Page :
Date : / /

OOMD Unit 6

Q. What is pattern?

→ - Each pattern is a three part rule, which expresses a relatio between a certain context, a problem and a solution. Patterns help to build an collective experience.
- They capture existing, well proven experience in software development. Every pattern deals with specific, recurring problem in design or implementation of software system.
- Patterns help to build complex software architectures with defined properties.
- It provides a common vocabulary and understanding for design principles.

Q. What makes a pattern?

→ - Three-part schema of every pattern is-

1) Context - a situation giving rise to problem.
- It describes the situations in which problems occur:
- Context of problem may be fairly general. eg.' developing software with human-computer interface'.
- specifying correct context is difficult & it is practically impossible to determine all situations for which pattern can be applied

2) Problem-
- It is the recurring problem arising in the context.
- It begins with general problem specification.
- The general problem statement is completed by set of forces.

3) Solution-
- It is proven resolution of the problem.
- It shows how to solve the recurring problem or how to balance associated forces.

Page :
Date : / /

Q. Pattern categories-

→ - There are 3 categories of patterns-

1) Architectural patterns-

- They are used to describe viable software architectures that are built according to some overall structuring principle.
- It expression a fundamental structural organization schema for software systems.
- It provides a set of pre-defined subsystems, specifies their responsibilities, organizes relationships.

eg. Model-view-controller pattern.

2) Design Patterns-

- They are used to describe subsystems of software architecture as well as relationships betⁿ them.
- They are smaller in scale than architectural patterns, but are independent of particular programming language or paradigm.
- They describe commonly recurring structure of communicating components.

eg. Publisher-subscriber pattern.

3) Idiom-

- An Idiom is a low-level pattern specific to a programming language.
- It describes how to implement particular aspects of components or relationships betⁿ them using features of given language.
- They address aspect of both body design & implementation.

eg. Counted body pattern.

Q. Pattern description-

→ - It is essential to prepare and present design pattern in widely accepted format.

- So, design pattern should comprise of following entities-

- Pattern Name- It is a small name that should specify the purpose of design pattern.
- Classification- It means design pattern must be categorized under creational, structural or behavioral patterns.
- Intent- It is short explanation of design pattern.
- Also know as- It enlists aliases for pattern.
- Motivation- Detailed description of design problem.
- Applicability- It guides about areas where pattern can be applied.
- Structure- It depicts how pattern actually works with help of class diagrams and sequence diagrams.

Other descriptions are participants, Collaborations, Consequences, Implementation, sample code, Related Patterns, etc.

Q. Forwarder Receiver Problem-

→ • Problem-

- Many components in distributed system communicate in a peer to peer fashion.
- The communication bet^n peers should not depend on particular IPC mechanism.
- Performance is always an issue.

• Solution-

- Encapsulate the inter process mechanism.
- Peer implements application services.
- Forwarders send request/messages to forwarders.
- Receivers are responsible for receiving IPC requests sent by remote peers using specific IPC mechanism.

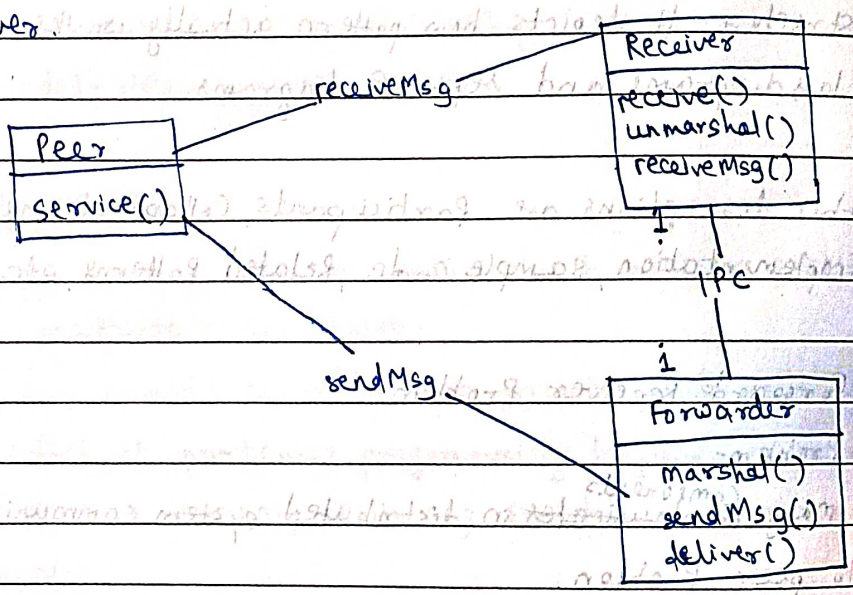
• Intent-

- The Forwarder-Receiver design pattern provides transparent interprocess communication for software systems with a

peer-to-peer interaction model.

