3

Architectures of Client/Server Systems

3.1 INTRODUCTION

The term Client/Server was first used in the 1980s in reference to personal computers (PCs) on a network. The actual Client/Server model started gaining acceptance in the late 1980s. The term Client/Server is in reality a logical concept. The client and server components may not exist on distinct physical hardware. A single machine can be both a client and a server depending on the software configuration. The Client/Server technology is a model, for the interaction between simultaneously executing software processes. The term architecture refers to the logical structure and functional characteristics of a system, including the way they interact with each other in terms of computer hardware, software and the links between them.

In case of Client/Server systems, the architecture means the way clients and servers along with the requisite software are configured with each others. Client/Server architecture is based on the hardware and the software components that interact to form a system. The limitations of file sharing architectures led to the emergence of the Client/Server architecture. This approach introduced a database server to replace the file server. Using a Relational Database Management System (RDBMS), user queries could be answered directly. The Client/Server architecture reduced network traffic by providing a query response rather than total file transfer. It improves multi-user updating through a GUI front-end to a shared database. In Client/Server architectures, Remote Procedure Calls (RPCs) or Structural Query Language (SQL) statements are typically used to communicate between the client and server.

File sharing architecture (not a Client/Server architecture):

File based database (flat-file database are very efficient to extracting information from large data files. Each workstation on the network has access to a central file server where the data is stored.

The data files can also reside on the workstation with the client application. Multiple workstations will access the same file server where the data is stored. The file server is centrally located so that it can be reached easily and efficiently by all workstations.

The original PC networks were based on file sharing architectures, where the server downloads files form the shared location to the desktop environment. The requested user job is then run (including logic and data) in the desktop environment.

File sharing architectures work if shared usage is low, update contention is low, and the volume of data to be transferred is low. In the 1990s, PC LAN (Local Area Network) computing changed because the capacity of file sharing was strained as the number of online users grew (it can only satisfy about 12 users simultaneously) and Graphical User Interfaces (GUIs)became popular (making mainframe and terminal displays appear out of data). PCs are now being used in Client/Server architectures.

Mainframe architecture (not a Client/Server architecture)

With mainframe software architectures all intelligence is within the central host computer. Users interact with the host through a terminal that captures keystrokes and sends that information to the host. Mainframe software architectures are not tied to a hardware platform. User interaction can be done using PCs and UNIX workstations. A limitation of mainframe software architectures is that they do not easily support graphical user interfaces or access to multiple databases from geographically dispersed sites. In the last few years, mainframes have found a new use as a server in distributed Client/Server architectures.

The Client/Server software architecture is a versatile, message-based and modular infrastructure that is intended to improve usability, flexibility, interoperability, and scalability as compared to centralized, mainframe, time sharing computing.

3.2 COMPONENTS

Client/Server architecture is based on hardware and software components that interact to form a system. The system includes mainly three components.

(*i*) Hardware (client and server).

(*ii*) Software (which make hardware operational).

(*iii*) Communication middleware. (associated with a network which are used to link the hardware and software).

The client is any computer **process** that requests services from server. The client uses the services provided by one or more server processors. The client is also known as the **front-end application**, reflecting that the end user usually interacts with the client process.

The server is any computer **process** providing the services to the client and also supports multiple and simultaneous clients requests. The server is also known as **back-end application**, reflecting the fact that the server process provides the background services for the client process.

The communication middleware is any computer **process through** which client and server communicate. Middleware is used to integrate application programs and other software components in a distributed environment. Also known as **communication layer**. Communication layer is made up of several layers of software that aids the transmission of data and control information between Client and Server. Communication middleware is usually associated with a network. The Fig. 3.1 below gives a general structure of Client/ Server System.

Now as the definition reveals, clients are treated as the front-end application and the server as the back-end application, the Fig. 3.2 given below shows the front-end and back-end functionality.

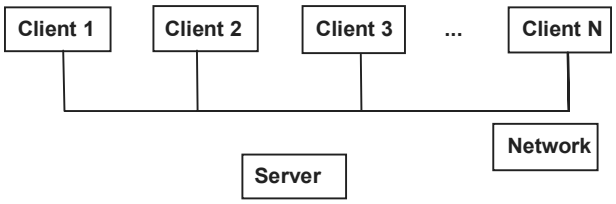

Fig.3.1: Structure of a Client/Server System

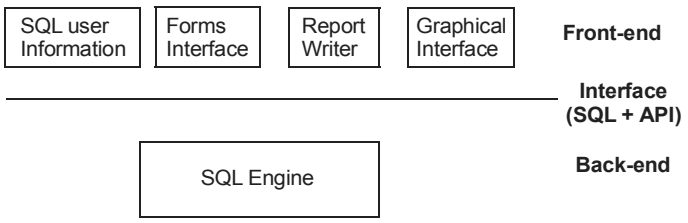

Fig.3.2: Front-end and Back-end Functionality

3.2.1 Interaction between the Components

The interaction mechanism between the components of Client/Server architecture is clear from the Fig. 3.3. The client process is providing the interface to the end users. Communication middleware is providing all the possible support for the communication taking place between the client and server processes. Communication middleware ensures that the messages between clients and servers are properly routed and delivered. Requests are handled by the database server, which checks the validity of the request, executes them, and send the result back to the clients.

3.2.2 Complex Client/Server Interactions

The better understanding about the functionality of Client/Server is observed when the clients and server interact with each other. Some noticeable facts are:

- A client application is not restricted to accessing a single service. The client contacts a different server (perhaps on a different computer) for each service.
- A client application is not restricted to accessing a single server for a given service.

