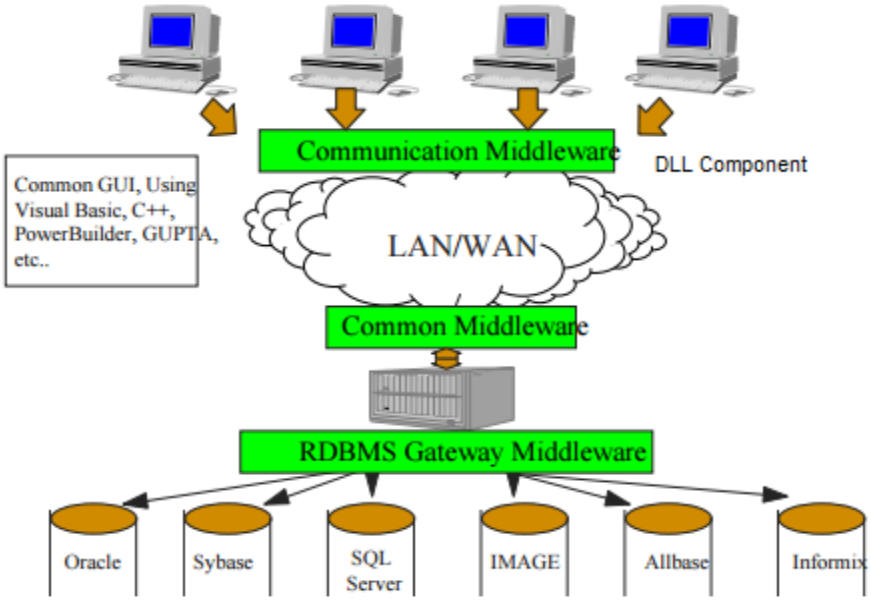
5.2 Basic view of Middle Ware

Middleware is software that runs between client and server processes. It is the "**GLUE**" between the client and server, which makes it possible for them to communicate to each other. *Middleware is written in such a way that the user never notices it's presence. It delivers secure and transparent services to users.* Connectivity software that joins applications thru communication mechanisms creating transparency, scalability, & interoperability.

" Middleware Lies between Applications Software & Platform "

Some of the **middleware services are :-**

- **Remote Data Access (RDA):** which implements a RDA protocol *for sending, data manipulation language statements to an appropriate database server for processing and transporting the result back to the invoking process*.
- **Remote procedure calls (RPCs):** that allows a program on one computer to execute a program on a server computer. The client program sends a message to the server with appropriate arguments and the server returns a message containing the results of the program executed.
- **Message-oriented middleware (MOM):** MOM can be *used as a mechanism for storing and forwarding messages queuing.*
- **Distributed transaction processing (DTP):** This type of mechanism *use execution semantics to interact between the client and the server.*



Software that connects two otherwise separate applications. *For example, there are a number of middleware products that link a database system to a Web server. This allows users to request data from the database using forms displayed on a Web browser, and it enables the Web server to return dynamic Web pages based on the user's requests and profile.*

The term *middleware* is used to describe separate products that serve as the **glue** between two applications. It is, therefore, distinct from import and export features that may be built into one of the applications. Middleware is sometimes called *plumbing* because it connects two sides of an application and passes data between them

