DISTRIBUTED ARCHITECTURE

In distributed architecture, components are presented on different platforms and several components can cooperate with one another over a communication network in order to achieve a specific objective or goal.

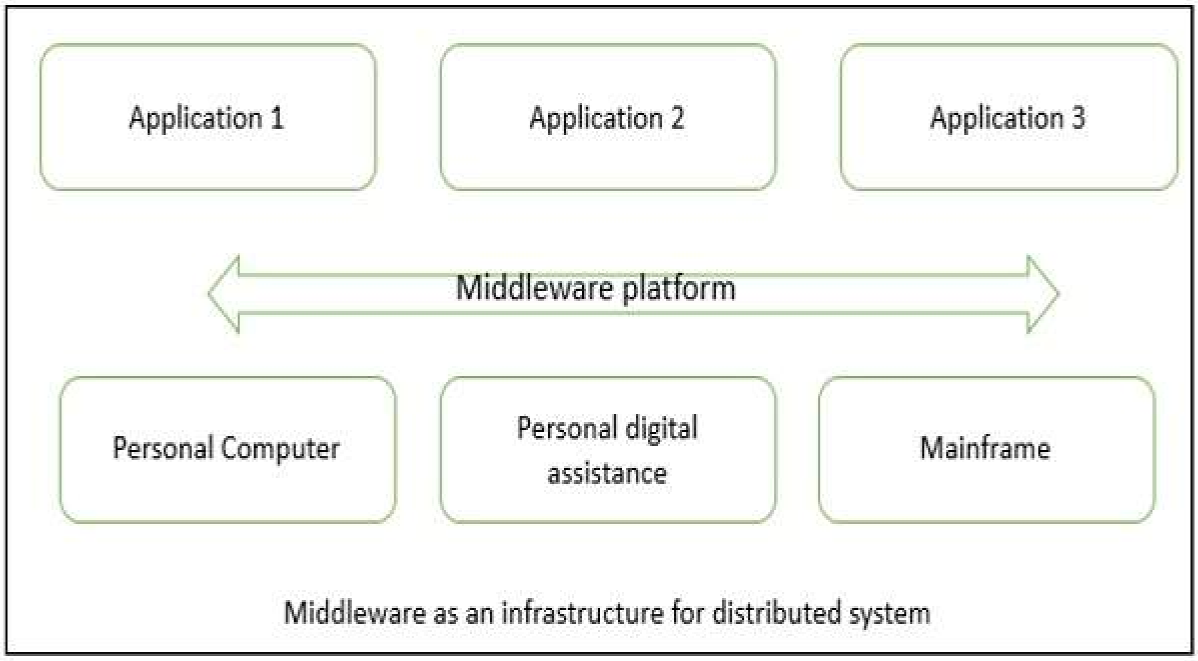
Concept of Distributed Architecture

A distributed system can be demonstrated by the client-server architecture, which forms the base for multi-tier architectures; alternatives are the broker architecture such as CORBA, and the Service-Oriented Architecture *SOA*. In this architecture, information processing is not confined to a single machine rather it is distributed over several independent computers.

There are several technology frameworks to support distributed architectures, including .NET, J2EE, CORBA, .NET Web services, AXIS Java Web services, and Globus Grid services. Middleware is an infrastructure that appropriately supports the development and execution of distributed applications. It provides a buffer between the applications and the network.

It sits in the middle of system and manages or supports the different components of a distributed system. Examples are transaction processing monitors, data convertors and communication controllers, etc.

Middleware as an infrastructure for distributed system −



Basis of Distributed Architecture

The basis of a distributed architecture is its transparency, reliability, and availability.

The following table lists the different forms of transparency in a distributed system −

Transparency	Description
Access	Hides the way in which resources are accessed and the differences in data platform.
Location	Hides where resources are located.
Technology	Hides different technologies such as programming language and OS from user.

Migration / Relocation	Hide resources that may be moved to another location which are in use.
Replication	Hide resources that may be copied at several location.
Concurrency	Hide resources that may be shared with other users.
Failure	Hides failure and recovery of resources from user.
Persistence	Hides whether a resource *software* is in memory or disk.