• Structure-

- F-R consists 3 components - forwarders, receivers and peers.
- Peer components are responsible for application tasks.
- It uses a forwarder to send messages to other peers. To send message to remote peer, it invokes method sendmsg of it's forwarder
- To receive message, it invokes receive msg method of it's receiver.



• Benefits and liability -

- Efficient inter process communication.
- Encapsulation of IPC facilities.

Q. Client-Dispatcher-Server

→ • Goals-

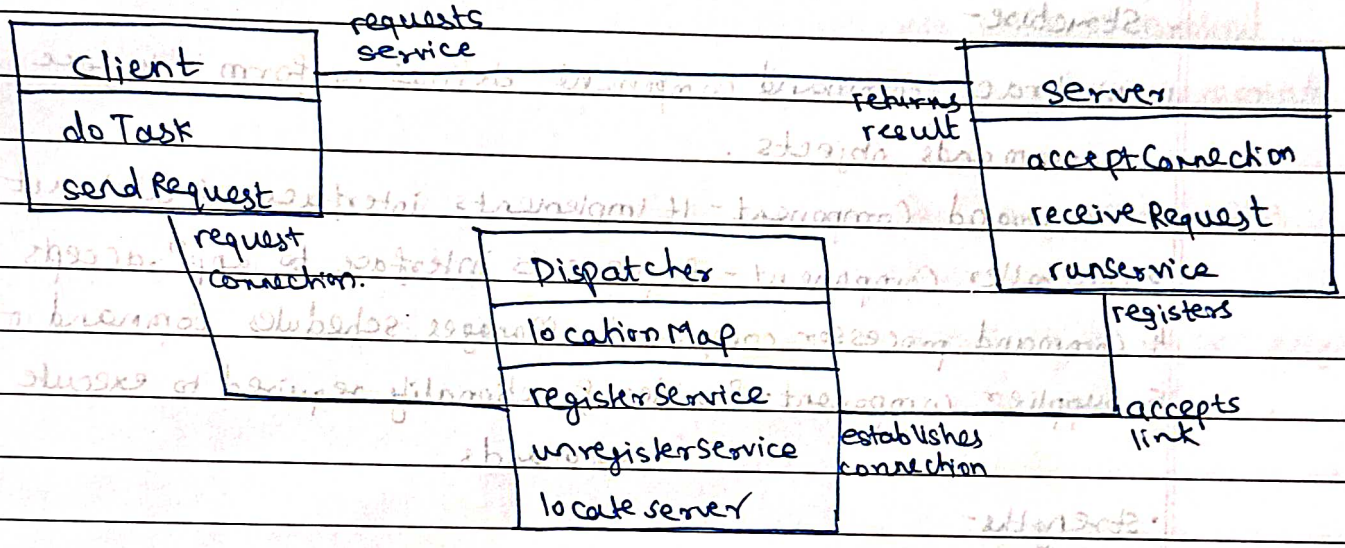
- Introduce intermediate layer betn clients & server.
- Provide location transparency.
- Hide details of establishment of communication.

• Components-

- Client - Perform some domain specific tasks.
  - Access operations offered by servers.

- Server - Provide services to clients.
  - Register itself with dispatcher

- Dispatcher - Establish communication channels.
  - Locate, register servers.
  - Maintain a map of server locations & names.



• Applicability -
- A software system integrating a set of distributed servers, with servers running locally or distributed over a network.

Q. Command processor pattern-

→ - Command processor design pattern separates the request for a service from it's execution

• Context -
Applications that need flexible and extensible user interfaces that provides services related to execution of user functions, such as scheduling or undo.

• Problem -
- Application needs a large set of features.

- Need a soln that is well structured for mapping it's int to internal functionality.

• Solution -
- use command processor pattern.
- Encapsulate request into objects.

• Structure -
1. Abstract command component - defines uniform interface for commands objects.
2. Command Component - It implements interface of abstract com
3. Controller Component - Represents interface to app^n, accepts servi
4. Command processor component - Manages, schedules command objects
5. Supplier component - Provides functionality required to execute conc commands.

• Strengths -
- Flexibility in the way requests are activated.
- Concurrency.
- Testability at application level.

• Weaknesses -
- Efficiency loss.
- Complexity in acquiring command parameters.

Q. Idioms - (Intro)
→ Written in Q. Pattern Categories.

Q. What can Idioms provide?
→ - A single Idiom helps to solve a recuring problem with programming language we normally use.
- It provides a vehicle for communication among developers as each idiom has a unique name.

- Idioms are less 'portable' between programming languages.

Idioms and style-

- If programmers use different styles, they should agree on a single coding style for their team's program.
- A program that uses mixture of styles might be harder to understand and maintain.
- Idioms from conflicting styles do not mix well if applied carelessly to program. Different set of idioms may be appropriate for different domains.
- A single style guide can be unsuitable for large companies that employ many teams in different domains.
- A coherent set of idioms leads to consistent style in our progra