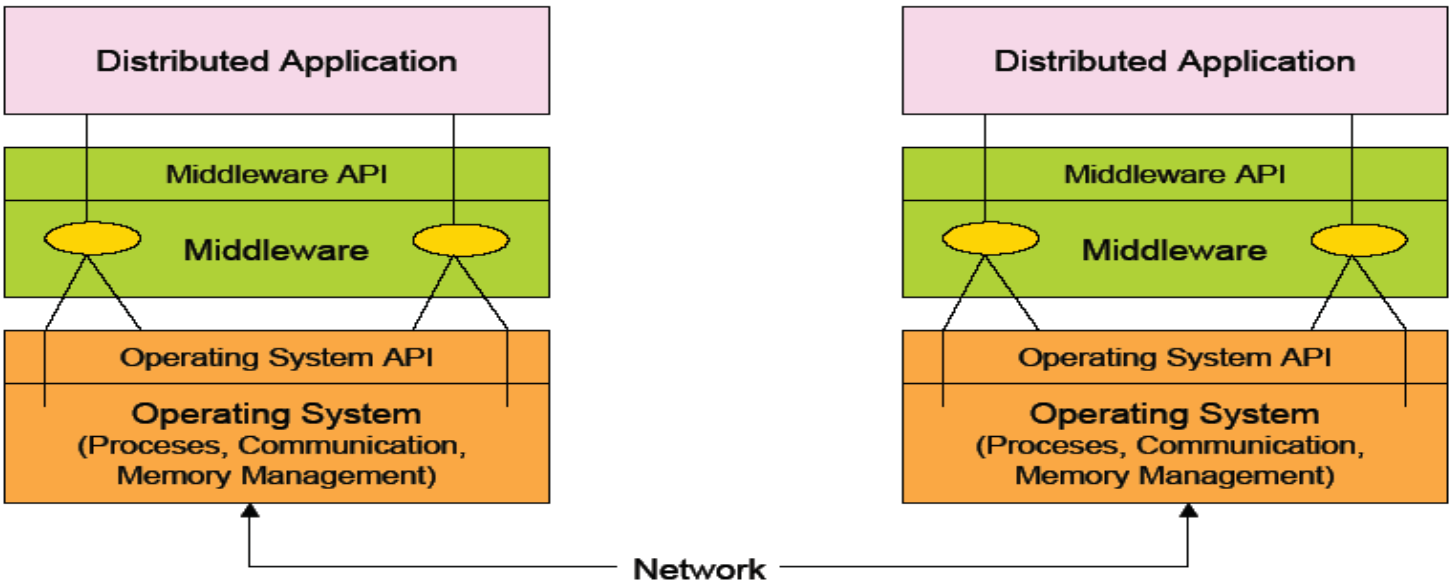


Fig. The Middleware Layer

5.2.1 General Characteristics

- Complement project, easier to design, develop, & maintain
- Interoperability, reliability, efficiency throughput
- Secure, dynamic, adaptable, scalable, available, fault-tolerant
- Flexible, portable, maintainable, reusable
- The primary role of middleware systems is to transfer data between applications, typically between Frontend and Backend applications, one or other of which, may be packaged applications. The data transfer may be in one, or both directions.
- Middleware applications decouple the Frontend and Backend applications in order to enable changes to the Backend systems to be transparent to the Frontend applications and vice versa.
- The primary functions of the middleware software are data generation, validation, authorization, conversion, re-formatting, transaction mapping and logging. They add value to transactions that are initiated by Frontend systems and are intended for processing on Backend systems (and vice versa).
- Application to application interfaces represent the majority of business functions in a middleware application.
- Middleware applications receive input from external applications and provide a response, based upon internal processing performed or responses received from destination applications
- Middleware processes tend to comprise multiple sub-processes.
- Middleware applications usually support online, real-time processing, as opposed to batch processing.

5.4 The main types of Middleware,

5.4.1 DCE (Distributed Computing Environment)

DCE (Distributed Computing Environment) is an industry-standard software technology for setting up and managing computing and data exchange in a system of distributed computers. DCE is typically used in a larger network of computing systems that include different size servers scattered geographically. DCE uses the client/server model. *Using DCE, application users can use applications and data at remote servers. Application programmers need not be aware of where their programs will run or where the data will be located.*

DCE was developed by the **Open Software Foundation (OSF)** using software technologies contributed by some of its member companies.

1. Computer networking scheme in which multiple software components (such as directory services, file services, security services) integrated to work closely toward well developed objectives. These objectives may include building of custom applications or providing of support to other applications.

2. Open system foundation's (OSF) vendor-independent standard technology that supports heterogeneous hardware and software products, and provides (1) security services to protect and control access to data, (2) name services that facilitate finding the distributed resources, and (3) a scalable scheme for organizing scattered data, services, and users.

Security In DCE : A Security Infrastructure That Provide: Authorization , Authentication , Encryption At Application Selectable Levels and Central Security Database.

DCE is based on three distributed computing models:

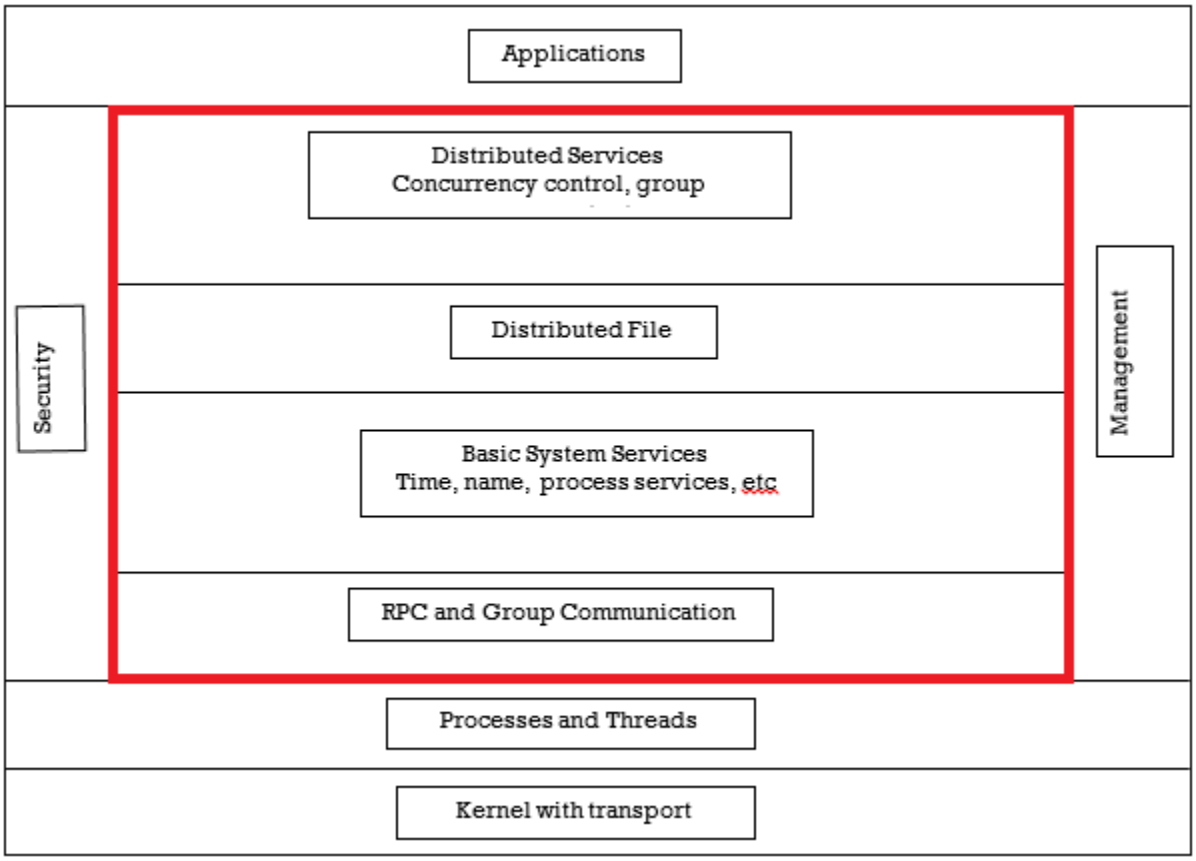
I. **Client/server:** A way of organizing a distributed application
II. **Remote procedure call (RPC):** A way of communicating between parts of a distributed application
III. **Shared files:** A way of handling data in a distributed system, based on a personal computer file access model.