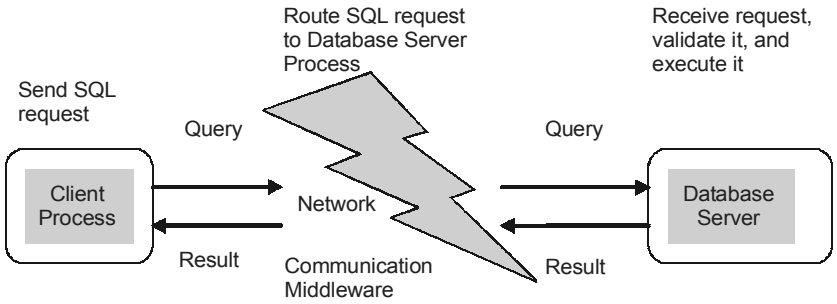

Fig.3.3: Components Interaction

- A server is not restricted from performing further Client/Server interactions — a server for one service can become a client of another.

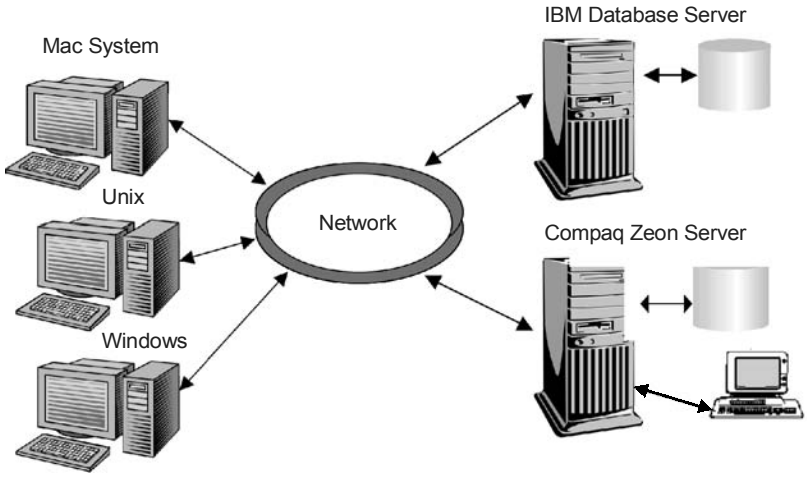

Fig.3.4: A Complex Client/Server Environment

Generally, the client and server processes reside on different computers. The Fig. 3.4 illustrates a Client/Server system with more than one server and several clients. The system comprises of the Back-end, Front-end Processes and Middleware.

Back-end processes as: IBM Database server process and Compaq Zeon Server

Front-end as: Application client processes (Windows, Unix and Mac Systems)

Middleware as: Communication middleware (network and supporting software)

The client process runs under different Operating Systems (Windows, Unix and Mac System), server process (IBM and Compaq computers) runs under different operating system

(OS/2 and Unix). The communication middleware acts as the integrating platform for all the different components. The communication can take place between client to client and as well as server to server.

3.3 PRINCIPLES BEHIND CLIENT/SERVER SYSTEMS

The components of the Client/Server architecture must conform to some basic principles, if they are to interact properly. These principles must be uniformly applicable to client, server, and to communication middleware components. Generally, these principles generating the Client/Server architecture constitute the foundation on which most current-generation Client/Server system are built. Some of the main principles are as follows:

(*i*) Hardware independence.
(*ii*) Software independence.
(*iii*) Open access to services.
(*iv*) Process distribution.
(*v*) Standards.

(*i*) Hardware independence: The principles of hardware independence requires that the Client, Server, and communication middleware, processes run on multiple hardware platforms (IBM, DEC, Compaq, Apple, and so on) without any functional differences.

(*ii*) Software independence: The principles of software independence requires that the Client, Server, and communication middleware processes support multiple operating systems (such as Windows 98, Windows NT, Apple Mac system, OS/2, Linux, and Unix) multiple network protocols (such as IPX, and TCP/IP), and multiple application (spreadsheet, database electronic mail and so on).

(*iii*) Open access to services: All client in the system must have open (unrestricted) access to all the services provided within the network, and these services must not be dependent on the location of the client or the server. A key issue is that the services should be provided on demand to the client. In fact, the provision of on-demand service is one of the main objectives of Client/Server computing model.

(*iv*) Process distribution: A primary identifying characteristic of Client/Server system is that the processing of information is distributed among Clients and Servers. The division of the application-processing load must conform to the following rules:

- Client and server processes must be autonomous entities with clearly defined boundaries and functions. This property enables us to clearly define the functionality of each side, and it enhances the modularity and flexibility of the system.
- Local utilization of resources (at both client and server sides) must be maximized. The client and server process must fully utilize the processing power of the host computers. This property enables the system to assign functionality to the

computer best suited to the task. In other words, to best utilize all resources, the server process must be shared among all client processes; that is, a server process should service multiple requests from multiple clients.

- Scalability and flexibility requires that the client and server process be easily upgradeable to run on more powerful hardware and software platforms. This property extends the functionality of Client/Server processes when they are called upon to provide the additional capabilities or better performance.
- Interoperability and integration requires that client and server processes be seamlessly integrated to form a system. Swapping a server process must be transparent the client process.

(*v*) Standards: Now, finally all the principles that are formulated must be based on standards applied within the Client/Server architecture. For example, standard must govern the user interface, data access, network protocols, interprocess communications and so on. Standards ensure that all components interact in an orderly manner to achieve the desired results. There is no universal standard for all the components. The fact is that there are many different standards from which to choose. For example, an application can be based on Open Database Connectivity (ODBC) instead of Integrated Database Application Programming Interface (IDAPI) for Data access (ODBC and IDAPI are database middleware components that enables the system to provide a data access standard for multiple processes.) Or the application might use Internet work Packet Exchange (IPX) instead of Transmission Control Protocol/Internet Protocol (TCP/IP) as the network protocol. The fact that the application does not use single standards does not mean that it will be a Client/Server application. The point is to ensure that all components (server, clients, and communication middleware) are able to interact as long as they use the same standards. What really defines Client/Server computing is that the splitting of the application processing is independent of the network protocols used.