Advantages

It has following advantages −

- **Resource sharing** − Sharing of hardware and software resources.
- **Openness** − Flexibility of using hardware and software of different vendors.
- **Concurrency** − Concurrent processing to enhance performance.
- **Scalability** − Increased throughput by adding new resources.
- **Fault tolerance** − The ability to continue in operation after a fault has occurred.

Disadvantages

Its disadvantages are −

- **Complexity** − They are more complex than centralized systems.
- **Security** − More susceptible to external attack.
- **Manageability** − More effort required for system management.
- **Unpredictability** − Unpredictable responses depending on the system organization and network load.

Centralized System vs. Distributed System

Criteria	Centralized system	Distributed System
Economics	Low	High
Availability	Low	High
Complexity	Low	High
Consistency	Simple	High
Scalability	Poor	Good
Technology	Homogeneous	Heterogeneous
Security	High	Low

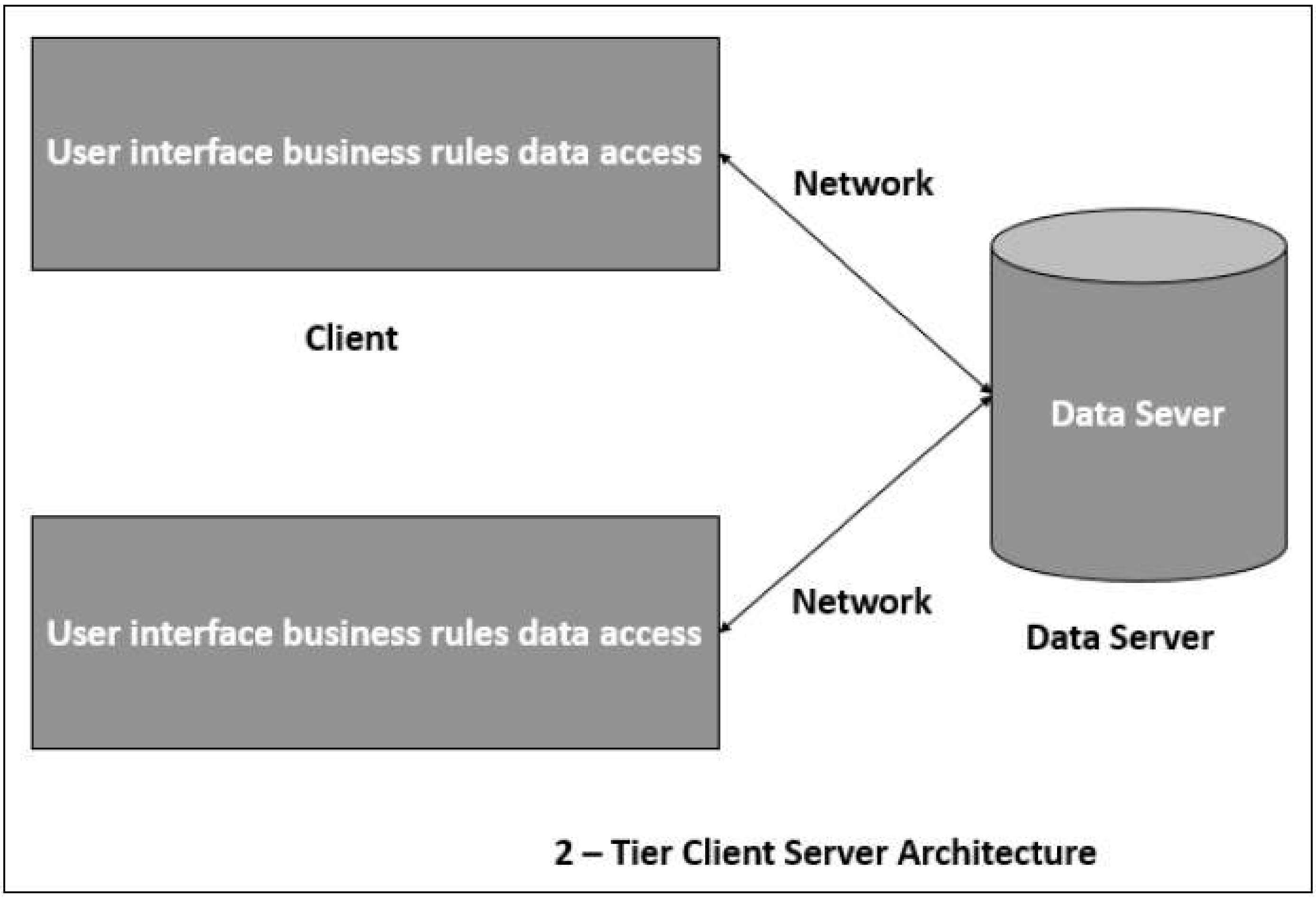
Client-Server Architecture

The client-server architecture is the most common distributed system architecture which decomposes the system into two major subsystems or logical processes −

- **Client** − This is the first process that issues a request to the second process i.e. the server.

- **Server** − This is the second process that receives the request, carries it out, and sends a reply to the client.

