CAS 703 Software Design

Dr. Ridha Khedri

Department of Computing and Software, McMaster University
Canada L8S 4L7, Hamilton, Ontario

Acknowledgments: Material based on *Software Architecture Design* by Tao et al. (Chapters 9 and 10)

- More and more software applications that involve user input and output interactions
- We focus on the software architecture that best supports user interaction
- Interaction oriented software architecture decomposes the system into three major partitions
  - Data module (provides the data abstraction& core business logic)
  - Flow control module (determines the flow controls, view selections, communications between modules, job dispatching, and certain data initializations and configurations)
  - View presentation module (responsible for visual or audio data output presentation)

- This architecture allows the separation of user interactions from data abstraction and business data processing
- Allows multiple views for a same data set
- Even for a specific view presentation, the interfaces may need to change very often (the loose coupling between data abstractions and its presentations is very helpful)
- A control module plays a central role that mediates the data module and view presentation modules
- All three modules may be completely connected

- There are two categories of interaction oriented architecture:
  - Presentation-Abstraction-Control (PAC)
  - Model-View-Controller (MVC).
- They are different in their flow controls and structure organization
- The MVC does not have a clear hierarchical structure and all three modules are connected together

- The PAC is an agent based hierarchical architecture
  - The system is decomposed into many cooperating agents
  - Each agent has three components (Presentation, Abstraction, and Control)
  - The Control component in each agent is in charge of communications with other agents
  - The top-level agent provides core data and business logics
  - The bottom level agents provide detailed specific data and presentations
  - A middle level agent may play a role of coordinator of low-level agents
  - There are no direct connections between Abstraction component and Presentation component in each agent

- A distributed system is a collection of computers connected through a communication network
  - Data is distributed
  - Software is distributed
  - Users are distributed

- The sub-systems or components within a distributed system communicate with each other via
  - message passing
  - remote procedure call
  - remote method invocation
  - etc.

Two important issues for designing a distributed system are:

- Topology: the way in which entities connect with each other
- Mode: the method by which they communicate with each other
  - Synchronous
  - Asynchronous
  - message driven
  - callback
  - event-driven

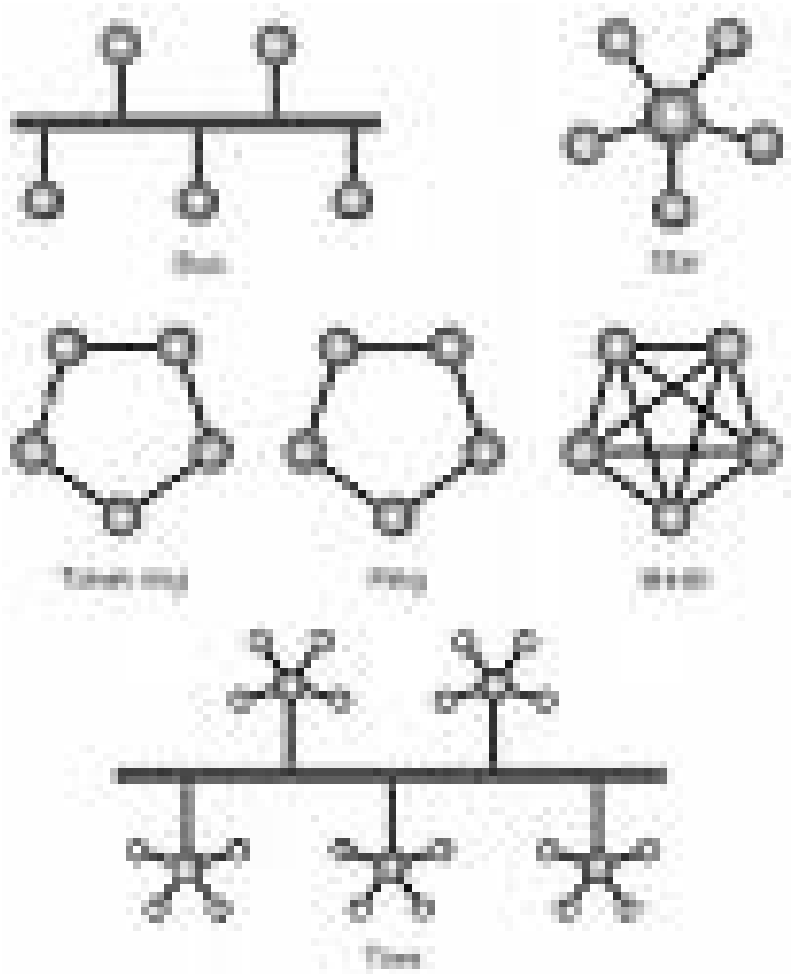

Figure: Examples of network topologies

- A distributed system can be modeled as a
  - Client/server architecture
  - Broker architecture
  - Service-Oriented Architecture (SOA)
- The important features of a distributed architecture include
  - its service location transparency
  - service reliability and availability

- Most of Web developers are familiar with MVC architecture
  - Widely adopted for Web server site interactive application design such as online shopping, online survey, online student registration, etc.
- MVC architecture is specifically used in applications where user interfaces are prone to data changes all the time
- MVC architecture typically supports "look and feel" features in GUI application
- The Java Swing components and Java Swing layout managers are designed in MVC architecture

Summary:

- The Controller
  - manages the user input requests
  - controls the sequence of user interactions
  - selects desired views for output displays
  - manages all initialization, instantiations, and registrations of other modules in the MVC system
- The Model module
  - provides all core functional services and encapsulates all data details
  - does NOT depend on other modules, and it does not know which views are registered with or attached to it
- The View module
  - is responsible for displaying the data provided by the Model module and updating the interfaces whenever the data changes

- The MVC-I is a simple version of MVC architecture
- The system is simply decomposed into two sub-systems:
  - Controller/View
    - It handles input and output processing and their interfaces
    - It registers with (attaches to) data module
  - Model
    - It copes with all core functionality and data
    - It notifies the Controller-View module of any data changes in the Model module

- The connection between the Controller/View and the Model can be designed in a pattern of subscribe/notify
- The Controller/View subscribes the Model and the Model notifies the Controller/View of any changes
- The Controller/View is an observer to the data in the Model of MVC
- Read the example given in Chaptre 9, Section 9.2.1 to see how MVC-I architecture concretely works

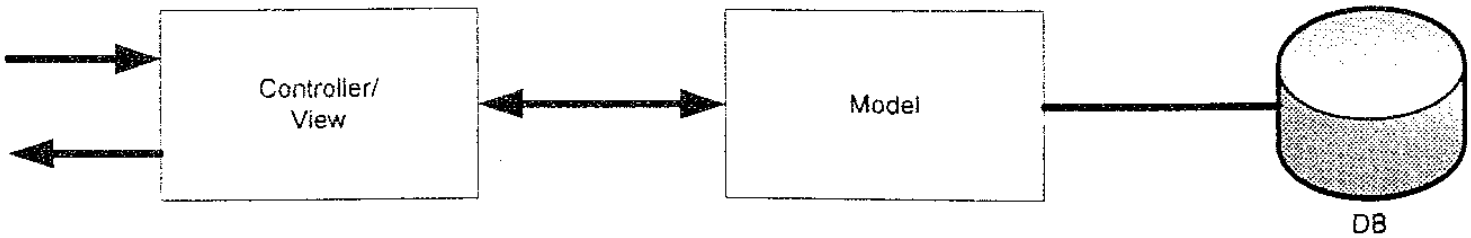

Figure: MVC-I architecture

MVC-II architecture

- The Model module provides all core functionality and data supported by database (Same as MVC-I)
- The View module displays the data from the Model module
- The Controller module
  - It takes input requests, validates input data, initiates the Model and the View and their connection
  - It dispatches tasks

- The Controller and the View register with the Model module
- Whenever the data in the Model module is changed the View module and the Controller module are notified
- Comparison to MVCI
  - In both MVC-I and MVC-II, Model module plays an active role
  - In MVC-II architecture, the View module and the Controller module are completely separated

