KTU NOTES

The learning companion.

KTU STUDY MATERIALS | SYLLABUS | LIVE NOTIFICATIONS | SOLVED QUESTION PAPERS

Module -2 (Client/Server Application Components)

Classification of Client/Server Systems- Two-Tier Computing, Middleware, Three-Tier Computing- Model View Controller (MVC), Principles behind Client/Server Systems. Client/Server Topologies. Existing Client/Server Architecture. Architecture for Business Information System.

CLASSIFICATION OF CLIENT SERVER SYSTEMS

Broadly, there are three types of Client/Server systems in existence.

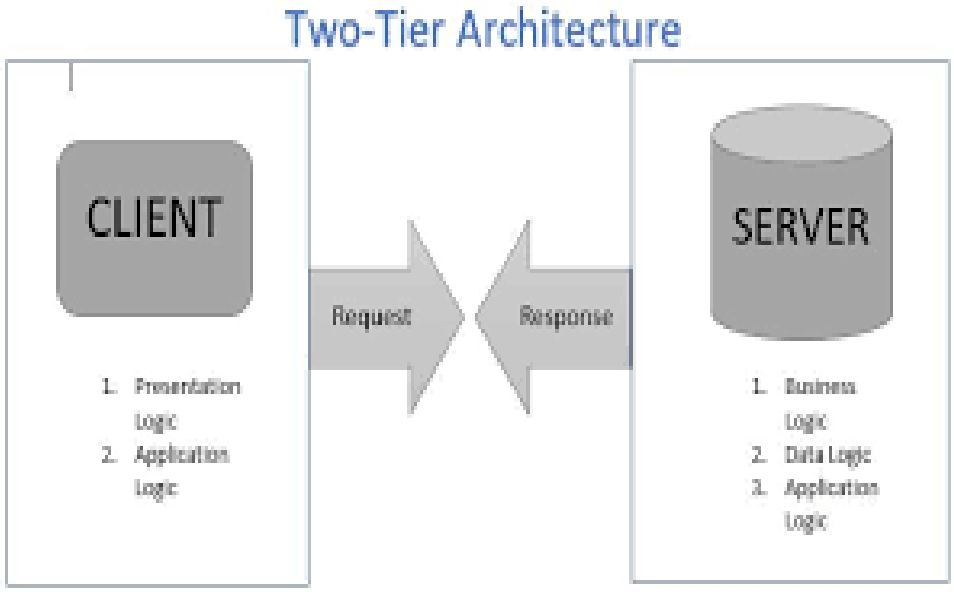
(i) Two-tier
(ii) Three-tier
(iii) N-Tier

A '**tier**' commonly refers to the logical or functional separation of software into layers on different physical locations or hardware.

Multi-tier architecture is a pattern of splitting software into several definite domains that handle particular aspects of the software such as presentation, logic or data management.

Two-tier Client/Server Model

The two-tier architecture is based on the Client-Server model.



It consists of Client-Application tier and Database tier.

The Client-Application server communicates directly with the Database server. Data or information transfer between the two components is fast due to the absence of a middleware.

- The Client-Application contains the codes for interfacing with the user and also for saving data in the database server.

- The Client-Application sends the request to the server and it processes the request and sends back with data. This means client Application handles both the Presentation layer (application interface) and the Application layer (logical operations).

- The client-Application layer can be build using languages such as C, C++, Java, Python, PHP, .NET.

- The database server handles the data management layer. Data management layer consists of data storage(database or file system) and methods for storing and retrieving data from the data storage. Commonly used databases include MySQL, MongoDB, SQLite.

Two-tiered application examples include desktop applications, games, and music players.

Advantages:

- It is fast and easy to implement
- communication is faster
- It is suitable in an environment where business rules or logic operations are static

Disadvantages:

- It is not easily scalable, thus performance degrades as users scale.
- Data integrity issue may arise due to the server responding to multiple requests at the same time.

- The architecture of any client/server environment is by definition at least a two-tier system, the client being the first tier and the server being the second.

The Client requests services directly from server i.e. client communicates directly with the server without the help of

another server or server process.

In a typical two-tier implementation, SQL statements are issued by the application and then handed on by the driver to the database for execution.