In this architecture, the application is modelled as a set of services those are provided by servers and a set of clients that use these services. The servers need not to know about clients, but the clients must know the identity of servers.


User interface business rules data access
Network
Client
Data Sever
User interface business rules data access
Network
Data Server
2 – Tier Client Server Architecture

Client-server Architecture can be classified into two models based on the functionality of the client −

Thin-client model

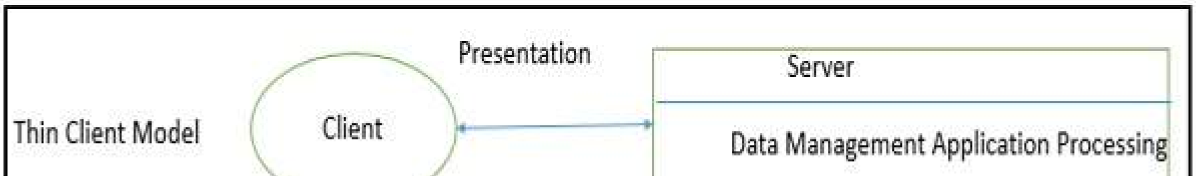
In thin-client model, all the application processing and data management is carried by the server. The client is simply responsible for running the GUI software. It is used when legacy systems are migrated to client server architectures in which legacy system acts as a server in its own right with a graphical interface implemented on a client.

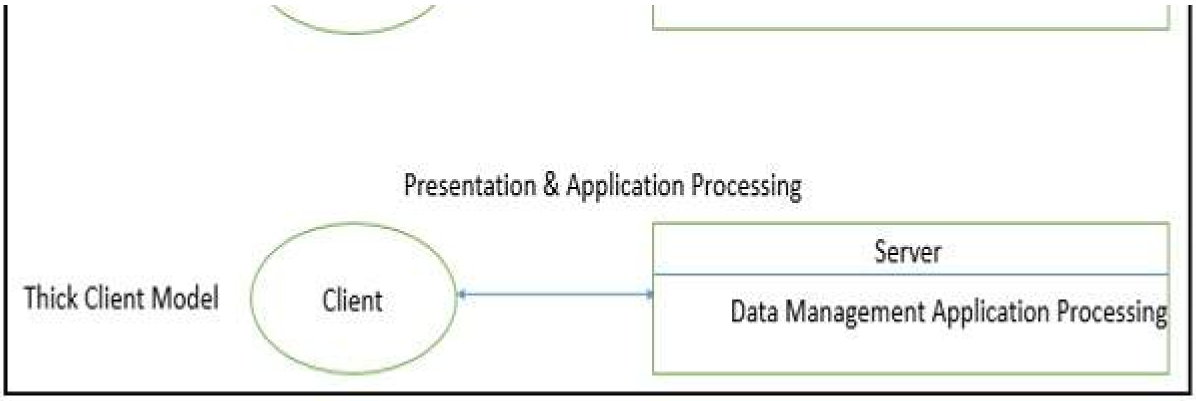
However, A major disadvantage is that it places a heavy processing load on both the server and the network.

Thick/Fat-client model

In thick-client model, the server is in-charge of the data management. The software on the client implements the application logic and the interactions with the system user. It is most appropriate for new client-server systems where the capabilities of the client system are known in advance.

