

Characterization of Distributed Systems:

Introduction A distributed system is one in which components located at networked computers communicate and coordinate their actions only by passing messages.

A distributed system as one in which hardware or software components located at networked computers communicate and coordinate their actions only by passing messages. This simple definition covers the entire range of systems in which networked computers can usefully be deployed.

Computers that are connected by a network may be spatially separated by any distance. They may be on separate continents, in the same building or in the same room.

Our definition of distributed systems has the following significant consequences:

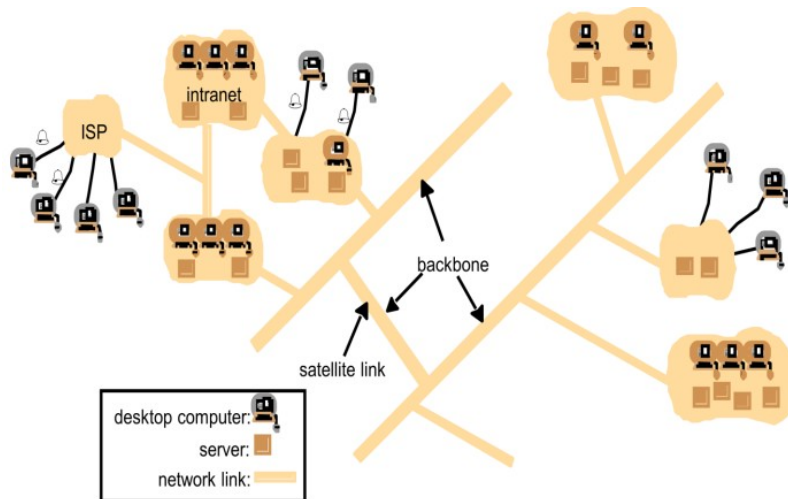
Concurrency: In a network of computers, concurrent program execution is the norm. I can do my work on my computer while you do your work on yours, sharing resources such as web pages or files when necessary. The capacity of the system to handle shared resources can be increased by adding more resources (for example. computers) to the network.

No global clock: When programs need to cooperate they coordinate their actions by exchanging messages. Close coordination often depends on a shared idea of the time at which the programs' actions occur. But it turns out that there are limits to the accuracy with which the computers in a network can synchronize their clocks – there is no single global notion of the correct time. This is a direct consequence of the fact that the only communication is by sending messages through a network.

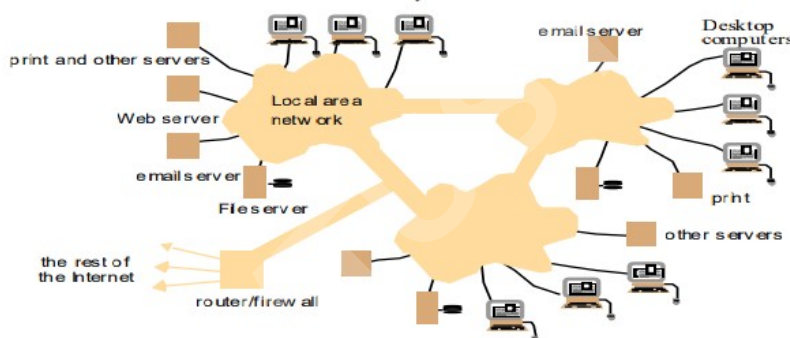
Independent failures: All computer systems can fail, and it is the responsibility of system designers to plan for the consequences of possible failures. Distributed systems can fail in new ways. Faults in the network result in the isolation of the computers that are connected to it, but that doesn't mean that they stop running. In fact, the programs on them may not be able to detect whether the network has failed or has become unusually slow. Similarly, the failure of a computer, or the unexpected termination of a program somewhere in the system (a crash), is not immediately made known to the other components with which it communicates. Each component of the system can fail independently, leaving the others still running.

Examples of Distributed systems To place distributed systems in a realistic context through examples: the Internet, an intranet and mobile computing.

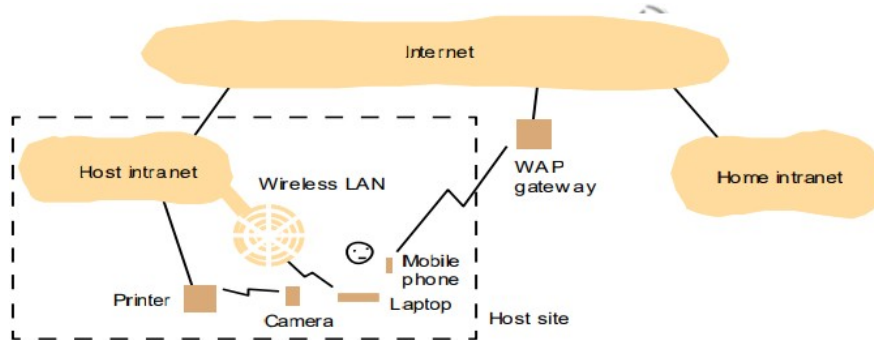
1. **The Internet** (Figure 1) : A vast interconnected collection of computer networks of many different types. Passing message by employing a common means of communication (Internet Protocol). The web is not equal to the Internet.