The results are then sent back via the same mechanism, but in the reverse direction. It is the responsibility of the driver to present the SQL statement to the database in a form that the database understands.

Three-tier Client/Server Model

- Reusability is hard to achieve if pieces of business logic must be distributed across systems and several databases are involved.
- To avoid embedding the application's logic at both the database side and the client side, a third software tier is inserted in between.
- In the three- tier architecture, most of the business logic is located in the middle tier (here business logic is encapsulated as a component in a separate tier).
- In this structure, when the business activity or business rules change, only the middle tier must be modified.
- In three-tier architecture application responsibilities are divided into three logical categories (in other words, the business system must provide three types of main services).

- Presentation (GUI) or user services: Include maintaining the graphical user interface and generating what users see on the monitor.
- Presentation Logic dealing with:
  - Screen formatting
  - Windows management
  - Input editing
- Application services or business rules: These include executing applications and controlling program flow.
  Business logic dealing with:

- Domain and range validation
- Data dependency validation
- Request/response architecture of Inter Process Communication level

- Database services or data server: Which refers to the management of underlying databases. Server logic deals with:
  - Data access
  - Data management
  - Data security
  - SQL parsing
  - Based on these three components, the three-tier architecture of Client/Server system is shown in fig. 1.8 below.

"In three-tier client/server system the client request are handled by intermediate servers which coordinate the execution of the client request with subordinate servers."

- In three-tier model, a third server is employed to handle requests from the client and then pass them off to the database server.
- The third server acts as proxy for all client requests. Or, in other words we can say:
-

All client requests for the database are routed through the proxy server, thereby creating a more secure environment for your database.