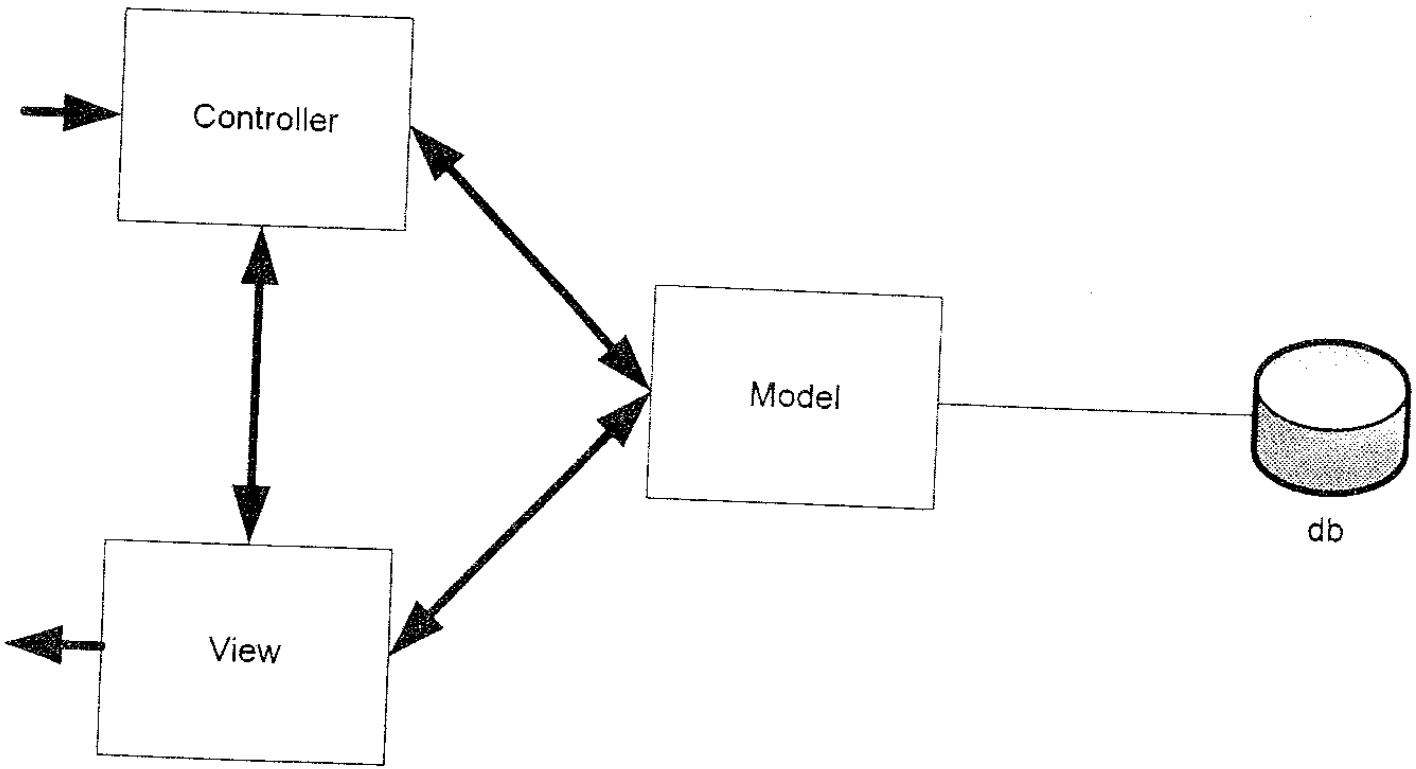

Figure: MVC-II architecture

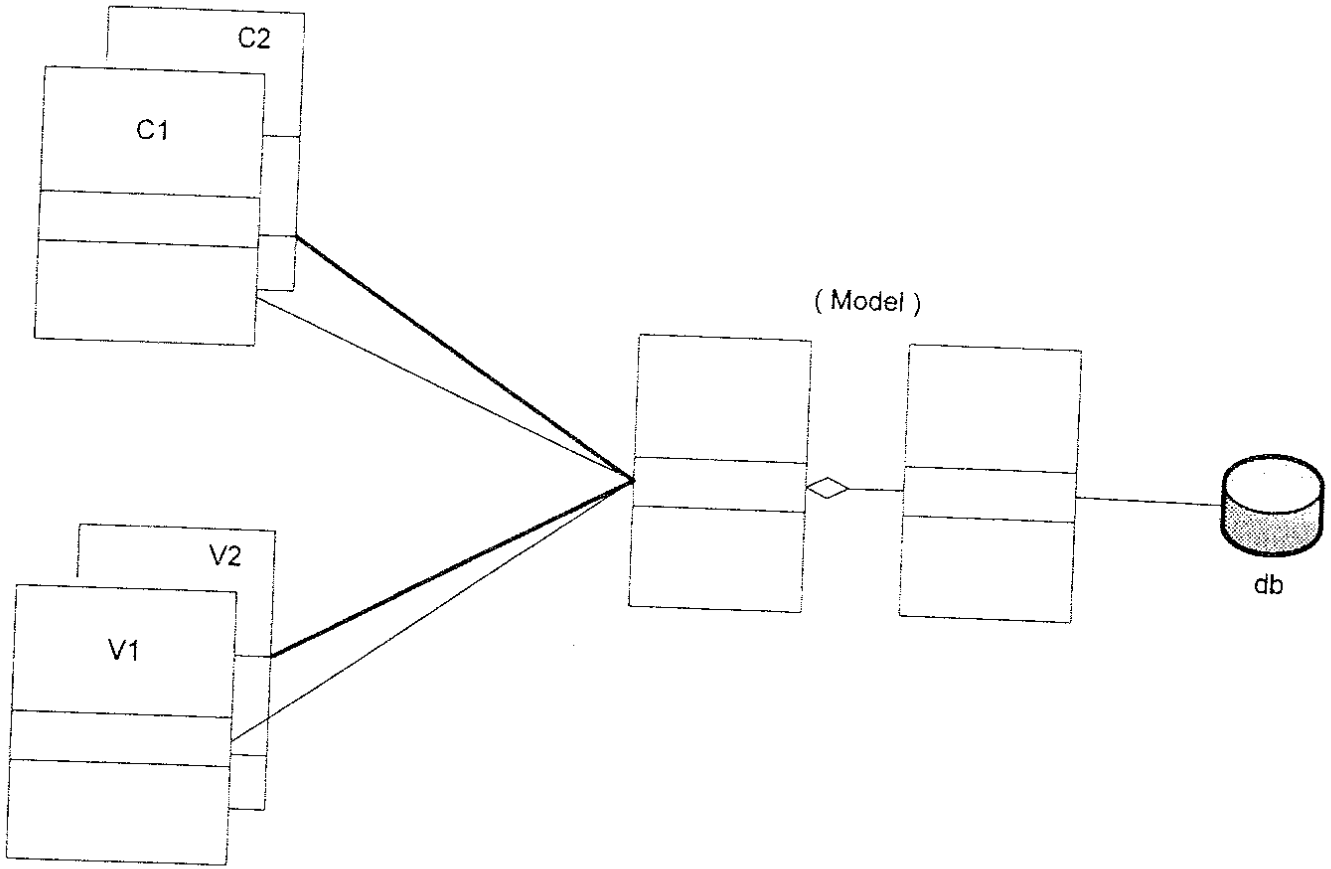

Figure: A detailed MVC-II architecture

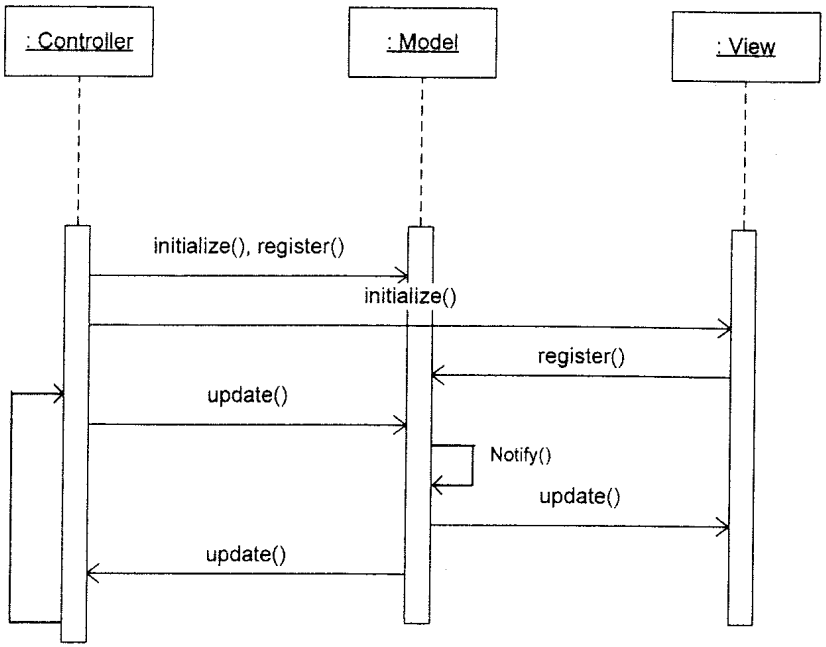

Figure: Sequence diagram for MVC architecture

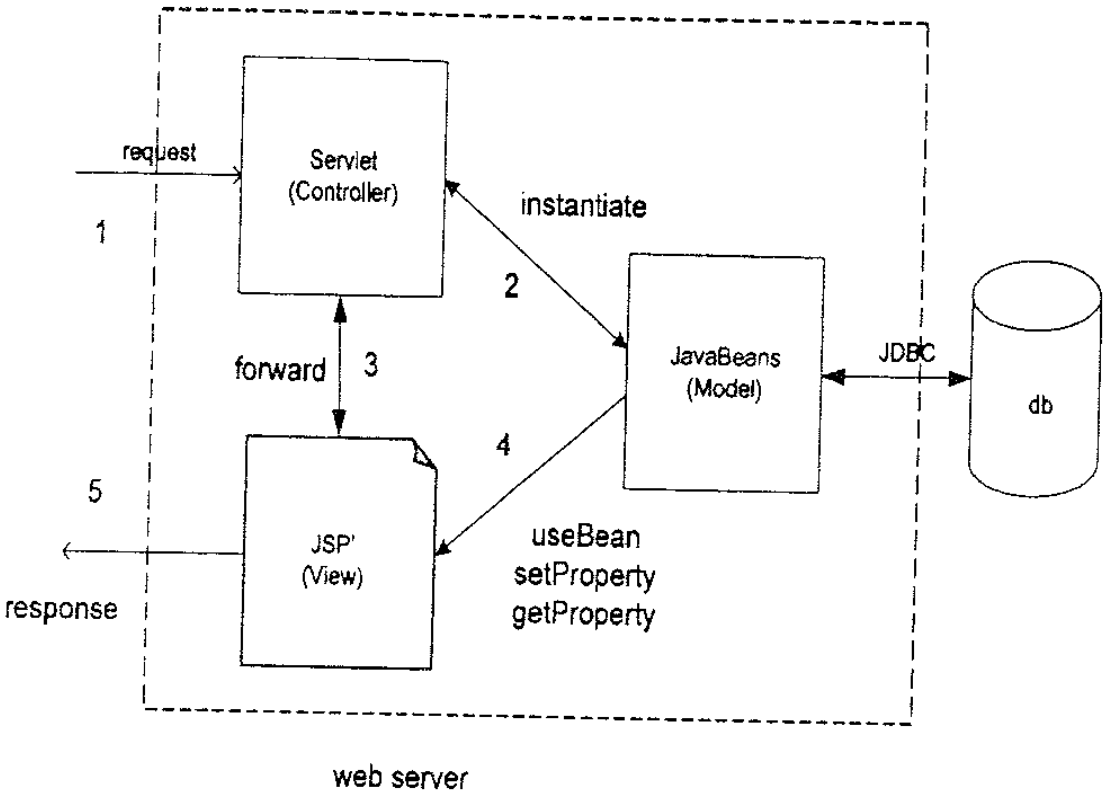

Figure: MVC architecture on Java Web platform

- Applicable domain of MVC architecture
  - Suitable for interactive applications (multiple views are needed + volatile graphics interfaces)
  - There are clear divisions between controller, view, and data modules (different professionals can be assigned to work on different aspects of the system)
- Benefits
  - Many MVC vendor frameworks available
  - Multiple views synchronized with same data model
  - Easy to plug in new or change interface views, update interface views with new technologies
  - Very effective for developments (team = graphics, programming, and data base professionals)

- Limitations
  - Does not fit agent-oriented application such as interactive mobile, robotics applications
  - Multiple pairs of controllers and views based on the same data model make any data model change expensive
  - The division between the View and the Controller is not very clear in some cases
- Related architecture
