Page No.
Date

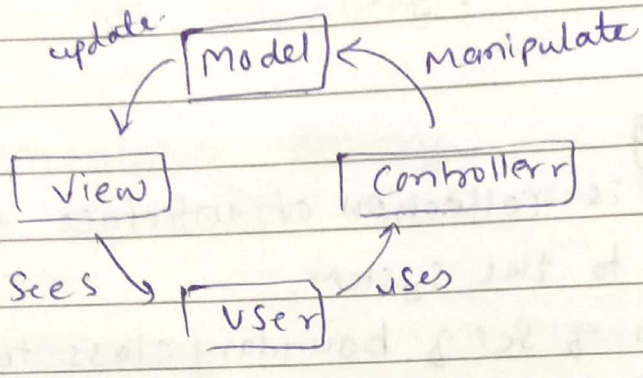
Unit-6

Design pattern.

Q What is pattern?

→ pattern represent a reusable solution to a common design problem.

eg MVC - model view controller.



Q What Makes a pattern?

There are 3 parts

① context → It is situation due to which the problem is raised.
→ make list of all situation

② problem →
→ It is condition in which pattern should be applied
→ It needs to solve.

③ Solution →
→ Answer to problem
→ Solution does not necessarily resolve all the forces associated with the problem.

Two aspects.
① Static aspect ② Dynamic aspect

Page No.
Date

pattern categories

(1) Architectural pattern.

- An architectural pattern expresses a fundamental structural organization schema for ~~sy~~ software system.
- It provides a set of predefined subsystem, specifies their responsibilities, and include rule and guidelines for organizing the relationship between them.

(2) Design pattern.

→ A design pattern provide a schema for refining the subsystems or components of a software system or the relationship between them. It describes a commonly recurring structure of communicating component that solves a general design problem within a particular context.

(3) Idioms.

An idiom is low level pattern specific to a programming language.
It describe how to implement particular aspects of components of or the relationship them using the feature of given language.

Page No.
Date

Relationship blw patterns

* one pattern may depend on another pattern
* one pattern problem can solve by another
* eg. model view and controller are dependent on controller.

Pattern Description.

→ Name :
→ Also known as (aka): other name
→ Example :
→ Context :
→ problem :
→ Solution :
→ Structure : a detail specification aspects of the pattern.
→ Dynamic : Describe Scenario of the pattern.
→ Implementation: Guideline for implementing the pattern.
→ variants: variants of pattern.
→ Known Uses:
→ Consequences: adv. & disadv. of pattern.

Communication patterns

→ Many distributed System.
* they required communication System
* Communication pattern is used for Communication blw Software module.

There are 2 aspects

(1) encapsulation: hiding the details of underlying communication mechanism from user.

(2) location transperency:- When apps are allowed to access remote components without any knowledge of their physical location is know Location transperency.

There are 3 pattern.

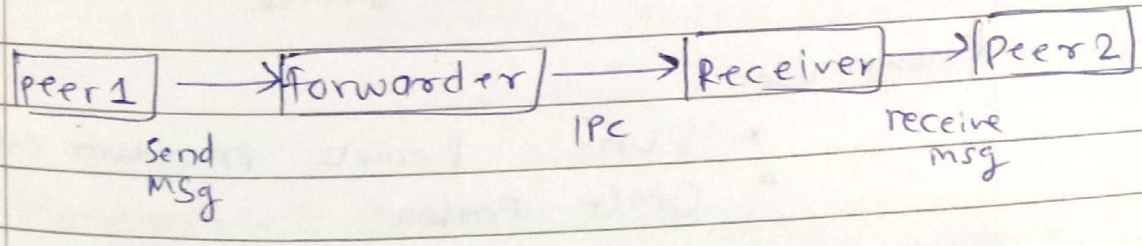
1. forwarder - Receiver [encapsulation]

Problem: 1) Diff platform uses diff Interprocess communication (IPC)
2) performance low.

Solution: encapsulate IPC mechanism.

Intent: use a transparent IPC.

