process.
- ✓ They are created when remote objects are instantiated and remain in use until they are no longer needed and finally servants are garbage collected or deleted.

The RMI software

- ✓ This consists of the application-level objects and the communication and remote reference modules. The RMI software has the following components:
  - **Proxy:** The class of the proxy contains a method corresponding to each method in the remote interface which marshalls arguments and unmarshalls results for that method. It hides the details of the remote object reference. Each method of the proxy marshals a reference to the target object using a request message (operationId, arguments). When the reply is received the proxy, unmarshals it and returns the results to the invoker.
  - **Dispatcher:** Server has a dispatcher & a skeleton for each class of remote object. The dispatcher receives the incoming message, and uses its method info to pass it to the right method in the skeleton.
  - **Skeleton:** Skeleton implements methods of the remote interface to unmarshall arguments and invoke the corresponding method in the servant. It then marshalls the result (or any exceptions thrown) into a reply message to the proxy.

Generation of the classes for proxies, dispatchers and skeletons

- ✓ The classes for the proxy, dispatcher and skeleton used in RMI are generated automatically by an interface compiler.
- ✓ For Java RMI, the set of methods offered by a remote object is defined as a **Java interface.**

Dynamic invocation:

- ✓ This is an alternative to proxies.
- ✓ The proxy just described (i.e.) its class is generated from an interface definition and then compiled into the client code.

- ✓ If a client program receives a remote reference to an object whose remote interface was not available at compile time. The method could not be invoked through proxies.
- ✓ Dynamic invocation is used in this case which gives the client access to a generic representation of a remote invocation, which is a part of the infrastructure for RMI.
- ✓ The client will supply the remote object reference, the name of the method and the arguments to doOperation and then wait to receive the results.

Dynamic skeletons:

The Dynamic Skeleton Interface (or DSI) allows applications to provide implementations of the operations on CORBA objects without static knowledge of the object's interface. It is the server-side equivalent of the Dynamic Invocation Interface.

Server programs

The server program contains

- ✓ the classes for the dispatchers and skeletons, with the implementations of the classes of all of the servants that it supports.
- ✓ an initialization section which is responsible for creating and initializing at least one of the servants to be hosted by the server.

Client programs

The client program will contain

- ➢ the classes of the proxies for all of the remote objects that it will invoke.

Factory methods

- ✓ Factory methods are static methods that return an instance of the native class.
- ✓ Factory method is used to create different object from factory often refereed as Item and it encapsulate the creation code.
- ✓ So instead of having object creation code on client side we encapsulate inside Factory method in Java.
- ✓ The term factory method is sometimes used to refer to a method that creates servants, and a factory object is an object with factory methods.
- ✓ Any remote object that needs to be able to create new remote objects on demand for clients must provide methods in its remote interface for this purpose. Such methods are called factory methods, although they are really just normal methods.

Binder

- ✓ A binder in a distributed system is a separate service that maintains a table containing mappings from textual names to remote object references.
- ✓ It is used by servers to register their remote objects by name and by clients to look them up.

Server threads

- ✓ Whenever an object executes a remote invocation, that execution may lead to further invocations of methods in other remote objects, which may take sometime to return.
- ✓ To avoid this, servers generally allocate a separate thread for the execution of each remote invocation.

Activation of remote objects

- ✓ Activation of servers to support remote objects is done by a service called **Inetd**.
- ✓ Certain applications demand the objects to persist for longer times.
- ✓ When the objects are maintained for longer period it is actually waste of resources.
- ✓ To avoid this the servers could be started on demand as and when needed through Inetd.
- ✓ Processes that start server processes to host remote objects are called **activators.**
- ✓ Based on this remote objects can be described as active or passive.
  - ➢ An object is said to be **active** when it is available for invocation.
  - ➢ An object is called **passive** if is currently inactive but can be made active.
- ✓ A passive object consists of two parts:
  - ➢ implementation of its methods
  - ➢ its state in the marshalled form.
- ✓ When activating a passive object, an active object is created from the corresponding passive object by creating a new instance of its class and initializing its instance variables from the stored state.
- ✓ Passive objects can be activated on demand.

Responsibilities of an activator:

- ✓ Registering passive objects that are available for activation. This is done by mapping the names of servers and passive objects.

- ✓ Starting named server processes and activating remote objects in them.
- ✓ Tracking the locations of the servers for remote objects that it has alreadyactivated.
- ✓ Java RMI provides the ability to make some remote objects activatable [java.sun.comIX].

Persistent object stores

An object that is guaranteed to live between activations of processes is called a persistent object.

- ✓ Persistent objects are managed by persistent object stores, which store their state in a marshalled form on disk.
- ✓ The objects are activated when their methods are invoked by other objects.
- ✓ When activating the object, the invoker will not know whether an object is in main memory or in a disk.
- ✓ The persistent objects that are not needed are maintained in the object store (active or passive state).
- ✓ The decision of making an active object to passive is done through the following ways:
  - ➢ in response to a request in the program that activated the objects
  - ➢ at the end of a transaction
  - ➢ when the program exits.
- ✓ There are two approaches to determine whether an object is persistent or not:
  - ➢ The persistent object store maintains some persistent roots, and any object that is reachable from a persistent root is defined to be persistent.
  - ➢ The persistent object store provides some classes. If the object belong to their subclasses, then it is persistent.

Object location

- ✓ Remote object reference can be done based on the Internet address and port number of the process that created the remote object.
- ✓ Some remote objects will exist in different processes, on different computers, throughout their lifetime.
- ✓ In this case, a remote object reference cannot act as an address for the objects.

- ✓ So clients making invocations require both a remote object reference and an address to which to send invocations.
- ✓ A **location service** helps clients to locate remote objects from their remote object references.
- ✓ It acts as a database that maps remote object references to their probable current locations.
- ✓ Alternatively a **cache/broadcast scheme** could also be used.
- ✓ Here, a member of a location service on each computer holds a small cache of remote object reference to-location mappings.
- ✓ If a remote object reference is in the cache, that address is tried for the invocation and will fail if the object has moved.
- ✓ To locate an object that has moved or whose location is not in the cache, the system broadcasts a request.

Distributed garbage collection

- ✓ It is always desirable to automatically delete the remote objects that are no longer referenced by any client.
- ✓ The memory held by those objects will be recovered.
- ✓ RMI uses a **reference-counting garbage collection** algorithm for automatic garbage collection.
- ✓ In this algorithm, the RMI runtime keeps track of all live references within each Java virtual machine.
- ✓ When a live reference enters a Java virtual machine, its reference count is incremented.
- ✓ The first reference to an object sends a referenced message to the server for the object.
- ✓ As live references are found to be unreferenced in the local virtual machine, the count is decremented.
- ✓ When the last reference has been discarded, an unreferenced message is sent to the server.
- ✓ The Java distributed garbage collection algorithm can tolerate the failure of client processes.
- ✓ To achieve this, servers lease their objects to clients for a limited period of time.

- ✓ The lease period starts when the client makes first referenceinvocation to the server.
- ✓ It ends either when the time has expired or when the client makes a last reference invocation to the server.
- ✓ The information stored by the server (i.e.) the leasecontains the identifier of the client‟s virtual machine and the period of the lease.
- ✓ Clients are responsible for requesting the server to renew their leases before they expire.

Leases in Jini

- ✓ Jini provides a mechanism for locating services on the network that conform to a particular (Java) interface, or that have certain attributes associated with them.
- ✓ Once a service is located, the client can download an implementation of that interface, which it then uses to communicate with the service.
- ✓ The Jini distributed system includes a specification for leases that can be used in a variety of situations when one object offers a resource to another object.
- ✓ The granting of the use of a resource for a period of time is called a **lease**.
- ✓ The object offering the resource will maintain it until the time in the lease expires.
- ✓ The resource users are responsible for requesting their renewal when they expire.
- ✓ The period of a lease may be negotiated between the grantor and the recipient in Jini.
- ✓ In Jini, an object representing a lease implements the Lease interface.
- ✓ It contains information about the period of the lease and methods enabling the lease to be renewed or cancelled.

2.9 CASE STUDY: JAVA RMI

In this case study we use shared white board example. This program that allows a group of users to share a common view of a drawing surface containing graphical objects, such as rectangles, lines and circles, each of which has been drawn by one of the users.

Remote interfaces in Java RMI

- ✓ Remote interfaces are defined by extending an interface called Remote in the java.rmipackage.
- ✓ The methods must throwRemoteException,contain an GraphicalObject which implements theSerializableinterface.

Java Remote interfaces Shape and ShapeList

```
import java.rmi.*;
import java.util.Vector;
public interface Shape extends Remote
 {
intgetVersion() throws RemoteException;
GraphicalObjectgetAllState() throws RemoteException;
}
public interface ShapeList extends Remote
 {
   Shape newShape(GraphicalObject g) throws RemoteException;
   Vector allShapes() throws RemoteException;
intgetVersion() throws RemoteException;
}
```

Parameter and result passing

The parameters of a method are input parameters and the result of a method is a single output parameter. Any object that implements the Serializable interface can be passed as an argument or result in Java RMI. All primitive types and remote objects are serializable.

- **Passing remote objects:**
  - ✓ When the type of a parameter or result value is defined as a remote interface, the corresponding argument or result is always passed as a remote object reference.
  - ✓ When a remote object reference is received, it can be used to make RMI calls on the remote object to which it refers.
- **Passing non-remote objects:**
  - ✓ All serializable non-remote objects are copied and passed by value.
  - ✓ When an object is passed by value, a new object is created in the receiver‟s process.
  - ✓ The methods of this new object can be invoked locally, possibly causing its state to differ from the state of the original object in the sender‟s process.
  - ✓ In this example, the client uses the method new Shape that passes an instance of GraphicalObject to the server.
  - ✓ The server creates a remote object of type Shape with the state of the GraphicalObject and returns a remote object reference to it.

- ✓ The arguments and return values in a remote invocation are used with the following modifications:

  1. Whenever an object that implements the Remote interface is serialized, it is replaced by its remote object reference, which contains the name of its (the remote object‟s) class.

  2. When any object is serialized, its class information is annotated with the location of the class (as a URL), enabling the class to be downloaded by the receiver.

Downloading of classes

- ✓ In Java classes can be downloaded from one virtual machine to another.
- ✓ If the recipient does not already possess the class of an object passed by value, its code is downloaded automatically.
- ✓ If the recipient of a remote object reference does not already possess the class for a proxy, its code is downloaded automatically.
- ✓ The advantages are:

  1. There is no need for every user to keep the same set of classes in their working environment.

  2. Both client and server programs can make transparent use of instances of new classes whenever they are added.

RMIregistry

- ✓ The RMIregistry is the binder for Java RMI.
- ✓ It maintains a table mapping URL-style names to references to remote objects.
- ✓ The Naming class is used to access the RMIregistry.
- ✓ The methods of the naming class is of the general form:

  //computerName:port/objectName
  where computerName and port refer to the location of the RMIregistry.

