Introduction
In a distributed system (client-server application), the various components are hosted on different machines. These components send requests and responses between themselves in a heterogeneous environment (hosts could be from different manufacturers and running different operating systems), which may present a challenge. To address this and other challenges, a software that facilitates exchanges between different hosts is deployed. This software is known as *middleware*.

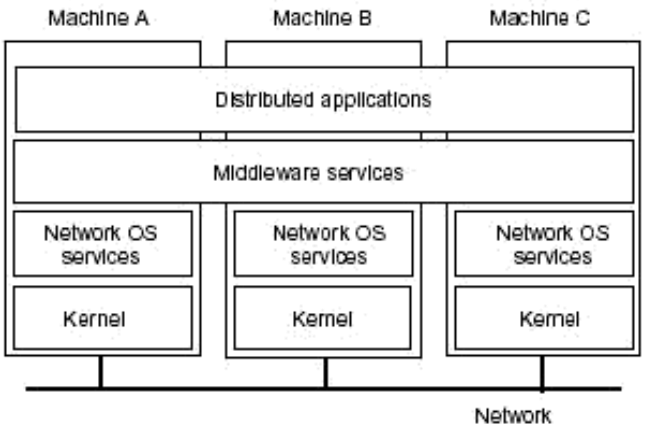
A middleware is *a layer of software running between client and server processes*. It shields the client (and application developers) from the complexity of the underlying communications protocol, network operating systems functions and hardware configurations. It can also be looked at as a common set of application programming interfaces (APIs) provided to application developers to enable rapid development of distributed applications and flexible access to distributed resources.

Middleware is as an additional layer of software between an application and the operating system and networking on a computer. It is the layer above the operating system but below the application program that provides a common programming abstraction (higher-level building blocks for programmers than the OS provides) across a distributed system and resolves heterogeneity inherent in distributed systems

Typically, middleware is based on one of two underlying mechanisms. These mechanisms are determined by the programming paradigms used in developing the distributed applications. The two main programming paradigms are structured/procedural and object-orientation. The middleware provides two types of communication between the client and the server:

- Message passing: This mechanism is for the object-oriented programming paradigm. The communication between the client and the server is via exchange of messages.
- Remote procedure calls (RPCs): This mechanism is for the structured/procedural programming paradigm and communication between the client and the server is effected by calling of procedures that are resident in either component (whether client or server).

Positioning Middleware
Many distributed applications make direct use of the programming interface offered by network operating systems. For example, communication is often expressed through operations on sockets, which allow processes on different machines to pass each other messages (Stevens, 1998). In addition, applications often make use of interfaces to the local file system. A problem with this approach is that distribution is hardly transparent and a solution is to place an additional layer of software between applications and the network operating system, offering a higher level of abstraction. Such a layer is called *middleware*. It sits in the middle between applications and the network operating system as shown below:



Each local system forming part of the underlying network operating system is assumed to provide local resource management in addition to simple communication means to connect to other computers. In other words, middleware itself will not manage an individual node; this is left entirely to the local operating system.

An important goal is to hide heterogeneity of the underlying platforms from applications. Therefore, many middleware systems offer a more-or-less complete collection of services and discourage using anything else but their interfaces to those services. In other words, skipping the middleware layer and immediately calling services of one of the underlying operating systems is often frowned upon.

Notice that after widespread use of network operating systems, many organizations found themselves having lots of networked applications that could not be easily integrated into a single system. At that point, manufacturers started to build higher-level, application-independent services into their systems. Typical examples include support for distributed transactions and advanced communication facilities.

Of course, agreeing on what the right middleware should be is not easy. An approach is to set up an organization that subsequently defines a common standard for some middleware solution. At present, there are a number of such standards available. The standards are generally not compatible with each other, and even worse, products implementing the same standard but from different manufacturers rarely inter-work.

Advantages of Middleware:

i). Reduce number of interfaces.
ii). Clients see only one system i.e. the middleware.
iii). Centralizes control.
iv). Functionality widely available to all clients.
v). It allows for the implementation of functionality that otherwise would be very difficult to provide.

Disadvantages of Middleware:

i). Complex software.
ii). Development platform (API) not complete system.
iii). Functionality is hard to understand.