3.4 CLIENT COMPONENTS

As we know, the client is any process that requests services from the server process. The client is proactive and will, therefore, always initiate the conversation with the server. The client includes the software and hardware components. The desirable client software and hardware feature are:

(i) Powerful hardware.

(ii) An operating system capable of multitasking.

(iii) Communication capabilities.

(iv) A graphical user interface (GUI).

(*i*) Powerful hardware: Because client processes typically requires a lot of hardware resources, they should be stationed on a computer with sufficient computing power, such as fast Pentium II, III, or RISC workstations. Such processing power

facilitates the creation of systems with multimedia capabilities. A Multimedia system handles multiple data types, such as voice, image, video, and so on. Client processes also require large amount of hard disk space and physical memory, the more such a resource is available, the better.

(*ii*) An operating system capable of multitasking: The client should have access to an operating system with at least some multitasking capabilities. Microsoft Windows 98 and XP are currently the most common client platforms. Windows 98 and XP provide access to memory, pre-emptive multitasking capabilities, and a graphical user interface, which makes windows the platform of choice in a majority of Client/Server implementations. However, Windows NT, Windows 2000 server, OS/2 from IBM corporation, and the many "flavours" of UNIX, including Linux are well-suited to handle the Client/Server processing that is largely done at the server side of the Client/Server equation.

(*iii*) Communication capabilities: To interact efficiently in a Client/Server environment, the client computer must be able to connect and communicate with the other components in a network environment. Therefore, the combination of hardware and operating system must also provide adequate connectivity to multiple network operating systems. The reason for requiring a client computer to be capable of connecting and accessing multiple network operating systems is simple services may be located in different networks.

(*iv*) A graphical user interface (GUI): The client application, or front-end, runs on top of the operating system and connects with the communication middleware to access services available in the network. Several third generation programming languages (3GLs) and fourth generation languages (4GLs) can be used to create the front-end application. Most front-end applications are GUI-based to hide the complexity of the Client/Server components from the end user. The Fig. 3.5 given below illustrates the basic client components.

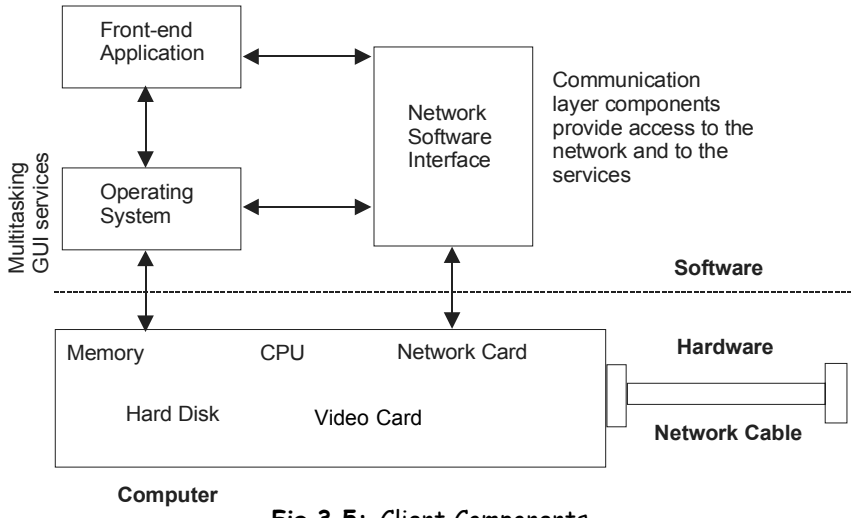

Fig.3.5: *Client Components*

3.5 SERVER COMPONENTS

As we have already discussed, the server is any process that provides services to the client process. The server is active because it always waits for the client's request. The services provided by server are:

(*i*) **File services**: For a LAN environment in which a computer with a big, fast hard disk is shared among different users, a client connected to the network can store files on the file server as if it were another local hard disk.

(*ii*) **Print services**: For a LAN environment in which a PC with one or more printers attached is shared among several clients, a client can access any one of the printers as if it were directly attached to its own computer. The data to be printed travel from the client's PC to the server printer PC where they are temporarily stored on the hard disk. When the client finishes the printing job, the data is moved from the hard disk on the print server to the appropriate printer.

(*iii*) **Fax services:** This requires at least one server equipped (internally or externally) with a fax device. The client PC need not have a fax or even a phone line connection. Instead, the client submits the data to be faxed to the fax server with the required information; such as the fax number or name of the receiver. The fax server will schedule the fax, dial the fax number, and transmit the fax. The fax server should also be able to handle any problems derived from the process.

(*iv*) **Communication services**: That let the client PCs connected to the communications server access other host computers or services to which the client is not directly connected. For example, a communication server allows a client PC to dial out to access board, a remote LA location, and so on.

(*v*) **Database services:** Which constitute the most common and most successful Client/Server implementation. Given the existence of database server, the client sends SQL request to the server. The server receives the SLQ code, validates it, executes it, and send only the result to the client. The data and the database engine are located in the database server computer.

(*vi*) **Transaction services:** Which are provided by transaction servers that are connected to the database server. A transaction server contains the database transaction code or procedures that manipulate the data in database. A front-end application in a client computer sends a request to the transaction server to execute a specific procedure store on the database server. No SQL code travels through the network. Transaction servers reduce network traffic and provide better performance than database servers.

(*vii*) **Groupware services:** Liable to store semi-structured information like Text, image, mail, bulletin boards, flow of work. Groupware Server provides services, which put people in contact with other people, that is because "groupware" is an ill-defined classification. Protocols differ from product to product. For examples: Lotus Notes/Domino, Microsoft Exchange.