- ✓ The LocateRegistryclass , which is in java.rmi.registry, is used to discover the names.
- ✓ The getRegistry()returns an object of typeRegistry representing the remote binding service:
  public static Registry getRegistry() throws RemoteException

The following table gives a description of the methods in the Naming class:

Method	**Description**
void rebind (String name, Remote obj)	This method is used by a server to register the identifier of a remote object by name.
void bind (String name, Remote obj)	This method is also used to register a remote object by name, but if the name is already bound to a remote object reference an exception is thrown.
void unbind (String name, Remote obj)	This method removes a binding.
Remote lookup(String name)String [] list()	This method returns an array of Strings containing the names bound in the registry .This method is used by clients to look up a remote object by name.

Server Program

```
import java.util.Vector;
public class ShapeListServant implements ShapeList
{
    private Vector theList; // contains the list of Shapes
    private int version;
    public ShapeListServant(){...}
    public Shape newShape(GraphicalObject g)
    {
        version++;
        Shape s = new ShapeServant( g, version);
theList.addElement(s);
        return s;
    }
public Vector allShapes(){...}
public intgetVersion() { ... }
}
```

Client Program

```
import java.rmi.*;
import java.rmi.server.*;
import java.util.Vector;
public class ShapeListClient
{
    public static void main(String args[])
    {
System.setSecurityManager(new RMISecurityManager());
ShapeListaShapeList = null;
            try
            {
aShapeList = (ShapeList) Naming.lookup("//project.ShapeList");
                    Vector sList = aShapeList.allShapes();
            }
        catch(RemoteException e)
        {
System.out.println(e.getMessage());
        }
        catch(Exception e)
        {
System.out.println("Client: " + e.getMessage());
        }
  }
}
```

Callbacks

Callback refers to a server‟s action of notifying clients about an event. Callbacks can be implemented in RMI as follows:

- ✓ The client creates a remote object that implements an interface that contains a method for the server to call.
- ✓ The server provides an operation allowing interested clients to inform it of the remote object references of their callback objects.
- ✓ Whenever an event of interest occurs, the server calls the interested clients.

The disadvantages of callbacks:

- The performance of the server may be degraded by the constant polling.
- Clients cannot notify users of updates in a timely manner.

Design and implementation of Java RMI

Use of reflection

- ✓ Reflection is used to pass information in request messages about the method to be invoked.
- ✓ This is achieved with the help of the class Method in the reflection package.
- ✓ Each instance of Method represents the characteristics of a particular method, including its class and the types of its arguments, return value and exceptions.
- ✓ An instance of Method can be invoked on an object of a suitable class by means of its invoke method.
- ✓ The invoke method requires two arguments: the first specifies the object to receive the invocation and the second is an array of Object containing the arguments. The result is returned as type Object.
- ✓ The proxy has to marshal information about a method and its arguments into the request message with array of Objects as its arguments and then marshals that array.
- ✓ The dispatcher ; arguments in the array of Objects from th
- ✓ The dispatcher method, supplying the target and the array od has been executed, the dispatcher mars e reply message.

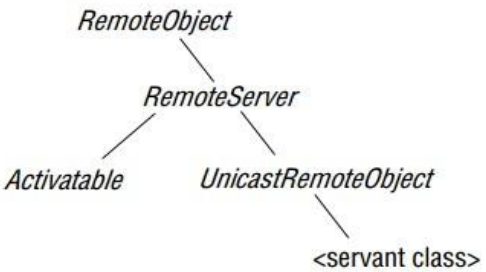


Fig 2.13: Classes in Java RMI

Java classes supporting RMI

- Every servent need to extend UnicastRemoteObject.

- The class UnicastRemoteObject extends an abstract class called RemoteServer, which provides abstract versions of the methods required by remote servers.
- Activatable class is available for providing activatable objects.
- The class RemoteServer is a subclass of RemoteObject that has an instance variable holding the remote object reference and provides the following methods:
  - equals: Compares remote object references.
  - toString: Returns the contents of the remote object reference as a String
  - readObject, writeObject: These methods deserialize/serialize remote objects

2.10 GROUP COMMUNICATION

Group communication is a service where a message is sent to a group and then this message is delivered to all members of the group.

Programming Model

A group has group members (processes) who can join or leave the group. This is done through multicast communication. There are three communication modes:

- **Unicast:** Process to process communication
- **Broadcast:** Communication to all the process
- **Multicast:** Communication only to a group of processes

Process Groups and Objects Groups

- ✓ An object group is a collection of objects that process the same set of invocations concurrently.
- ✓ Client objects invoke operations on a single, local object, which acts as a proxy for the group.
- ✓ The proxy uses a group communication system to send the invocations to the members of the object group.
- ✓ Object parameters and results are marshalled as in RMI and the associated calls are dispatched automatically to the right destination objects/methods.

Types of group communication

- **Closed and open groups:** A group is said to be closed if only members of the group may multicast to it A process in a closed group delivers to itself any message that it multicasts to the group.

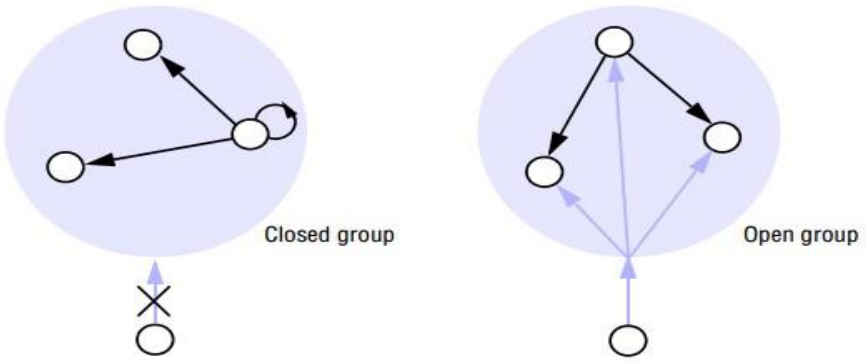


Fig 2.15: Open and Closed group communication

- **Overlapping and non-overlapping groups:** In overlapping groups, entities may be members of multiple groups, and non-overlapping groups imply that membership does not overlap to at most one.
- **Synchronous and asynchronous systems**: Group communication is possible in both environments.

Reliability and ordering in multicast

- ✓ In group communication, all members of a group must receive copies of the messages sent to the group, with delivery guarantees.
  - ➢ Reliable multicast operation is based on:
  - ➢ Integrity: delivering the message correctly only once
  - ➢ Validity: guaranteeing that a message sent will eventually be delivered.
- ✓ Group communication demands relative ordering of messages delivered to multiple destinations.
- ✓ The following are the types of message ordering:
  - ➢ **FIFO ordering:** If a process sends one message before another, it will be delivered in this order at all processes in the group.
  - ➢ **Causal ordering:** If a message happens before another message in the distributed
    system this so-called causal relationship will be preserved in the delivery of the associated messages at all processes.
  - ➢ **Total ordering:** In total ordering, if a message is delivered before another message at one process, then the same order will be preserved at all processes.

Group membership management

A group membership service has four main tasks:

- **Providing an interface for group membership changes:** The membership service provides operations to create and destroy process groups and to add or withdraw a process to or from a group.
- **Failure detection:** The service monitors the group members in case of crash and unreachability. The detector marks processes as Suspected or Unsuspected. The process is excluded from membership if it is not reachable or crashed.
- **Notifying members of group membership changes:** The service notifies the group‟s members when a process is added, or when a process is excluded.
- **Performing group address expansion:** When a process multicasts a message, it supplies the group identifier rather than a list of processes in the group.

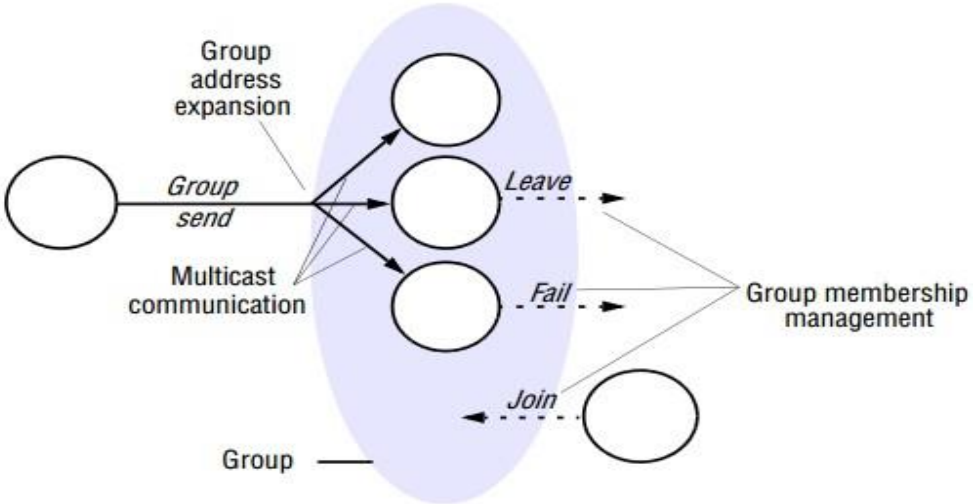


Fig 2.15: Group membership management

Case study: the JGroups toolkit

JGroups is a toolkit for reliable group communication written in Java. JGroups supports process groups in which processes are able to join or leave a group, send a message to all members of the group or indeed to a single member, and receive messages from the group. The toolkit supports a variety of reliability and ordering guarantees.