5.4.1.1 Components of DCE

Major components of DCE within every cell are:

1. The **Security Server** that is responsible for authentication
2. The **Cell Directory Server** (CDS) that is the repository of resources and ACLs and
3. The **Distributed Time Server** that provides an accurate clock for proper functioning of the entire cell

- Modern DCE implementations such as IBM's are fully capable of interoperating with Kerberos as the security server, LDAP for the CDS and the Network Time Protocol implementations for the time server.
- Users, machines, and other resources of a DCE system are grouped to form a *cell*--administrative units.
- Lowest layer, hardware, and then operating systems. DCE requires a thread implementation to run. If the underlying OS does not support threads, then there is a DCE threads package that can be used. The next layer is an RPC layer, which includes authentication services so that authenticated RPC can be used.



- **Applications:** any software used by clients.
- **Security Services** : protection of shared resources via authentication
- **Transaction services** : for concurrent data access/update
- **Distributed services:** concurrency control used to address conflicts with the simultaneous accessing or altering of data that can occur with a multi-user system i.e. access the same data or data set are executed concurrently with time overlap.
- **Distributed File:** allow users of physically distributed computers to share data and storage resources by using a common file system. A typical configuration for a DFS is a collection of workstations and mainframes connected by a local area network (LAN). A DFS is implemented as part of the operating system of each of the connected computers.
- **Time Service:** A service to keep the clocks on the different machines "almost synchronized"--difficult to be exact. Working off a common clock is one of the distinguishing characteristics between tighly-coupled multiprocessors and loosely coupled distributed systems. At least want to get within a few seconds (distributed make example).
- **System management services** : service monitoring, management, and administration
- **Threads are part of process**. i.e. one process can spawn multiple Threads. If you run a Java program in UNIX based system e.g. Linux and if that program creates 10 Threads, it still one process.

Process : is an executing instance of an application . They run in separate memory space unlike threads
Threads : are path of execution within a process . Threads within the same process runs in a shared memory space .
The **kernel** is the central module of an operating system (OS). It is the part of the operating system that loads first, and it remains in main memory

5.4.1.2 Application of DCE

In the enterprise, distributed computing has often meant putting various steps in business processes at the most efficient places in a network of computers. For example, *in the typical distribution using the 3-tier model, user interface processing is performed in the PC at the user's location, business processing is done in a remote computer, and database access and processing is conducted in another computer that provides centralized access for many business processes.* Typically, this kind of distributed computing uses the client/server communications model.
The Distributed Computing Environment (DCE) is a widely-used industry standard that supports this kind of distributed computing. On the Internet, third-party service providers now offer some generalized services that fit into this model.
Much of DCE setup requires the preparation of distributed directories so that DCE applications and related data can be located when they are being used. DCE includes security support and some implementations provide support for access to popular databases such as IBM's CICS, IMS, and DB2 databases.

- ✓ **Distributed Services :** Concurrency control, group management, etc
- ✓ **Distributed File Service**
- ✓ **Basic System Services :** Time, name, process services, etc
- ✓ **RPC and Group Communication**

5.4.2 MOM (Message Oriented Middleware) : *communication via message exchange*

Message-oriented middleware (MOM) is middleware where transactions or event notifications are delivered between dissimilar systems or components by way of messages, often via an enterprise messaging system. *With MOM, messages sent to the client are collected and stored until they are acted upon, while the client continues with other processing.*
Context: The application components of a Distributed Application are hosted on multiple cloud resources and have to exchange information with each other. Often, the integration with other cloud applications and non-cloud applications is also required.
Solution : Communication partners exchange information asynchronously using messages. The message-oriented middleware handles the complexity of addressing, availability of communication partners and message format transformation.

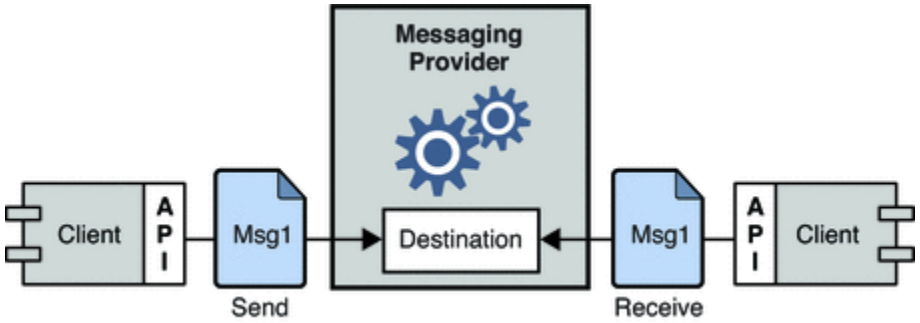


Figure 1–2 MOM-Based System

Message Oriented Middleware makes use of messaging provider to mediate messaging operations. The basic elements of a MOM system are clients, messages, and the MOM provider, which includes an API and administrative tools. The MOM provider uses different architectures to route and deliver messages: it can use a centralized message server or it can distribute routing and delivery functions to each client machine. Some MOM products combine these two approaches.
Using a MOM system, a client makes an API call to send a message to a destination managed by the provider. The call invokes provider services to route and deliver the message. Once it has sent the message, the client can continue to do other work, confident that the provider retains the message until a receiving client retrieves it.