(*viii*) Object application services: Communicating distributed objects reside on server. Object server provides access to those objects from client objects. Object Application Servers are responsible for Sharing distributed objects across the network. Object Application Servers uses the protocols that are usually some kind of Object Request Broker (ORB). Each distributed object can have one or more remote method. ORB locates an instance of the object server class, invokes the requested method, and returns the results to the client object. Object Application Server provides an ORB and application servers to implement this.

(*ix*) Web application services: Some documents, data, etc., reside on web servers. Web application provides access to documents and other data. "Thin" clients typically use a web browser to request those documents. Such services provide the sharing of the documents across intranets, or across the Internet (or extranets). The most commonly used protocol is HTTP (Hyper Text Transport Protocol). Web application servers are now augmenting simple web servers.

(*x*) Miscellaneous services: These include CD-ROM, video card, backup, and so on. Like the client, the server also has hardware and software components. The hardware components include the computer, CPU, memory, hard disk, video card, network card, and so on. The computer that houses the server process should be the more powerful computer than the "average" client computer because the server process must be able to handle concurrent requests from multiple clients. The Fig. 3.6 illustrates the components of server.

The server application, or back-end, runs on the top of the operating system and interacts with the communication middleware components to "listen" for the client request for the services. Unlike the front-end client processes, the server process need not be GUI based. Keep in mind that back-end application interacts with operating system (network or stand alone) to access local resources (hard disk, memory, CPU cycle, and so on). The back-end server constantly "listens" for client requests. Once a request is received the server processes it locally. The server knows how to process the request; the client tells the server only what it needs do, not how to do it. When the request is met, the answer is sent back to the client through the communication middleware.

The server hardware characteristics depend upon the extent of the required services. For example, a database is to be used in a network of fifty clients may require a computer with the following minimum characteristic:

- Fast CPU (RISC, Pentium, Power PC, or multiprocessor)
- Fault tolerant capabilities:
  - Dual power supply to prevent power supply problem.
  - Standby power supply to protect against power line failure.

- Error checking and correcting (ECC) memory to protect against memory module failures.
- Redundant Array to Inexpensive Disk (RAID) to provide protection against physical hardware failures.

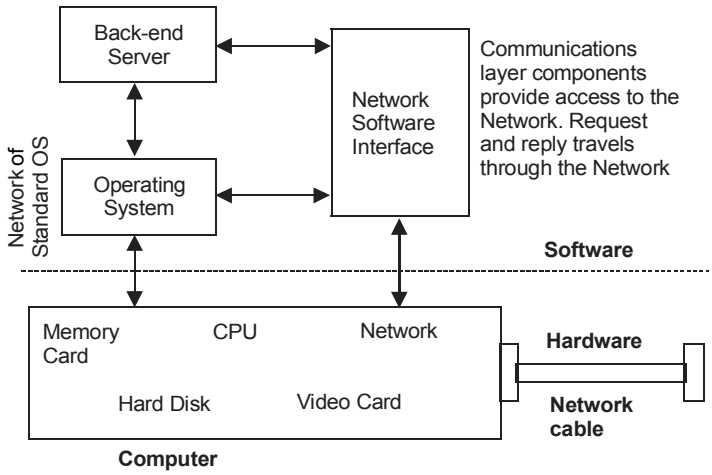

Fig.3.6: Server Components

- Expandability of CPU, memory disk, and peripherals.
- Bus support for multiple add-on boards.
- Multiple communication options.

In theory, any computer process that can be clearly divided into client and server components can be implemented through the Client/Server model. If properly implemented, the Client/Server architectural principles for process distribution are translated into the following server process benefits:

- **Location independence.** The server process can be located anywhere in the network.
- **Resource optimization.** The server process may be shared.
- **Scalability.** The server process can be upgraded to run on more powerful platforms.
- **Interoperability and integration.** The server process should be able to work in a "Plug and Play" environment.

These benefits added to hardware and software independence principles of the Client/ Server computing model, facilitate the integration of PCs, minicomputer, and mainframes in a nearly seamless environment.

3.5.1 The Complexity of Servers

The server processes one request at a time; we can say servers are fairly simple because they are sequential. After accepting a request, the server forms a reply and sends it before requesting to see if another request has arrived. Here, the operating system plays a big role in maintaining the request queue that arrives for a server.

Servers are usually much more difficult to build than clients because they need to accommodate multiple concurrent requests. Typically, servers have two parts:

- A single master program that is responsible for accepting new requests.
- A set of slaves that are responsible for handling individual requests.

Further, master server performs the following five steps (Server Functions):

(*i*) Open port: The master opens a port at which the client request reached.

(*ii*) Wait for client: The master waits for a new client to send a request.

(*iii*) Choose port: If necessary, the master allocates new local port for this request and informs the client.

(*iv*) Start slave: The master starts an independent, concurrent slave to handle this request (for example: in UNIX, it forks a copy of the server process). Note that the slave handles one request and then terminates–the slave does not wait for requests from other clients.

(*v*) Continue: The master returns to the wait step and continues accepting new requests while the newly created slave handles the previous request concurrently.

Because the master starts a slave for each new request, processing proceeds concurrently. In addition to the complexity that results because the server handles concurrent requests, complexity also arises because the server must enforce authorization and protection rules. Server programs usually need to execute with the highest privilege because they must read system files, keep logs, and access protected data. The operating system will not restrict a server program if it attempts to access a user files. Thus, servers cannot blindly honour requests from other sites. Instead, each server takes responsibility for enforcing the system access and protection policies.

Finally, servers must protect themselves against malformed request or against request that will cause the server program itself to abort. Often it is difficult to foresee potential problems. Once an abort occurs, no client would be able to access files until a system programmer restarts the server.