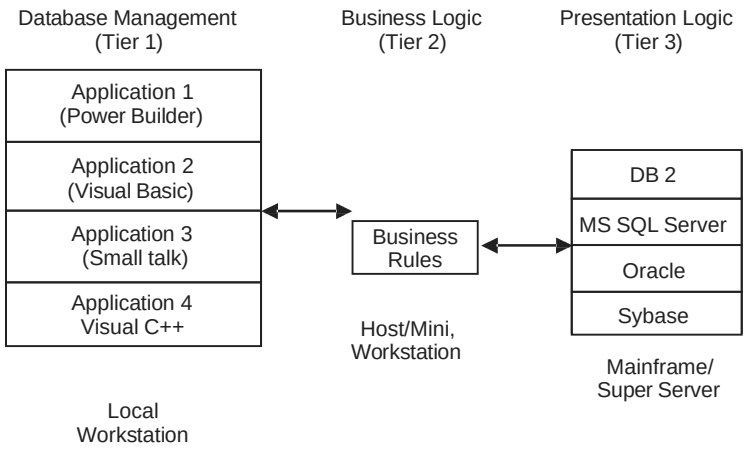


Fig.1.7: Three-tier Architecture

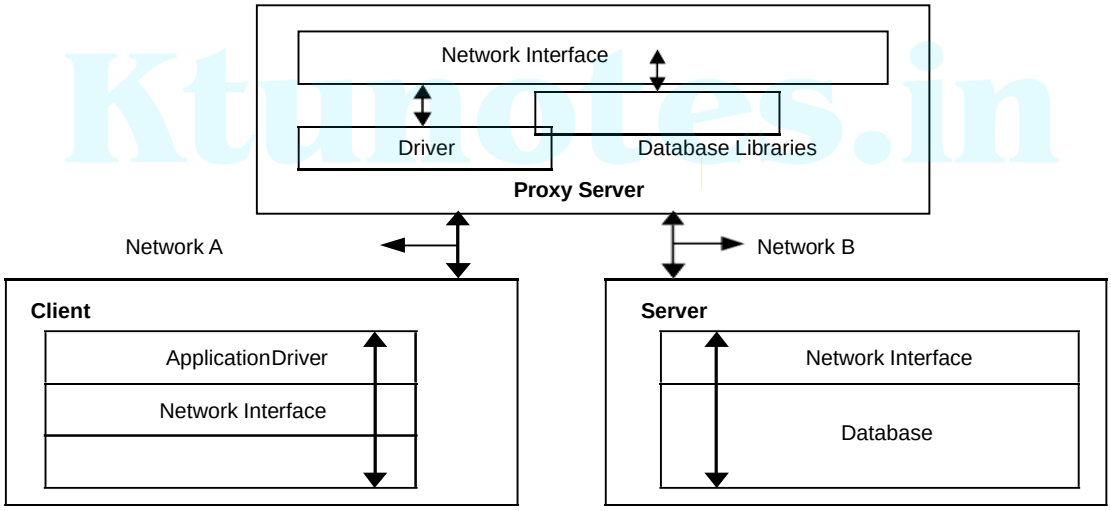


Fig.1.8: Three-tier Client/Server Model

First-tier (client-tier): The main responsibility of this tier is to receive user events and to control the user interface and presentation of data. As most of the software is removed from the client, the client is called "Thin Client". Mainly browser and presentation code resides on this tier.

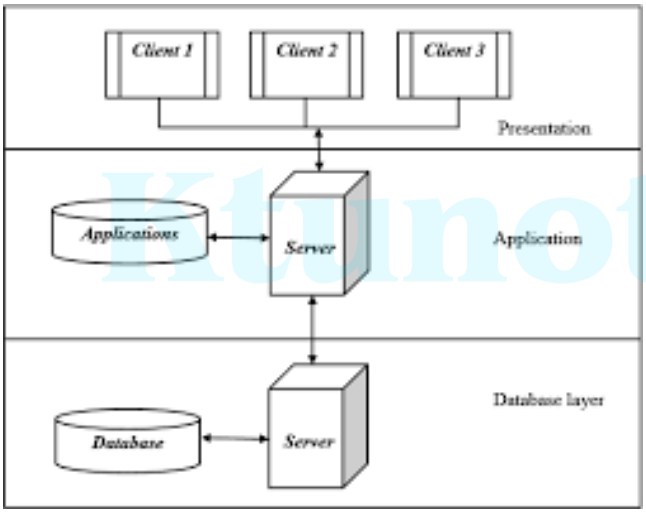
Second-tier (application-server-tier): The complex application logic is loaded here and available to the client tier on request from client. This level forms the central key

towards solving the 2-tier problem. This tier can protect direct access of data.

Three-tier (database-server-tier): This tier is responsible for data storage. This server mostly operates on a relational database.

. Some of the advantages of using three-tier model include:

- Application maintenance is centralized with the transfer of the business logic for many end users into a single application server. This eliminates the concern of software distribution that are problematic in the traditional two-tier Client/Server model.



- Clear separation of user-interface-control and data presentation from application- logic.
- Many users are able to access a wide variety of server applications, as all application logic are loaded in the applications server.
- As a rule servers are "trusted" systems. Data protection and security is simpler to obtain. Therefore, it makes sense to run critical business processes that work with security

sensitive data, on the server.

- Load balancing is easier with the separation of the core business logic from the database server.
- Dynamic load balancing: if bottlenecks in terms of performance occur, the server process can be moved to other servers at runtime.
- Business objects and data storage should be brought as close together as possible. Ideally, they should be together physically on the same server. This way network load for complex access can be reduced.
- The need for less expensive hardware because the client is 'thin'.
- Change management is easier and faster to execute.

- The added modularity makes it easier to modify or replace one tier without affecting the other tier.
- Drivers can be managed centrally.
- Your database server does not have to be directly visible to the Internet.

n-tier architecture

In an n-tier architecture, application objects are distributed across multiple logical tiers, typically three or four.

In a three-tier architecture, the database server does not share a server machine with the web application server.

In this approach, hardware and software components of the second and third tiers share responsibility for the availability, scalability, and performance characteristics of the web environment.

The three-tier architecture can be easily extended to N-tier, with additional tiers added to provide more flexibility and scalability. For example, the middle tier of the three-tier architecture could be split into two, with one tier for the Web server and another for the application server.

Some disadvantages are:

- The client does not maintain a persistent database connection.
- A separate proxy server may be required.
- The network protocol used by the driver may be proprietary.
- You may see increased network traffic if a separate proxy server is used.

The most common approach used when designing N-tier system is the three-tier architecture. Three-tier and N-tier notations are similar, although N-tier architecture provides finer-grained layers.

Architectures often decide to layout much more than three-tiers to deploy services (An infrastructure that supports three-tier is often made of several machines and services whose functionalities aren't part of the three-tier design).

