Software Architecture:

Software Architecture is the process of designing the global organization of a software system, including dividing software into subsystems deciding how these will interact, and determing their interfaces. Supports object oriented and procedural oriented development.

Importance of Software architecture / Architectural Pattern

1) To enable everyone to better understand the system:

As a system becomes more and more complex, making it understandable is an increasing challenge. This is especially true for large distributed system that use sophisticated technology. A good architectural model allows people to understand how the system as a whole works. It also includes the term that people use when they communicate with each other about lowel-level details.

2) To allow people to work on individual pieces of the system in isolation:

The work of developing a complex software system must be distributed among a large number of people. The architecture allows the planning and coordination of this distributed work.

The architecture should provide sufficient information so that the work of the individual people or teams can later on be integrated to form the final system.

3. To prepare for extension of the system:

- With a complete architectural model, it becomes easier to plan the evolution of the system.
- Subsystems that are planned to be part of a future release can be included in the architecture, even though they are not to be developed immediately.
- It is then possible to see how the new elements will be integrated, and where they will be connected to the system.

Thus, it helps to prepare for extension of the system.

4. To facilitate reuse and reusability:

- The architectural model makes each system component visible.
- By analyzing the architecture, we can discover those components that can be obtained from the past projects or from third parties. (reuse)
- We can also identify components that have high potential reusability in our system.

Hence, the architectural model is the core of the design; therefore all software engineer need to understand it. Poor decision made while creating this model will constraint/limit the subsequent design.

Contents of good architectural model

1) the logical breakdown of the system into subsystems. (packages)
2) The interfaces among the sub-system.
3) The way in which the components interact with each other at runtime. (Behavioral diagram)
4) The data that will be shared among the subsystem. (class diagram)
5) The components that will exist at runtime and the machines or devices on which they will be located. (component and deployment diagram)

Challenges in Architectural Modeling:

1) To produce a relevant ~~and~~ picture of a large and complex system.
2) Reader must be able to understand the system very quickly by looking at the different views.
3) Architectural model must be designed to be stable to ensure maintainability and reliability.
(Being stable means that the new features can be easily added with only small changes to the architecture).
4) The architecture expressed clearly enough that it can be used to communicate effectively with clients.

Steps for developing the architectural Model

1. Start by sketching ~~this~~ an outline of architecture
   - based on the principle, requirements and use cases
   - Determine the main components that will be needed.

based on principle & ~~requirement of architecture~~

- how the components interact
- how to distribute data and functionality
- how we can use reuse existing frameworks.

- choose among the various architectural pattern.

② Refine the architecture
- Identify the main ways in which the components will interact and their interfaces.
- Describe how data and functionality will be distributed among the various components.
- Determine if we can reuse an existing framework.

③ Consider each use case and adjust the architecture to make it realizable.
- finalize interface for each component
- make each use case realizable.

④ Mature the architecture by defining final class diagram and interaction diagram.

Make architecture mature by defining final class diagram
- Interaction system

Architecture - Centric Process:
- The unified process insists that architecture sit at the heart of the project team's efforts to shape the system
- Since no single model is sufficient to cover all aspects of a system the up supports multiple architectural models and views.
- The architecture description is a view of the models of the system; views of the use case, analysis, design, implementation, and deployment models. It describes the part of the system that are important for all developers and stakeholders to understand

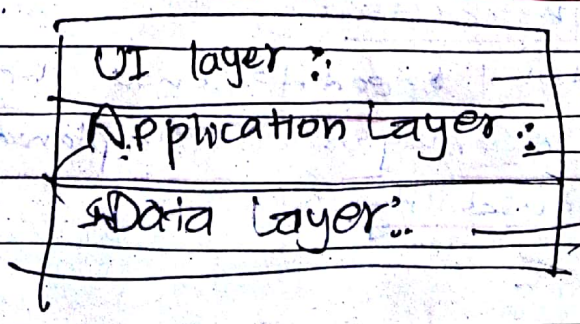
* Architecture description lets the architect control the development of the system from a technical prospective.
* It focuses on subsystems, classes, components and nodes as well as their collaborations.