  - Implicit invocation architecture such as event-based, Multi-tierarchitecture, and Presentation-Abstraction-Control (PAC)

- The PAC architecture is quite similar to MVC
- The PAC was developed from MVC to support the application requirement of multiple agents in addition to the interactive application requirement
- The PAC three components concepts are applied to all concrete sub-system architecture
- It is very suitable for any distributed system where each remote agent has its own functionalities with data and interactive interface
- Another feature: all agents need to communicate with other agents in a well structured way

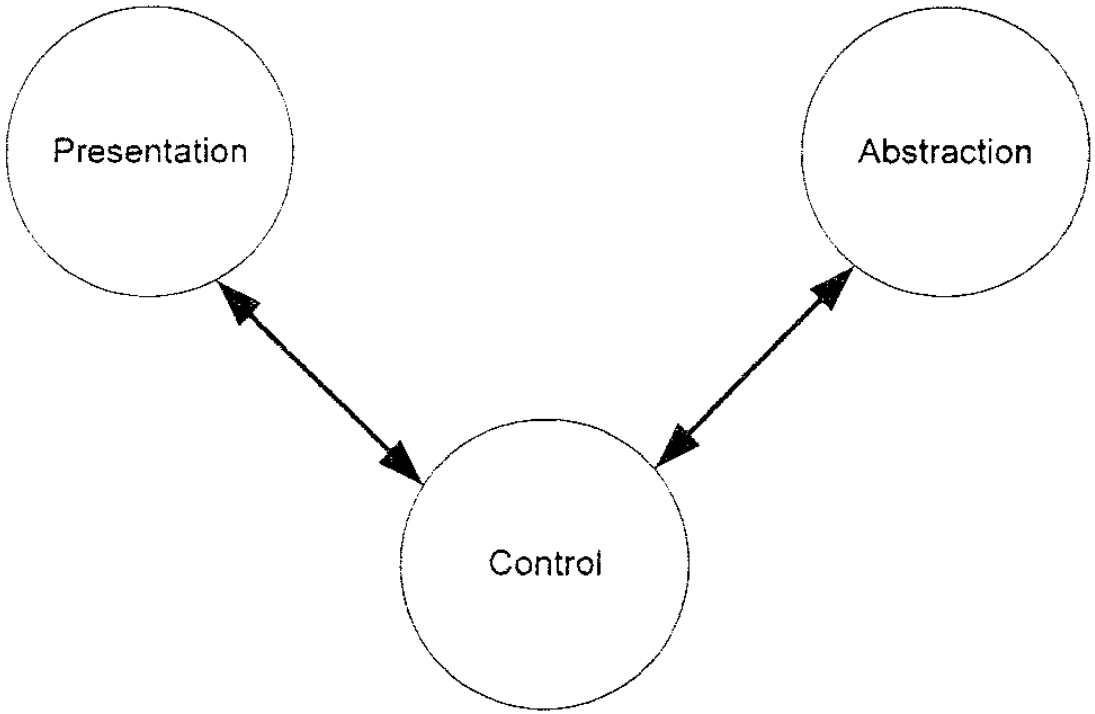

Figure: A single agent in PAC

- Applicable domain of PAC architecture
  - Interactive system where the system can be divided into many cooperating agents in a hierarchical structure (Each agent has its specific job)
  - The coupling among the agents is expected very loose (change of one agent will not affect the others)
- Benefits
  - Supporting multi-tasking, multi-viewing
  - Supporting agent reusability and extensibility
  - Easy to plug in new agent or replace an existing agent
  - Supporting concurrency (agents are in different threads or different devices or computers)