"Servers are usually more difficult to build than clients because, although they can be implemented with application programs, server must enforce all the access and protection policies of the computer system on which they run, and must protect themselves against all possible errors."

3.6 COMMUNICATIONS MIDDLEWARE COMPONENTS

The communication middleware software provides the means through which clients and servers communicate to perform specific actions. It also provides specialized services to the client process that insulates the front-end applications programmer from the internal working of the database server and network protocols. In the past, applications programmers had to write code that would directly interface with specific database language (generally a version of SQL) and the specific network protocol used by the database server. For example, when writing a front-end application to access an IBM OS/2 database manager database, the programmer had to write SQL and Net BIOS (Network Protocol) command in the application. The Net BIOS command would allow the client process to establish a session with the database server, send specific control information, send the request, and so on. If the same application is to be used with a different database and network, the application routines must be rewritten for the new database and network protocols. Clearly, such a condition is undesirable, and this is where middleware comes in handy. The definition of middleware is based on the intended goals and main functions of this new software category.

Although middleware can be used in different types of scenarios, such as e-mail, fax, or network protocol translation, most first generation middleware used in Client/Server applications is oriented toward providing transport data access to several database servers. The use of database middleware yields:

- **Network independence:** by allowing the front-end application to access data without regard to the network protocols.
- **Database server independence:** by allowing the front-end application to access data from multiple database servers without having to write code that is specific to each database server.

The use of database middleware, make it possible for the programmer to use the generic SQL sentences to access different and multiple database servers. The middleware layer isolates the program from the differences among SQL dialects by transforming the generic SQL sentences into the database server's expected syntax. For example, a problem in developing a front-end system for multiple database servers is that application programmers must have in-depth knowledge of the network communications and the database access language characteristic of each database to access remote data. The problem is aggravated by the fact that each DBMS vendor implements its own version of SQL (with difference in syntax, additional functions, and enhancement with respect to the SQL standard). Furthermore, the data may reside in a non-relational DBMS (hierarchical, network or flat files) that does not support SQL, thus making it harder for the programmers to access the data given such cumbersome requirements, programming in Client/Server systems becomes more difficult than programming in traditional mainframe system. Database middleware

eases the problem of accessing resources of data in multiple networks and releases the program from details of managing the network communications. To accomplish its functions, the communication middleware software operates at two levels:

- The physical level deals with the communications between client and server computers (computer to computer). In other words, it addresses how the computers are physically linked. The physical links include the network hardware and software. The network software includes the network protocol. Recall that network protocols are rules that govern how computers must interact with other computers in network, and they ensure that computers are able to send and receive signal to and from each other. Physically, the communication middleware is, in most cases, the network. Because the Client/Server model allows the client and server to reside on the same computer, it may exist without the benefit of a computer network.
- The logical level deals with the communications between client and server. Process (process to process) that is, with how the client and server process communicates. The logical characteristics are governed by process-to-process (or interprocess) communications protocols that give the signals meaning and purpose. It is at this level that most of the Client/Server conversation takes place.

Although the preceding analogy helps us understand the basic Client/Server interactions, it is required to have a better understanding of computer communication to better understand the flow of data and control information in a client server environment. To understand the details we will refer to Open System Interconnection (OSI) network reference model which is an effort to standardize the diverse network systems. Figure 3.7 depicts the flow of information through each layer of OSI model.

From the figure, we can trace the data flow:

- The client application generates a SQL request.
- The SQL request is sent down to the presentation layer, where it is changed to a format that the SQL server engine can understand.
- Now, the SQL request is handed down to session layer. This layer establishes the connection to the client processes with the server processes. If the database server requires user verification, the session layer generates the necessary message to log on and verify the end user. And also this layer will identify which mesaages are control messages and which are data messages.
- After the session is established and validated, the SQL request is sent to the transport layer. The transport layer generates some error validation checksums and adds some transport-layer-specific ID information.
- Once the transport layer has performed its functions, the SQL request is handed down to the network layer. This layer takes the SQL request, identifies the address of the next node in the path, divides the SQL request into several smaller packets,

and adds a sequence number to each packet to ensure that they are assembled in the correct order.

- Next the packet is handed to the data-link layer. This layer adds more control information, that depends on the network and on which physical media are used. The data-link layer sends the frame to the next node.
- When the data-link layer determines that it is safe to send a frame, it hands the frame down to the physical layer, which transmits it into a collection of ones and zeros(bits), and then transmit the bits through the network cable.
- The signals transmitted by the physical layer are received at the server end at the physical layer, which passes the data to the data-link layer. The data-link layer reconstructs the bits into frames and validates them. At this point, the data-link layer of the client and server computer may exchange additional messages to verify that the data were received correctly and that no retransmission is necessary. The packet is sent up to the network layer.
- The network layer checks the packet's destination address. If the final destination is some other node in network, the network layer identifies it and sends the packet down to the data-link layer for transmission to that node. If the destination is the current node, the network layer assembles the packets and assigns appropriate sequence numbers. Next, the network layer generates the SQL request and sends it to the transport layer.
- Most of the Client/Server "conversation" takes place in the session layer. If the communication between client and server process is broken, the session layer tries to reestablish the session. The session layer identifies and validates the request, and sends it to the presentation layer.
- The presentation layer provides additional validation and formatting.
- Finally, the SQL request is sent to the database server or application layer, where it is executed.

Although the OSI framework helps us understand network communications, it functions within a system that requires considerable infrastructure. The network protocols constitute the core of network infrastructure, because all data travelling through the network must adhere to some network protocol. In the Client/Server environment, it is not usual to work with several different network protocols. In the previous section, we noted that different server processes might support different network protocols to communicate over the network.