Structure:



Dynamics

Peer 1 Send request to forwarder and name of receiver it marshal the msg then Receiver get the msg and unmarshals and deliver to P2

Page No.
Date

Known use: TASL
REBOOT

Consequences. (1) efficient IPC
(2) encapulation.

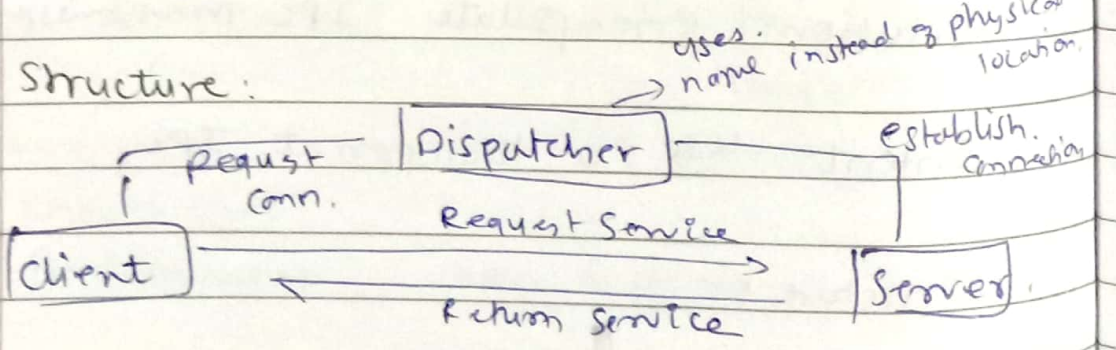
2 Client - Dispatcher ~~System~~ Server [local Transparency]

problem: component should use a service independent of the location of service provider.

Solution: Dispatcher establish communication b/w client & server.

Intent: IPC for system.

Structure:



Known use:
* Sun's Remote procedure call (RPC)
* OMG Corba

Consequence. * Exchangability of server is possible
* location and migration transparency exist
* Support fault Mechanism.

Page No.
Date

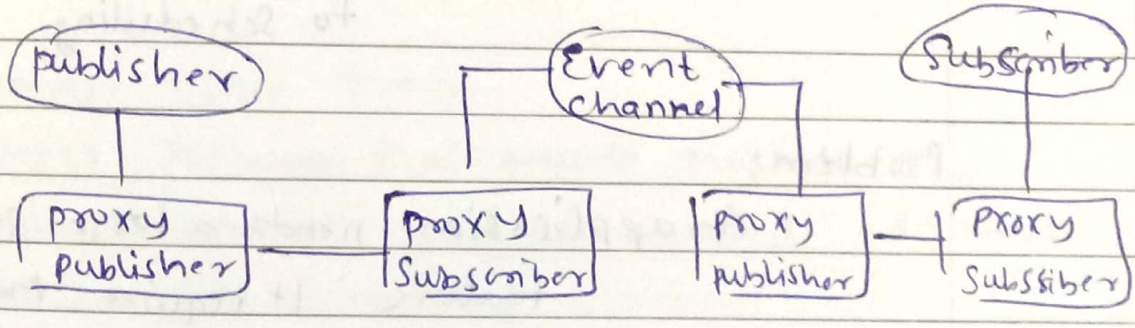
3. publisher - Subscriber

problem: data changes at one place and dependent component can affected.

Solution: a dedicated component who inform to all that data is change called publisher and dependent component is called subscribes.

* a object can be publisher or subscriber.

Structure:



Known use.

Online shopping

Consequences. Advantages: ① Extensibility

② Support Distributed Sys.

Disadvantage ① less efficient

② complex pattern to implement.

Management pattern.

* Seperate Manager component are cased to handle homogeneous collection of objects.

2 pattern.

1) Command processor.

- It manages request as separate object
- It schedule the execution of all the participating objects.

eg. undo in text editor.