(i) Figure 1.10 shown below depicts the N-tier architecture.

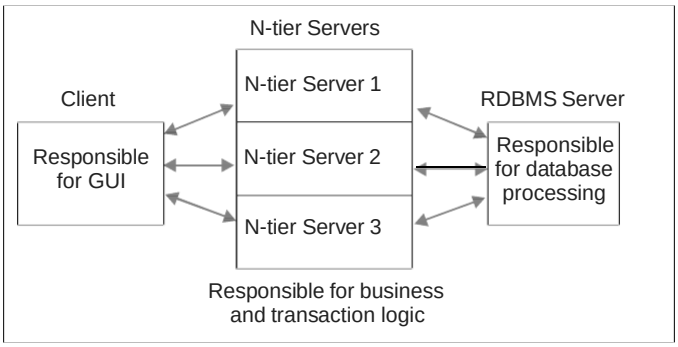


Fig.1.10: N-tier Architecture

N-tier computing provides many advantages over traditional two-tier or single-tier design, which includes the following:

- Overall performance has been improved.
- The business logic is centralized.
- Enhanced security level is attained.

Middleware: It is software that helps to connect the operating system and applications. Or it is software that runs between client and server processes. Generally, this software is written in such a way that the user never notices the presence of middleware. It also helps in delivering secure and transparent services to users.

Types of Middleware Services:

The different types of middleware services are RDA, RPC, and MOM.

- **RDA:** Remote Data Access(RDA), which implements an RDA protocol for sending data manipulation language statements to an appropriate database server for processing and transporting the result back to the invoking process.
- **RPC:** Remote procedure calls (RPCs). RPC is used in network operating system services.
- **MOM:** Message-oriented middleware(MOM) is used as a mechanism for storing and forwarding message queuing. It

helps when client and server processes communicate asynchronously.

Middleware is Useful For

1. Security
2. Transaction Management
3. Message Queues
4. Application Server
5. Web Server

Types of Middleware:

1. **Message Oriented Middleware:** It is an infrastructure that is helpful in sending and receiving messages over distributed applications / It helps applications to be paid over different platforms and makes the process of creating software applications time many OS and network protocols less complicated.

2. **Object-Oriented Middleware**: This method is also called an Object request broker. Object Middleware gives applications the ability to send objects and request services through an object-oriented system. Object Middleware main use is to manage the communication between objects.

3. **Remote Procedure Call (RPC) Middleware:** RPC is used to perform synchronous or asynchronous interactions between applications by calling procedure remotely and it is utilized within a software application.

4. **Database Middleware:** It allows direct interaction to access the database. In the market there are many database gateways and connectivity options, we can select anyone based on our requirements. It is one of the common types of middleware that includes SQL database software.

Role of Middleware in a Client-Server Architecture:

Middleware

1. In a client-server architecture, the most important entities are the clients and the browsers. A problem in this client-server architecture is that the client can be heterogeneous. For example, consider, our server is set up for a restaurant management case. In that, the client-side request can come from a mobile application, a web browser, or even from other applications. So our server should handle this by serving the client of different types separately.
2. Also sometimes the server needs to communicate with other software or entities. In that case to server the data to the client, the server needs to take some help from other applications. This application which helps the server to provide some help is known as middleware. Almost all the client-server architects will be having middleware. In the above example,(restaurant management) the data that the client sends should be stored in the server. For the structured storage and management of that data, we will be using a database. A database management system is another middleware that helps our server to handle the data.
3. Likewise, there are multiple middlewares that will perform single operations to help the server.

Two broad classes of middleware in the client-server environment:

1. **General middleware:**

The general middleware is one of the broad classes of middleware in client-server environments that includes the communication stacks, distributed directories, authentication services, network time, RPC, Queuing services along with the network OS extensions such as the distributed file and print services. The print services and distributed file services also come under this category

For Example, The distributed computing environment is an example of general middleware, where a common set of distributed services are available to the application.
The main services provided by distributed computing environment for supporting distributed applications are security services, RPC, time services, IDL compiler, Threading services, Directory service.