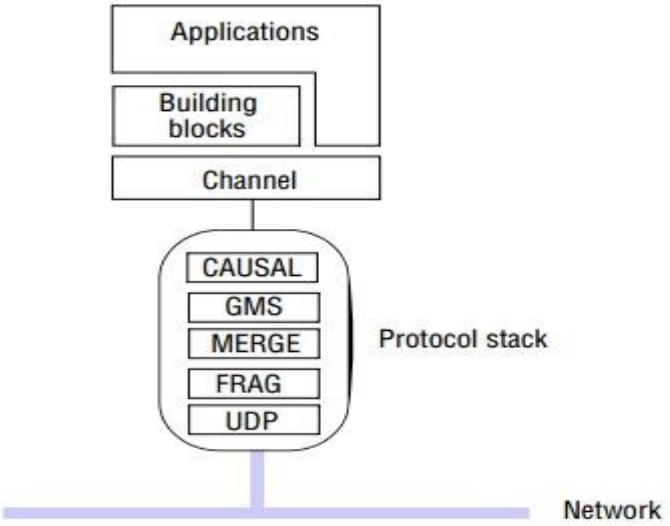


Fig 2.16: Architecture of JGroups

The following are the important components in the architecture:

- **Channels**
  - ✓ They represent the most primitive interface for application developers, offering the core functions of joining, leaving, sending and receiving.
  - ✓ A process interacts with a group through a channel object, which acts as a handle onto a group.
  - ✓ The following operations are allowed in channel:
    - Connect: binds that handle to a particular named group.If the named group does not exist, it is implicitly created at the time of the first connect.
    - Disconnect: To leave the group, the process executes the corresponding disconnect operation.
    - Close: will render the channel unusable.
    - getView: returns the current view defined in terms of the current member list
    - getState: returns the historical application state associated with the group
- **Building blocks**
  - ✓ This offers higher-level abstractions, building on the underlying service offered by channels.

- ✓ Some of the building blocks are:
  - **MessageDispatcher**
    - A sender to send a message to a group and then wait for some or all of the replies.
    - MessageDispatcher supports this by providing a castMessage method that sends a message to a group and blocks until a specified number of replies are received.
  - **RpcDispatcher**
    - This takes a specific method and invokes this method on all objects associated with a group.
  - **NotificationBus**
    - This is an implementation of a distributed event bus, in which an event is any serializable Java object.
    - This class is often used to implement consistency in replicated caches.

➢ **Protocol stack**

- ✓ This provides the underlying communication protocol, constructed as a stack of protocol layers.
- ✓ Here, a protocol is a bidirectional stack of protocol layers with each layer implementing the following two methods:
  public Object up (Event evt);
  public Object down (Event evt);
- ✓ Each layer process the message, including modifying its contents, adding a header or indeed dropping or reordering the message.
- ✓ This shows a protocol that consists of five layers:
  - UDP: This is the transport layer that utilizes IP multicast and UDP datagrams specifically for point-to-point communication. For larger-scale systems operating over wide area networks, a TCP layer may be preferred.
  - FRAG: This implements message packetization and the maximum packet size is 8,192 bytes by default.
  - MERGE: This deal with unexpected network partitioning and the subsequent merging of subgroups after the partition.
  - GMS: This implements a group membership protocol to maintain consistent views of membership across the group
  - CAUSAL: This implements causal ordering.

2.11 PUBLISH -SUBSCRIBER SYSTEMS

- ✓ Publish–subscribe is a messaging pattern where senders of messages, called **publishers,** do not program the messages to be sent directly to specific receivers, called **subscribers**.
- ✓ Instead, published messages are characterized into classes. The subscribers express interest in one or more classes, and only receive messages that are of interest.
- ✓ The task of the publish subscribe system is to match subscriptions against published events and ensure the correct delivery of event notifications.

Characteristics of publish-subscribe systems

- ➢ **Heterogeneity:** In distributed environment, the different types of components are made to interoperate with each other. To maintain integrity an interface is designed for receiving and dealing with the resultant notifications.
- ➢ **Asynchronicity:** Notifications are sent asynchronously by event-generating publishers to all the subscribers that have expressed an interest. This avoids the need for synchronizing.

The programming model

- ✓ Publishers disseminate an event e through a publish(e) operation and subscribers express an interest in a set of events through subscribe(f).
- ✓ This refers to a filter or a pattern defined over the set of all possible events.
- ✓ Subscribers can later revoke this interest through a corresponding unsubscribe(f) operation.
- ✓ The events are delivered to the subscriber using a notify(e) operation.
- ✓ The advertise(f) allows the publishers to have the option of declaring the nature of future events.

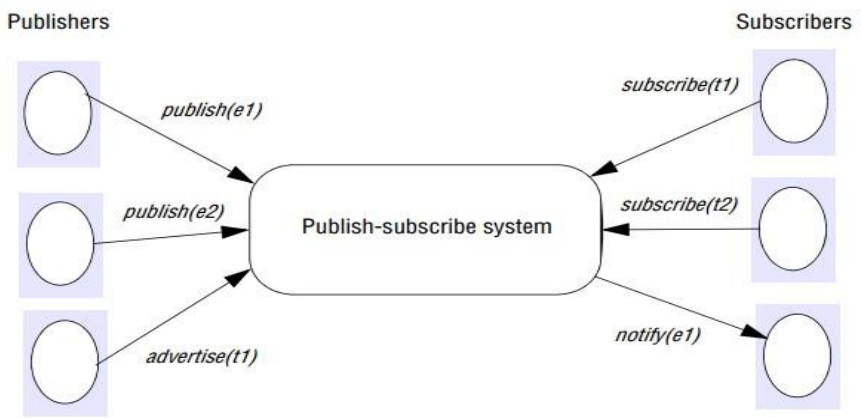


Fig 2.17: Publish Subscriber Model

The following are the subscriber or filter models:

- **Channel-based:**
  - ✓ The publishers publish events to named channels and subscribers then subscribe to one of these named channels to receive all events sent to that channel.
- **Topic-based (subject-based):**
  - ✓ Each notification from the publisher is expressed in terms of a number of fields, with one field denoting the topic.
  - ✓ Subscriptions are then defined in terms of the topic of interest.
- **Content-based:**
  - ✓ Content-based approaches are a generalization of topic-based approaches allowing the expression of subscriptions over a range of fields in an event notification.
- **Type-based:**
  - ✓ This is intrinsically linked with object-based approaches where objects have a specified type.
  - ✓ In type-based approaches, subscriptions are defined in terms of types of events and matching is defined in terms of types or subtypes of the given filter.

Implementation Issues

The publish subscribe model also demands of security, scalability, failure handling, concurrency and quality of service. This makes this model complex.

Centralized versus distributed implementations

- ✓ The centralized implementation of this model is forming a single node with a server on that node acting as an event broker.
- ✓ Publishers then publish events to this broker, and subscribers send subscriptions to the broker and receive notifications in return.
- ✓ Interaction with the broker is then through a series of point-to-point messages; this can be implemented using message passing or remote invocation.
- ✓ This lacks resilience and scalability, since the centralized broker represents a single point for potential system failure and a performance bottleneck.

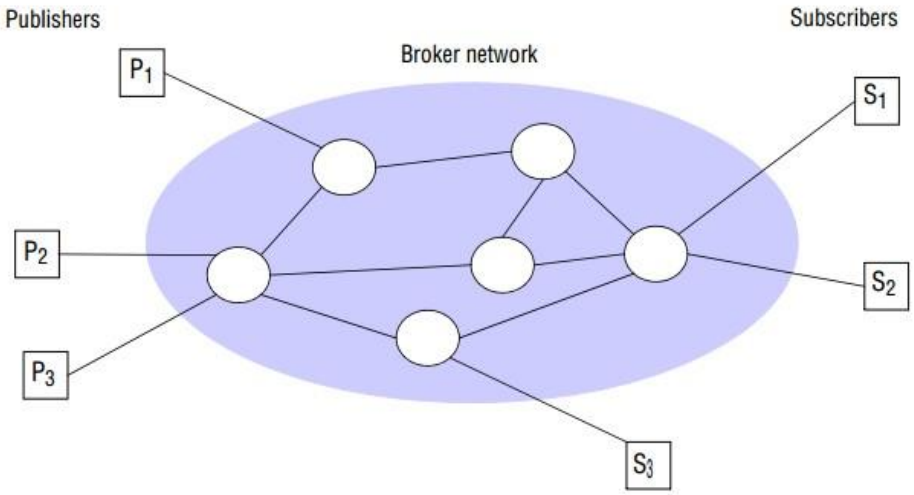


Fig 2.17: Network of Brokers

- ✓ Alternatively, the centralized broker is replaced by a network of brokers that cooperate to offer the desired functionality.
- ✓ A fully peer-to-peer implementation of a publish-subscribe system is also possible.
- ✓ In this approach, there is no distinction between publishers, subscribers and brokers; all nodes act as brokers, cooperatively implementing the required event routing functionality.

Overall systems architecture

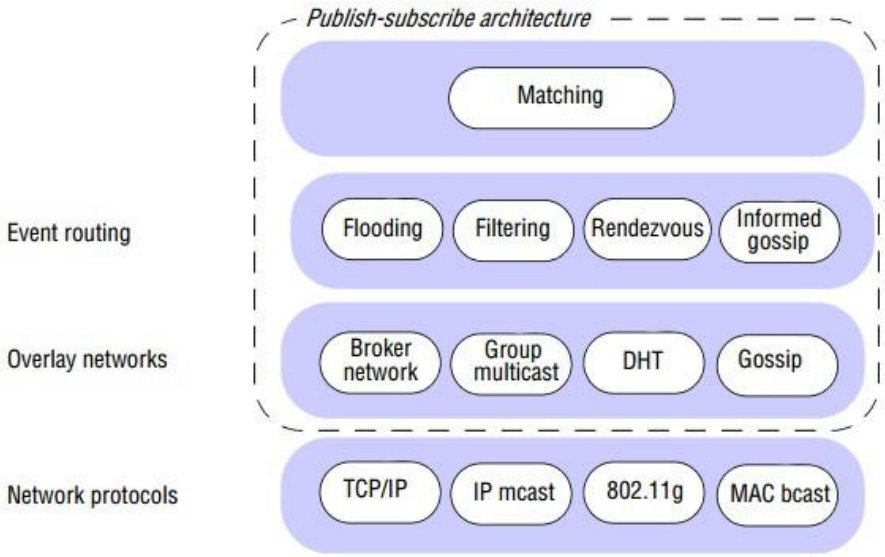


Fig 2.17: Overall system architecture

- ✓ In the bottom layer, publish-subscribe systems make use of a range of interprocess communication services.
- ✓ The event routing layer is supported by a network overlay infrastructure. Event routing performs the task of ensuring that event notifications are routed as efficiently as possible to appropriate subscribers
- ✓ The overlay infrastructure supports this by setting up appropriate networks of brokers or peer-to-peer structures.
- ✓ For content-based approaches, this problem is referred to as content-based routing (CBR), with the goal being to exploit content information to efficiently route events to their required destination.
- ✓ The top layer implements matching ensure that events match a given subscription.