However, it is more complex than a thin client model especially for management, as all clients should have same copy/version of software application.




Presentation & Application Processing
Thick Client Model
Client
Server
Data Management Application Processing

Advantages

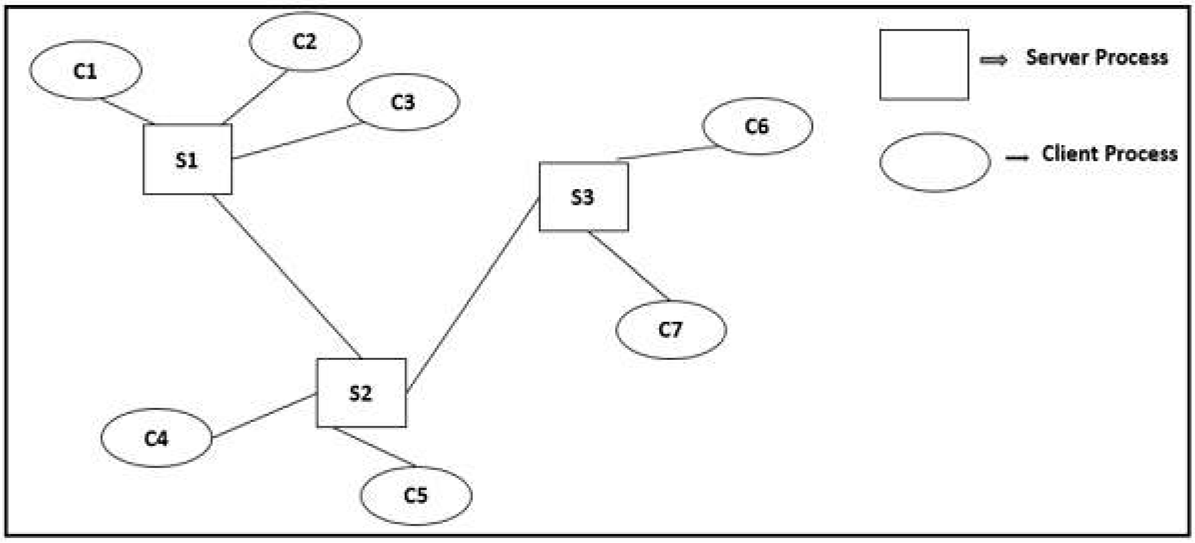
- Separation of responsibilities such as user interface presentation and business logic processing.
- Reusability of server components and potential for concurrency
- Simplifies the design and the development of distributed applications
- It makes it easy to migrate or integrate existing applications into a distributed environment.
- It also makes effective use of resources when a large number of clients are accessing a high-performance server.

Disadvantages

- Lack of heterogeneous infrastructure to deal with the requirement changes.
- Security complications.
- Limited server availability and reliability.
- Limited testability and scalability.
- Fat clients with presentation and business logic together.

Multi-Tier Architecture $n - tierArchitecture$

Multi-tier architecture is a client–server architecture in which the functions such as presentation, application processing, and data management are physically separated. By separating an application into tiers, developers obtain the option of changing or adding a specific layer, instead of reworking the entire application. It provides a model by which developers can create flexible and reusable applications.



The most general use of multi-tier architecture is the three-tier architecture. A three-tier architecture is typically composed of a presentation tier, an application tier, and a data storage tier and may execute on a separate processor.