- Limitations
  - Overhead due to
    - the control bridge between presentation and abstraction
    - the communications of controls of many agents
  - Difficult to determine the right numbers of the agents based on the loose couplings between agents and high independence of each other
  - Development complexity: due to complete separation of presentation and abstraction (communications between agents only take place between the controls of agents)
  - Increased complexity of the system design and implementation
- Related Architecture
  - Layered architecture, multi-tier architecture, MVC architecture

- The client-server model is the most common distributed system
- It is based on two communicating subsystems (usually running on different processors)
  - Client issues a request to the second process server
  - Server process receives the request, carries it out, and sends a reply to the client

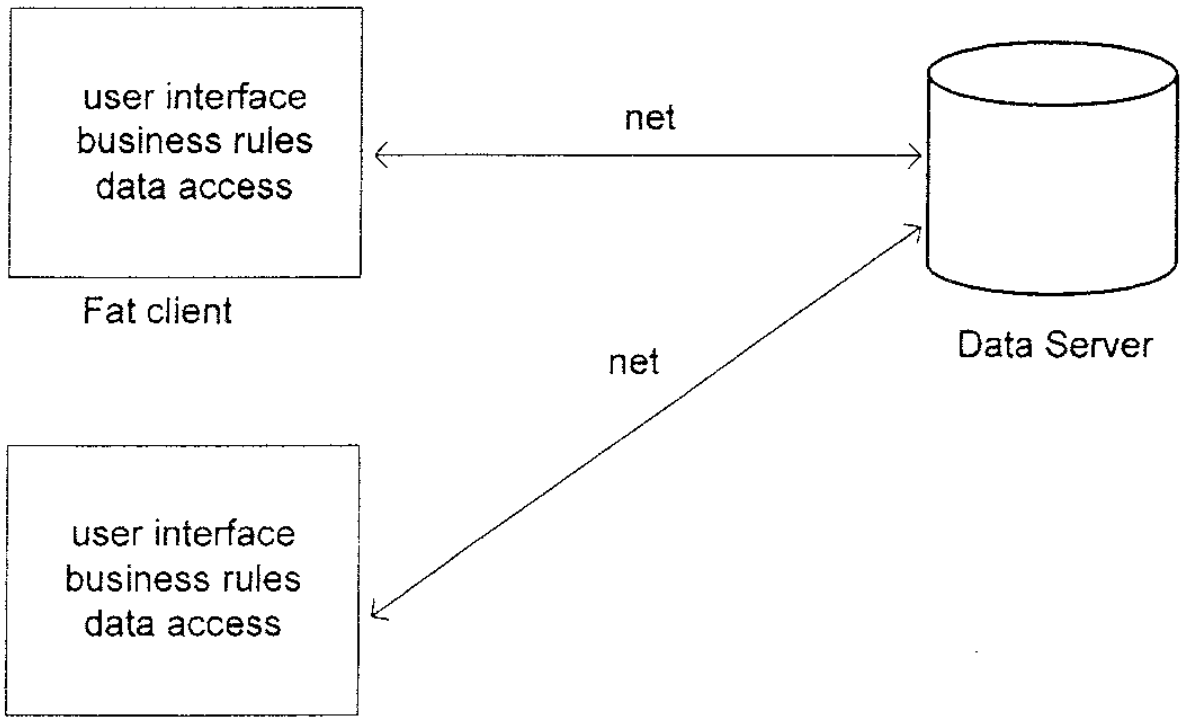

Figure: Two tier client/server architecture

- Advantages
  - Separation of responsibilities such as user interface presentation and business logic processing
  - Reusability of server components
- Disadvantages
  - Lack of heterogeneous infrastructure to deal with the requirement changes
  - Security complications
  - Server availability and reliability
  - Testability and scalability
  - Fat-clients/ Thin-clients (depends on the application)

- The front tier in a multi-tier architecture is the user interface presentation tier
- The middle-tier(s) take(s) care of business logic, application decision, and execution
- The back-end tier usually works on database management, or on a (virtual) machine
- The advantages of multi-tier over the two-tier architecture are
  - the enhancement of reusability
  - scalability by the middle tier
    - The middle tier can also provide multi-threading supports for scalability
  - Multi-tier architecture also reduces the traffic on the network
- Disadvantage: complex testability

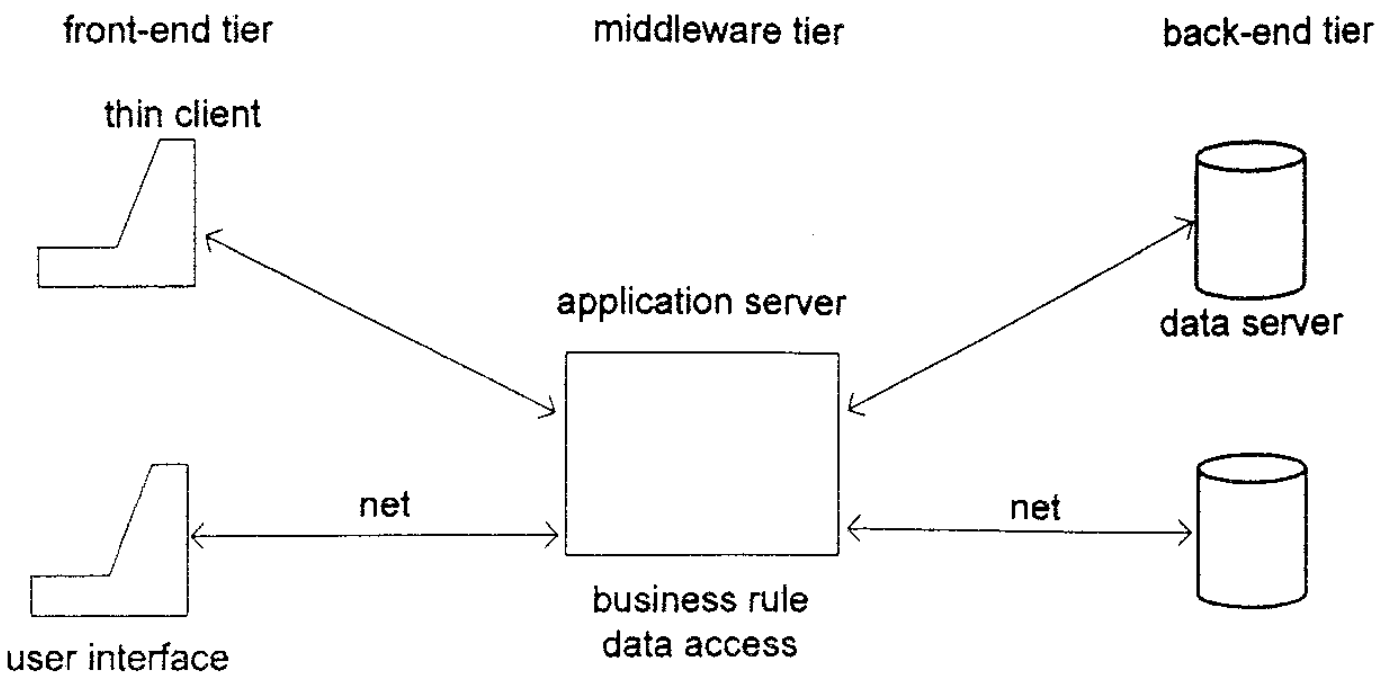

Figure: Three tier architecture

- The broker architecture is a middleware architecture widely used in distributed computing
- It is suitable for distributed computing that coordinates and facilitates communication
  - brokering the service requests
  - locating proper server
  - forwarding and dispatching requests
  - sending responses or exceptions back to clients
- It can be used to structure distributed software systems with decoupled components that interact by remote service invocations
- The most important quality of this architecture = better decoupling between clients and servers

- Servers make their services available to their clients by registering and publishing their interfaces with the broker
- Clients can request the services of servers from the broker statically or dynamically by look-up
- A broker acts as a policeman in a busy intersection who controls and interacts with the client components and server components
- The connection between clients and servers is maintained by the broker