2. **Intranets (Figure 2):** An intranet is a private network that is contained within an enterprise. It may consist of many interlinked local area networks and also use leased lines in the Wide Area Network. It separately administrated and enforces local security policies. It is connected to the Internet via a router .It uses firewall to protect an Intranet by preventing unauthorized messages leaving or entering .Some are isolated from the Internet .Users in an intranet share data by means of file services. the rest of email server Web server Desktop computer.



3. **Mobile and Ubiquitous Computing (Figure 1.3)** a. Distributed systems techniques are equally applicable to mobile computing involving laptops, PDAs and wearable computing devices. b. Mobile computing (nomadic computing) - perform of computing tasks while moving (nomadic computing) c. Ubiquitous computing - small computers embedded in appliances i. harness of many small, cheap computation devices ii. It benefits users while they remain in a single environment such as home. Distributed In Figure 3 user has access to three forms of wireless connection: d. A laptop is connected to host's wireless LAN. e. A mobile (cellular) phone is connected to Internet using Wireless Application Protocol (WAP) via a gateway. f. A digital camera is connected to a printer over an infra-red link.



Resource Sharing and Web

- Equipments are shared to reduce cost. Data shared in database or web pages are high-level resources which are more significant to users without regard for the server or servers that provide these.

- Patterns of resource sharing vary widely in their scope and in how closely users work together: Search Engine: Users need no contact between users .Computer Supported Cooperative Working (CSCW): Users cooperate directly share resources. Mechanisms to coordinate users' action are determined by the pattern of sharing and the geographic distribution.

- For effective sharing, each resource must be managed by a program that offers a communication interface enabling the resource to be accessed and updated reliably and consistently.

- Server is a running program (a process) on a networked computer that accepts requests from programs running on other computers to perform a service and responds appropriately .

- The requesting processes are referred to as a client.

- An executing web browser is a client. It communicates with a web server to request web pages from it.

- When a client invokes an operation upon the server, it is called the remote invocation.

- Resources may be encapsulated as objects and accessed by client objects. In this case a client object invokes a method upon a server object. The World Wide Web (WWW)

- WWW is an evolving system for publishing and accessing resources and services across Internet. Web is an open system. Its operations are based on freely published communication standards and documents standards.

- Key feature: Web provides a hypertext structure among the documents that it stores. The documents contain links - references to other documents or resources. The structures of links can be arbitrarily complex and the set of resources that can be added is unlimited.

- Three main standard technological components:
- HTML (Hypertext Markup Language) specify the contents and layout of web pages.
- Contents: text, table, form, image, links, information for search engine, ..
- Layout: text format, background, frame, ...
- URL (Uniform Resource Location): identify a resource to let browser find it
 - scheme : scheme-specific-location
 - http://web.cs.twsu.edu/ (HyperText Transfer Protocol)
- URL (continued):
 - ftp://ftp.twsu.edu/ (File Transfer Protocol)
 - telnet://kirk.cs.twsu.edu (log into a computer)
 - mailto:chang@cs.twsu.edu (identify a user's email address)
- HTTP (HyperText Transfer Protocol) defines a standard rule by which browsers and any other types of client interact with web servers.

Main features: • Request-reply interaction

• Content types may or may not be handled by browser - using plug-in or external helper
One resource per request - Several requests can be made concurrently.

- Simple access control
- Services and dynamic pages
- form - Common Gateway Interface program on server (Perl)
- JavaScript (download from server and run on local computer)
- Applet (download from server and run on local computer)

Challenges

- Heterogeneity
- Openness
- Security
- Scalability
- Failure handling

- Concurrency
- Transparency
- Quality of service

Heterogeneity: The Internet enables users to access services and run applications over a heterogeneous collection of computers and networks. Heterogeneity (that is, variety and difference) applies to all of the following:

- o Hardware devices: computers, tablets, mobile phones, embedded devices, etc.
- o Operating System: Ms Windows, Linux, Mac, Unix, etc.
- o Network: Local network, the Internet, wireless network, satellite links, etc.
- o Programming languages: Java, C/C++, Python, PHP, etc.
- o Different roles of software developers, designers, system managers

Different programming languages use different representations for characters and data structures such as arrays and records. These differences must be addressed if programs written in different languages are to be able to communicate with one another. Programs written by different developers cannot communicate with one another unless they use common standards, for example, for network communication and the representation of primitive data items and data structures in messages. For this to happen, standards need to be agreed and adopted – as have the Internet protocols.