Context: 1) App that need flexible and extensible user Interface

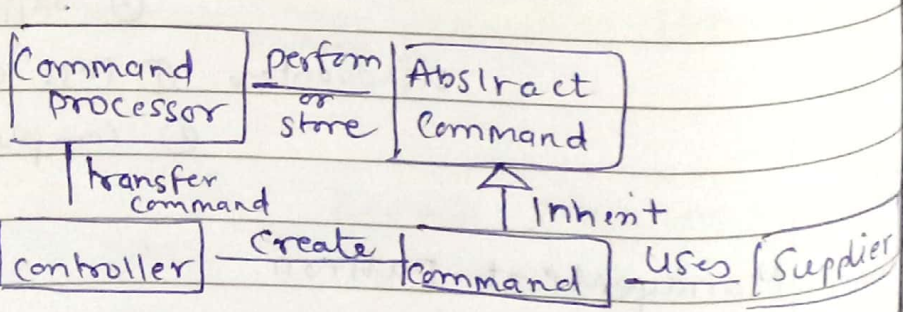
2) App that provide Service related to scheduling.

Problem:

An application needs a large set of feature. It require Management

Solution: * Encapsulate request into object
* use Command processor pattern.

Structure:



Known uses: MacApp use the pattern to do undo

Page No.
Date

consequence.

Advantage (1) flexibility
(2) Concurrency

Disadvantage : less efficient.

2) View handler.

This pattern manage all the view that a Software System provide.

Intent: It allow User to open, manipulate or dispose the view.

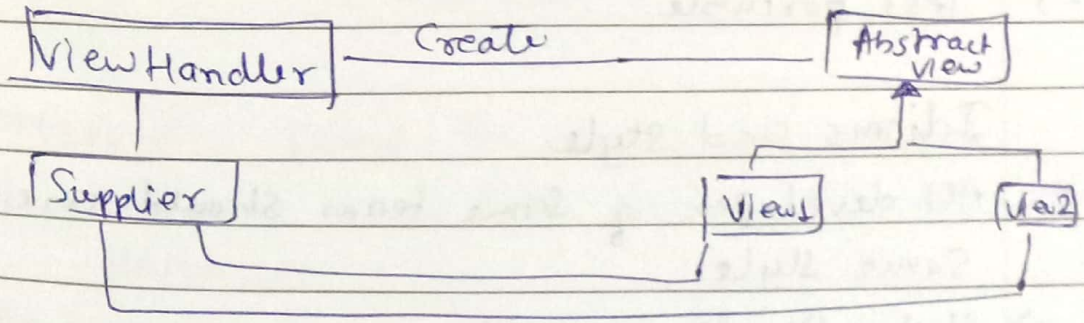
Example: Split screen.

Context: Software that provide multiple view.

problem: multiple view app required additional functionality to manage all view.

Solution: View handler pattern.

Structure.



Dynamic:

1. Scenario : View handler create a new view.
2. Scenario : Viewhandler organize the filling of views.

Page No.
Date

Known uses: Microsoft word.

Consequences: Adv. (1) It help in uniform handling the view.

(2) Extensibility and changeability

Disadv. (1) loss of information.

(2) It has restricated applicability.

Idioms.

→ low level pattern

→ It describe how to solve a specific problem using programming language

→ Speedup.

What can Idioms provide.

- help to solve recurring problem with programming lang

→ It help to perform Memory management, Object creation.

→ less portable.

Idioms and style

- All developer of same team should use same style.

→ style guide is used.

Where to find idioms?

→ Internet, Book, class library.

Page No.
Date

Counted pointer.

→ The counted pointer idiom is used for memory management of dynamically allocated shared objects in c++.

Problem: - passing object by value is inappropriate

* Several client want to acess same object
* object is deleted but client trying to acess
* If object is not required delete it.

Solution. 2 classes ① Handle ② Body

Client	hold	Handle	refer to	Body
handle b	by value	Body *body		int refcount
		operator → ()		Service()
		Handle (--)		Body(...)
		Handle (& Handle)		~Body()
		~Handle()		

refcount - count the no. of reference
If refcount is 0 then destroy object call destructor.