Presentation Tier

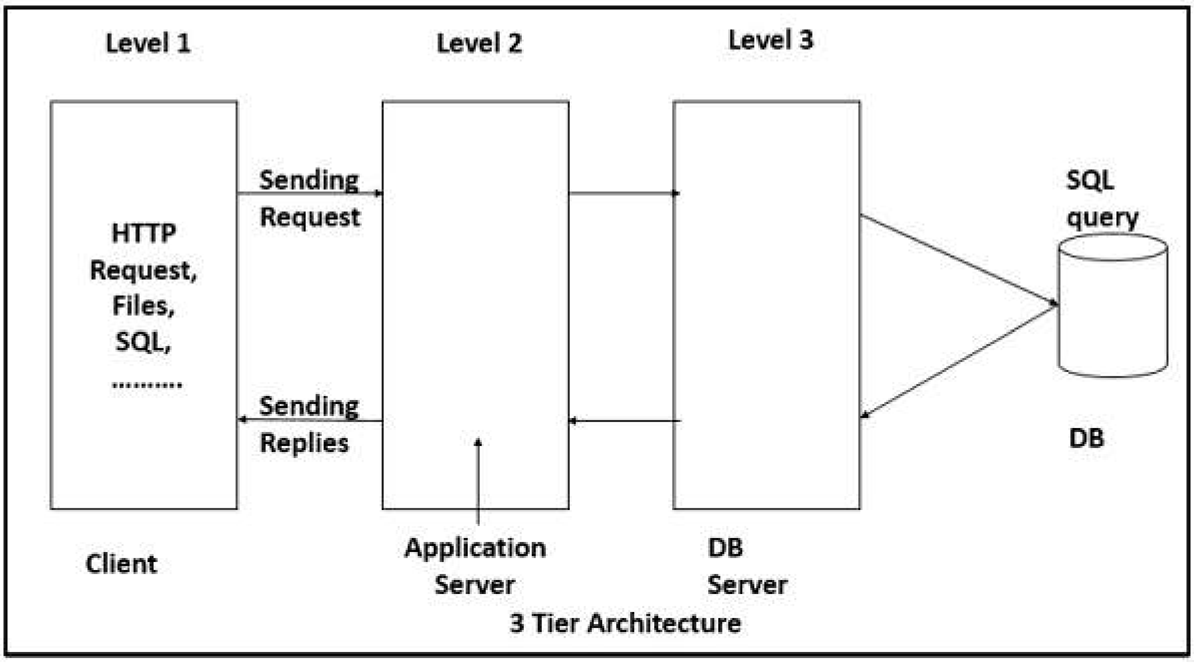
Presentation layer is the topmost level of the application by which users can access directly such as webpage or Operating System GUI *GraphicalUserinterface*. The primary function of this layer is to translate the tasks and results to something that user can understand. It communicates with other tiers so that it places the results to the browser/client tier and all other tiers in the network.

Application Tier *BusinessLogic, LogicTier, orMiddleTier*

Application tier coordinates the application, processes the commands, makes logical decisions, evaluation, and performs calculations. It controls an application’s functionality by performing detailed processing. It also moves and processes data between the two surrounding layers.

Data Tier

In this layer, information is stored and retrieved from the database or file system. The information is then passed back for processing and then back to the user. It includes the data persistence mechanisms *databaseservers, fileshares, etc.* and provides API *ApplicationProgrammingInterface* to the application tier which provides methods of managing the stored data.



Advantages

- Better performance than a thin-client approach and is simpler to manage than a thick-client approach.
- Enhances the reusability and scalability − as demands increase, extra servers can be added.
- Provides multi-threading support and also reduces network traffic.
- Provides maintainability and flexibility

Disadvantages

- Unsatisfactory Testability due to lack of testing tools.
- More critical server reliability and availability.

Broker Architectural Style

Broker Architectural Style is a middleware architecture used in distributed computing to coordinate and enable the communication between registered servers and clients. Here, object communication takes place through a middleware system called an object request broker *softwarebus*.

However, client and the server do not interact with each other directly. Client and server have a direct connection to its proxy, which communicates with the mediator-broker. A server provides services by registering and publishing their interfaces with the broker and clients can request the services from the broker statically or dynamically by look-up.

CORBA *CommonObjectRequestBrokerArchitecture* is a good implementation example of the broker architecture.

Components of Broker Architectural Style

The components of broker architectural style are discussed through following heads −

Broker

Broker is responsible for coordinating communication, such as forwarding and dispatching the results and exceptions. It can be either an invocation-oriented service, a document or message - oriented broker to which clients send a message.

Broker is responsible for brokering the service requests, locating a proper server, transmitting requests, and sending responses back to clients. It also retains the servers' registration information including their functionality and services as well as location information.

Further, it provides APIs for clients to request, servers to respond, registering or unregistering server components, transferring messages, and locating servers.

Stub

Stubs are generated at the static compilation time and then deployed to the client side which is used as a proxy for the client. Client-side proxy acts as a mediator between the client and the broker and provides additional transparency between them and the client; a remote object appears like a local one.