Middleware : The term middleware applies to a software layer that provides a programming abstraction as well as masking the heterogeneity of the underlying networks, hardware, operating systems and programming languages. Most middleware is implemented over the Internet protocols, which themselves mask the differences of the underlying networks, but all middleware deals with the difference in operating systems and hardware Heterogeneity and mobile

code : The term mobile code is used to refer to program code that can be transferred from one computer to another and run at the destination – Java applets are an example. Code suitable for running on one computer is not necessarily suitable for running on another because executable programs are normally specific both to the instruction set and to the host operating system.

Transparency: Transparency is defined as the concealment from the user and the application programmer of the separation of components in a distributed system, so that the system is perceived as a whole rather than as a collection of independent components. In other words, distributed systems designers must hide the complexity of the systems as much as they can. – 8 forms of transparency:

- Access transparency – access to local and remote resources using identical operations
- Location transparency – access to resources without knowing the physical location of the machine
- Concurrency transparency – several processes operate concurrently without interfering each other

- Replication transparency – replication of resources in multiple servers. Users are not aware of the replication

- Failure transparency – concealment of faults, allows users to complete their tasks without knowing of the failures

- Mobility transparency – movement of resources and clients within a system without knowing of the failures.

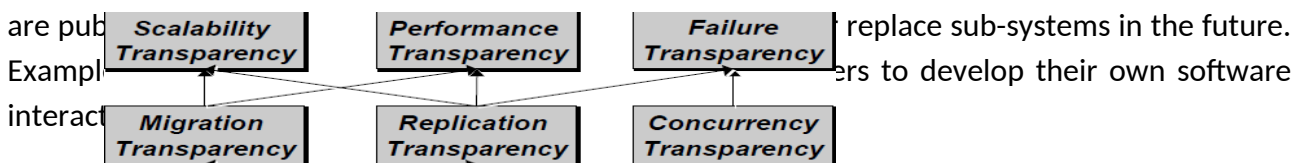
Mobility transparency – movement of resources and clients within a system without affecting users operations

- Performance transparency – systems can be reconfigured to improve performance by considering their loads

- Scaling transparency – systems and applications can be expanded without changing the structure or the application algorithms.

Openness

The openness of a computer system is the characteristic that determines whether the system can be extended and re-implemented in various ways. The openness of distributed systems is determined primarily by the degree to which new resource-sharing services can be added and be made available for use by a variety of client programs. If the well-defined interfaces for a system are published, users can replace sub-systems in the future.



Both services and applications provide resources that can be shared by clients in a distributed system. There is therefore a possibility that several clients will attempt to access a shared resource at the same time. For example, a data structure that records bids for an auction may be accessed very frequently when it gets close to the deadline time. For an object to be safe in a concurrent environment, its operations must be synchronized in such a way that its data remains consistent. This can be achieved by standard techniques such as semaphores, which are used in most operating systems

Security

Many of the information resources that are made available and maintained in distributed systems have a high intrinsic value to their users. Their security is therefore of considerable importance. Security for information resources has three components: confidentiality (protection against disclosure to unauthorized individuals) integrity (protection against alteration or corruption), availability for the authorized (protection against interference with the means to access the resources).

Scalability

Distributed systems must be scalable as the number of user increases. The scalability is defined by B. Clifford Neuman as A system is said to be scalable if it can handle the addition of users and resources without suffering a noticeable loss of performance or increase in administrative complexity Scalability has 3 dimensions: o Size o Number of users and resources to be processed. Problem associated is overloading o Geography o Distance between users and resources. Problem associated is communication reliability o Administration o As the size of distributed systems increases, many of the system needs to be controlled. Problem associated is administrative message.

Failure Handling

Computer systems sometimes fail. When faults occur in hardware or software, programs may produce incorrect results or may stop before they have completed the intended computation. The handling of failures is particularly difficult. – Dealing with failures in distributed systems:

- Detecting failures – known/unknown failures
- Masking failures – hide the failure from become severe. E.g. retransmit messages, backup of file data
- Tolerating failures – clients can be designed to tolerate failures – e.g. inform users of failure and ask them to try later
- Recovery from failures - recover and rollback data after a server has crashed
- Redundancy- the way to tolerate failures – replication of services and data in multiple servers.