The general principles behind content-based routing are:

- **Flooding:**
  - ✓ Sending an event notification to all nodes in the network and then carrying out the appropriate matching at the subscriber end is flooding.
  - ✓ Alternatively, flooding can also be used to send subscriptions back to all possible publishers, with the matching carried out at the publishing end and matched events sent directly to the relevant subscribers using point-to-point communication.
  - ✓ Flooding can be implemented using an underlying broadcast or multicast facility or arranging brokers in an acyclic graph in which each forwards incoming event notifications to all its neighbors.
- **Filtering:**
  - ✓ In filtering-based routing, filtering is done in the network of brokers.
  - ✓ Brokers forward notifications through the network only where there is a path to a valid subscriber.
  - ✓ This is achieved by propagating subscription information through the network towards potential publishers and then storing associated state at each broker.
  - ✓ Each node maintains a neighbors list containing a list of all connected neighbors in the network of brokers, a subscription list containing a list of all directly connected subscribers serviced by this node, and a routing table

- ✓ When a broker receives a publish request from a given node, it must pass this notification to all connected nodes where there is a corresponding matching subscription and also decide where to propagate this event through the network of brokers.
- ✓ This approach can cause heavy traffic due to propagation of subscription.

➢ **Advertisements:**

- ✓ Here the advertisements are propagated towards subscribers in a symmetrical way.

➢ **Rendezvous:**

- ✓ Here, the set of all possible events are viewed as an event space and responsibility is partitioned for this event space between the set of brokers in the network.
- ✓ This approach defines rendezvous nodes (broker nodes), that are responsible for a given subset of the event space.
- ✓ The rendezvous-based routing algorithm defines two functions.
  - SN(s) takes a given subscription(s), and returns one or more rendezvous nodes that take responsibility for that subscription. Each rendezvous node maintains a subscription list and forwards all matching events to the set of subscribing nodes.
  - When an event e is published, the function EN(e) also returns one or more rendezvous nodes, which are responsible for matching e against subscriptions in the system.
- ✓ Both SN(s) and EN(e) return more than one node if reliability is a concern.
- ✓ This approach works if the intersection of EN(e) and SN(s) is non-empty for a given e that matches s .

Rendezvous-based routing algorithm

```
upon receive publish(event e) from node x at node i
rvlist := EN(e);
if i in rvlistthen begin
matchlist<- match(e, subscriptions);
send notify(e) to matchlist;
end
send publish(e) to rvlist - i;
```

upon receive subscribe(subscription s) from node x at node i

rvlist := SN(s);

if i in rvlistthen

add s to subscriptions;

else

send subscribe(s) to rvlist - i;

2.12 MESSAGE QUEUES

Message Queuing technology enables applications running at different times to communicate across heterogeneous networks and systems that may be temporarily offline. Applications send messages to queues and read messages from queues. Message Queuing provides guaranteed message delivery, efficient routing, security, and priority-based messaging.

It can be used to implement solutions to both asynchronous and synchronous scenarios requiring high performance.

Programming Model

Communication in distributed systems is through queues. The producer processes can send messages to a specific queue and consumer processes can receive messages from this queue. There are three receiving styles

- blocking receive: block until an appropriate message is available
- non-blocking receive: check the status of the queue and return a message as available, or not available.
- Notify operation: issue an event notification when a message is available in the associated queue.

The queue can be build based on FIFO or priority.

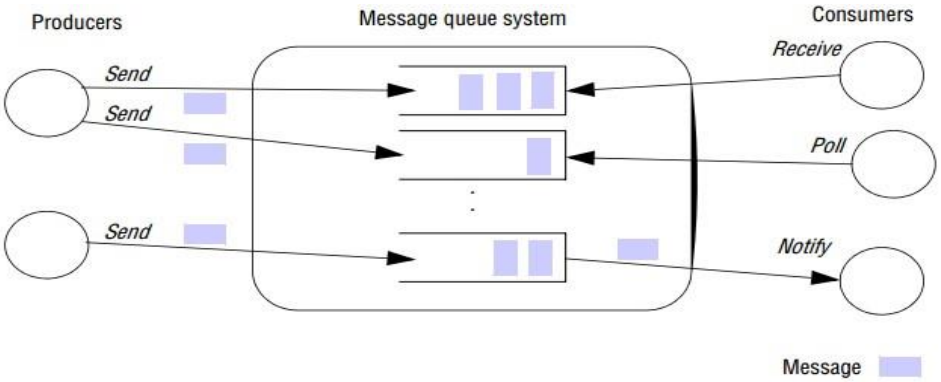


Fig 2.17: Message Queue

Properties of message passing system:

- Reliability
- Persistence
- Validity
- Integrity
- Guaranteed delivery
- Atomicity
- Transformation messages between formats to deal with heterogeneity in underlying data representations.
- Security

Implementation Issues

The important implementation decision for message queuing systems is the choice between centralized and distributed implementations.

WebSphere MQ

- ✓ This is a IBM developed middleware based on the concept of message queues.
- ✓ Queues in WebSphere MQ are managed by **queue managers** which host and manage queues and allow applications to access queues through the **Message Queue Interface**(MQI).
- ✓ This allows applications to carry out operations such as connecting to or disconnecting from a queue (MQCONN and MQDISC) or sending/receiving messages to/from a queue (MQPUT and MQGET).
- ✓ Multiple queue managers can reside on a single physical server. Client applications can also reside on the same physical server.
- ✓ The communication with the queue manager through is known as a **client channel.**
- ✓ MQI commands are issued on the proxy and then sent transparently to the queue manager for execution using RPC.
- ✓ The message channel is a unidirectional connection between two queue managers that is used to forward messages asynchronously from one queue to another.
- ✓ A message channel is managed by a message channel agent (MCA) at each end.
- ✓ Routing tables are also included in each queue manager, and together with channels this allows arbitrary topologies to be created.

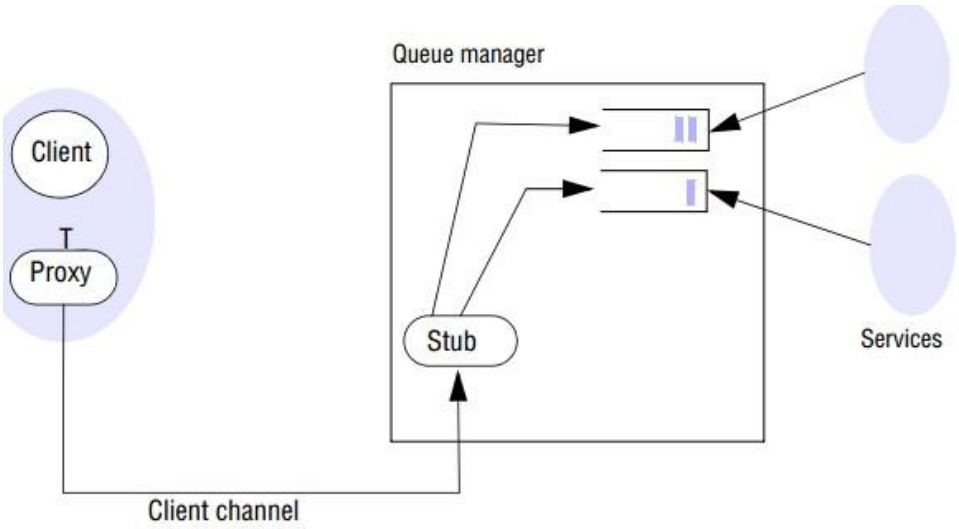


Fig 2.17: Network Topology in Websphere

The hub-and-spoke approach

- ✓ In the hub-and-spoke topology, one queue manager is designated as the hub.
- ✓ Client applications do not connect directly to this hub but rather connect through queue managers called **spokes.**
- ✓ Spokes relay messages to the message queue of the hub for processing by the various services.
- ✓ Spokes are placed strategically around the network to support different clients.
- ✓ The hub is placed on a node with sufficient resources to deal with the volume of traffic.
- ✓ This topology is heavily used with WebSphere MQ for large-scale deployments.
- ✓ The drawback of this architecture is that the hub can be a potential bottleneck and a single point of failure.

2.12.1 Java Messaging Services (JMS)

- ✓ The Java Messaging Service (JMS) is a standardized way for distributed Java programs to communicate indirectly.
- ✓ This unifies the publish-subscribe and message queue paradigms.
- ✓ A JMS client is a Java program or component that produces or consumes messages.

Programming with JMS

- ✓ Fir ough a cor
- ✓ Th
- ✓ Th Queue Co
  - ➢
  - ➢
- ✓ Co
- ✓ A n and cor

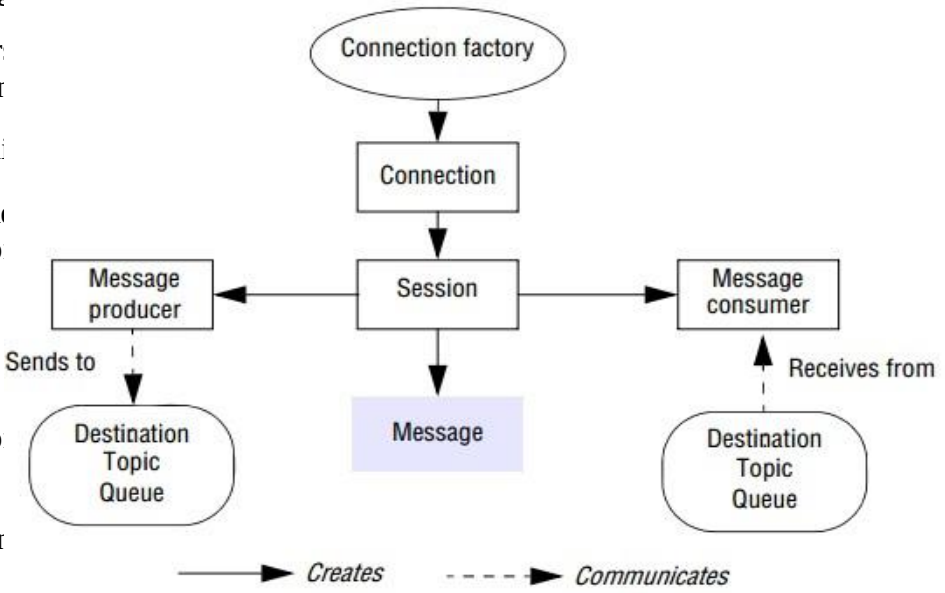


Fig 2.18: JMS Programming Model

Parts of a JMS Message

There are three parts: a header, a set of properties and the body of the message.

- ➢ **Header**
  - ✓ The header contains all the information needed to identify and route the message: destination, priority, expiration date, a message ID and a timestamp.

- ✓ These fields are either created by the underlying system, or constructor methods.
- ✓ The properties can be used to express additional context associated with the message, including a location field.

➢ **Body**

- ✓ The body is opaque and untouched by the system.
- ✓ The body can be any one of a text message, a byte stream, a serialized Java object, a stream of primitive Java values or a more structured set of name/value pairs.
- ✓ A message producer is an object used to publish messages under a particular topic or to send messages to a queue.
- ✓ A message consumer is an object used to subscribe to messages concerned with a given topic or to receive messages from a queue.
- ✓ The consumer is complicated:
  - Consumers apply message filters.
  - Consumer program either can block using a receive operation or it can establish a message listener object that identifies a message.