- A distributed client can access distributed services simply by calling a remote method of a remote object
- This concept is similar to unix Remote Procedure Call (RPC) and Java Remote Method Invocation (RMI)
- The clients can dynamically invoke the remote methods even if the interfaces of the remote objects are not available at the compilation time

- Client has a direct connection to its client-proxy
- Server has direct connection to its server-proxy
- The proxy talks to the mediator-broker
- The proxy is a well known pattern for hiding low-level detailed communication processing
  - It intercepts the client's request
  - gets all arguments
  - packets it
  - marshals (streamlines) and formats the package in the format of communication protocol
  - sends it to the broker
- A broker system is also called proxy-based system

Sub-components of a broker architecture

- Broker
- Stub (client-side proxy): It mediates between client and broker
- Skeleton (server-side proxy)
  - It is statically generated by the service interface compilation and then deployed to the server side
  - It receives the requests, unpacks the requests, unmarshals the method arguments, and calls the appropriate service
  - It also marshals results from the sever before it sends it back to the client
- Bridges (Optional)
  - Used to hide implementation details when two brokers interoperate
  - Can connect two different networks based on different communication protocols

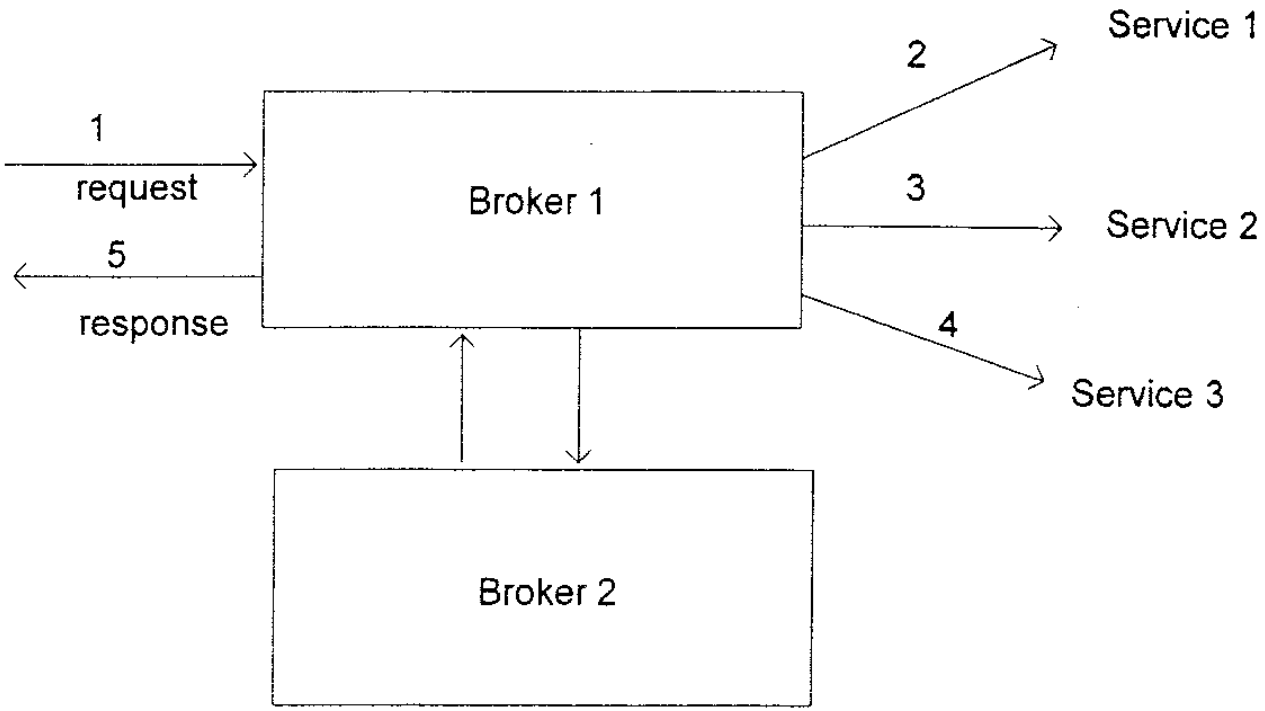

Figure: Broker model

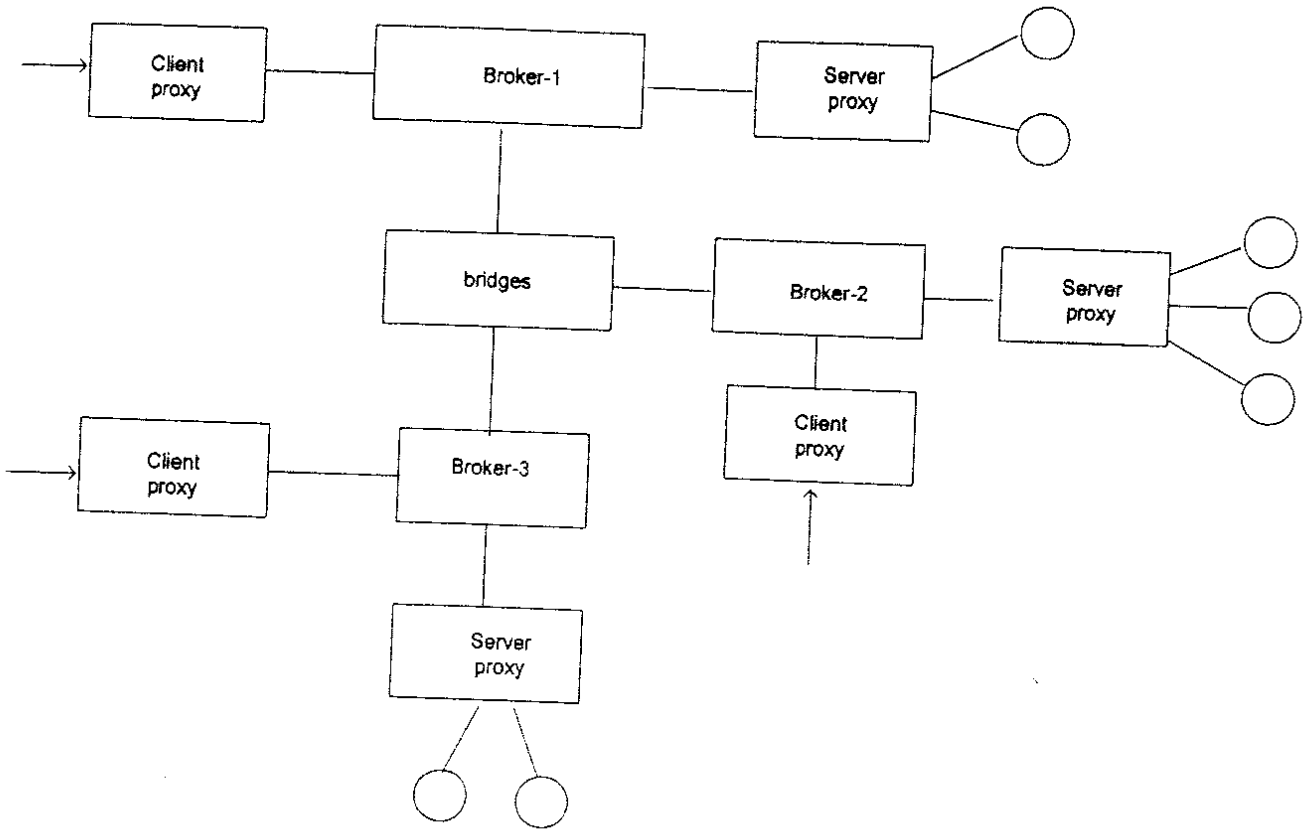

Figure: Connected brokers with client/server proxy

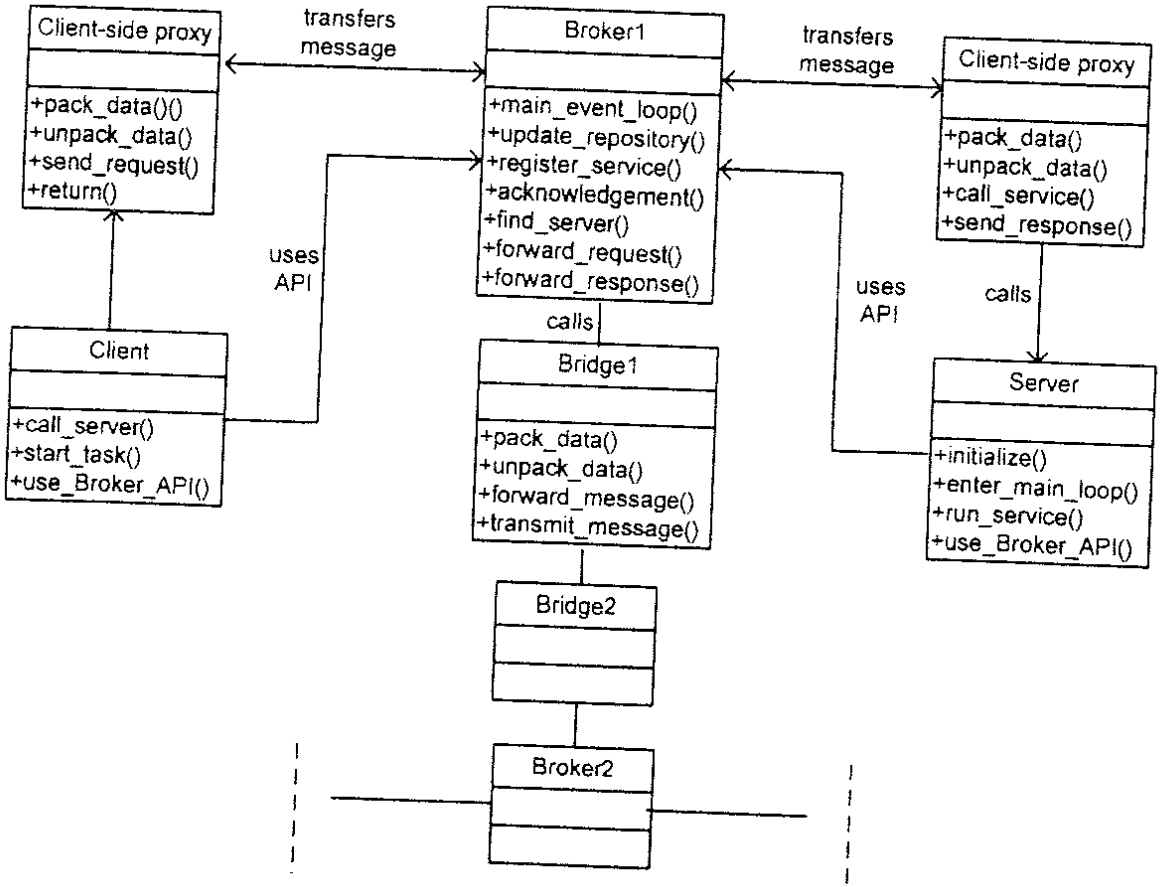

Figure: Class diagram for broker architecture

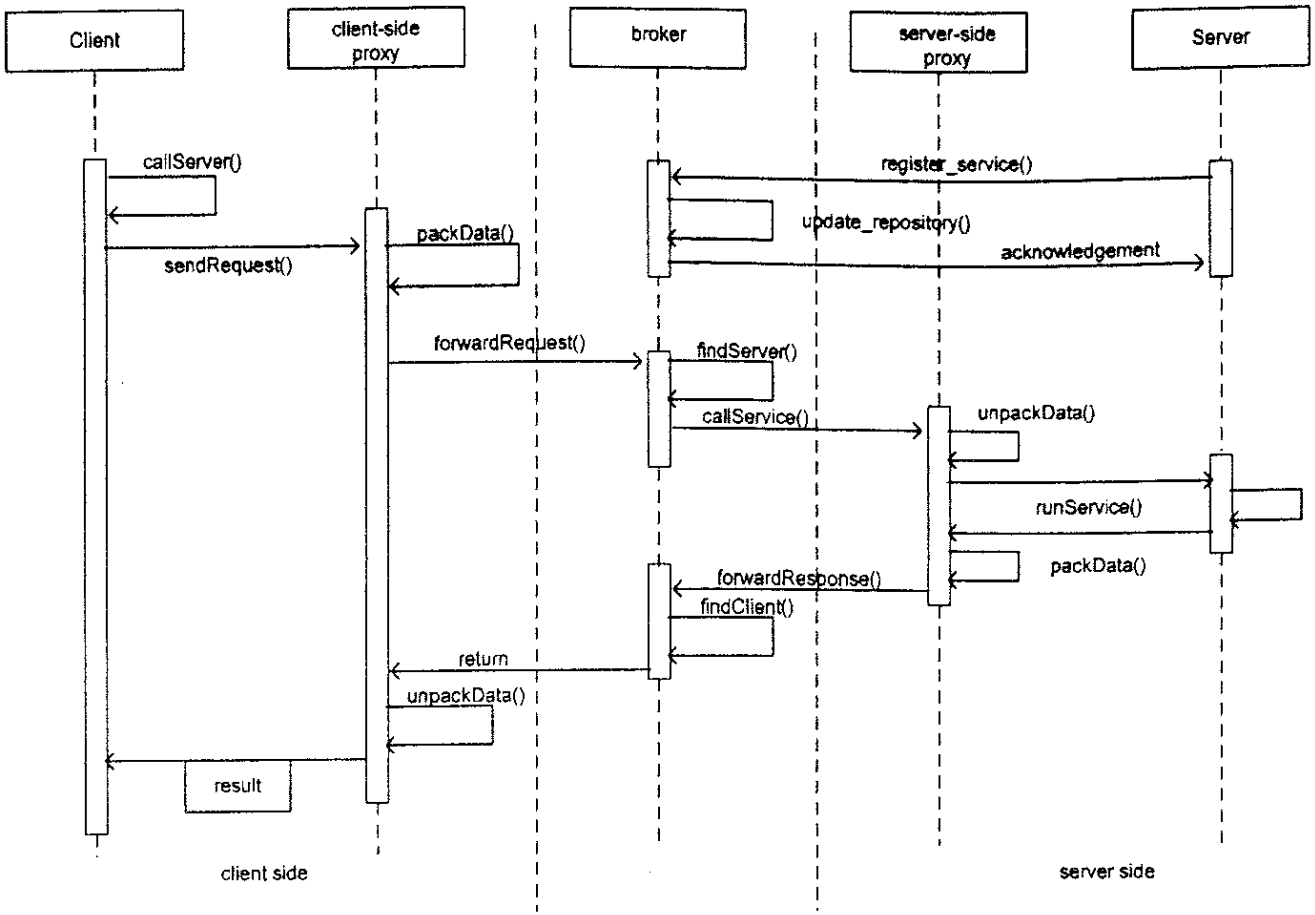

Figure: Sequence diagram for broker architecture

- Advantages
  - Server component implementation and location transparency
  - Changeability and extensibility
  - Simplicity for clients to access server and server portability
  - Interoperability via broker bridges
  - Reusability
  - Feasibility of runtime changes of server components (add or remove server components)
- Disadvantages
  - Inefficiency due to the overhead of proxies
  - Low fault-tolerance
  - Difficulty in testing

- A service is a business functionality that is
  - well-defined and self-contained
  - independent from other services
  - published and available to be used via an interface

- SOA services can be reused extensively regardless of whether they are based on new or legacy applications
- Loose coupling of service-orientation architecture provides a great flexibility for enterprises to make use of all available service resourses

- The connections between services are conducted by common and universal message oriented protocols such as the SOAP Web service protocol
- A connection can be established statically or dynamically