5.4.2.1 Working Mechanism of MOM

Message-Oriented Middleware – Advantages

* ***Asynchronous Messaging:*** Message-oriented middleware comprises a category of inter-application communication software that usually relies on asynchronous message-passing, as opposed to a request-response architecture. In case of asynchronous systems, message queues provide temporary storage when the destination program is busy or unable to get connected. Apart from this, most asynchronous MOM systems provide persistent storage to back up the message queue which means sender and receiver does not necessarily connect to the network at the same time (asynchronous delivery), and problems with irregular connectivity are also solved. If the receiver application fails for any reason, the senders can continue uninterrupted, as messages which are sent will automatically accumulate in the message queue for processing when the receiver restarts.

**Routing :* Many message-oriented middleware implementations depend on a message queue system. Most implementations have routing logic provided by the messaging layer itself, while very few depend on client applications to provide routing information or allow both paradigms. Some implementations make use of broadcast or multicast distribution paradigms.

**Transformation :* In a message-based middleware system, the message received at the destination need not be identical to the message originally sent. A MOM system with built-in intelligence can transform messages route to match the requirements of the sender or of the recipient. In conjunction with the routing and broadcast/multicast facilities, one application can send a message in its own native format, and two or more other applications may each receive a copy of the message in their own native format. Many modern MOM systems provide sophisticated message transformation (or mapping) tools which allow programmers to specify transformation rules applicable to a simple GUI drag-and-drop operation.

MOM – Implementation

The message queue is a fundamental concept within MOM. *Queues provide the ability to store messages on a MOM platform. MOM clients are able to send and receive messages to and from a queue. Queues are central to the implementation of the asynchronous interaction model within MOM.* Usually the messages contained within a Queue is sorted in a particular order. The standard queue found in a messaging system is the First-In First-Out (FIFO) queue; as the name suggests, the first message sent to the queue is the first message to be retrieved from the queue

MOM – Messaging Models

A solid understanding of the available messaging models within MOM is key to appreciate the unique capabilities it provides. Two main message models that are commonly available, the point-to-point and publish/subscribe models. Both of these models are based on the exchange of messages through a channel (queue). A typical system will utilize a mix of these models to achieve different messaging objectives.

Point-to-Point Model : The point-to-point messaging model provides a straightforward asynchronous exchange of messages between software entities. In this model, messages from producing clients are routed to consuming clients via a queue. As discussed earlier, the most common queue used is a FIFO queue, in which messages are sorted in the order. While there is no restriction on the number of clients who can publish to a queue, there is usually only a single consuming client, although this is not a strict requirement. The model allows multiple receivers to connect to the queue, but only one of the receivers will consume the message. In the point-to-point model, messages are always delivered and will be stored in the queue until a consumer is ready to retrieve them.

Publish/Subscribe Model : The publish/subscribe messaging model, is a mechanism used to disseminate information between anonymous message consumers and producers. These one-to-many and many-to-many distribution mechanisms allow a single producer to send a message to one user or potentially hundreds of thousands of consumers. It only needs to send the information to a destination within the publish/subscribe engine. The engine will then send it to the consumer. *Clients producing messages "publish" to a specific topic or channel, these channels are then "subscribed" to by clients wishing to consume messages.*

5.4.2.2 Application of MOM

Enterprise-level messaging platforms will usually come with a number of built-in services for transactional messaging, reliable message delivery, load balancing, and clustering. If the distributed systems are geographically dispersed deployments with poor network connectivity and stringent demands in reliability, flexibility, and scalability, then MOM is the ideal solution. MOM-based systems are proficient in coping with traffic bursts while offering a flexible and robust solution for disperse deployments. Remote systems do not need to be available for the calling program to send a message. MOM also provides an abstract interface for communications. When MOM is used in conjunction with XML messages and web services, we are able to create highly flexible service oriented architectures. This form of architecture allows for the flexible integration of multiple systems.

5.4.3 Transaction processing (TP) Monitors : *two-phase commit for distributed transactions*

A transaction processing monitor (TPM) is a program that monitors transactions from one stage to the next, ensuring that each one completes successfully; if not, or if an error occurs, the TM Monitor takes the appropriate action. A transaction processing monitor's main purpose/objective is to allow resource sharing and assure optimal use of the resources by applications i.e. Load Balancing.

TP monitors are especially important in three-tier architectures that employ load balancing because a transaction may be forwarded to any of several servers. In fact, many TP monitors handle all the load balancing operations, forwarding transactions to different servers based on their availability.

A TP Monitor is a subsystem that groups together, set of related database updates and submits them together to a relational database. The result is that the database server does not need to do all of the work of managing the consistency/correctness of the database; the **TP Monitor** makes sure that groups of updates take place together or not at all. The advantages of this include increased system robustness as well as throughput.

- TP monitors initially developed as multithreaded servers to support large numbers of terminals from a single process.
- Provide infrastructure for building and administering complex transaction processing systems with a large number of clients and multiple servers.

Provide services such as:

- Presentation facilities to simplify creating user interfaces
- Persistent *queuing of client requests and server responses*
- Routing of client messages to servers
- Coordination of *two-phase commit when transactions access multiple servers*.

A **Transaction Processing System (TPS)** is a type of information system that collects, stores, modifies and retrieves the data transactions of an enterprise. *Transaction processing systems also attempt to provide predictable response times to requests, although this is not as critical as for real-time systems. Rather than allowing the user to run arbitrary programs as time-sharing, transaction processing allows only predefined, structured transactions*. Each transaction is usually short duration and the processing activity for each transaction is programmed in advance.