2.13 SHARED MEMORY APPROACHES

They are indirect communication paradigms that offer an abstraction of shared memory.

Distributed shared memory (DSM)

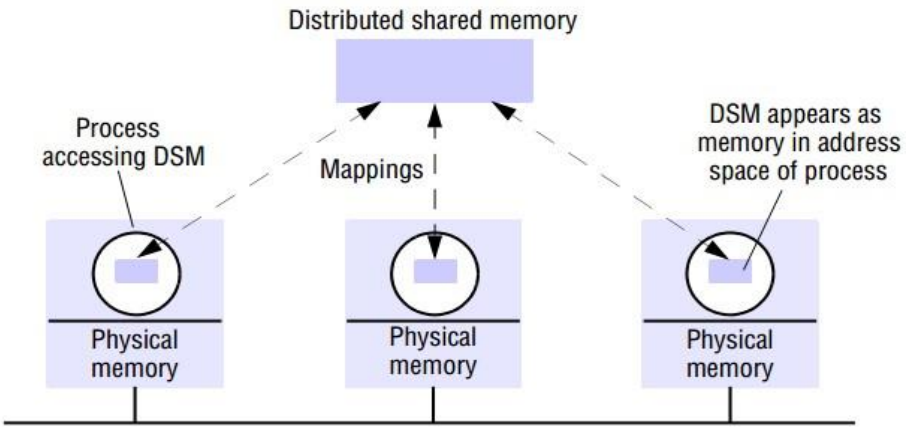


Fig 2.18: Distributed Shared Memory

- ✓ This is an abstraction used for sharing data between computers that do not share physical memory.
- ✓ Processes access DSM by reads and updates to what appears to be ordinary memory within their address space.
- ✓ The programmer need not worry about the message passing.
- ✓ DSM runtime support has to send updates in messages between computers.
- ✓ DSM systems must also manage replicated data.
- ✓ Each computer has a local copy of recently accessed data items stored in DSM, for speed of access.

Message passing versus DSM

The message passing and DSM can be compared based on services they offer and in terms of their efficiency.

Message Passing	**Distributed Shared Memory**
Services Offered: Variables have to be marshalled from one process, transmitted and unmarshalled into other variables at the receiving process.	The processes share variables directly, so no marshalling and unmarshalling. Shared variables can be named, stored and accessed in DSM.
Processes can communicate with other processes. They can be protected from one another by having private address spaces.	Here, a process does not have private address space. So one process can alter the execution of other.
This technique can be used in heterogeneous computers.	This cannot be used to heterogeneous computers.
Synchronization between processes is through message passing primitives.	Synchronization is through locks and semaphores.
Processes communicating via message passing must execute at the same time.	Processes communicating through DSM may execute with non-overlapping lifetimes.
Efficiency: All remote data accesses are explicit and therefore the programmer is always aware of whether a particular operation is in-process or involves the expense of communication.	Any particular read or update may or may not involve communication by the underlying runtime support.

2.13.1 Tuple space communication

Processes communicate indirectly by placing tuples in a tuple space, from which other processes can read or remove them.

The programming model

- ✓ Processes communicate through a shared collection of tuples.
- ✓ Any combination of types of tuples may exist in the same tuple space.
- ✓ Processes share data by accessing the same tuple space.
- ✓ The tuples are stored in tuple space using the write operation and read or extract them from tuple space using the reador take operation.
- ✓ No direct access to tuples in tuple space is allowed and processes have to replace tuples in the tuple space instead of modifying them. Thus, tuples are immutable.

Properties of tuple space

- ➢ **Space uncoupling:** A tuple placed in tuple space may originate from any number of sender processes and may be delivered to any one of a number of potential recipients.
- ➢ **Time uncoupling:** A tuple placed in tuple space will remain in that tuple space until removed and hence the sender and receiver do not need to overlap in time.

Implementation Issues

Centralized solution where the tuple space resource is managed by a single server is a common implementation.

Replication:

- ✓ A tuple space behaves like a state machine, maintaining state and changing this state in response to events received from other replicas or from the environment.
- ✓ To ensure consistency the replicas
  - (ii) Must start in the same state
  - (iii) must execute events in the same order
  - (iv) must react deterministically to each event.
- ✓ Alternatively, a multicast algorithm can be used.
- ✓ Here, updates are carried out in the context of the agreed set of replicas and tuples are also partitioned into distinct tuple sets based on their associated logical names.
- ✓ The system consists of a set of workers carrying out computations on the tuple space, and a set of tuple space replicas.

- ✓ A given physical node can contain any number of workers, replicas or indeed both; a given worker therefore may or may not have a local replica.
- ✓ Nodes are connected by a communications network that may lose, duplicate or delay messages and can deliver messages out of order.
- ✓ A write operation is implemented by sending a multicast message over the unreliable communications channel to all members of the view.
- ✓ The write request is repeated until all acknowledgements are received.
- ✓ The read operation consists of sending a multicast message to all replicas.

Write

1. The requesting site multicasts the write request to all members of the view;
2. On receiving this request, members insert the tuple into their replica and acknowledge this action;
3. Step 1 is repeated until all acknowledgements are received.

Read

1. The requesting site multicasts the read request to all members of the view;
2. On receiving this request, a member returns a matching tuple to the requestor;
3. The requestor returns the first matching tuple received as the result of the operation (ignoring others);
4. Step 1 is repeated until at least one response is received.

Phase 1: Selecting the tuple to be removed

1. The requesting site multicasts the take request to all members of the view;
2. On receiving this request, each replica acquires a lock on the associated tuple set and, if the lock cannot be acquired, the take request is rejected;
3. All accepting members reply with the set of all matching tuples;
4. Step 1 is repeated until all sites have accepted the request and responded with their set of tuples and the intersection is non-null;
5. A particular tuple is selected as the result of the operation .
6. If only a minority accept the request, this minority are asked to release their locks and phase 1 repeats.

Phase 2: Removing the selected tuple

1. The requesting site multicasts a remove request to all members of the view citing the tuple to be removed;

2. On receiving this request, members remove the tuple from their replica, send an acknowledgement and release the lock;
3. Step 1 is repeated until all acknowledgements are received

2.14 DISTRIBUTED OBJECTS

Distributed object communication realizes communication between distributed objects in the distributed computing environment. The main role is to interconnect objects residing in non-local memory space and allow them to perform remote calls and exchange data. Distributed object middleware offers a programming abstraction based on object oriented principles.

The following table gives a brief description about relation between objects and distributed objects:

Objects	**Distributed Objects**	**Descriptions**
Object references	Remote object references	Unique reference for each distributed object.
Interfaces	Remote interfaces	Provides methods to be invoked o remote object using DLL.
Actions	Distributed actions	Invocation of methods
Exceptions	Distributed Exception	Includes exceptions generated in distributed environment
Garbage Collection	Distributed Garbage Collection	Implements distributed garbage collection algorithm

Differences between objects and distributed objects

Objects	**Distributed Objects**
Class is a fundamental concept in object-oriented languages.	Classes are not prominent here.
In the object oriented world, class can be the description of the behavior associated with a group of objects or, the place to go to instantiate an object with a given behavior or even the group of objects that adhere to that behavior.	Here the term „class" is avoided, more specific terms such as „factory" and „template" are readily used.
Object oriented languages offer implementation inheritances	This offers interface inheritances.

Additional Complexities

- **Inter-object communication:**
  - ✓ This is normally provided by remote method invocation.
- **Lifecycle management**:
  - ✓ Lifecycle management is concerned with the creation, migration and deletion of objects, with each step having to deal with the distributed nature of the underlying environment.
- **Activation and deactivation:**
  - ✓ In distributed systems, the numbers of objects may be very large, and hence it would be waste of resources to have all objects available (active) at any time.
  - ✓ Activation is the process of making an object active in the distributed environment by providing the necessary resources for it to process incoming invocations.
  - ✓ Deactivation is rendering an object temporarily unable to process invocations.
- **Persistence:**
  - ✓ Objects have state, and it maintains this state across possible cycles of activation and deactivation and indeed system failures.
  - ✓ Distributed object middleware must therefore offer persistency management for stateful objects.
- **Additional services:**
  - ✓ This must provide support for the range of distributed system services considered in this book, including naming, security and transaction services.

2.15 FROM OBJECTS TO COMPONENTS

Component based computing is out powering distributed objects.

Problems in object-oriented middleware

The following four problems in object oriented middleware led to the development of component based technology:

- **Implicit dependencies:**
  - ✓ A distributed object communicates with outside world through interfaces.
  - ✓ The interfaces provide a complete contract for the deploying and use of this object.

- ✓ The problem in distributed objects is that the internal behavior of an object is hidden.
- ✓ Implicit dependencies in the distributed configuration are not safe.
- ✓ The third-party developers to implement one particular element in a distributed configuration.

  Requirement: Dependencies of the object with otherobjects in the distributed configuration must be specified.

➢ **Interaction with the middleware:**

- ✓ The distributed objects must interact even with low-levelmiddleware architecture.

  Requirement: While programming distributed applications, a clean separation must be made between code related to middleware framework and code associated with the application.

➢ **Lack of separation of distribution concerns:**

- ✓ Distributed programmers must also focus on issues related to security, transactions, coordination and replication.
- ✓ To provide the non functional requirements :
  - Programmers must have knowledge of the full details of all the associated distributed system services.
  - The implementation of an object will contain calls to distributed system services and to middleware interfaces. This increases the programming complexity.

  Requirement: These complexities should be hidden from the programmer.

➢ **No support for deployment:**

- ✓ Objects must be deployed manually on individual machines and activated when required.
- ✓ This is a tiresome and error-prone process.

  Requirement: Middleware platforms should support deployment.

2.15.1 Fundamentals of Components

A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only.

The term contract refers to:

- a set of provided interfaces
- a set of required interfaces

Every required interface must be bound to a provided interface of another component. This is called **software architecture.** This consists of components, interfaces and connections between interfaces.

Programming in component-based systems is concerned with the development of components and their composition. So third party development of software components is possible in **component based technology (CBT).**

Advantages of CBT

- ✓ Independent extensions
- ✓ Component Market
- ✓ Component models lessen unanticipated interactions between components
- ✓ Reduced time to market
- ✓ Reduced Costs

Disadvantages of CBT:

- ✓ Time to develop software components takes a big effort.
- ✓ Components can be pricey.
- ✓ Requirements in component technologies lacking
- ✓ Conflict between usability and reusability of components.
- ✓ Maintenance cost for components increased

Components and distributed systems Containers:

- ✓ The containers are absolutely central to component-based middleware.
- ✓ Containers support a common pattern often encountered in distributed applications, which consists of:
    - a front-end client
    - a container holding one or more components that implement the application or business logic
    - system services that manage the associated data in persistent storage.