Architectural Patterns:

- The notations of patterns can be applied to software architecture. These are called architectural patterns which allows us to design flexible systems using components that are as independent of each other as possible.

① The Multi-layered Architectural Pattern:

- In a layered system each layer communicates only with the layers below it.
- Each layer has a well-defined interface used by the layer immediately above it.
- The higher layer sees the lower layer as a set of services.
- A complex system can be built by superposing layers at increasing layers of abstraction.
- It is important to have a separate layer for the UI.
- Layer immediately below the UI layer provide the application function determined by the use cases.
- Bottom layers provide general services such as data storage and transmission.
- Most operating systems and communication systems are built according to the multi-layered architectural pattern.



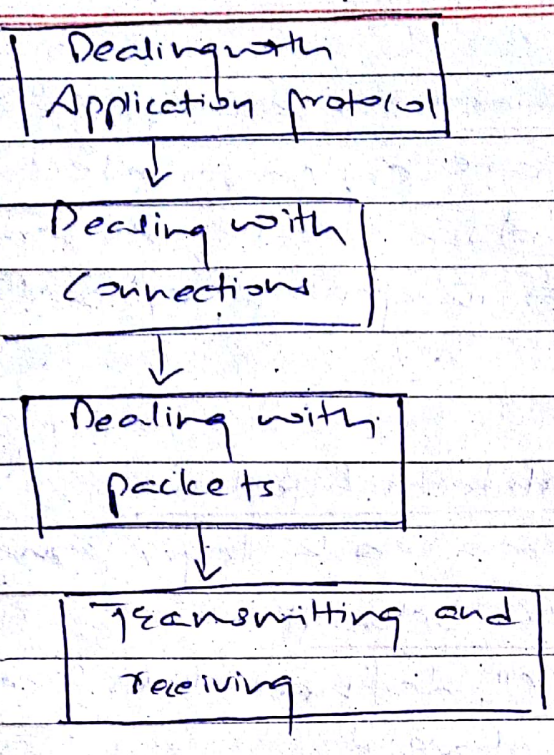


fig: simplified view of layers in a communication system.

Design Principles:

i) Divide and Conquer:
The layers can be independently designed.

ii) Increase cohesion:
well designed layers have layer cohesion.

iii) Reduce coupling:
well designed lower layers donot know about the higher layers. and the only connection between layer is through the API.

iv) Increase Abstraction:
We donot need to know the detail of how the lower layers are implemented.

v) Increase Reusability:

The lower layers can often be designed more generically

vi) Increase Reuse:

We can reuse layers built by others that provides the services we need.

vii) Increase flexibility:

We can add new facilities or replace layers.

viii) Anticipate Obsolescence:

By isolating components in seperate layers the system becomes more resistant to obsolescence.

ix) Design for portability:

All the facilities that are dependent on a particular platform can be isolated in one of the lower layers to increase portability.

x) Design for Testability:

Individual layers can be tested independently.

xi) Design Defensively:

Since layers communicate only through APIs we can design a system so that if a higher layer fails, the lower layers continues to run and vice-versa

Client-Server And other Distributed Architectural Patterns:

Client-Server:

- There is atleast one component that has the role of server, waiting for and then handling connections.
- There is atleast one component that role of client initiating connections in order to obtain some services.
- 2-tier

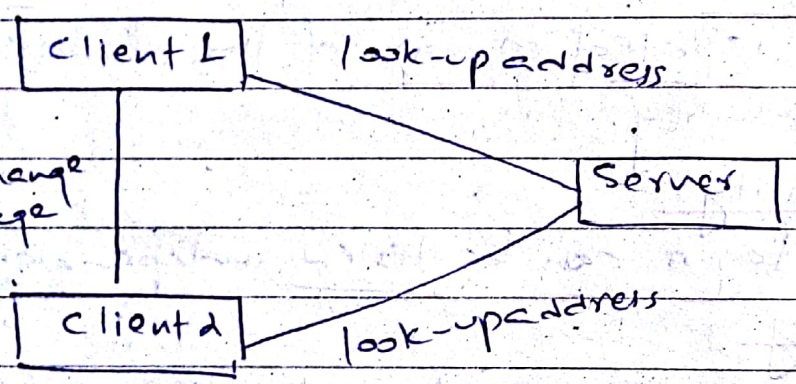


fig: client_server model

Distributed architecture:

- When we have more than one machine involved in a system, it is called distributed architecture.
- Client-Server is distributed to an extent but usually client-server is tied to two or three tier.
- distributed could be n-tier.