Quality of service

– The main nonfunctional properties of distributed systems that affect the quality of service experienced by users or clients are: reliability, security, performance, adaptability

- Reliability
- Security

- Performance
- Adaptability

In practice, patterns of resource sharing vary widely in their scope and in how closely users work together. At one extreme, a search engine on the Web provides a facility to users throughout the world, users who need never come into contact with one another directly. At the other extreme, in *computer-supported cooperative working* (CSCW), a group of users who cooperate directly share resources such as documents in a small, closed group. The pattern of sharing and the geographic distribution of particular users determines what mechanisms the system must supply to coordinate users' actions.

We use the term *service* for a distinct part of a computer system that manages a collection of related resources and presents their functionality to users and applications. For example, we access shared files through a file service; we send documents to printers through a printing service; we buy goods through an electronic payment service. The only access we have to the service is via the set of operations that it exports. For example, a file service provides *read*, *write* and *delete* operations on files.

The fact that services restrict resource access to a well-defined set of operations is in part standard software engineering practice. But it also reflects the physical organization of distributed systems. Resources in a distributed system are physically encapsulated within computers and can only be accessed from other computers by means of communication. For effective sharing, each resource must be managed by a program that offers a communication interface enabling the resource to be accessed and updated reliably and consistently.

The term *server* is probably familiar to most readers. It refers to a running program (a *process*) on a networked computer that accepts requests from programs running on other computers to perform a service and responds appropriately. The requesting processes are referred to as *clients*, and the overall approach is known as *client-server computing*. In this approach, requests are sent in messages from clients to a server and replies are sent in messages from the server to the clients. When the client sends a request for an operation to be carried out, we say that the client *invokes an operation* upon the server. A complete interaction between a client and a server, from the point when the client sends its request to when it receives the server's response, is called a *remote invocation*.

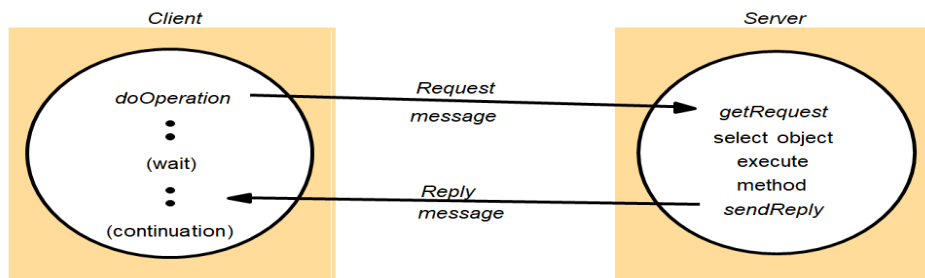
The same process may be both a client and a server, since servers sometimes invoke operations on other servers. The terms 'client' and 'server' apply only to the roles played in a single request. Clients are active (making requests) and servers are passive (only waking up when they receive requests); servers run continuously, whereas clients last only as long as the applications of which they form a part.

Request-reply communication

Request-reply communication is synchronous because the client process blocks until the reply arrives from the server. It can also be reliable because the reply from the server is effectively an acknowledgement to the client.

Asynchronous request-reply communication is an alternative that may be useful in situations where clients can afford to retrieve replies later.

Request-reply communication



Operations of the request-reply protocol

The request-reply protocol:

- The protocol is based on a trio of communication primitives, *doOperation*, *getRequest* and *sendReply*,
- This request-reply protocol matches requests to replies. It may be designed to provide certain delivery guarantees. If UDP datagrams are used, the delivery guarantees must be provided by the request-reply protocol, which may use the server reply message as an acknowledgement of the client request message.
- The *doOperation* method is used by clients to invoke remote operations. Its arguments specify the remote server and which operation to invoke, together with additional information (arguments) required by the operation. Its result is a byte array containing the reply.
- *getRequest* is used by a server process to acquire service requests.
- *sendReply* is used to send the reply message to the client. When the reply message is received by the client the original *doOperation* is unblocked and execution of the client program continues.

```
public byte[] doOperation (RemoteRef s, int operationId, byte[] arguments)
```

sends a request message to the remote server and returns the reply.