For example, when several processes run on the client, each process may be executing a different SQL request, or each process may access a different database server. The transport layer ID helps the transport layer identify which data corresponds to which session.

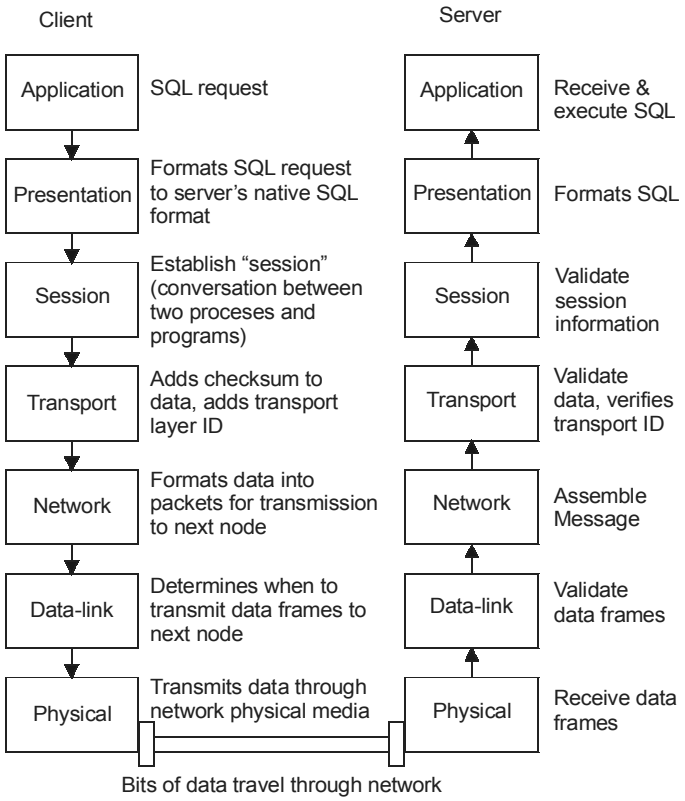

Fig.3.7: Flow of Information through the OSI Model

3.7 ARCHITECTURE FOR BUSINESS INFORMATION SYSTEM

3.7.1 Introduction

In this section, we will discuss several patterns for distributing business information systems that are structured according to a layered architecture. Each distribution pattern cuts the architecture into different client and server components. All the patterns discussed give an answer to the same question: How do I distribute a business information system? However, the consequences of applying the patterns are very different with regards to the forces influencing distributed systems design. Distribution brings a new design dimension into the architecture of information systems. It offers great opportunities for good systems design, but also complicates the development of a suitable architecture by introducing a lot of new design aspects and trap doors compared to a centralized system. While

constructing the architecture for a business information system, which will be deployed across a set of distributed processing units (e.g., machines in a network, processes on one machine, threads within one process), you are faced with the question:

How do I partition the business information system into a number of client and server components, so that my users' functional and non-functional requirements are met?

There are several answers to this question. The decision for a particular distribution style is driven by users' requirements. It significantly influences the software design and requires a very careful analysis of the functional and non-functional requirements.

3.7.2 Three-Layer Architecture

A Business Information System, in which many (spatially distributed) users work in parallel on a large amount of data. The system supports distributed business processes, which may span a single department, a whole enterprise, or even several enterprises. Generally, the system must support more than one type of data processing, such as On-Line Transaction Processing (OLTP), off-line processing or batch processing. Typically, the application architecture of the system is a *Three-Layer Architecture*, illustrated in Fig. 3.8.

The user interface handles presentational tasks and controls the dialogue the application kernel performs the domain specific business tasks and the database access layer connects the application kernel functions to a database. Our distribution view focuses on this coarse-grain component level. In developing distributed system architecture we mainly use the *Client/Server Style.* Within these model two roles, client and server classify components of a distributed system. Clients and servers communicate via a simple request/response protocol.

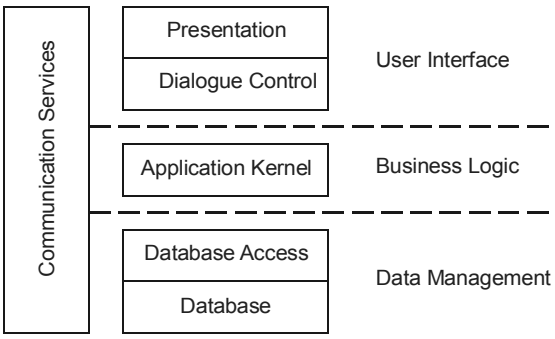

Fig.3.8: Three-Layer Architecture for Business Information System

3.7.3 General Forces

- *Business needs vs. construction complexity:* On one hand, allocating functionality and data to the places where it is actually needed supports distributed business processes

very well, but on the other hand, distribution raises a system's complexity. Client server systems tend to be far more complex than conventional host software architectures. To name just a few sources of complexity: GUI, middleware, and heterogeneous operating system environments. It is clear that it often requires a lot of compromises to reduce the complexity to a level where it can be handled properly.

- *Processing style:* Different processing styles require different distribution decisions. Batch applications need processing power close to the data. Interactive processing should be close to input/output devices. Therefore, off-line and batch processing may conflict with transaction and on-line processing.
- *Distribution vs. performance:* We gain performance by distributed processing units executing tasks in parallel, placing data close to processing, and balancing workload between several servers. But raising the level of distribution increases the communication overhead, the danger of bottlenecks in the communication network, and complicates performance analysis and capacity planning. In centralized systems the effects are much more controllable and the knowledge and experience with the involved hardware and software allows reliable statements about the reachable performance of a configuration.
- *Distribution vs. security:* The requirement for secure communications and transactions is essential to many business domains. In a distributed environment the number of possible security holes increases because of the greater number of attack points. Therefore, a distributed environment might require new security architectures, policies and mechanisms.
- *Distribution vs. consistency:* Abandoning a global state can introduce consistency problems between states of distributed components. Relying on a single, centralized database system reduces consistency problems, but legacy systems or organizational structures (off-line processing) can force us to manage distributed data sources.
- *Software distribution cost:* The partitioning of system layers into client and server processes enables distribution of the processes within the network, but the more software we distribute the higher the distribution, configuration management, and installation cost. The lowest software distribution and installation cost will occur in a centralized system. This force can even impair functionality if the software distribution problem is so big that the capacities needed exceed the capacities of your network. The most important argument for so called diskless, Internet based network computers is exactly software distribution and configuration management cost.
- *Reusability vs. performance vs. complexity:* Placing functionality on a server enforces code reuse and reduces client code size, but data must be shipped to the server and the server must enable the handling of requests by multiple clients.