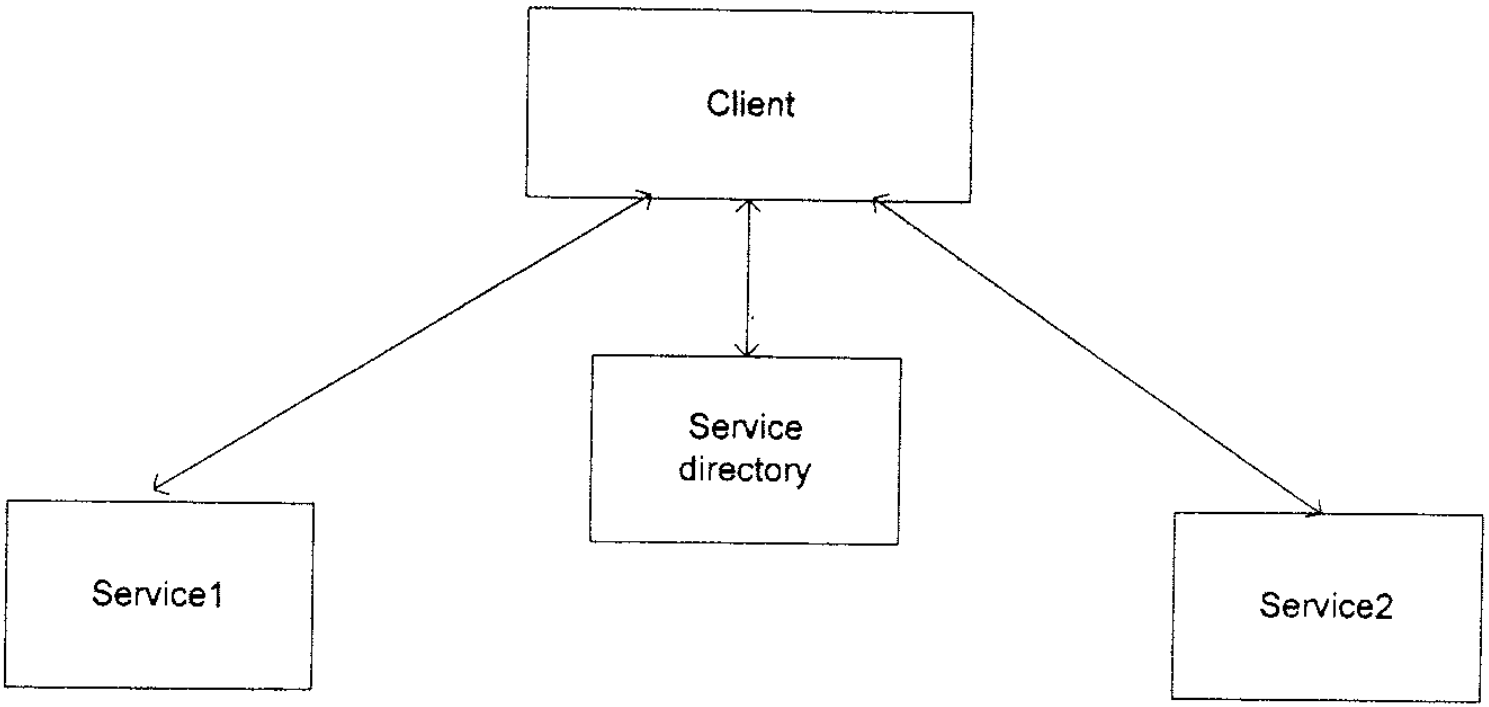

Figure: Client with services and service directory

- A service-oriented application might make use of many available services
- For that one needs a flow control language
  - allows specifying the sequence and logical order of the business executions based on the business logic
- Some services can be reused by other applications that they are not originally designed for
- We can build a new service out of existing services
  - aggregation: extends one endpoint of a service to make a new interface of the new service
  - containment structure: has one interface that wraps all used services
- Services can be recursively constructed to satisfy a more complex business needs (through aggr. and cont.)