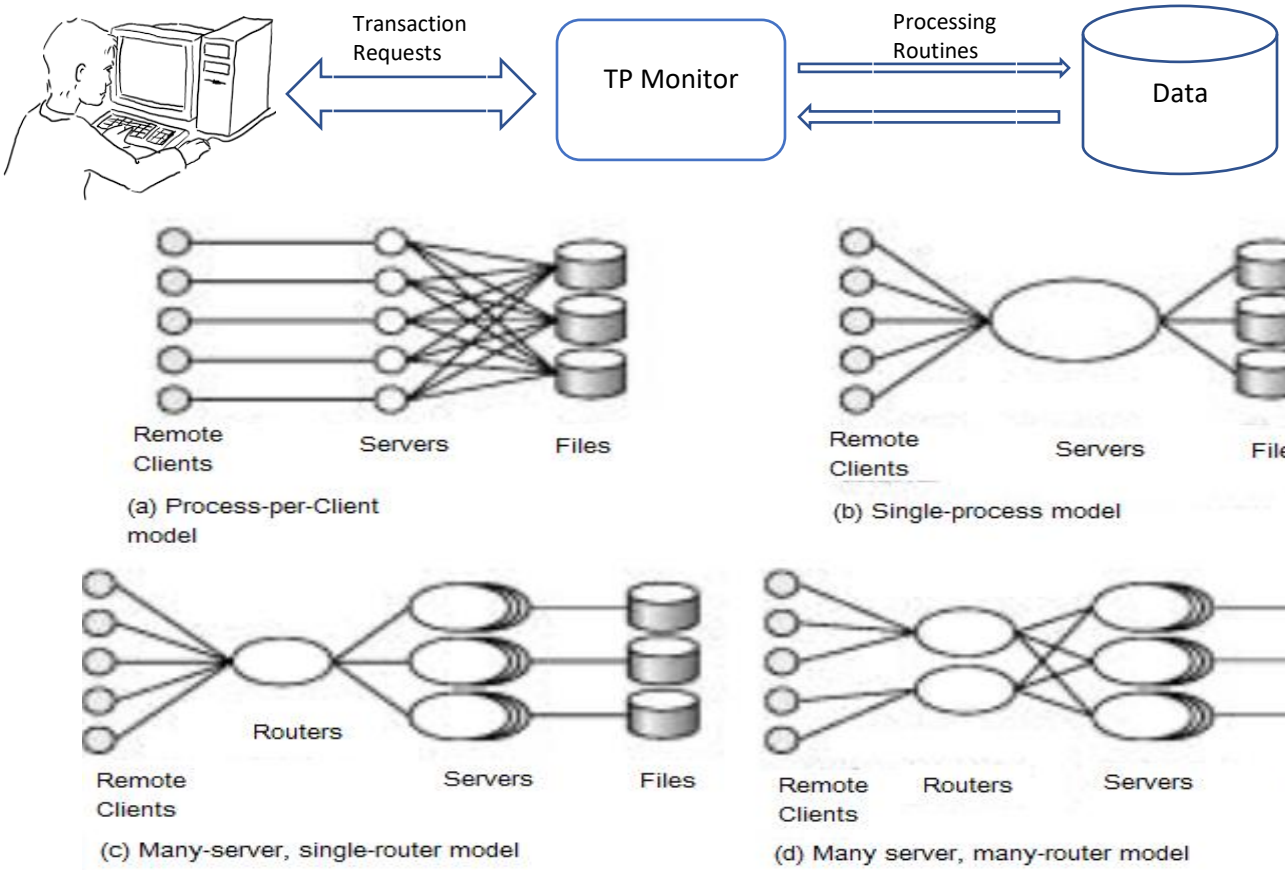


Fig. TP Monitor Architecture

(a)Process per client model - instead of individual login session per terminal, server process communicates with the terminal, handles authentication, and executes actions.

- Memory requirements are high
- Multitasking- high CPU overhead for context switching between processes

(b)Single process model - all remote terminals connect to a single server process.

- Used in client-server environments
- Server process is multi-threaded; low cost for thread switching
- No protection between applications
- Not suited for parallel or distributed databases

(c)Many-server single-router model - multiple application server processes access a common database; clients communicate with the application through a single communication process that routes requests.

- Independent server processes for multiple applications
- Multithread server process
- Run on parallel or distributed database

(d)Many server many-router model - multiple processes communicate with clients.

- Client communication processes interact with router processes that route their requests to the appropriate server.
- Controller process starts up and supervises other processes.