The arguments specify the remote server, the operation to be invoked and the arguments of that operation.

```
public byte[] getRequest ();
```

acquires a client request via the server port.

```
public void sendReply (byte[] reply, InetAddress clientHost, int clientPort);
```

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

Styles of exchange protocols

Three protocols that produce differing behaviours in the presence of communication failures are used for implementing various types of request behaviour. They were originally identified by Spector [1982]:

- the request (R) protocol;
- the request-reply (RR) protocol;
- the request-reply-acknowledge reply (RRA) protocol.

Request-reply message structure

messageType	<i>int (0=Request, 1= Reply)</i>
requestId	<i>int</i>
remoteReference	<i><u>RemoteRef</u></i>
operationId	<i>int or Operation</i>
arguments	<i>array of bytes</i>

Distributed Objects and Remote Invocation

- Remote procedure call – client calls the procedures in a server program that is running in a different process.
- Remote method invocation (RMI) – an object in one process can invoke methods of objects in another process
- Event notification – objects receive notification of events at other objects for which they have registered.

Middleware:

The important aspects of middleware are:

- **Location transparency:** In RPC ,the client that calls a procedure cannot tell whether the procedure runs in the same process or in different process, possibly on a different computer.

Similarly in RMI the object making the invocation cannot tell whether the object it invokes is local or not and does not need to know the location.



It is also free from the specifics of communication protocols, operating system and communication hardware

Interfaces:

In most of the programming languages program is divided into set of modules and these modules communicate with each other. In distributed systems these modules are present indifferent processes. The interface of a module specifies the procedures and the variables that can be accessed from other modules.

There are two types of interfaces:

- **Service interface:** In client server model ,The server specifies set of procedures and input-output parameters available to the client.
- **Remote interface:** In Distributed object model ,a remote interface specifies the methods of an object that are available for invocation by other objects and also the input output arguments.

Interface Definition Language: It provides a notation for defining interfaces. It can also specify type of arguments.

Examples:CORBAIDL for RMI,SunXDR for RPCCORBA IDLExample:

```
struct Person {  
    string name;  
    string place;  
    long year;  
};  
interface PersonList {  
    readonly attribute string listname;  
    void addPerson(in Person p) ;  
    void getPerson(in string name, out  
    Person p);  
    long number();  
};
```

In the above example, add person and get person are methods that are available for RPC

Communication between Distributed Objects:

1. The Object Model:

- ✓ An object encapsulates both data and methods. Objects can be accessed via **object references**.
- ✓ An **interface** provides a definition of the signatures of a set of methods
- ✓ Actions are performed by **method invocations**:

The invocation of a method has three effects:

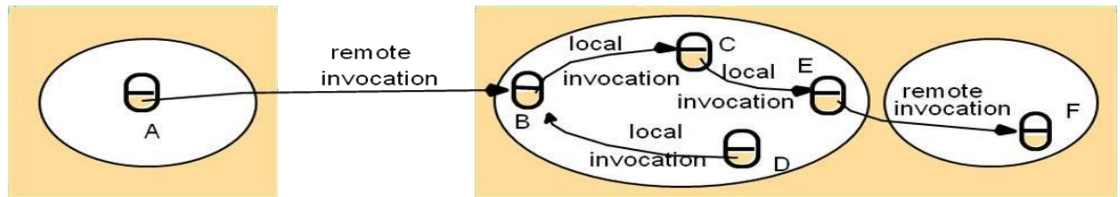
- i. The state of the receiver may be changed
 - ii. A new object maybe instantiated
 - iii. Further invocations on methods in other objects may take place.
- ✓ Exceptions may be thrown to caller when an error occurs.
 - ✓ **Garbage collection** frees the space occupied by objects when they are no longer needed.

The Distributed Objects Model:

Here we discuss the object model that is applicable to distributed objects.

- Remote method invocation – Method invocations between objects in different processes, whether in the same computer or not.

- Local method invocation – Method invocations between objects in the same process.