- ✓ The containers provide a managed server-side hosting environment for components.
- ✓ The components deal with application concerns and the container deals with distributed systems and middleware issues.
- ✓ A container includes a number of components that require the same configuration in terms of distributed system support.

Support for Deployment

- ✓ Component-based middleware supports the deployment of component configurations.
- ✓ They include releases of software architectures (components and their interconnections) together with deployment descriptors.
- ✓ These descriptors describe how the configurations should be deployed in a distributed environment.
- ✓ Deployment descriptors are typically written in XML and include sufficient information to ensure that:
    - o components are correctly connected using appropriate protocols and associated middleware support
    - o the underlying middleware and platform are configured to provide the right level of support to the component configuration
    - o the associated distributed system services are set up to provide the right level of security, transaction support and so on.

2.16 ENTERPRISE JAVA BEANS

- ✓ Enterprise JavaBeans (EJB) is a specification of a server-side, managed component architecture
- ✓ It supports the development of the classic style of application, where potentially large numbers of clients interact with a number of services realized through components or configuration of components.
- ✓ The components, which are known as beans in EJB, are intended to capture the application (or business) logic
- ✓ The container in EJB issues calls to the associated services to provide the required properties.
- ✓ Beans act as:
    - ➢ Container Managed: The transaction manager or security services is completely hidden from the developer of the associated beans.

- Bean Managed: The bean developer takes control over these operations not the transaction manager.

✓ The EJB specification has the following roles:

- bean provider: develops the application logic of the component(s)
- application assembler: assembles beans into application configurations
- deployer: takes a given application assembly and ensures it can be correctly deployed in a given operational environment;
- service provider: does transaction management
- persistence provider: maps persistent data to databases
- container provider: build on the above two roles and is responsible for configuring containers with the required level of distributed systems support in terms of non-functional properties
- system administrator: responsible for monitoring a deployment at runtime and making any adjustments to ensure its correct operation. **The EJB component model**

✓ A bean in EJB is a component offering one or more business interfaces to potential clients of that component.

✓ A bean is represented by the set of remote and local business interfaces together with an associated bean class that implements the interfaces.

✓ Two main styles of bean are supported in the EJB specification:

- **Session beans:** A session bean is a component implementing a particular task within the application logic of a service.
- **Message-driven beans:** Clients interact with session beans using local or remote invocation.

POJOs and annotations

The task of programming in EJB has been simplified significantly through the use of Enterprise JavaBean POJOs(plain old Java objects). It consists of the application logic written simply in Java with no other code relating to it being a bean. Annotations are then used to ensure the correct behaviour in the EJB context. **Enterprise JavaBean containers in EJB**

✓ Beans are deployed to containers, and the containers provide implicit distributed system management.

✓ The container provides the policies in areas including transaction management, security, persistence and lifecycle management.

- ✓ The developer needs to focus only on application logic.
- ✓ EJB manages transactions.
- ✓ Transactions are sequences of operations, and that the sequences must be identified by the transaction manager.
- ✓ The first thing to declare is whether transactions associated with an enterprise bean should be bean-managed or container-managed. @TransactionManagement (BEAN)@TransactionManagement (CONTAINER)
- ✓ The bean managed transaction uses the following methods:
    - ➢ javax.transaction.UserTransaction– the User.Transaction.begin
    - ➢ UserTransaction.commitmethods – within the code of the bean.

Transaction Attributes in EJB

Attribute	**Policy**
REQUIRED	If the client has an associated transaction running, execute within this transaction; otherwise, start a new transaction.
REQUIRES_NEW	Always start a new transaction for this invocation.
SUPPORTS	If the client has an associated transaction, execute the method within the context of this transaction; if not, the call proceeds without any transaction support.
NOT_SUPPORTED	If the client calls the method from within a transaction, then this transaction is suspended before calling the method and resumed afterwards – that is, the invoked method is excluded from the transaction.
MANDATORY	The associated method must be called from within a client transaction; if not, an exception is thrown.
NEVER	The associated methods must not be called from within a client transaction; if this is attempted, an exception is thrown

Dependency injection:

- ✓ In dependency injection a third party (container), is responsible for managing and resolving the relationships between a component and its dependencies.
- ✓ A component refers to a dependency using an annotation and the container is responsible for resolving this annotation and ensuring that, at runtime, the associated attribute refers to the right object.
- ✓ This is typically implemented by the container usingreflection.

Enterprise JavaBean Interception

The Enterprise JavaBeans allows interception of two types of operation to alter their default behavior:

- method calls associated with a business interface
- life cycle events.

Invocation Contexts

Method	**Description**
public Object getTarget()	Returns the bean instance associated with the incoming invocation or event
public Method getMethod()	Returns the method being invoked
public Object[] getParameters()	Returns the set of parameters associated with the intercepted business method
public void setParameters (Object[] params)	Allows the parameter set to be altered by theinterceptor, assuming type correctness ismaintained
public Object proceed() throws Exception	Execution proceeds to next interceptor in the chain (if any) or the method that has been intercepted

REVIEW QUESTIONS

PART-A

1. What is IPC?

Interprocess communication (IPC) is a set of programming interfaces that allows a programmer to coordinate activities among different program processes that can run concurrently in an operating system.

2. List the models in distributed system.

- Physical models: It is the most explicit way in which to describe a system in terms of hardware composition.
- Architectural models: They describe a system in terms of the computational and communication tasks performed by its computational elements.
- Fundamental models: They examine individual aspects of a distributed system. They are again classified based on some parameters as follows:
  - ✓ Interaction models: This deals with the structure and sequencing of the communication between the elements of the system
  - ✓ Failure models: This deals with the ways in which a system may fail to operate correctly
  - ✓ Security models: This deals with the security measures implemented in the system against attempts to interfere with its correct operation or to steal its data.

3. Define ULS.

The ultra large scale (ULS) distributed systems is defined as a complex system consisting of a series of subsystems that are systems in their own right and that come together to perform a particular task or tasks.

4. What are the characteristics of ULS?

- very large size
- global geographical distribution
- operational and managerial independence of their member systems.

5. What are the evaluation elements of architectural model?

Architectural elements, architectural patterns and middleware platforms.

6. What are the architectural entities?

– Communicating entities (objects, components and web services);

– Communication paradigms (inter process communication, remote invocation and indirect communication);

– Roles responsibilities and placement

7. What are the communicating entities in architectural model?

↑ Objects: They are used to implement object oriented approaches in distributed systems. Objects are accessed through interfaces.

↑ Components: Components resemble objects and they offer problem-oriented abstractions for building distributed systems. They are also accessed through interfaces. The key difference between component and object is that a components holds all the assumptions specified to other components and their interfaces in the system. Components and objects are used to develop tightly coupled applications.

↑ Web services: Web services are closelyrelated to objects and components. Web services areintegrated into the World Wide Web. They are partially defined by the web-based technologies theyadopt. Web services are generally viewed as complete servicesthat can be combined to achieve value-added services across organizational boundaries.

8. List the communication mechanisms in DS.

IPC, Remote invocation and indirect communication.

9. Give the features of IPC.

This has the following two key features:

✓ Space uncoupling: Senders do not need to know who they are sending to

✓ Time uncoupling: Senders and receivers do not need to exist at the same time

10. What is Group communication?

Group communication is concerned with the delivery of messages to a set of recipients and hence is a multiparty communication paradigm supporting one-to-many communication. A group identifier uniquely identifies the group. The recipients join the group and receive the messages. Senders send messages to the group based on the group identifier and hence do not need to know the recipients of the message.

11. What are Publish-subscribe systems?

This is a messaging pattern where senders of messages, called publishers, do not program the messages to be sent directly to specific receivers, called subscribers. Instead, published messages are characterized into classes, without knowledge of what, subscribers there may be. Similarly, subscribers express interest in one or more classes, and only receive messages that are of interest, without knowledge of what, if any, publishers there are. They offer one-to-many style of communication.

12. What are Message queues?

They offer a point-to-point service whereby producer processes can send messages to a specified queue and consumer processes can receive messages from the queue or be notified of the arrival of new messages in the queue. Queues therefore offer an indirection between the producer and consumer processes.

13. What are Tuple spaces?

The processes can place arbitrary items of structured data, called tuples, in a persistent tuple space and other processes can either read or remove such tuples from the tuple space by specifying patterns of interest. This style of programming is known as generative communication.

14. Write about Distributed shared memory.

In computer architecture, distributed shared memory (DSM) is a form of memory architecture where the (physically separate) memories can be addressed as one (logically shared) address space. Here, the term shared does not mean that there is a single centralized memory but shared essentially means that the address space is shared (same physical address on two processors refers to the same location in memory

15. What are the architectures based on roles and responsibilities?

Client server and peer to peer.

16. What are the strategies for placement of objects?

- Mapping of services to multiple servers
- Caching
- Mobile code
- Mobile agents

17. What is mobile code?

Applets are a well-known and widely used example of mobile code. The user running a browser selects a link to an applet whose code is stored on a web server; the code is downloaded to the browser and runs. They provide interactive response.

18. What are mobile agents?

A mobile agent is a running program (including both code and data) that travels from one computer to another in a network carrying out a task on someone‟s behalf, such as collecting information, and eventually returning with the results. A mobile agent is a complete program, code + data, that can work (relatively) independently.

19. What is layering?

In a layered approach, a complex system is partitioned into a number of layers, with a given layer making use of the services offered by the layer below.

20. Define tiering.

Tiering is a technique to organize functionality of a given layer and place this functionality into appropriate servers and, as a secondary consideration, on to physical nodes.

21. Write about two tier architecture.

The two tiered architecture refers to client/server architectures in which the user interface (presentation layer) runs on the client and the database (data layer) is stored on the server. The actual application logic can run on either the client or the server.

22. Write about three tier architecture.

Presentation tier: This is the topmost level of the application. is concerned with handling user interaction and updating the view of the application as presented to the user;

Application tier (business logic, logic tier, or middle tier): The logical tier is pulled out from the presentation tier and, as its own layer, it controls an application‟s functionality by performing detailed processing.

Data tier: The data tier includes the data persistence mechanisms (database servers, file shares, etc.) and the data access layer that encapsulates the persistence mechanisms and exposes the data.