3.7.4 Distribution Pattern

To distribute an information system by assigning client and server roles to the components of the layered architecture we have the choice of several distribution styles. Figure 3.9 shows the styles, which build the pattern language. To take a glance at the pattern language we give an abstract for each pattern:

- *Distributed presentation:* This pattern partitions the system within the presentation component. One part of the presentation component is packaged as a distribution unit and is processed separately from the other part of the presentation, which can be packaged together with the other application layers. This pattern allows of an easy implementation and very thin clients. Host systems with 3270-terminals is a classical example for this approach. Network computers, Internet and intranet technology are modern environments where this pattern can be applied as well.
- *Remote user interface:* Instead of distributing presentation functionality the whole user interface becomes a unit of distribution and acts as a client of the application kernel on the server side.
- *Distributed application kernel:* The pattern splits the application kernel into two parts which are processed separately. This pattern becomes very challenging if transactions span process boundaries (distributed transaction processing).

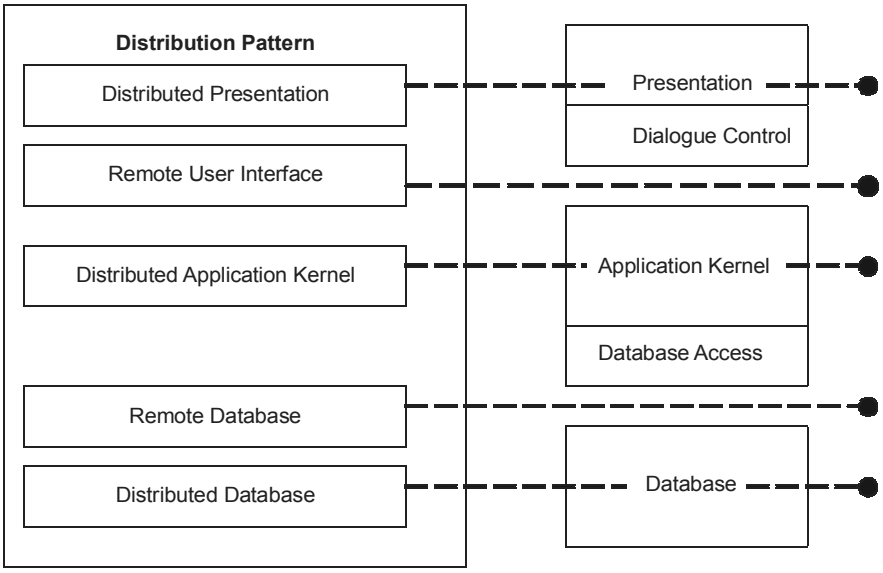

Fig.3.9: Pattern Resulting from Different Client/Server Cuts

- *Remote database:* The database is a major component of a business information system with special requirements on the execution environment. Sometimes, several applications work on the same database. This pattern locates the database component on a separate node within the system's network.

- *Distributed database:* The database is decomposed into separate database components, which interact by means of interprocess communication facilities. With a distributed database an application can integrate data from different database systems or data can be stored more closely to the location where it is processed.

3.8 EXISTING CLIENT/SERVER ARCHITECTURE

3.8.1 Mainframe-based Environment

In mainframe systems all the processing takes place on the mainframe and usually dumb terminals that are known as end user platform are used to display the data on screens.

Mainframes systems are highly centralized known to be integrated systems. Where dumb terminals do not have any autonomy. Mainframe systems have very limited data manipulation capabilities. From the application development point of view. Mainframe systems are over structured, time-consuming and create application backlogs. Various computer applications were implemented on *mainframe computers* (from IBM and others), with lots of attached (dumb, or semi-intelligent) terminals see the Fig. 3.10.

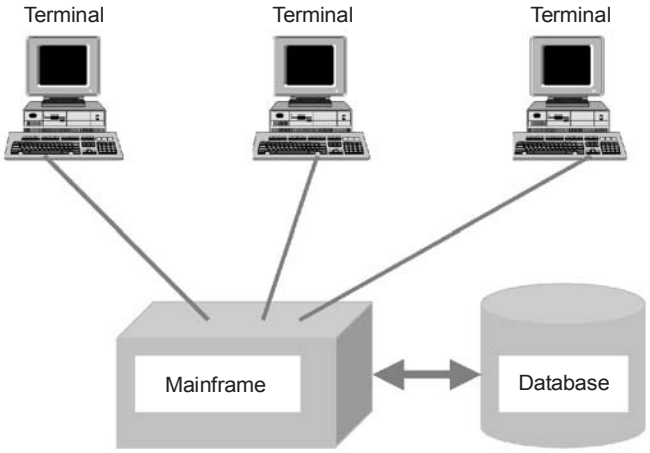

Fig.3:10: Mainframe-based Environment

There are some major problems with this approach:

- ⇨ Very inflexible.
- ⇨ Mainframe system are very inflexible.
- ⇨ Vendor lock-in was very expensive.
- ⇨ Centralized DP department was unable to keep up with the demand for new applications.