5.4.3.1 Working Mechanism (ACID) of TPM

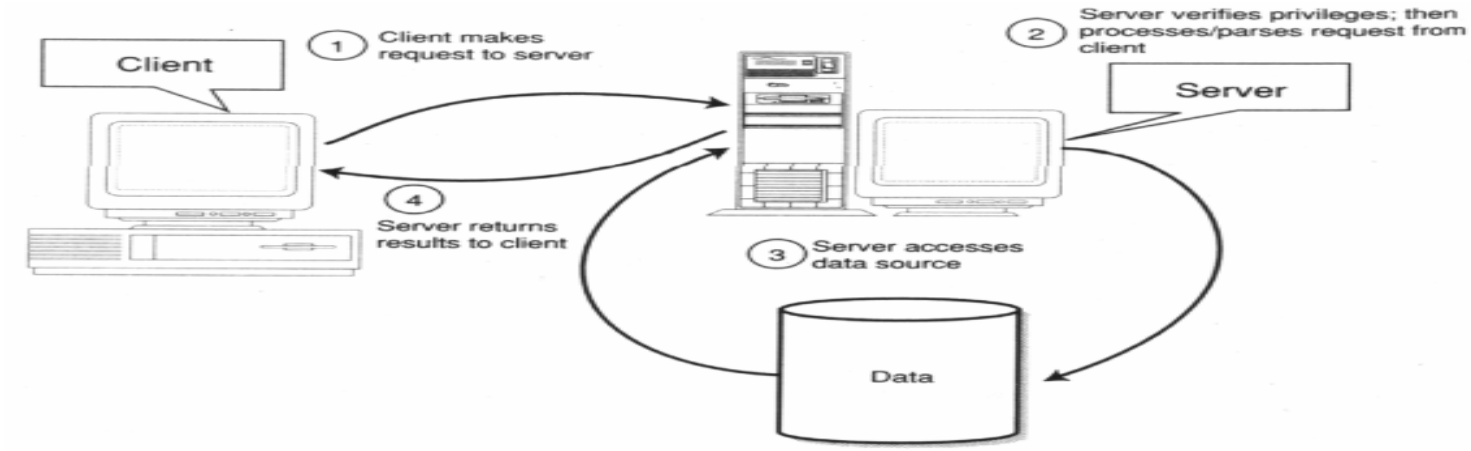
- **Atomicity** – (ALL or NONE) All transactions are either performed completely - *committed,* or are not done at all; a partial transaction that is aborted must be *rolled back*.. *E.g. a transaction to transfer funds from one account to another involves making a withdrawal operation from the first account and a deposit operation on the second. If the deposit operation failed, you don't withdrawal operation to happen either.*
- **Consistency** स्थिर रहनु- (Correctness) A transaction either creates a new and valid state of data, or, if any failure occurs, returns all data to its state before the transaction was started. *E.g. if funds are transferred between accounts, a deposit and withdrawal must both be committed to the database, so that the accounting system does not fall out of balance..*
- **Isolation** अलग गर्नु- A transaction in process or not yet committed must remain isolated from any other transaction. *E.g. a teller looking up a balance must be isolated from a concurrent transaction involving a withdrawal from the same account. Only when the withdrawal transaction commits successfully and the teller looks at the balance again will the new balance be reported.*
- **Durability** Permanently - Committed data is saved by the system such that, even in the event of a failure and system restart, the data is available in its correct state. *E.g. A system crash or any other failure must not be allowed to lose the results of a transaction or the contents of the database. Durability is often achieved through separate transaction logs that can "re-create" all transactions from some picked point in time (like a backup).*

5.4.3.2 Application of TPM

- CustomerInformationControlSystem, or CICS in short. Mostly found in IBM environments
- TuxedoMonitor for the Unix environment
- ApplicationServers using the JavaTwoEnterpriseEdition for the Unix environment
- Encina for Unix (Is this in the same league as the other products?)
- MicrosoftTransactionServer, or MTS. Now superseded by ComPlus which is also used in DotNet environments.
- IbmIms

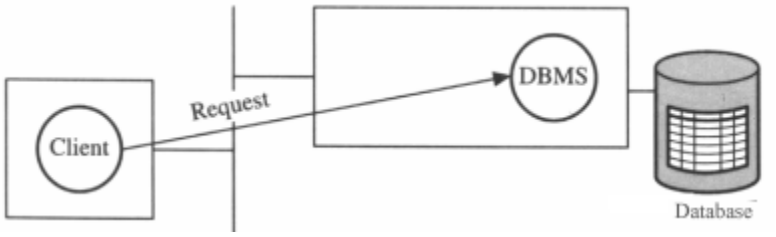
5.1 The Database Connectivity Challenge

If you want to communicate between the client and server than both machines must run ODBC drivers to communicate with each other.



Remote Request

What happens if someone wants to perform an operation on the database server? In this example the DBMS, Database Management System will convert the client's request to perform an operation on the database server.

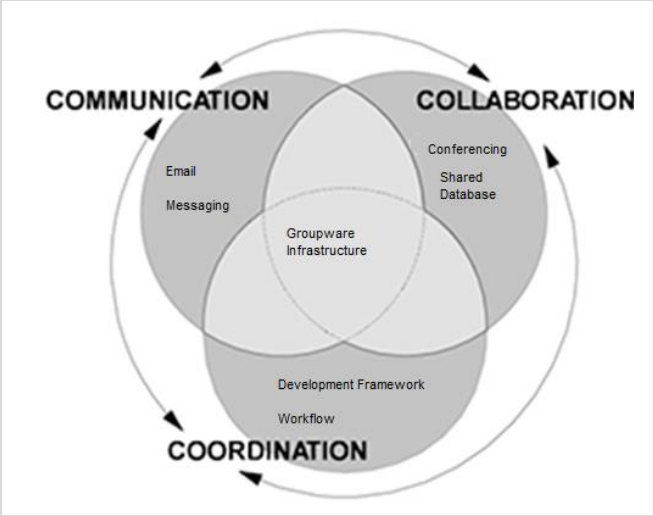


5.3 Introduction to Groupware.

Efficient way for 3 C' : Communication, Collaboration and Coordination

Groupware refers to programs that help people work together collectively while located remotely from each other.
Programs that enable real time collaboration are called synchronous groupware.

Groupware services can include the sharing of calendars, collective writing, e-mail handling, shared database access, electronic meetings with each person able to see and display information to others, and other activities. Groupware systems are classified based on functions, specifically:



- Computer mediated communication supporting direct participant communication
- Meeting and decision support systems capturing the common understanding of participants
- Shared applications
- Artifacts supporting the interaction of participants through shared work objects

Benefits and Features of Groupware

- Groupware allows both on-site workers and geographically dispersed team members to collaborate with each other over the internet or an internal network/intranet
- a centralized place for saving and accessing documents or files (along with document version and change management), shared calendars and task management, and communication tools, from instant messaging and video conferencing to white boards and message boards.
- groupware apps serve as a virtual office hub--everything is stored, shared, and documented in one place, which makes finding information and sharing data so much easier
- For entrepreneurs and freelancers, these tools enable easy file sharing, collaboration, and communication over projects with remote clients, all from the comfort of the home office.