The proxy hides the IPC *inter − processcommunication* at protocol level and performs marshaling of parameter values and un-marshaling of results from the server.

Skeleton

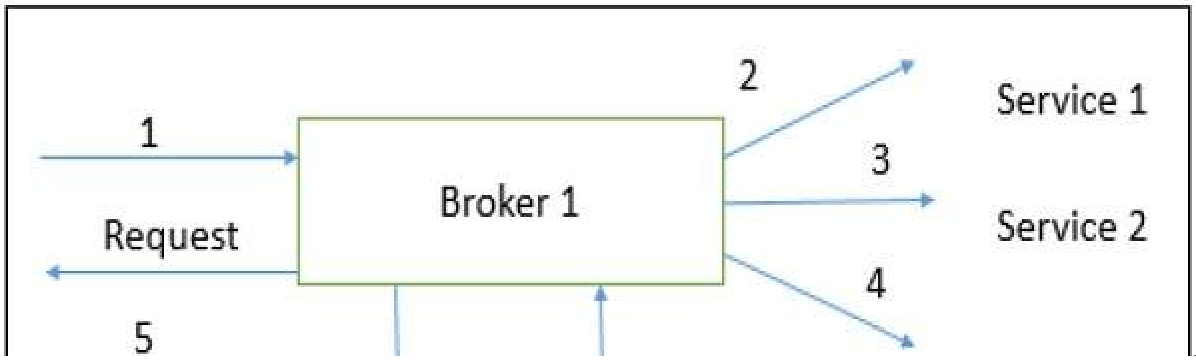
Skeleton is generated by the service interface compilation and then deployed to the server side, which is used as a proxy for the server. Server-side proxy encapsulates low-level system-specific networking functions and provides high-level APIs to mediate between the server and the broker.

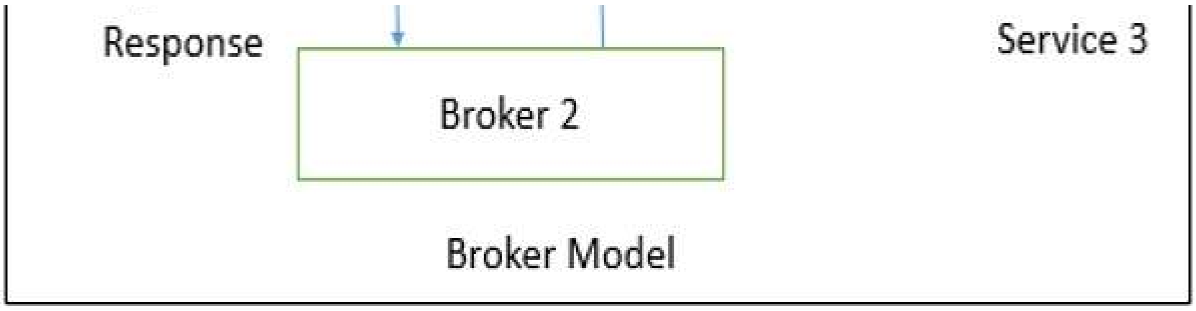
Further, it receives the requests, unpacks the requests, unmarshals the method arguments, calls the suitable service, and also marshals the result before sending it back to the client.

Bridge

A bridge can connect two different networks based on different communication protocols. It mediates different brokers including DCOM, .NET remote, and Java CORBA brokers.