3.8.2 LAN-based Environment

LAN can be configured as a Client/Server LAN in which one or more stations called servers give services to other stations, called clients. The server version of network operating system is installed on the server or servers; the client version of the network operating system is installed on clients. A LAN may have a general server or several dedicated servers. A network may have several servers; each dedicated to a particular task for example database servers, print servers, and file servers, mail server. Each server in the Client/Server based LAN environment provides a set of shared user services to the clients. These servers enable many clients to share access to the same resources and enable the use of high performance computer systems to manage the resources.

A file server allows the client to access shared data stored on the disk connected to the file server. When a user needs data, it access the server, which then sends a copy, a print server allows different clients to share a printer. Each client can send data to be printed to the print server, which then spools and print them. In this environment, the file server station server runs a server file access program, a mail server station runs a server mail handling program, and a print server station a server print handling program, or a client print program.

Users, applications and resources are distributed in response to business requirements and linked by single Local Area Networks. See the Fig. 3.11 illustrated below:

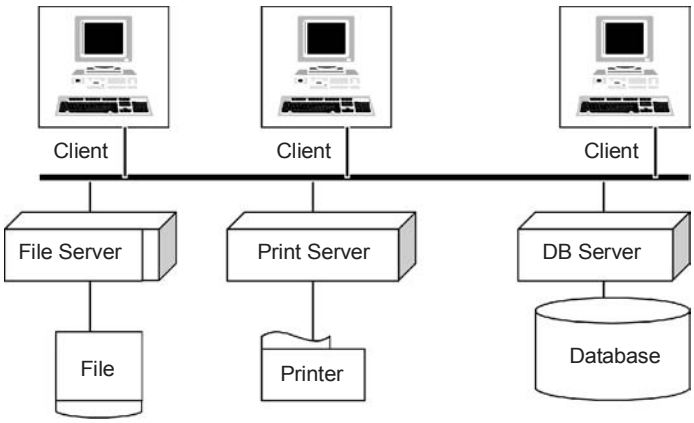

Fig.3.11: LAN Environment

3.8.3 Internet-based Environment

What the Internet brings to the table is a new platform, interface, and architectures. The Internet can employ existing Client/Server applications as true Internet applications, and integrate applications in the Web browser that would not normally work and play well together. The Internet also means that the vast amount of information becomes available

from the same application environment and the interface. That's the value. See the Fig. 3.12 given below:

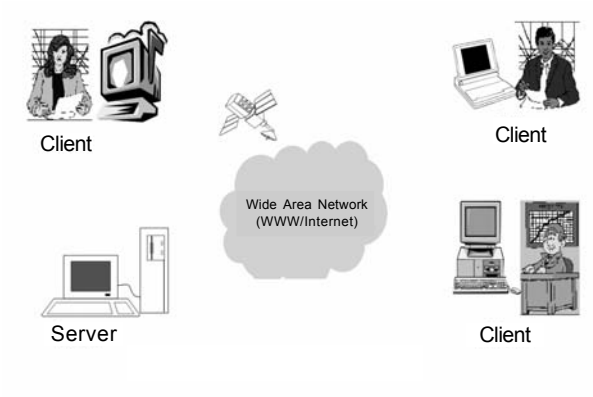

Fig.3.12: Internet-based Environment

The internet also puts fat client developers on a diet. Since most internet applications are driven from the Web server, the application processing is moving off the client and back onto the server. This means that maintenance and application deployment become much easier, and developers don't have to deal with the integration hassles of traditional Client/Server (such as loading assorted middleware and protocol stacks).

The web browsers are universal clients. A web browser is a minimalist client that interprets information it receives from a server, and displays it graphically to a user. The client is simply here to interpret the server's command and render the contents of an HTML page to the user. Web browsers-like those from Netscape and Spyglass – are primarily interpreters of HTML commands. The browser executes the HTML commands to properly display text and images on a specific GUI platform; it also navigates from one page to another using the embedded hypertext links. HTTP server produce platform independent content that clients can then request. A server does not know a PC client from a Mac client – all web clients are created equal in the eyes of their web server. Browsers are there to take care of all the platform-specific details.

At first, the Web was viewed as a method of publishing informaton in an attractive format that could be accessed from any computer on the internet. But the newest generation of the Web includes programmable clients, using such programming environments as Sun Microsystem's Java and Microsoft's ActiveX. With these programming environments, the Web has become a viable and compelling platform for developing Client/Server applications on the Internet, and also platform of choice for Client/Server computing. The World Wide Web (WWW) information system is an excellent example of client server "done right". A server system supplies multimedia documents (pages), and runs some application programs (HTML forms and CGI programs, for example) on behalf of the client. The client takes complete responsibility for displaying the hypertext document, and for the user's response to it. Whilst the majority of "real world" (i.e., commercial) applications of Client/Server are in database applications.

EXERCISE 3

1. What is the role of mainframe-centric model in Client/Server computing?
2. Explain Connectivity and Communication Interface Technology in Client/Server application. How does transmission protocol work in Client/Server application?
3. Explain Peer to Peer architecture. What is the basic difference between Client/Server and Peer to Peer Computing?
4. Draw the block diagram of Client/Server architecture and explain the advantage of Client/Server computing with the help of suitable diagram.
5. Explain shared tiered Client/Server architecture.
6. How are connectivity and interoperability between Client/Server achieved? Explain.
7. Explain Client/Server architecture. What is the basic difference between Client/Server and peer to peer computing?
8. Explain the three-level architecture of database management system. Also explain the advantages and disadvantages of DBMS.
9. Draw the block diagram of Client/Server architecture and explain the advantage of Client/Server computing with the help of suitable example.
10. What are the various ways available to improve the performance of Client/Server computing?