Distributed System in Modern Context

There may be millions of request made to the server of some popular web applications such as facebook, Google. This cannot be handled by a single computer. So, distributed system having many server computers are connected together so that all the requests are handled properly. Distributed system in this context is easy to scale-up and is reliable.

Principles:

1) Divide And conquer:

Dividing the system into client and server processes is a strong way to divide the system. Each can be separately developed.

2) Increase cohesion:

The server can provide a cohesive service to clients

3) Reduce Coupling:

There is usually only one communication channel exchanging simple message.

4) Increase abstraction:

A client do not need to understand the details of how a server operates.

5) Increase Reuse:

We can find suitable framework to develop good distributed systems. However, reusability may not be high since client server systems are often very application specific.

Broker Architectural patt - Broker
Transaction Processing AP - Transaction dispatcher
Pipe and filter A.P - Pipe and filter

7) Design for flexibility:
Distributed system can often be easily reconfigured by adding extra servers or clients.

9) Design for portability:
We can write clients for new platforms without having to port the server.

10) Design for testability:
We can test client and server independent

11) Design Defensively:
We can put strict checks in the message handling code.
Server ↔ client ko req check krte h

Model-View Controller (MVC) Architectural Pattern:
- An architectural pattern used to help seperate the user interface from other parts of the system.
- The model contains the underlying classes whose instances are to be viewed and manipulated.
- The view contains objects used to render the appearance of the data from the model in the user interface.
- The controller contains the objects that control and handle the users information with the view and model.

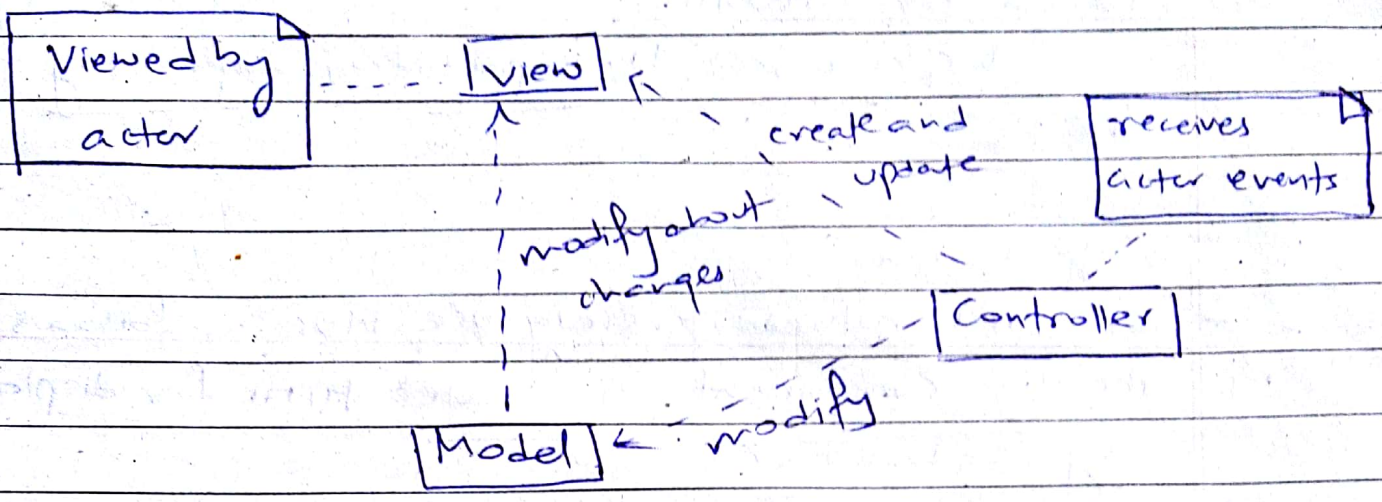


fig: Model-View Controller.