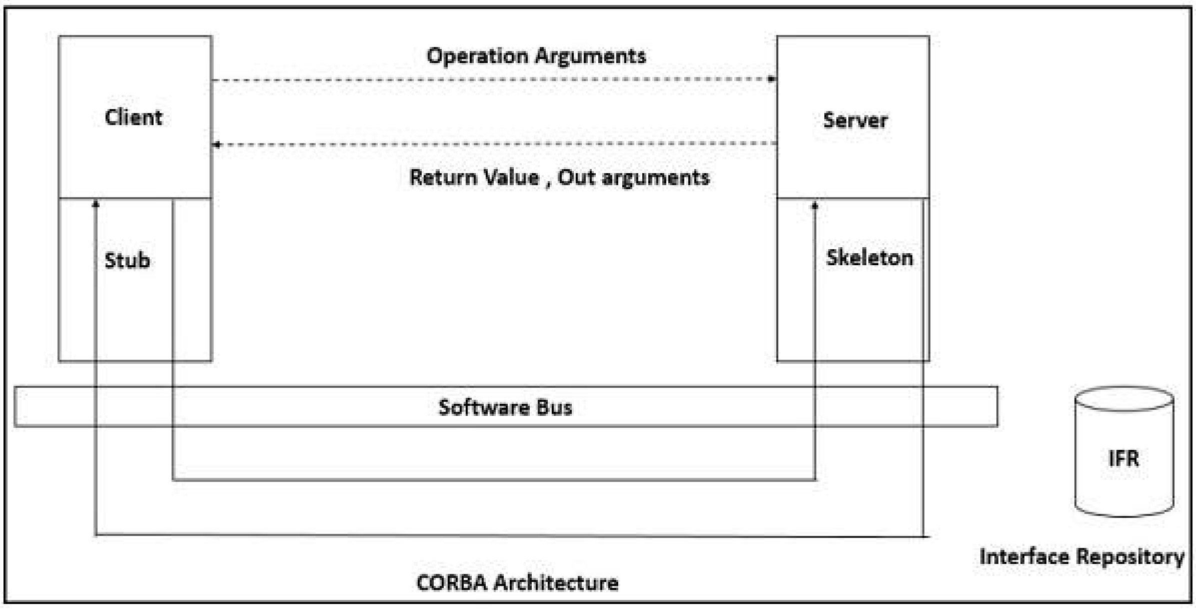
Bridges are optional component, which hides the implementation details when two brokers interoperate and take requests and parameters in one format and translate them to another format.





Broker implementation in CORBA

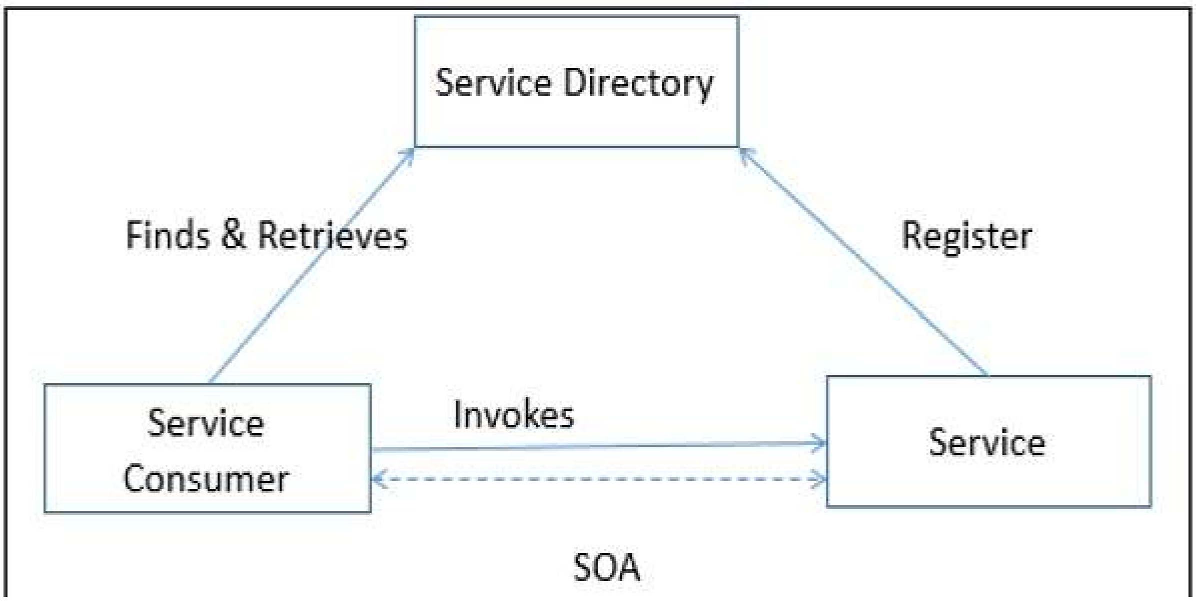
CORBA is an international standard for an Object Request Broker – a middleware to manage communications among distributed objects defined by OMG *objectmanagementgroup*.