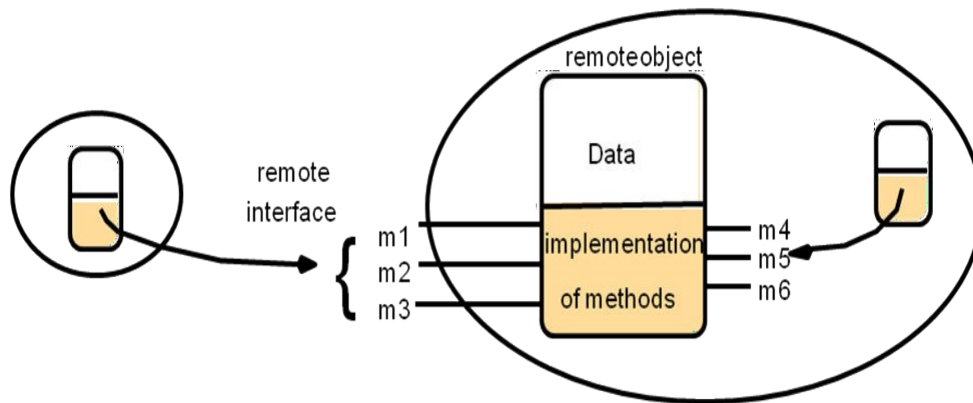


- Remote object – Objects that can receive remote invocations. Remote and local method invocations are shown in Figure 5.3.
- Each process contains objects, some of which can receive remote invocations, others only local invocations.
- Those that can receive remote invocations are called *remote objects*.
- Objects need to know the *remote object reference* of an object in another process in order to invoke its methods.
- The *remote interface* specifies which methods can be invoked remotely. The two fundamental concepts that are heart of distributed object model are:
 - Remote object reference: An object must have the remote object reference of an object in order to do remote invocation of an object. Remote object references may be passed as input arguments or returned as output arguments.
 - Remote interface: Objects in other processes can invoke only the methods that belong to its remote interface (Figure 5.4).

CORBA – uses IDL to specify remote interface

JAVA – extends interface by the **Remote** keyword

A remote object and its remote interface



Here the methods m1,m2,m3 are provided in the remote interface .so, client can access only these methods.

2. Design Issues for RMI:

Two important design issues in making RMI a natural extension of local method are: 1. Invocation semantics and ii. Transparency

Remote Invocation Semantics:

To provide a more reliable request-reply protocol, these fault-tolerant measures can be employed:

Retry request message: whether to transmit the request message until either a reply is received or server is assumed to be failed.

Duplicate Filtering: when transmissions are used , whether to filter out duplicate requests at the server.

Retransmission of results: whether to keep a history of result messages to enable lost results to be retransmitted without re-executing the operations at the server.

Combinations of these measures lead to a variety of possible semantics for the reliability of remote invocations.

The choices of RMI invocation semantics are defined as follows:

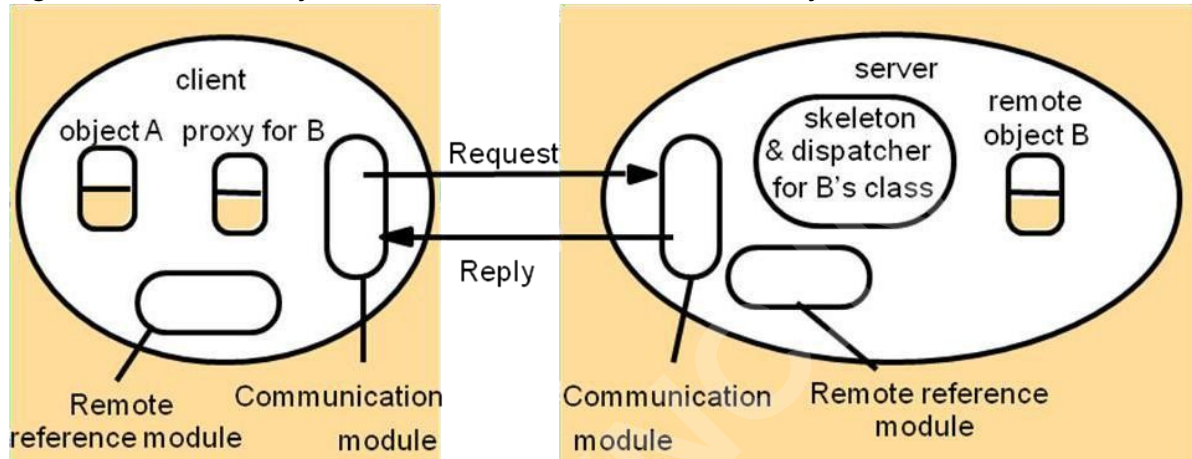
- i. **Maybe invocation semantics:** Remote method may be executed once or not at all. If then result message is not received after a timeout there will be no retries, it is uncertain whether the method has been executed. . On the other hand, the procedure may have been executed and the result message has been lost. Maybe semantics is useful only for applications in which occasional



<i>Fault tolerance measures</i>			<i>Invocation semantics</i>
<i>Retransmit request message</i>	<i>Duplicate filtering</i>	<i>Re-execute procedure or retransmit reply</i>	
No	Not applicable	Not applicable	<i>Maybe</i>
Yes	No	Re-execute procedure	<i>At-least-once</i>
Yes	Yes	Retransmit reply	<i>At-most-once</i>

1. Syntax may be made identical but behavioral differences exist. The cause could be **Failure and latency.**
2. Exceptions and exception handling are needed
3. **Implementation of RMI:**

Figure 5.6 shows an object A invokes a method in a remote object B



Remote Reference Module: Responsibilities:

- i. Translation between local and remote object references
- ii. the remote reference module in each process has a remote object table that includes:
 - An entry for all the remote objects held by the process. For example in the above fig the remote object B will be recorded in the table at the server
 - An entry for each local proxy. for example in the above fig the proxy for B will be recorded in the table at the client.