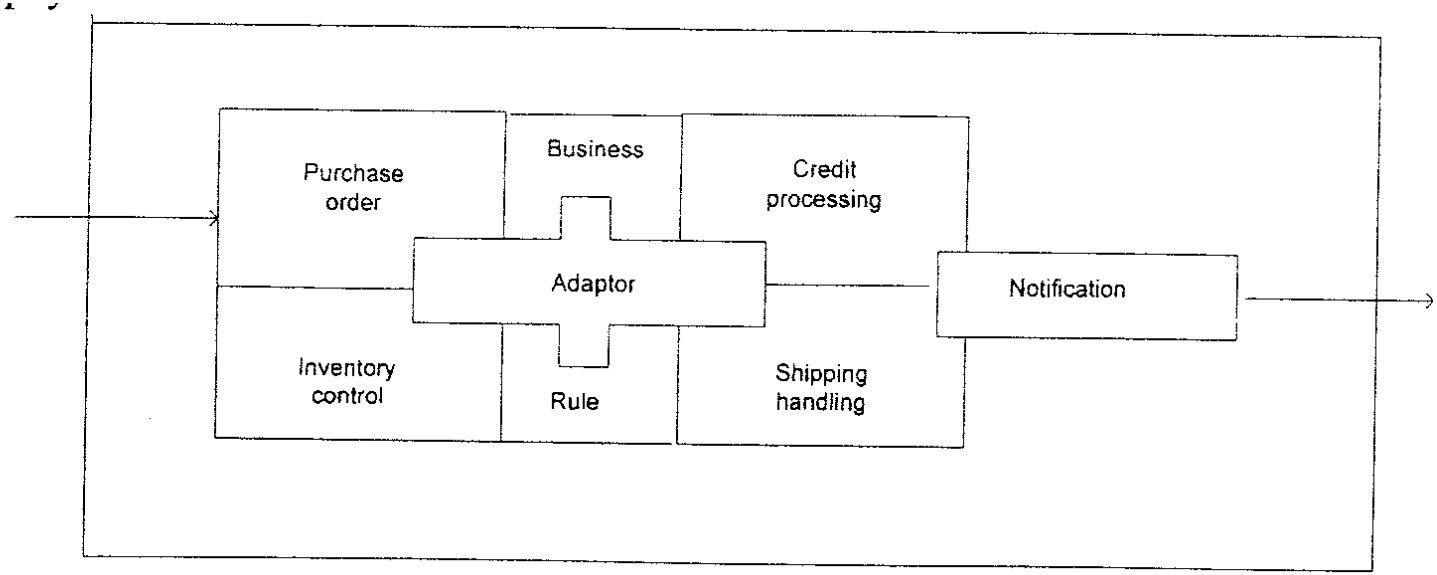

Figure: Service composition

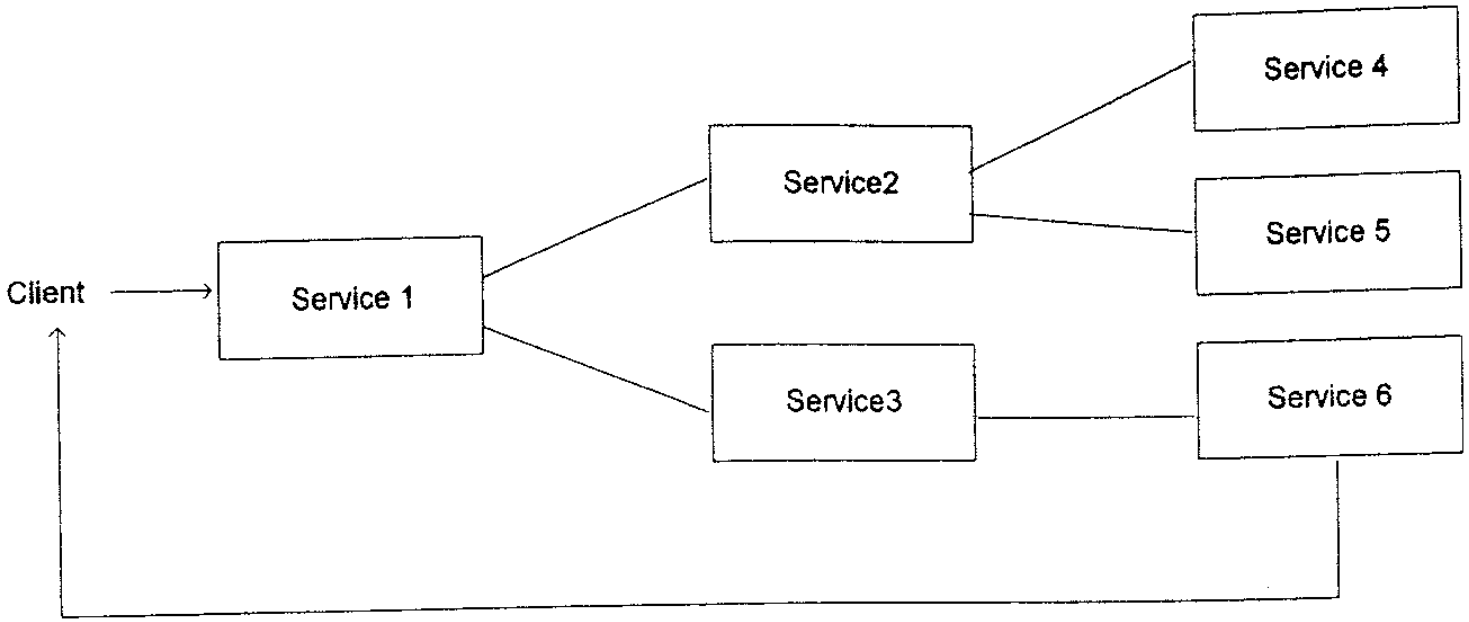

Figure: Service reuse

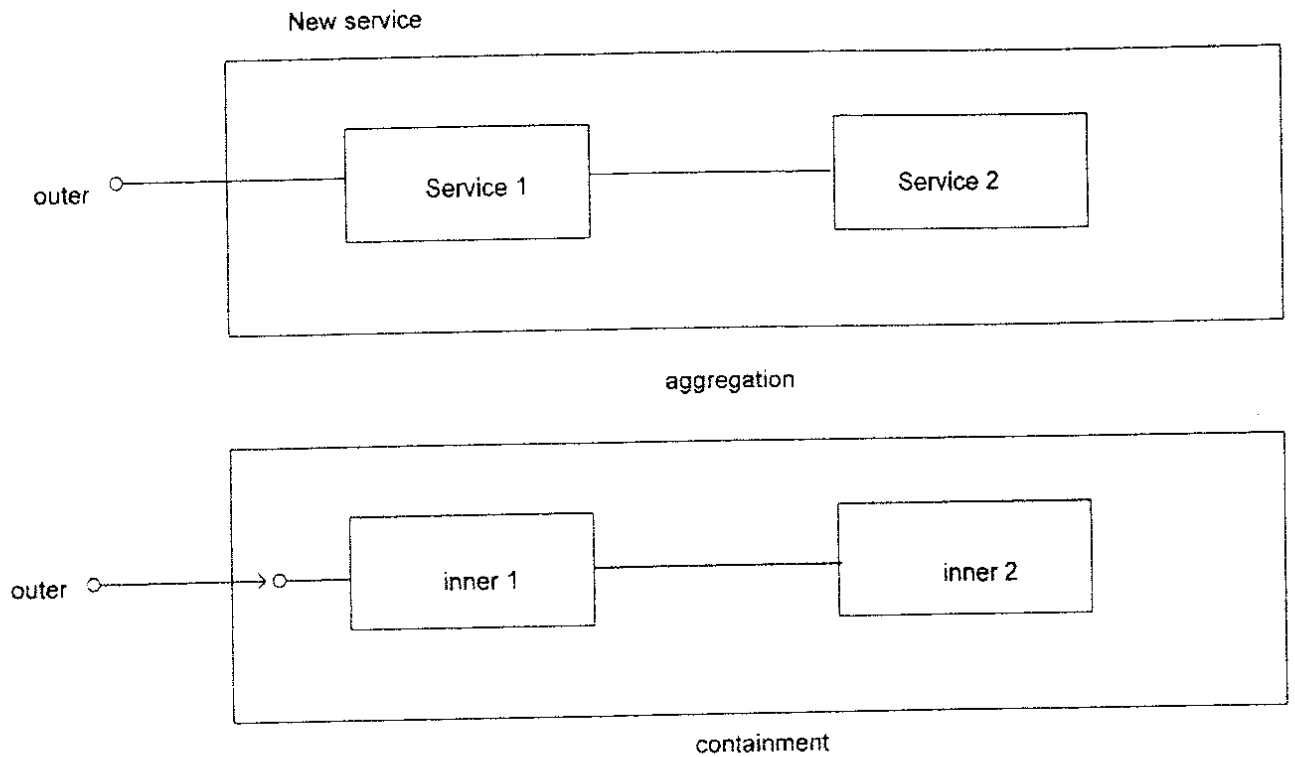

Figure: Service composition model

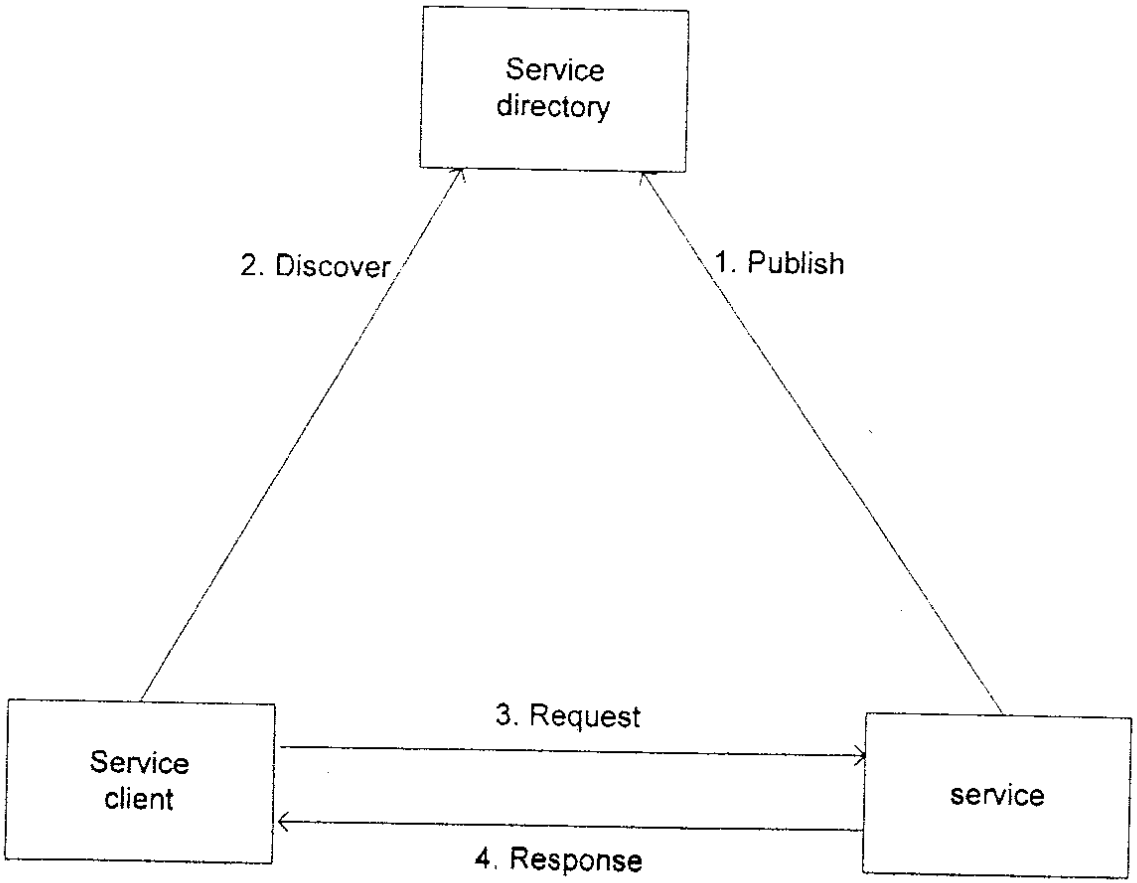

Figure: Service working model

Advantages of SOA

- Loosely-coupled connection
- Each service component is independent from other services due to the stateless service feature
- Interoperability: Technically any client or any service can access other services regardless of their platform, technology, vendors, or language implementations
- Reusability: Any service can be reused by any other services and service is developed to be reused as well
- Scalability: Loosely coupled services make themselves easy to scale