Service-Oriented Architecture *SOA*

A service is a component of business functionality that is well-defined, self-contained, independent, published, and available to be used via a standard programming interface. The connections between services are conducted by common and universal message-oriented protocols such as the SOAP Web service protocol, which can deliver requests and responses between services loosely.

Service-oriented architecture is a client/server design which support business-driven IT approach in which an application consists of software services and software service consumers *alsoknownasclientsorservicerequesters*.



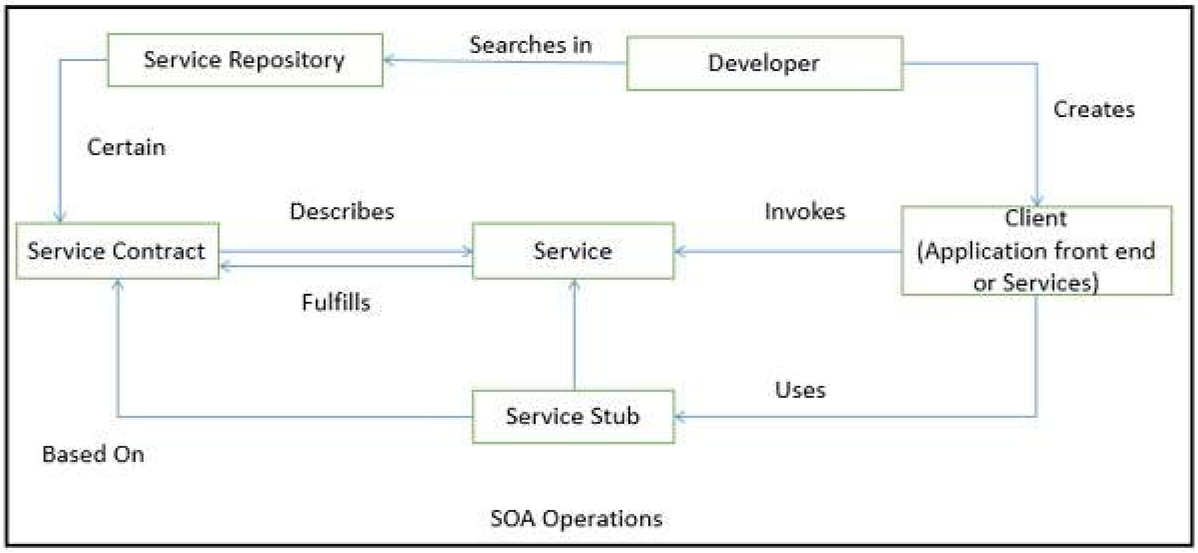
Features of SOA

A service-oriented architecture provides the following features −

- **Distributed Deployment** − Expose enterprise data and business logic as loosely, coupled, discoverable, structured, standard-based, coarse-grained, stateless units of functionality called services.
- **Composability** − Assemble new processes from existing services that are exposed at a desired granularity through well defined, published, and standard complaint interfaces.
- **Interoperability** − Share capabilities and reuse shared services across a network irrespective of underlying protocols or implementation technology.
- **Reusability** − Choose a service provider and access to existing resources exposed as services.

SOA Operation

The following figure illustrates how does SOA operate −



Advantages

- Loose coupling of service–orientation provides great flexibility for enterprises to make use of all available service recourses irrespective of platform and technology restrictions.
- Each service component is independent from other services due to the stateless service feature.
- The implementation of a service will not affect the application of the service as long as the exposed interface is not changed.
- A client or any service can access other services regardless of their platform, technology, vendors, or language implementations.
- Reusability of assets and services since clients of a service only need to know its public interfaces, service composition.
- SOA based business application development are much more efficient in terms of time and cost.
- Enhances the scalability and provide standard connection between systems.
- Efficient and effective usage of ‘Business Services’.
- Integration becomes much easier and improved intrinsic interoperability.

- Abstract complexity for developers and energize business processes closer to end users.

Loading [MathJax]/jax/output/HTML-CSS/jax.js