Principles:

1) Divide and conquer:
The three components can somewhat independently designed.

2) Increase cohesion:
The components have stronger layer cohesion ~~than if the~~ view and controller are separate layers.

3) Reduce coupling:
The communication channels between the three components are minimal

6) Increase Reuse:
The view and controller normally make extensive use of reusable components for various kinds of UI controls.

7) Design for flexibility:
It is usually quite easy to change the UI by changing the view, the controller, or both.

10) Design for testability:
We can test the application separately from the UI.

Application:
- Web Architectures generally use MVC as follows:
1. The view component generates HTML for display by the browser.
2. There is a component that interprates HTTP 'post' transmission coming back from the browser - this is the controller.
3. There is an underlying system for managing the information - this is the model.

- Components of JAVA APIs are based on MVC architectural pattern.

Service-Oriented Architectural Pattern
- The service-oriented architectural pattern organizes an application as a collection of services that communicates using well-defined interfaces.
- In the context of internet, the services are called web services.
- A web service is an application, accessible through the internet, that can be integrated with other services to form a complete system.

- The different components generally communicate with each other using open standard such as XML:

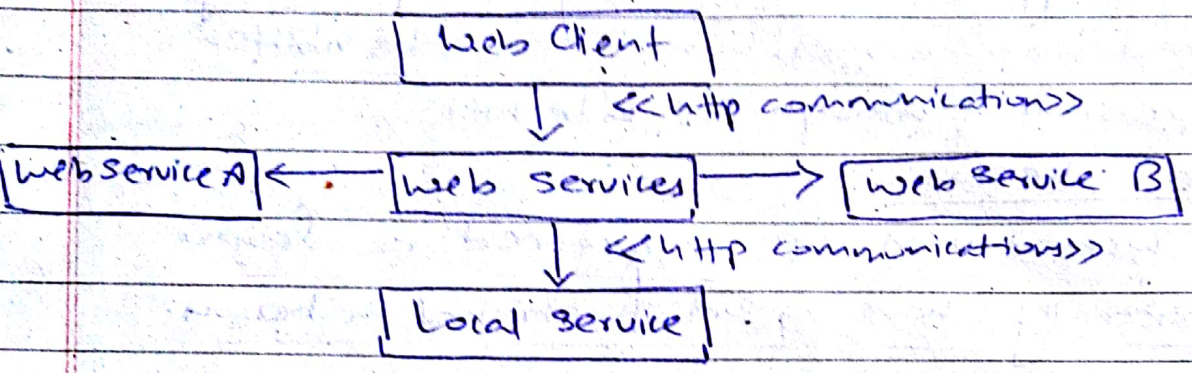


fig: Service oriented Architectural Pattern

- In the above figure a client program we are developing obtains information from a web-service provided by some company on the internet. This inturns call upon two other web services 'A and B' also available on the internet.
- In addition it uses web services protocols to access a local server operated by the same company.
- The different components of web service architecture ~~is~~ communicate with each other using open web standards.
- A web service may be bound to many other web-services over the internet.
- A web service can be accessed by many applications in many locations.

Not 4, 7

Principles:

1) Divide And conquer:
The application is made of independently designed web services, which are distributed and accessible through the internet

2) Increase cohesion:
The web services are structured as layers and generally have good functional cohesion.

3) Reduce coupling:
web based applications are loosely coupled built by binding together distributed components.

5) Increase Reusability:
A web service is a highly reusable component.

6) Increase Reuse:
web based application are built by reusing existing web services.

8) Anticipate Obsolescence:
Obsolete services can be replaced by new implementation without impacting the applications that use them.

9) Design for portability:
A Service can be implemented on any platform that supports the required standards.

10) Design for testability:
Each service can be tested independently.