What is Groupware server?

- Groupware server is software that allows the collaboration of users, irrespective of location through the internet or intranet to work together in an atmosphere which is virtual.
- The number of clients that need to be connected to this server mostly depends on the scope and nature of the project.
- Here a software is installed on various client computers so that better communication takes place between the clients and access to the server.
- This server helps to reduce the unnecessary or repetitive communication between the team members which also helps to increase the productivity.

5.1.1 Data Source Differences, Approaches to Database Connectivity

Approaches to database connectivity are JDBC, ODBC.

5.4.4 ODBC (Open Database Connectivity) & JDBC (Java Database Connectivity)

- ODBC stands for Open Database Connectivity. It is a standard which makes it is possible for any database application that supports ODBC to manage the databases which also supports ODBC with for example SQL statements.
- Once you have the ODBC driver in place users can gain immediate access to your DBMS. If you write an application which supports the ODBC standard, you don't have to worry about how to approach the database. For instance: *A Microsoft Client Application which supports ODBC can communicate with an Oracle Database which also supports ODBC. ODBC makes sure that the commands are converted in a proper way.*

Java Database Connectivity (**JDBC**) is an application programming interface (API) for the programming language Java, which defines how a client may access a database. *It is part of the Java Standard Edition platform, from Oracle Corporation. It provides methods to query and update data in a database, and is oriented towards relational databases. A JDBC-to-ODBC bridge enables connections to any ODBC-accessible data source in the Java virtual machine (JVM) host environment.*

5.4.4.1 ODBC and JDBC Components

1) JDBC callable API: It is used by Java application programs for making function calls.
2) JDBC drivers: For every DBMS brand there is a separate driver.
3) Driver manager: It is used for managing the different JDBC drivers.
Types of JDBC drivers: JDBC drivers are classified based on how JDBC calls are handled by the DBMS. There are 4 types of drivers:
Type 1 driver: JDBC/ODBC bridge: It uses ODBC driver for connection. Then, ODBC driver is used to connect to the DBMS.
Type 2 driver: Native API driver: It is used for translating JDBC calls directly to native API of DBMS.
Type 3 driver: Network Protocol driver: It is used for translating JDBC calls directly to particular DBMS network.

Type 4 driver: Native protocol driver: It is used to translate JDBC calls to particular DBMS network in proprietary format. It is a full Java driver and is written completely in Java. There is no middle layer in this driver.