2. Service-specific middleware:
The service-specific middleware is another broad class of middleware in a client-server environment that needs to accomplish a particular Client/Server type of services which includes:

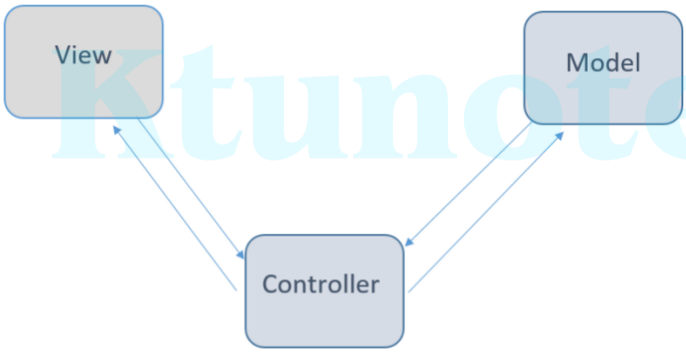
1. **Database-specific middleware**: It allows direct access to data structures and provides interaction with the database directly. For example, ORACLE, SQL, ODBC, etc.
2. **OLTP specific middleware**: It is a type of transaction scheduling, message queuing where a client connects to middleware and in turn connects to the database back end. For example, RPC, ATMI.
3. **Object-specific middleware**: It helps in the reusability and interoperability of distributed objects. For example, CORBA, Microsoft DCOM.
4. **Workflow management middleware**: It focuses on the management of activity flow in client or server systems.
5. **Internet-specific middleware**: It is one type of communication middleware consists of tools like HTTP, S-HTTP, and SSL.

6. **Message-oriented middleware**: It is used as a mechanism for storing and forwarding message queuing. It helps when client and server processes communicate asynchronously.

MVC Architecture

MVC is an architectural pattern which means it rules the whole architecture of the applications.

It has three main components:
-Model
-View
-Controller
and each of them has specific responsibilities



MVC Architecture

It helps to create a solid structure of our web applications.

Model

It is known as the lowest level which means it is responsible for maintaining data. Handle data logically so it basically deals with data.

The model is actually connected to the database so anything you do with data.

Adding or retrieving data is done in the model component.

It responds to the controller requests because the controller never talks to the database by itself.

The model talks to the database back and forth and then it gives the needed data to the controller.

Note: the model never communicated with the view directly.

View

Data representation is done by the view component. It actually generates UI or user interface for the user.

So at web applications when you think of the view component just think the Html/CSS part.

Views are created by the data which is collected by the model component but these data aren't taken directly but through the controller, so the view only speaks to the controller.

Controller

It's known as the main man because the controller is the component that enables the interconnection between the views and the model so it acts as an intermediary.

The controller doesn't have to worry about handling data logic, it just tells the model what to do.

After receiving data from the model it processes it and then it takes all that information it sends it to the view and explains how to represent to the user. Note: Views and models can not talk directly.

Advantages of MVC

- MVC architecture will separate the user interface from business logic
- Components are reusable.
- Easy to maintain.

- Different components of the application in MVC can be independently deployed and maintained.
- This architecture helpt to test components independently.

Disadvantages of MVC

-The complexity is high.
-Not suitable for small applications.
-The inefficiency of data access in view.

Conclusion

MVC is an architecture that divides your software into smaller components. The model deals with data and the logic of our system. The view only displays data and the controller maintains the connection between the model and the view. This 'division' enables readability and modularity as well it easier the testing part.

Just keep in mind these key points:
-MVC is an architectural pattern consisting of three parts: Model, View, Controller.
Model: Handles data logic.
View: It displays the information from the model to the user.
Controller: It controls the data flow into a model object and updates the view whenever data changes.

- main problem is complexity.

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 serversides) 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.

- **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.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. Windows98 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 hardwareand 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 servicesmay 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 createthe front-end application. Most front-end applications are GUI-based to hide thecomplexity of the Client/Server components from the end user. The Fig. 3.5 given below illustrates the basic client components.