11) Design Defensively:
web services enforces defensive design since different applications can access the services

The Message-oriented Architectural Pattern

- In this architecture, the different subsystems communicate and collaborate to accomplish some task only by exchanging messages.
- Also known as Message-oriented Middleware (MOM)
- The core of this architecture is an application-to-application messaging system.
- Senders and receivers need only to know what are the message formats. (ie the receiving application doesnot have to know anything about the software component that sent the messages
- In addition the communicating applications do not have to be available at the same time

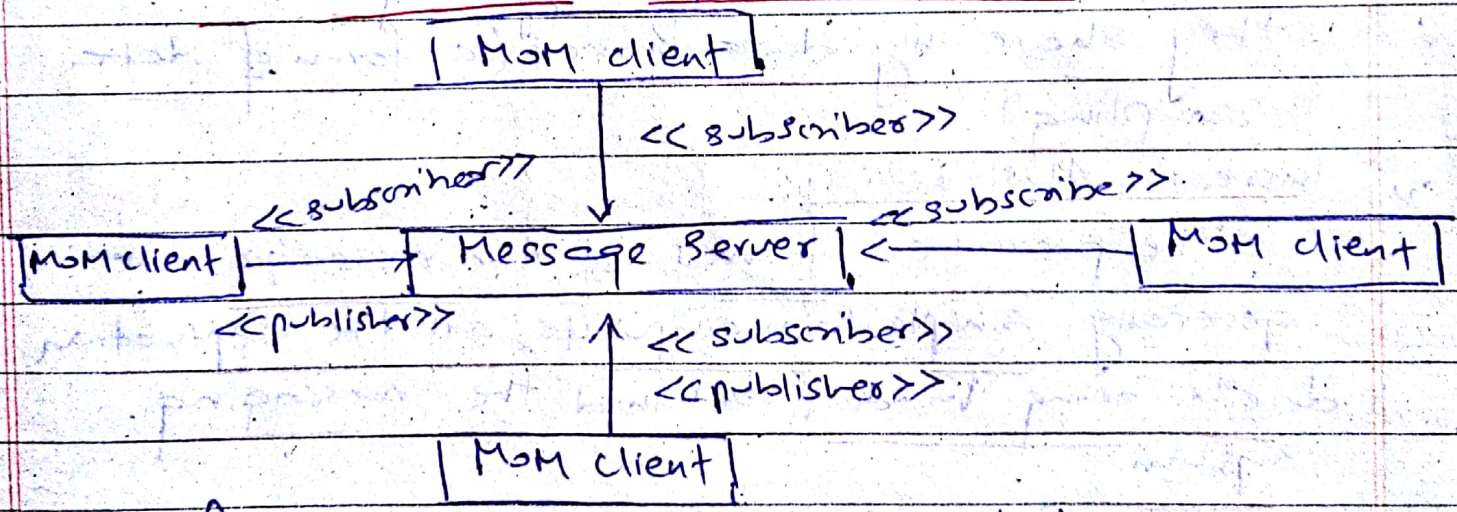


fig: The message oriented architectural pattern.

- In message oriented architectural pattern, the messages are sent through virtual channels - also called topics
- Software components that send messages to topics are called publishers.
- To receive messages an application must suscribe to a topic.

- Any message sent to a given topic is delivered to all the topics subscribers, each of them receiving a copy of message
- Message delivery is completely asynchronous.
- Given message is delivered once and only once to each subscriber.

Principles: Not 2, 8, 9

1) Divide And conquer:
The applications is made of isolated software components independently designed and distributed across a network.

3. Reduce coupling:
The components are loosely coupled since they share only data format (a form of data coupling)

4) Increase Abstraction:
The prescribed format of the messages are generally simple to manipulate, all the application details being hidden behind the messaging system.

5) Increase Reusability:
A component will be reusable if the message formats are flexible enough.

6) Increase Reuse:
The new system can reuse different components as long as it can accomodate the message format of the reused component.

9. Design for flexibility:

The functionality of a message oriented system can be easily updated or enhanced by adding or replacing components in the system.

10. Design for testability:

Each component can be tested independently.

11. Design defensively:

Defensive Design consists simply of validating all received messages before processing them.