RMI Software:

Proxy—provides remote invocation transparency

- marshal arguments, unmarshal results, send and receive messages
- Dispatcher—handles transfer of requests to correct method
- receive requests, select correct method, and pass on request message
- Skeleton—implements methods of remote interface

- unmarshal arguments from request, invoke the method of the remote object, and marshal the results

RMI Server and Client Programs:

Server: contains

- classesfordispatchers,skeletons and remote objects
- initializationsectionforcreatingsomeremoteobjects
- registration of remote objects with the binderClientcontains:
- classes for proxies of all remote objects
- binder to look up remote object references

RMI Binder and Server Threads:

A binder in a distributed system is a separate service that maintains a table containing mappings from textual names to remote object references

Server threads:

- sometimes implemented so that remote invocation causes a new thread to be created to handle the call
- server with several remote objects might also allocate separate threads to handle each object Activation of remote objects:
- A remote object is described as **active** when it is a running process.
- A remote object is described as **passive** when it can be made active if requested.
- An object that can live between activations of processes is called a **persistent object**

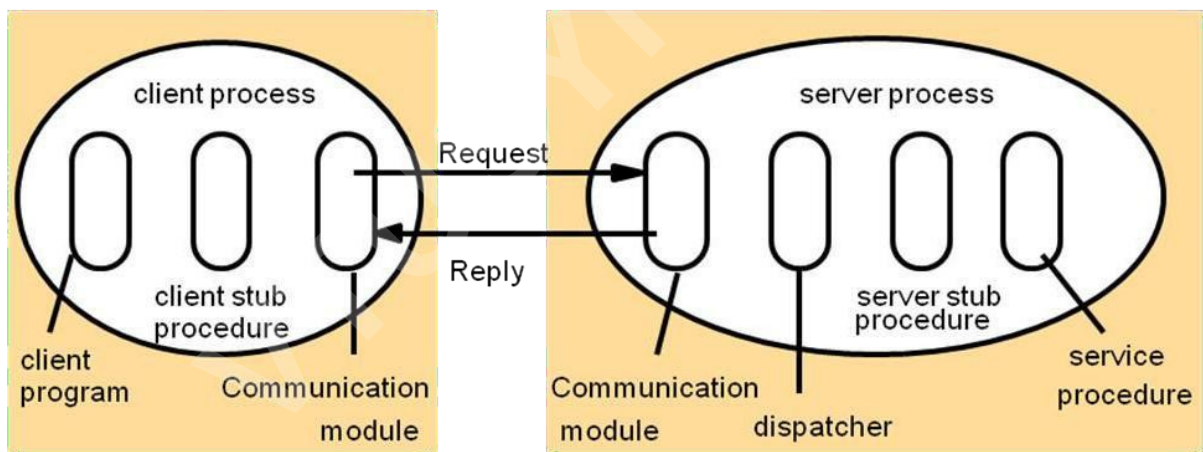
Allocation servicehelpsclients to locate remote objects from their remote references

RMI Distributed Garbage Collection:

Aim-recover memory if no reference t o an object exists. If there is a reference object

A RPC call is very similar to RMI ,in which a client program calls a procedure in another program running in server process