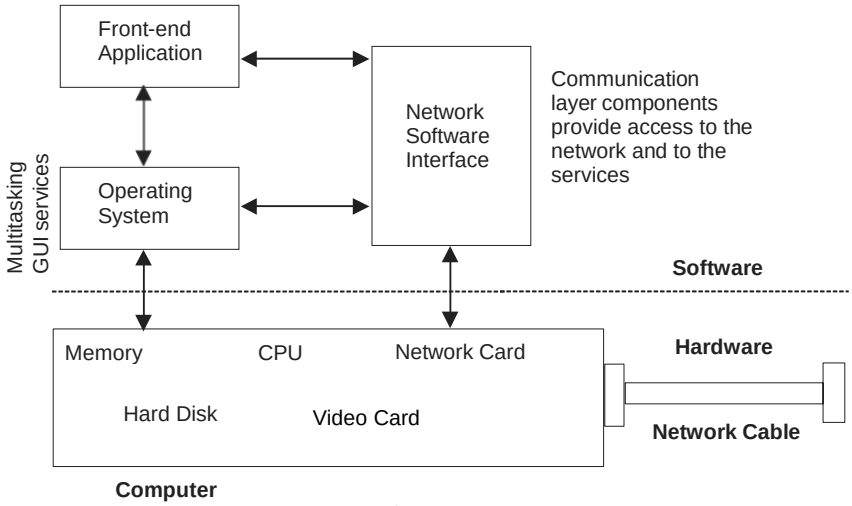


Fig.3.5: *Client Components*

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 storedon 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 bethe 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 needsdo, 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.

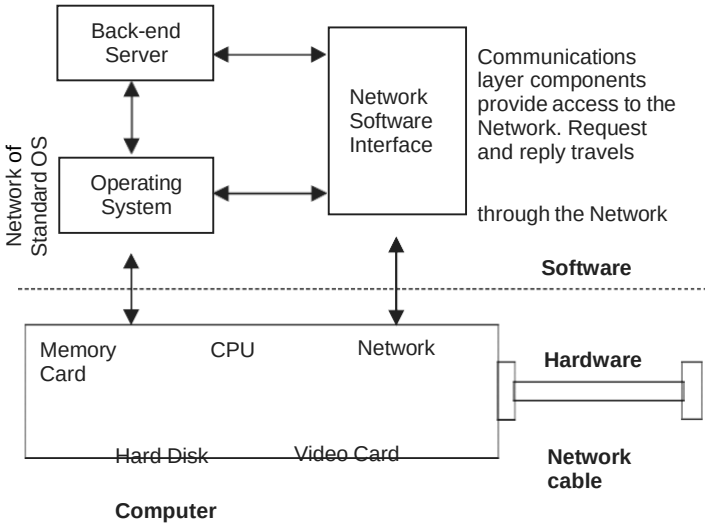


Fig.3.6: Server Components

- 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.

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 thatthe 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 responsibilityfor 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 systemprogrammer 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."***

ARCHITECTURE FOR BUSINESS INFORMATION SYSTEM

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.

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.

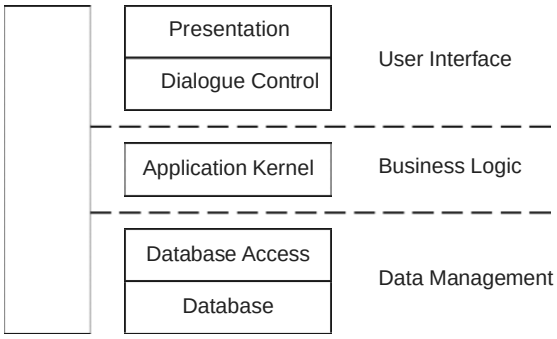


Fig.3.8: Three-Layer Architecture for Business Information System

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 every 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 processingmay 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 softwaredistribution problem is so big that the capacities needed exceed the capacities ofyour 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 serverand the server must enable the handling of requests by multiple clients.

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, whichcan 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).