23. Define thin client.

In a thin-client model, all of the application processing and data management is carried out on the server. The client is simply responsible for running the responsible for running the presentation software. This is used when legacy systems are migrated to client server architectures.

24. What is VNC?

The virtual network computing (VNC) has emerged to overcome the disadvantages of thin clients. VNC is a type of remote-control software that makes it possible to control another computer over a network connection.

25. What is Proxy?

- This facilitates location transparency in remote procedure calls or remote method invocation.
- A proxy is created in the local address space to represent the remote object with same interface as the remote object.
- The programmer makes calls on this proxy object and he need not be aware of the distributed nature of the interaction.

26. Define Brokerage.

- It is used to bring interoperability in potentially complex distributed infrastructures.
- The service broker is meant to be a registry of services, and stores information about what services are available and who may use them.

27. What is reflection?

- The Reflection architectural pattern provides a mechanism for changing structure and behaviour of software systems dynamically.
- It supports the modification of fundamental aspects, such as type structures and function call mechanisms.

28. What are the levels in reflection?

Meta Level: This provides information about selected system properties and makes the software self-aware.

Base level: This includes the application logic. Its implementation builds on the meta level. Changes to information kept in the meta level affect subsequent base-level behavior.

29. Define middleware.

Middleware is a general term for software that serves to "glue together" separate, often complex and already existing, programs.

30. What are Distributed Objects?

The term distributed objects usually refers to software modules that are designed to work together, but reside either in multiple computers connected via a network or in different processes inside the same computer. One object sends a message to another object in a remote machine or process to perform some task. The results are sent back to the calling object.

31. What are Distributed Components?

A component is a reusable program building block that can be combined with other components in the same or other computers in a distributed network to form an application. Examples: a single button in a graphical user interface, a small interest calculator, an interface to a database manager. Components can be deployed on different servers in a network and communicate with each other for needed services. A component runs within a context called a container . Examples: pages on a Web site, Web browsers, and word processors.

32. Write about Publish subscriber model.

Publish–subscribe is a messaging pattern where senders of messages, called publishers, do not program the messages to be sent directly to specific receivers, called

subscribers. Instead, published messages are characterized into classes, without knowledge of what, if any, subscribers there may be. Similarly, subscribers express interest in one or more classes, and only receive messages that are of interest, without knowledge of what publishers are there.

33. Write about Message Queues.

Message queues provide an asynchronous communications protocol, meaning that the sender and receiver of the message do not need to interact with the message queue at the same time. Messages placed onto the queue are stored until the recipient retrieves them. Message queues have implicit or explicit limits on the size of data that may be transmitted in a single message and the number of messages that may remain outstanding on the queue.

34. Define Web services.

A web service is a collection of open protocols and standards used for exchanging data between applications or systems. Software applications written in various programming languages and running on various platforms can use web services to exchange data over computer networks like the Internet in a manner similar to inter-process communication on a single computer. This interoperability (e.g., between Java and Python, or Windows and Linux applications) is due to the use of open standards.

35. What is Peer to peer computing?

Peer-to-peer (P2P) computing or networking is a distributed application architecture that partitions tasks or work load between peers. Peers are equally privileged, equipotent participants in the application. They are said to form a peer-to-peer network of nodes.

36. List the advantages of middleware

- ✓ Real time information access among systems.
- ✓ Streamlines business processes and helps raise organizational efficiency.
- ✓ Maintains information integrity across multiple systems.
- ✓ It covers a wide range of software systems, including distributed Objects and components, message-oriented communication, and mobile application support.
- ✓ Middleware is anything that helps developers create networked applications.

37. List the disadvantages of middleware

- ✓ Prohibitively high development costs.
- ✓ There are only few people with experience in the market place to develop and use a middleware.

- ✓ There are only few satisfying standards.
- ✓ The tools are not good enough.
- ✓ Too many platforms to be covered.
- ✓ Middleware often threatens the real-time performance of a system.
- ✓ Middleware products are not very mature.

38. Define fundamental models.

Fundamental Models are concerned with a formal description of the properties that are common in all of the architectural models.

39. What are the types of fundamental models?

Interaction Model – deals with performance and the difficulty of setting of time limits in a distributed system.

Failure Model – specification of the faults that can be exhibited by processes

Secure Model – discusses possible threats to processes and communication channels.

40. Give the purpose of the fundamental model.

- ✓ Make explicit all the relevant assumptions about the systems we are modelling.
- ✓ Make generalizations (general purpose algorithms) concerning what is possible or impossible with given assumptions.

41. What are the communication performance metrics?

a. Latency: A delay between the start of a message‟s transmission from one process to the beginning of reception by another.

b. Bandwidth: The total amount of information that can be transmitted over in a given time. The communication channels using the same network, have to share the available bandwidth.

c. Jitter: The variation in the time taken to deliver a series of messages. It is very relevant to multimedia data.

42. What is synchronous DS?

- This is practically hard to achieve in real life.
- In synchronous DS the time taken to execute a step of a process has known lower and upper bounds.

- Each message transmitted over a channel is received within a known bounded time.
- Each process has a local clock whose drift rate from real time has known bound.

43. What is Asynchronous DS?

- There are no bounds on: process execution speeds, message transmission delays and clock drift rates.

44. What are the common types of failures?

- Omission Failure
- Arbitrary Failure
- Timing Failure

45. What is Omission failure?

Omission failures occur due to communication link failures. They are detected through timeouts.

46. **Write about Process omission failures**.

- ✓ Omission failures that occur due to process crash (i.e.) the execution of the process could not continue.
- ✓ This is detected using timeouts.
- ✓ A fixed period of time is fixed for all the methods to complete its execution. If the method takes time longer than the allowed time, a time out has occurred.
- ✓ In an asynchronous system a timeout can indicate only that a process is not responding.
- ✓ A process crash is called fail-stopif other processes can detect certainly that the process has crashed.
- ✓ Fail-stop behavior can be produced in a synchronous system if the processes use timeouts to detect when other processes fail to respond and messages are guaranteed to be delivered.

47. Write about Communication omission failures.

- ✓ This occurs when the messages are dropped between sender and the receiver.
- ✓ The message can be lost in sender buffer, receiver buffer or even in the communication channel.

48. What is send omission failure?

The loss of messages between the sending process and the outgoing message buffer is known as send omission failures.

49. What is receive-omission failure?

The loss of messages between the incoming message buffer and the receiving process as receive-omission failures.

50. What is channel omission failure?

The loss of messages in a channel is channel omission failure.

51. What are Arbitrary failures (Byzantine failure)?

This includes all possible errors that could cause failure.

52. Classify the arbitrary failures.

Class	Affects	Comments
Fail stop	Process	Process halts and remains halted. Other processes may detect that this process has halted.
Crash	Process	Process halts and remains halted. Other processes may not detect that this process has halted.
Omission	Channel	A message inserted in an outgoing message buffer never arrives at the other end‟s incoming messagebuffer.
Send-omission	Process	A process completes a send operation but the message is not put in its outgoing message buffer.
Receiveomission	Process	A message is put in a process‟s incoming messagebuffer, but that process does not receive it.
Arbitrary	Process or channel	Process/channel exhibits arbitrary behaviour: it may send/transmit arbitrary messages at arbitrary times or commit omissions; a process may stop or take an incorrect step.

53. Define timing failure.

Timing failures refer to a situation where the environment in which a system operates does not behave as expected regarding the timing assumptions, that is, the timing constraints are not met.

54. Define reliable communication.

The term reliable communication is defined in terms of validity and integrity.

- Validity: Any message in the outgoing message buffer is eventually delivered to the incoming message buffer.
- Integrity: The message received is identical to one sent, and no messages are delivered twice.

55. What is security model?

The security of a DS can be achieved by securing the processes and the channels used in their interactions and by protecting the objects that they encapsulate against unauthorized access.

56. What is DOS attack?

Denial of service (DoS) attack is an incident in which a user or organization is deprived of the services of a resource they would normally expect to have. In a distributed denial-of-service, large numbers of compromised systems (sometimes called a botnet) attack a single target.

57. Define IPC.

Interprocess communication (IPC) is a set of programming interfaces that allow a programmer to coordinate activities among different program processes that can run concurrently in an operating system.

58. What are the characteristics of IPC?

Characteristics of IPC:

- Synchronous and asynchronous communication
- Message destinations
- Reliability
- Ordering

59. Define socket.

A socket is one endpoint of a two-way communication link between two programs unning on the network.

60. What are the UDP failure modes?

UDP datagrams suffer from following failures:

- Omission failure: Messages may be dropped occasionally
- Ordering: Messages can be delivered out of order.

61. List the issues in stream communication.

Matching of data items: Two communicating processes need to agree as to the contents of the data transmitted over a stream.

Blocking: The process that writes data to a stream may be blocked by the TCP flow-control mechanism if the socket at the other end is queuing as much data as the protocol allows.

Threads: When a server accepts a connection, it generally creates a new thread in which to communicate with the new client. In an environment in which threads are not provided, an alternative is to test whether input is available from a stream before attempting to read it;

62. Define external data representation.

External Data Representation is an agreed standard for the representation of data structures and primitive values.

63. Define marshalling and unmarshalling.

Marshalling is the process of taking a collection of data items and assembling them into a form suitable for transmission in a message. Unmarshalling is the process of disassembling a collection of data on arrival to produce an equivalent collection of data items at the destination.

64. Define reflection.

Reflection is inquiring about class properties, e.g., names, types of methods and variables, of objects

65. List the requirements of remote object reference.

The following are mandatory for remote object reference

- internet address/port number: process which created object
- time: creation time
- object number: local counter, incremented each time an object is created in the creating process
- interface: how to access the remote object (if object reference is passed from one client to another)

66. Define multi cast communication.

Multicast (one-to-many or many-to-many distribution) is group communication where information is addressed to a group of destination computers simultaneously.

67. List the characteristics of multicasting.

The following are the characteristics of multicasting:

- Fault tolerance based on replicated services:
- Finding the discovery servers in spontaneous networking
- Better performance through replicated data
- Propagation of event notifications

68. What is IP multicast?

IP multicast is built on top of the Internet protocol. IP multicast allows the sender to transmit a single IP packet to a multicast group. A multicast group is specified by class D IP address with1110 as it starting byte.

69. List some methods in MulticastSocket class.

- getInterface():Retrieve the address of the network interface used for multicast packets.
- getTTL():Get the default time-to-live for multicast packets sent out on the socket.
- joinGroup(InetAddress): Joins a multicast group.
- leaveGroup(InetAddress):Leave a multicast group.
- send(DatagramPacket, byte):Sends a datagram packet to the destination, with a TTL (time- to-live) other than the default for the socket.
- setInterface(InetAddress): Set the outgoing network interface for multicast packets on this socket, to other than the system default.
- setTTL(byte): Set the default time-to-live for multicast packets sent out on this socket.