Role of client and server stub procedures in RPC

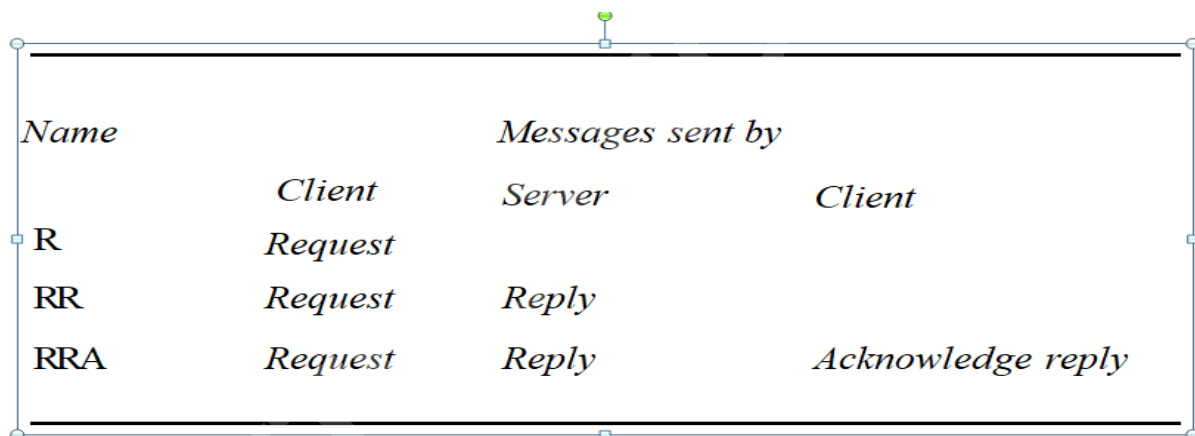


- The software components required to implement RPC are shown in the above Figure.
- The stub procedure behaves like a local procedure to the client, but instead of executing the call, it marshals the procedure identifier and the arguments into a request message, which it sends via its communication module to the server. When the reply message arrives, it unmarshals the results.
- The server process contains a dispatcher together with one server stub procedure and one service procedure for each procedure in the service interface
- The dispatcher selects one of the server stub procedures according to the

procedure identifier in the request message

- The server stub procedure then unmarshals the arguments in the request message, calls the corresponding service procedure and marshals there turn values for the reply message
- The service procedures implement the procedures in the service interface
- Client and servers tub procedures and the dispatcher can be generated automatically by an interface compiler from the interface definition no f the service.

RPC exchange protocols



HTTP

HTTP is a protocol that specifies the messages involved in a request-reply exchange, the methods, arguments and results, and the rules for representing them in the messages.

It supports a fixed set of methods (GET, PUT, POST, etc) that are applicable to all of the server's resources.

In addition to invoking methods on web resources, the protocol allows for content negotiation and password-style authentication:

Content negotiation: Clients' requests can include information as to what data representations they can accept (for example, language or media type), enabling the server to choose the representation that is the most appropriate for the user.

Authentication: Credentials and challenges are used to support password-style authentication. On the first attempt to access a password-protected area, the server reply contains a challenge applicable to the resource. When a client receives a challenge, it gets the user to type a name and password and submits the associated credentials with subsequent requests

HTTP request message

<i>method</i>	<i>URL or pathname</i>	<i>HTTP version</i>	<i>headers</i>	<i>message body</i>
GET	//www.dcs.qmw.ac.uk/index.html	HTTP/ 1.1		

HTTP Reply message

<i>HTTP version</i>	<i>status code</i>	<i>reason</i>	<i>headers</i>	<i>message body</i>
HTTP/1.1	200	OK		resource data

Case study :Sun RPC:

- It is designed for client-server communication over Sun NFS network file system.
- UDP or TCP can be used. If UDP is used, the message length is restricted to 64 KB

Interface Definition Language:

The notation is rather primitive compared to CORBA IDL or JAVA

- Instead of no interface definition, a program number and a version



```
const MAX = 1000;
typedef int FileIdentifier;
typedef int FilePointer;
typedef int Length;
struct Data {
    int length;
    char buffer[MAX];
};
struct writeargs {
    FileIdentifier f;
    FilePointer position;
    Data data;
};

struct readargs {
    FileIdentifier f;
    FilePointer position;
    Length length;
};

program FILEREADWRITE {
    version VERSION {
        void WRITE(writeargs)=1; 1
        Data READ(readargs)=2; 2
    }=2;
} = 9999;
```

Binding • Sun RPC runs a local binding service called the *port mapper* at a well-known port number on each computer. Each instance of a port mapper records the program number, version number and port number in use by each service running locally. When a server starts up it registers its program number, version number and port number with the local port mapper. When a client starts up, it finds out the server's port by making a remote request to the port mapper at the server's host, specifying the program number and version number.

RPC call semantics

Request-reply protocols showed that doOperation can be implemented in different ways to provide different delivery guarantees.

The main choices are:

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.

Call semantics

<i>Fault tolerance measures</i>			<i>Call semantics</i>
<i>Retransmit request message</i>	<i>Duplicate filtering</i>	<i>Re-execute procedure or retransmit reply</i>	
No	Not applicable	Not applicable	<i>Maybe</i>
Yes	No	Re-execute procedure	<i>At-least-once</i>
Yes	Yes	Retransmit reply	<i>At-most-once</i>