Middleware Services

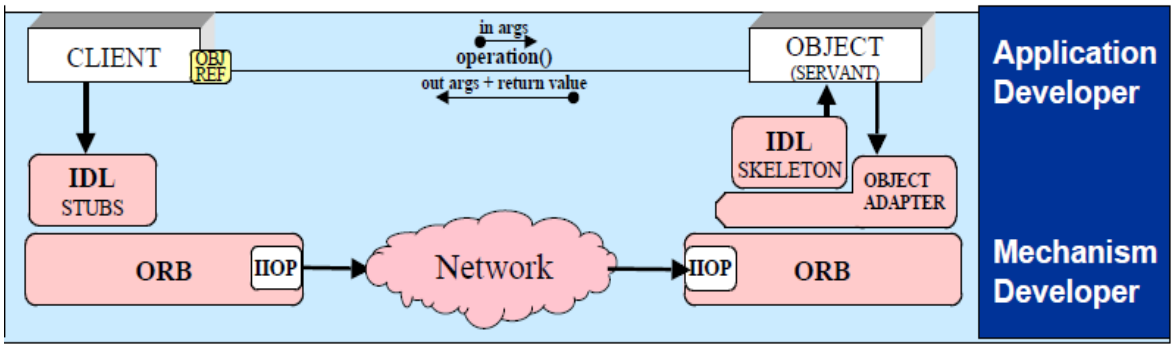
- Middleware provides services such as name-to-address resolution, dynamic server process invocation, locating particular servers, load balancing, security, failure recovery, message routing, reliable message delivery etc.
- Middleware can be viewed as a set of services that are accessible to application programmers through an API.
- There are several middleware services available, e.g. remote procedure call (RPC)
- Many middleware services are high-level services.

Examples of Middleware

- Distributed Computing Environment from OSF
    - based on RPC and IDL

- Common Object Request Broker Architecture (CORBA) from Object Management Group (OMG)
    - Open standard for distribution middleware that allows objects to interoperate across networks regardless of the language in which they were written or the platform on which they are deployed

- based on objects and IDL (Integrated Definition Language) interface
- objects locate each other through Object Request Broker (ORB)
- defines Internet Inter-ORB protocol (IIOP)



- Component Object Model(COM) and Distributed Component Object Model (DCOM)
  - Distribution middleware that enables software components to communicate over a network via remote component instantiation and method invocations
    - builds Object RPC (ORPC) on top of DCE RPC (Distributed Computing Environment RPC)
  - supports integration of binary components from different languages (e.g. VB, Java, C+ +)
  - DCOM is implemented primarily on Windows platforms.

- Remote Method Invocation (RMI)
  - based on a single language (Java)
  - Distribution middleware that enables developers to create distributed Java-to-Java applications, in which the methods of remote Java objects can be invoked from other JVMs, possibly on different hosts
- CORBA IDL used for defining interfaces Jini from Sun
  - based on RMI
  - proposed for network-aware appliances

- Simple Object Access Protocol (SOAP)
  - Distribution middleware technology based on a lightweight and simple XML-based protocol that allows applications to exchange structured and typed information on the Web.
  - Designed to enable automated Web services based on a shared and open Web infrastructure.
  - Applications can be written in a wide range of programming languages, used in combination with a variety of Internet protocols and formats (such as HTTP, SMTP, and MIME)
  - Support a wide range of applications from messaging systems to RPC.

International Standards Organization: Forms of Transparency

Any user of a distributed application should be protected from knowing that the resource(s) that they are accessing is remote. They should have an impression that all resources are local to them. For this to be achieved, a number of transparencies have been proposed by the ISO. These transparencies are listed below:

i). **Access** – hide differences in data representation and how a resource is accessed
ii). **Location** – hide where a resource is located
iii). **Migration** – hide that a resource may move to another location
iv). **Relocation** – hide that a resource may move while in use
v). **Replication** – hide that a resource is replicated

vi). **Concurrency** – hide that a resource may be shared by several competitive users
vii). **Failure** – hide the failure and recovery of a resource
viii). **Persistence** – hide whether a software resource is in memory or on disk

Note: Masking heterogeneity and providing transparency makes programming distributed systems much easier.

Middleware Standard Interfaces

Middleware provides a comprehensive set of higher-level distributed computing capabilities and a set of standards-based interfaces.

These interfaces allow applications to be distributed more easily and to take advantage of other services provided over the network.