70. Define network virtualization.

Network virtualization refers to the management and monitoring of an entire computer network as a single administrative entity from a single software-based administrator‟s console.

71. Define overlay network.

An overlay network is a virtual network of nodes and logical links that is built on top of an existing network with the purpose to implement a network service that is not available in the existing network.

72. List the Advantages of overlay network.

- ✓ They enable new network services to be defined without requiring changes to the underlying network.
- ✓ They encourage experimentation with network services and the customization of services to particular classes of application.
- ✓ Multiple overlays can be defined and can coexist, with the end result being a more open and extensible network architecture.

73. List the disadvantages of overlay network.

- ✓ They introduce an extra level of indirection.
- ✓ They add to the complexity of network services

74. Define message passing interface.

Message Passing Interface (MPI) is a standardized and portable message-passing system to function on a wide variety of parallel computers.

75. What are the types of message passing?

- ➢ Synchronous message passing: This is implemented by blocking send and receive calls.
- ➢ Asynchronous message passing: This is implemented by a non-blocking form of send.

76. What is remote invocation?

Remote invocations are method invocations for objects in different processes.

77. What are remote objects?

Remote objects are objects that can receive remote invocation. The core part of the distributed object model is:

- remote object reference;
- remote interface: specifies which methods can be invoked remotely;

78. Give the advantages of UDP over TCP.

UDP is preferred over TCP for the following reasons:

- ♦ Acknowledgements are redundant, since requests are followed by replies.
- ♦ Establishing a connection involves two extra pairs of messages in addition to the pair required for a request and a reply.
- ♦ Flow control is redundant for the majority of invocations, which pass only small arguments and results.

79. What is message id?

Each message have a unique message identifier for referencing it. A message identifier consists of two parts:

- request Id: Increasing sequence of integers assigned by the sending process. This makes the message unique to the sender.
- an identifier: for the sender process. This makes the message unique in the distributed system.

80. What is history of messages?

History is used to refer to a structure that contains a record of reply messages that have been transmitted by the server.

81. List the styles of exchange protocols.

There are three exchange protocols:

- request (R) protocol: a single request message is sent by the client to the server
- request-reply (RR) protocol: used in client-server exchanges because it is based on the request-reply protocol. Special acknowledgement messages are not required.
- request-reply-acknowledge reply (RRA) protocol: It is based on request-reply-acknowledge reply. The Acknowledge reply message contains the requested from the reply message being acknowledged. This will enable the server to discard entries from its history. The arrival of a requested in an acknowledgement message will be interpreted as acknowledging the receipt of all reply messages with lower request Ids.

82. Give the differences between persistent and non persistent connections

Persistent Connection	Non persistent Connection
On the same TCP connection the server, parses request, responds and waits for new requests.	The server parses request, responds, and closes TCP connection.
It takes fewer RTTs to fetch the objects.	It takes 2 RTTs to fetch each object.
It has less slow start.	Each object transfer suffers from slow start

83. Define RPC.

Remote Procedure Call (RPC) is a protocol that one program can use to request a service from a program located in another computer in a network without having to understand network details.

84. List the design issues of RPC.

The following are the design issues in RPC:

- Programming with interfaces
- Interfaces in distributed systems
- Interface definition languages:
- RPC call semantics

85. Define stub.

A stub is a piece of code that is used to convert parameters during a remote procedure call (RPC). An RPC allows a client computer to remotely call procedures on a server computer.

86. Define RMI.

RMI is a set of protocols being developed by Sun's JavaSoft division that enables Java objects to communicate remotely with other Java objects.

87. List the differences between RMI and RPC

RMI	RPC
RMI uses an object oriented paradigm where the user needs to know the object and the method of the object he needs to invoke.	RPC is not object oriented and does not deal with objects. Rather, it calls specific subroutines that are already established
With RPC looks like a local call. RPC handles the complexities involved with passing the call from the local to the remote computer.	RMI handles the complexities of passing along the invocation from the local to the remote computer. But instead of passing a procedural call, RMI passes a reference to the object and the method that is being called.

88. Give the Limitations of RMI

- ✓ RMI is not language independent. It is limited only to Java.
- ✓ Interfaces to remote objects are defined using ordinary Java interfaces not with special IDLs.

89. Give the differences between Interfaces and Class

Interfaces	Class
Interfaces cannot be instantiated.	Classes can be instantiated.
They do not contain constructors.	They can have constructors.
They cannot have instance fields (i.e.) all the fields in an interface must be declared both static and final.	No constraints over the fields.
The interface can contain only abstract method.	They can contain method implementation.
Interfaces are implemented by a class not extended.	Classes are extended by other classes.

90. What is remote object reference?

A remote object reference is an identifier that can be used throughout a distributed system to refer to a particular unique remote object.

91. Give the differences between remote and local objects

Remote Objects	Local Objects
Creating a remote object requires creating a stub and a skeleton, which will cost more time cycles.	They consume less time cycles.
Marshalling and unmarshalling is required.	Marshalling and unmarshalling is not required.
Longer time is taken by the remote object to return to the calling program.	Comparatively shorter return time.

92. What is reference counting?

Distributed garbage collection is usually based on reference counting. The total number of references to the object is maintained by a *reference count* field.

93. Give the use of Proxy in RMI.

The class of the proxy contains a method corresponding to each method in the remote interface which marshalls arguments and unmarshalls results for that method. It hides the details of the remote object reference. Each method of the proxy marshals a reference to the target object using a request message (operationId, arguments). When the reply is received the proxy, unmarshals it and returns the results to the invoker.

94. Give the use of Dispatcher in RMI.

Server has a dispatcher & a skeleton for each class of remote object. The dispatcher receives the incoming message, and uses its method info to pass it to the right method in the skeleton.

95. Give the use of Skeleton in RMI.

Skeleton implements methods of the remote interface to unmarshall arguments and invoke the corresponding method in the servant. It then marshalls the result (or any exceptions thrown) into a reply message to the proxy.

96. What are factory methods?

Factory methods are static methods that return an instance of the native class.Factory method is used to create different object from factory often refereed as Item and it encapsulate the creation code.

97. **List the responsibilities of activator**.

Responsibilities of an activator:

- ✓ Registering passive objects that are available for activation. This is done by mapping the names of servers and passive objects.
- ✓ Starting named server processes and activating remote objects in them.
- ✓ Tracking the locations of the servers for remote objects that it has alreadyactivated.
- ✓ Java RMI provides the ability to make some remote objects activatable [java.sun.comIX].

98. Define persistent object.

An object that is guaranteed to live between activations of processes is called a persistent object.

99. What is Jini?

Jini provides a mechanism for locating services on the network that conform to a particular (Java) interface, or that have certain attributes associated with them. Once a service is located, the client can download an implementation of that interface, which it then uses to communicate with the service.

100. What is callback?

Callbackrefers to a server‟s action of notifying clients about an event.

101. Define group communication.

Group communication is a service where a message is sent to a group and then this message is delivered to all members of the group.

102. What is publish subscribe model?

Publish–subscribe is a messaging pattern where senders of messages, called publishers, do not program the messages to be sent directly to specific receivers, called subscribers.

103. What is message queuing?

Message Queuing technology enables applications running at different times to communicate across heterogeneous networks and systems that may be temporarily offline.

104. Define session.

A session is a series of operations involving the creation, production and consumption of messages related to a logical task.

105. What are the parts of JMS message?

There are three parts: a header, a set of properties and the body of the message.

106. What is DSM?

This is an abstraction used for sharing data between computers that do not share physical memory. Processes access DSM by reads and updates to what appears to be ordinary memory within their address space.

107. Give the differences between message passing and DSM.

Message Passing	Distributed Shared Memory
Services Offered: Variables have to be marshalled from one process, transmitted and unmarshalled into other variables at the receiving process.	The processes share variables directly, so no marshalling and unmarshalling. Shared variables can be named, stored and accessed in DSM.
Processes can communicate with other processes. They can be protected from one another by having private address spaces.	Here, a process does not have private address space. So one process can alter the execution of other.
This technique can be used in heterogeneous computers.	This cannot be used to heterogeneous computers.
Synchronization between processes is through message passing primitives.	Synchronization is through locks and semaphores.
Processes communicating via message passing must execute at the same time.	Processes communicating through DSM may execute with non-overlapping lifetimes.

Efficiency: All remote data accesses are explicit and therefore the programmer is always aware of whether a particular operation is in-process or involves the expense of communication.	Any particular read or update may or may not involve communication by the underlying runtime support.

108. Give the properties of tuple space.

- Space uncoupling: A tuple placed in tuple space may originate from any number of sender processes and may be delivered to any one of a number of potential recipients.
- Time uncoupling: A tuple placed in tuple space will remain in that tuple space until removed and hence the sender and receiver do not need to overlap in time.

109. Give the differences between objects and distributed objects.

Objects	Distributed Objects
Class is a fundamental concept in object-oriented languages.	Classes are not prominent here.
In the object oriented world, class can be the description of the behavior associated with a group of objects or, the place to go to instantiate an object with a given behavior or even the group of objects that adhere to that behavior.	Here the term „class" is avoided, more specific terms such as „factory" and „template" are readily used.
Object oriented languages offer implementation inheritances	This offers interface inheritances.

110. Define software component.

A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only.

111. Give the advantages and disadvantages of CBT.

Advantages of CBT

- ✓ Independent extensions

- ✓ Component Market
- ✓ Component models lessen unanticipated interactions between components
- ✓ Reduced time to market
- ✓ Reduced Costs

Disadvantages of CBT:

- ✓ Time to develop software components takes a big effort.
- ✓ Components can be pricey.
- ✓ Requirements in component technologies lacking
- ✓ Conflict between usability and reusability of components.
- ✓ Maintenance cost for components increased

112. What is EJB?

Enterprise JavaBeans (EJB) is a specification of a server-side, managed component architecture. It supports the development of the classic style of application, where potentially large numbers of clients interact with a number of services realized through components or configuration of components.

PART-B

1. Describe the physical system model.
2. Explain architectural models.
3. Write notes about architectural patterns.
4. Explain the middleware.
5. Describe the fundamental models.
6. Explain about interprocess communication.
7. Describe external data representation and marshaling.
8. Elucidate multicast communication.
9. Explain overlay networks.
10. Write notes on message passing interface.
11. Describe remote invocation.
12. Write in detail about Java RMI.

13. Explain group communications.
14. Describe publish subscriber systems.
15. Write about message queues.
16. Explain shared memory approach.
17. Describe distributed objects.
18. How to convert from objects to components?
19. Write about EJB.