Patterns in Software Engineering

Lecturer: Raman Ramsin

Lecture 9

GoV Patterns: Design
GoV Design Patterns: Categories

- **Structural Decomposition**
  - Support a suitable decomposition of subsystems and complex components into cooperating parts.
  - *Whole-Part*

- **Organization of Work**
  - Define how components collaborate together to solve a complex problem.
  - *Master-Slave*

- **Access Control**
  - Guard and control access to services or components.
  - *Proxy*

- **Management**
  - Handle homogenous collections of objects, services and components in their entirety.
  - *Command Processor* and *View Handler*

- **Communication**
  - Help to organize communication between components.
  - *Forwarder-Receiver*, *Client-Dispatcher-Server*, and *Publisher-Subscriber*
Design: Structural Decomposition

- **Whole-Part**: Helps with the aggregation of components that together form a semantic unit.
  - An aggregate component, the *whole*:
    - encapsulates its constituent components: the *parts*;
    - organizes the parts’ collaboration; and
    - provides a common interface to its functionality.
  - Direct access to the parts is not possible.
  - Analogous to the *State-over-a-Collection* and *Behavior-over-a-Collection* patterns

- The **Composite** pattern also belongs to this category.
Design: Organization of Work

- **Master-Slave:** supports fault tolerance, parallel computation and computational accuracy. A *master* component:
  - distributes work to identical *slave* components, and
  - computes a final result from the results these slaves return.

- The **Chain of Responsibility, Command** and **Mediator** patterns also belong to this category.
Organization of Work: Master-Slave – Structure

- A *master* component
  - distributes work to identical *slave* components;
  - computes a final result from the results these slaves return.

- The master component provides a service that can be solved by applying the 'divide and conquer' principle.
Organization of Work: Master-Slave – Dynamics
Design: Access Control

- **Proxy**: makes the clients of a component communicate with a representative rather than to the component itself.

- The **Facade** and **Iterator** patterns also belong to this category.
Design: Management

- **Command Processor**: separates the request for a service from its execution. A command processor component:
  - manages requests as separate objects,
  - schedules the execution of the requests, and
  - provides additional services such as the storing of request objects for later undo.

- **View Handler**: helps to manage views in a software system. A view handler component:
  - allows clients to open, manipulate and dispose of views,
  - coordinates dependencies between views, and
  - organizes the update of the views.

- The **Memento** pattern also belongs to this category.
Management: Command Processor – Structure

- Manages requests as separate objects, schedules their execution, and provides relevant services.
- Builds on the **Command** design pattern.
  - Whenever a user calls a specific function of the application, the request is turned into a Command object.
  - A Command object delegates the execution of its task to supplier components.
  - The Command Processor takes care of all command objects.

![Command Processor Diagram]

Department of Computer Engineering

Sharif University of Technology
Management: Command Processor – Dynamics
Management: View Handler

- A *view handler* component
  - manages all views that the software system provides,
  - allows clients to open, manipulate and dispose of views, and
  - coordinates dependencies between views and organizes their update.

- Specific views, together with functionality for their presentation and control, are encapsulated within separate *view* components - one for each kind of view.

- *Suppliers* provide views with the data they must present.
Management: View Handler – Structure

```
ViewHandler

myViews

update
open
close
top
split
clean
tile

creates, closes, and coordinates

AbstractView

display
update
initialize
open
close
move
size

Supplier

attach
detach
getData
setData

notifies

SpecificViewA

display
initialize
update
open
close
move
size

+1

SpecificViewB

display
initialize
update
open
close
move
size

+1

notifies

retrieves data

retrieves data
```
Scenario I: The view handler creates a new view.
Scenario II: The view handler organizes the tiling of views.
Design: Communication

- **Forwarder-Receiver**: provides transparent inter-process communication for software systems with a peer-to-peer interaction model.
  - Introduces *forwarders* and *receivers* to decouple peers from the underlying communication mechanisms.

- **Client-Dispatcher-Server**: introduces an intermediate layer between *clients* and *servers*: the *dispatcher* component.
  - Provides location transparency by means of a name service.
  - Hides the details of the establishment of the communication connection between clients and servers.

- **Publisher-Subscriber**: The same as the *Observer* pattern.
Communication: Forwarder-Receiver

- **Peer** components:
  - are responsible for application tasks,
  - know the names of the remote peers with which they need to communicate, and
  - use a *forwarder* to send messages to other peers and a *receiver* to receive messages from other peers.

- **Forwarder** components:
  - send messages across process boundaries,
  - provide a general interface for sending messages,
  - marshal and deliver messages to remote receivers, and
  - map names to physical addresses.

- **Receiver** components:
  - are responsible for receiving messages,
  - provide a general interface for receiving messages, and
  - receive and unmarshal messages from remote forwarders.
Communication: Forwarder-Receiver – Structure
Communication: Forwarder-Receiver – Dynamics
Communication: Client-Dispatcher-Server

- Provides a *dispatcher* component to act as an intermediate layer between *clients* and *servers*. The dispatcher
  - implements a name service that allows clients to refer to servers by names instead of physical locations, thus providing location transparency, and
  - is responsible for establishing the communication channel between a client and a server.

- Before sending a request to a server, the client asks the dispatcher for a communication channel.
  - The client uses this channel to communicate with the server.

- A server provides a set of operations to clients.
  - It either registers itself or is registered with the dispatcher by its name and address.
Communication: Client-Dispatcher-Server – Structure

**Client**
- `doTask`
- `sendRequest`

**Dispatcher**
- `locationMap`
- `registerService`
- `unregisterServer`
- `locateServer`
- `establishChannel`
- `getChannel`

**Server**
- `acceptConnection`
- `runService`
- `receiveRequest`

**Relationships**
- The client requests a service from the dispatcher.
- The dispatcher requests a connection from the server.
- The server returns a result to the client.
- The server accepts links from the client.

Department of Computer Engineering
Sharif University of Technology
Communication: Client-Dispatcher-Server – Dynamics
Reference