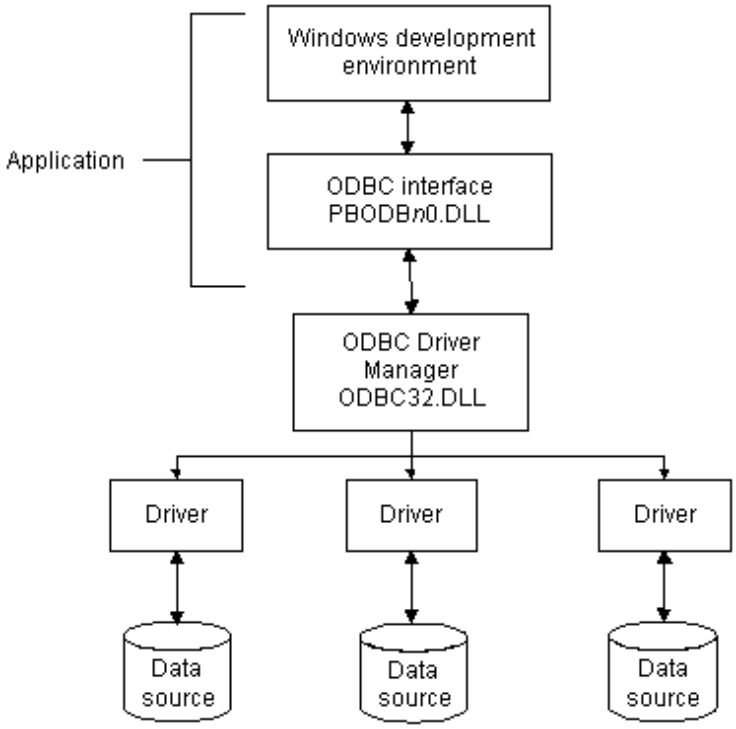


Fig. ODBC Components

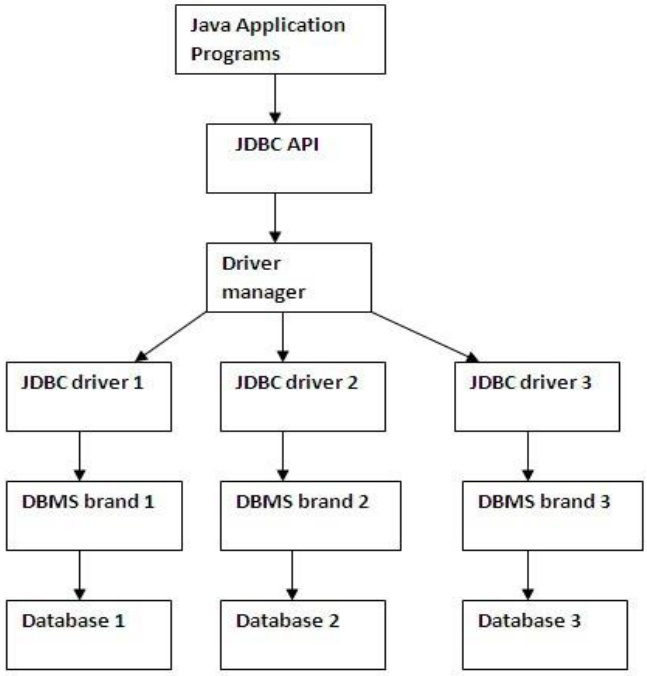


Fig. JDBC Components

ODBC Components

Component	Provider	What it does
Application	Sybase	Performs processing and calls ODBC functions to submit SQL statements and retrieve results.
ODBC Driver Manager	Microsoft	Loads and unloads drivers on behalf of an application. Processes ODBC function calls or passes them to a driver.
Driver	Driver vendor	Processes ODBC function calls, submits SQL requests to a specific data source, and returns results to the application. If necessary, the driver modifies an application's request so that the request conforms to syntax supported by the associated DBMS.
Data source	DBMS or database vendor	Consists of the data the user wants to access and its associated operating system, DBMS, and network platform (if any) used to access the DBMS.

ODBC v/s JDBC

Basis of Distinction	ODBC	JDBC
Origin	Oracle	Microsoft
Drivers	OBDC is Microsoft drivers and uses dlls	JDBC is Java drivers and uses jar files..
Types	ODBC is procedural oriented	JDBC is object oriented.
Complexity	ODBC can't be directly used with Java because it uses a C interface.	ODBC makes use of pointers which have been removed totally from Java.
Installation Process	ODBC requires manual installation of the ODBC driver manager and driver on all client machines.	JDBC drivers are written in java and JDBC code is automatically installable, secure, and portable on all platforms.
Features	JDBC API is a natural Java Interface and is built on ODBC. JDBC retains some of the basic feature of ODBC.	JDBC is Java Database Connectivity. It enables java programs to execute SQL statements.
Usage	Only becomes concrete for applications used by Java system i.e. language dependent	Becomes suitable for different applications irrespective of their originator.

5.4.4.2 Features and Application of JDBC and ODBC

- OBDC is Microsoft drivers and uses DLLs, JDBC is Java drivers and uses JAR files.
- ODBC is procedural oriented and JDBC is object oriented.
- ODBC can't be directly used with Java because it uses a C interface.
- ODBC makes use of pointers which have been removed totally from Java.
- ODBC mixes simple and advanced features together and has complex options for simple queries. But JDBC is designed to keep things simple while allowing advanced capabilities when required.
- ODBC requires manual installation of the ODBC driver manager and driver on all client machines.
- JDBC drivers are written in java and JDBC code is automatically installable, secure, and portable on all platforms.
- JDBC API is a natural Java Interface and is built on ODBC. JDBC retains some of the basic feature of ODBC.
- JDBC is Java Database Connectivity. It enables java programs to execute SQL statements. JDBC makes it possible to write a single database application that can run on different platforms and interact with different DBMS.
- JDBC is language dependent i.e it is used only to make connectivity with java. ODBC is open database connectivity. The goal of ODBC is to make it possible to access any data from any application, regardless of which database management system (DBMS) is handling the data so it is language independent.

Sample Code

This sample example can serve as a **template** when you need to create your own JDBC application in the future.
This sample code has been written based on the environment and database setup done in the previous chapter.
Copy and paste the following example in FirstExample.java, compile and run as follows –

```
//STEP 1. Import required packages
import java.sql.*;

public class FirstExample {
  // JDBC driver name and database URL
  static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
  static final String DB_URL = "jdbc:mysql://localhost/EMP";

  //  Database credentials
  static final String USER = "username";
  static final String PASS = "password";

  public static void main(String[] args) {
  Connection conn = null;
  Statement stmt = null;
  try{
    //STEP 2: Register JDBC driver
    Class.forName("com.mysql.jdbc.Driver");

    //STEP 3: Open a connection
    System.out.println("Connecting to database...");
    conn = DriverManager.getConnection(DB_URL,USER,PASS);

    //STEP 4: Execute a query
    System.out.println("Creating statement...");
    stmt = conn.createStatement();
    String sql;
    sql = "SELECT id, first, last, age FROM Employees";
    ResultSet rs = stmt.executeQuery(sql);

    //STEP 5: Extract data from result set
    while(rs.next()){
      //Retrieve by column name
      int id  = rs.getInt("id");
      int age = rs.getInt("age");
      String first = rs.getString("first");
      String last = rs.getString("last");
```

```
      //Display values
      System.out.print("ID: " + id);
      System.out.print(", Age: " + age);
      System.out.print(", First: " + first);
      System.out.println(", Last: " + last);
    }
    //STEP 6: Clean-up environment
    rs.close();
    stmt.close();
    conn.close();
  }catch(SQLException se){
    //Handle errors for JDBC
    se.printStackTrace();
  }catch(Exception e){
    //Handle errors for Class.forName
    e.printStackTrace();
  }finally{
    //finally block used to close resources
    try{
      if(stmt!=null)
        stmt.close();
    }catch(SQLException se2){
    }// nothing we can do
    try{
      if(conn!=null)
        conn.close();
    }catch(SQLException se){
      se.printStackTrace();
    }//end finally try
  }//end try
  System.out.println("Goodbye!");
}//end main
}//end FirstExample
```

Now let us compile the above example as follows −

```
C:\>javac FirstExample.java
C:\>
```

When you run **FirstExample**, it produces the following result −

```
C:\>java FirstExample
Connecting to database...
Creating statement...
ID: 100, Age: 18, First: Zara, Last: Ali
ID: 101, Age: 25, First: Mahnaz, Last: Fatma
ID: 102, Age: 30, First: Zaid, Last: Khan
ID: 103, Age: 28, First: Sumit, Last: Mittal
C:\>
```