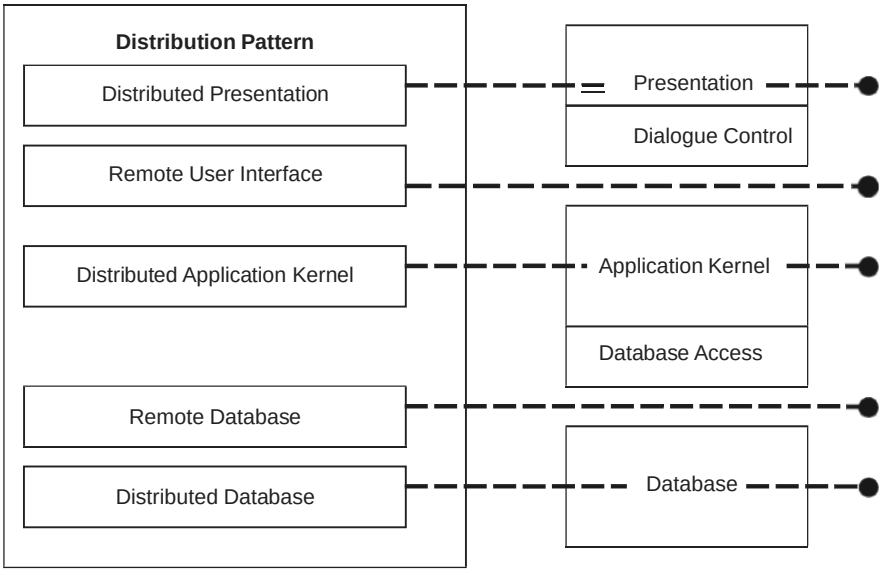


Fig.3.9: Pattern Resulting from Different Client/Server Cuts

- *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

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.

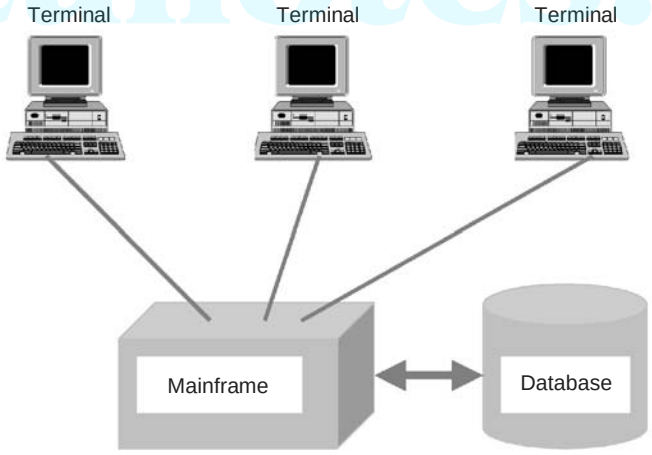


Fig.3:10: Mainframe-based Environment

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.

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:

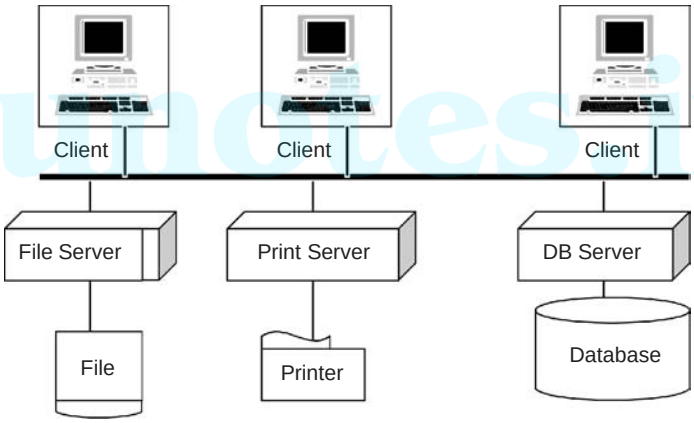


Fig.3.11: LAN Environment

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:

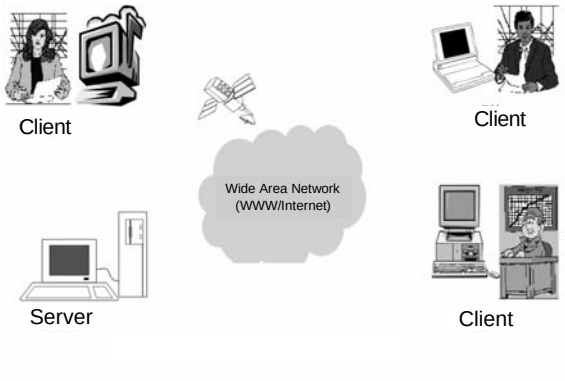


Fig.3.12: Internet-based Environment

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 andback 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 thatinterprets 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 anHTML 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 Macclient – 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.