These are the functional interfaces that middleware uses in the normal delivery of the service. The following are the various types of interfaces

i). Application Programming Interfaces

API is the mechanism by which application programs interact with a middleware service.
Each API specification describes the abstract syntax and semantics of the service, including parameter definitions and usage rules.

ii). System Programming Interfaces

SPI is the mechanism by which system programmers can extend or modify a middleware service or framework. SPI is identical in form to an API, but it is a lower level interface.
Application programmers do not get involved in changing the behavior of a service. For example, messaging services and the transaction processing monitor framework. These SPIs allow system programmers to add support for:

- Additional network transports (messaging service) and
- Additional user agents (e.g. transaction processing monitor framework) beyond the ones supported by the service.

iii). Communications Interfaces

CI is the mechanism by which various middleware components, running on different systems in a network, can exchange data and control information.

Implementer of services use application- level communication protocols to implement communications interface. Application developers rarely need to be concerned with the CIs because middleware services are of a high level and transparently distributed.

The CI describes the application protocols the service can use for each type of network transport (TCP/IP, OSI, DECnet, PC LANs)

iv). User Interfaces

UI is the mechanism by which users interact with the information system. The UI must handle a variety of device types and a wide range of user roles, languages, and cultures.

The middleware presentation services handle UIs.
Common UI devices include personal computers (running windows), Workstations that use Motif and the X Window System, video display terminals, and printers. Other devices include voice recognition, text-to-speech equipment, image scanners, and graphics tablets.

v). Data Interfaces

DI is an information structure that enables another routine to exchange (read and/or write) data with a service without going through a calling or messaging mechanism.

The DI includes storage formats and any data syntax that programmers or users can rely on. DI mechanisms include file passing, shared database access, or common memory (for example, using the X Window System paste buffer)

vi). Management Interfaces

MI permits system managers and system programmers to install, monitor, and control information system components. MI can include a management API, a management CI, a management SPI, a management UI, and/or a management DI.

Middleware Attributes

In view of the many different middleware that are in existence, one should be able to evaluate them using some attributes in order to identify the best middleware for their deployment. The following list makes up some of the desired attributes that a middleware should have. It will be a tall order to expect any single middleware to possess all these attributes and one is advised to identify those attributes that are crucial and most important to them and use those specific attributes to identify a suitable middleware. The attributes are:

i). Usability

It is the extent to which a system or component helps the users get jobs done efficiently and effectively. Many aspects that make a system easy to use are captured in the other attributes.

ii). Distributability

Distributability enables clients and components of a service, tool, or application to execute across multiple hardware components of a network. The goals for middleware distributability include support for the following:

- Transparent distribution of services for ease of use
- Multiple distribution models, with particular emphasis on the client/server model
- Distribution of data, processing, and presentation functions.
- Immediate and queued operation (to permit the client and server to be distributed in time and space)
- Distribution over all geographic ranges (single and multiprocessors, LANs, and WANs)

iii). Integration

Integration is the ability of applications to work together in a consistent manner to perform tasks for the user. Interoperability and uniformity are two key aspects of integration.

- Interoperability is the extent to which a set of components invoke and exchange information effectively, such as getting data in and out of a component and converting data as needed.
- Uniformity is the extent to which a set of components are constant with respect to a set of attributes. (Uniformity of interfaces reduces system complexity)

iv). Extensibility

It is the ease with which a system can be adapted to meet new requirements. There are three types of extensibility:

1. Customization: allowing the end user to change the system presentation characteristics.
2. Configuration: enable installers or system integrators to change the system and system components.

3. Evolution: enables component architects and developers to change the internals of components.

v). Internationalization

It is the extent to which an information system is suitable for users and data from multiple cultures.

vi). Manageability

It is the extent to which system managers can economically configure, monitor, diagnose, maintain, and control the resources of a computing environment.

vii). Performance

It is a measurement of the resources required to perform an operation (elapsed time to perform an operation)

Efficiency is the ratio of resources used to the number of operations performed.
Utilization: the relative use of resources
Throughput: number of completed operations per unit of time.

viii). Portability

It is the ease with which developer can move software from one platform to another.

ix). Reliability

Reliability is measured by the availability of the system (or the mean time between failures.)
The extent to which a system produces the same output on repeated trials.

x). Scalability

Scalability is the extent to which developers can apply solutions to problems of different sizes.
Complexity grows with the number of system nodes, users, and other network resources.

xi). Security

Security is the protection of information from unauthorized access, the protection of clients from accessing rogue services, and the protection of resources from unauthorized use.

Security services include authentication, access control, and auditing.

Questions

1) Identify the different middleware that exist in the market today.
2) Describe how the middleware facilitates communication between clients and servers.
3) Identify the six interfaces that a middleware uses to offer its services.