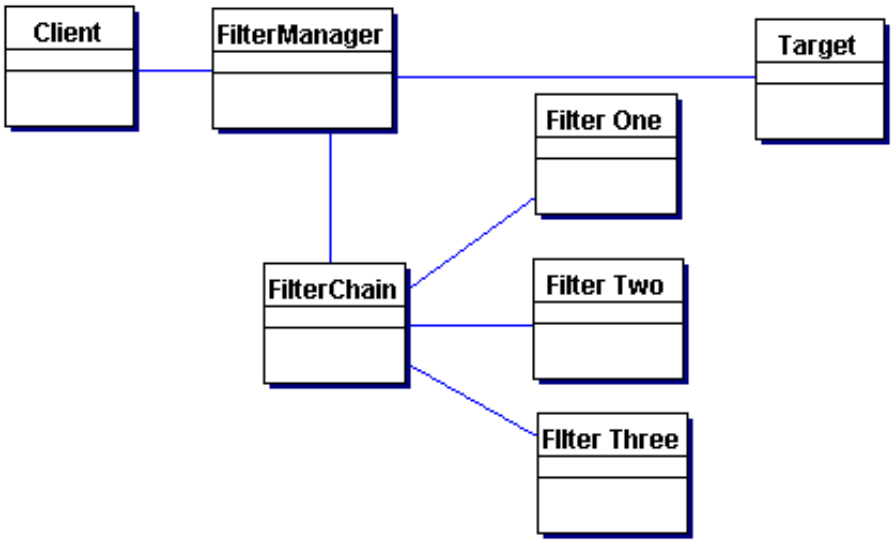
Core JEE patterns - Presentation tier design patterns

Intercepting filter



Purpose

- To do pre-processing on requests or post-processing on responses.

Notes

- Implementation available with JEE.

More info

http://www.oracle.com/technetwork/java/interceptingfilter-142169.html

Front controller

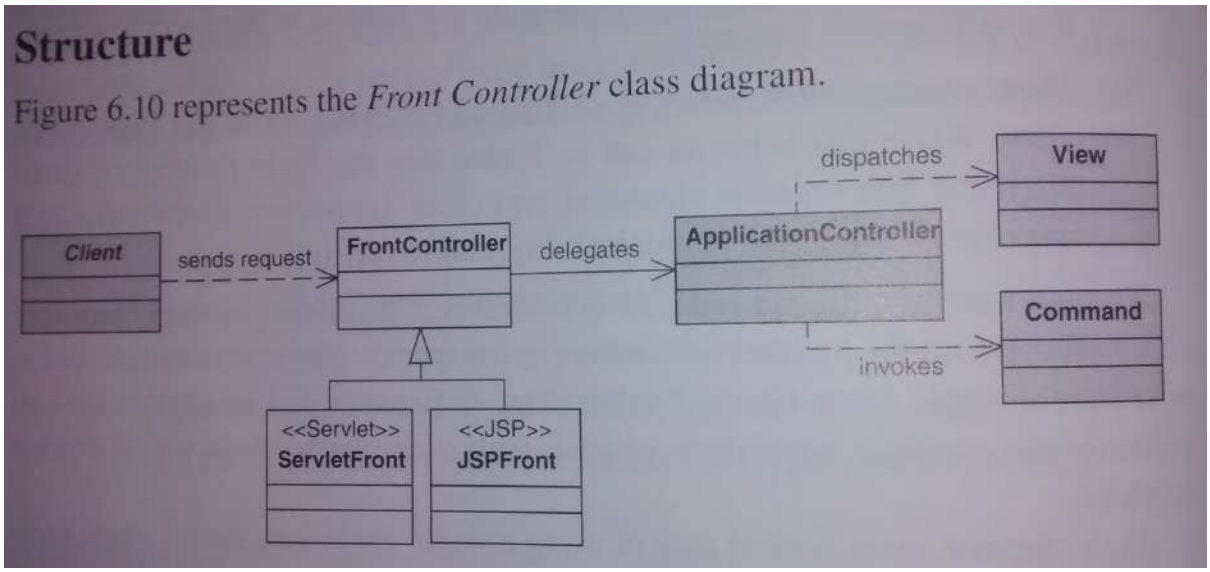
Purpose

Have better control over request routing. Front controller could technically be a servlet or a jsp. Generally it's made as servlet.

Without the Frontcontroller, resources (jsp) has to be directly called using urls (physical resource mapping). With front controller logical resource mapping is possible.

Variant 1

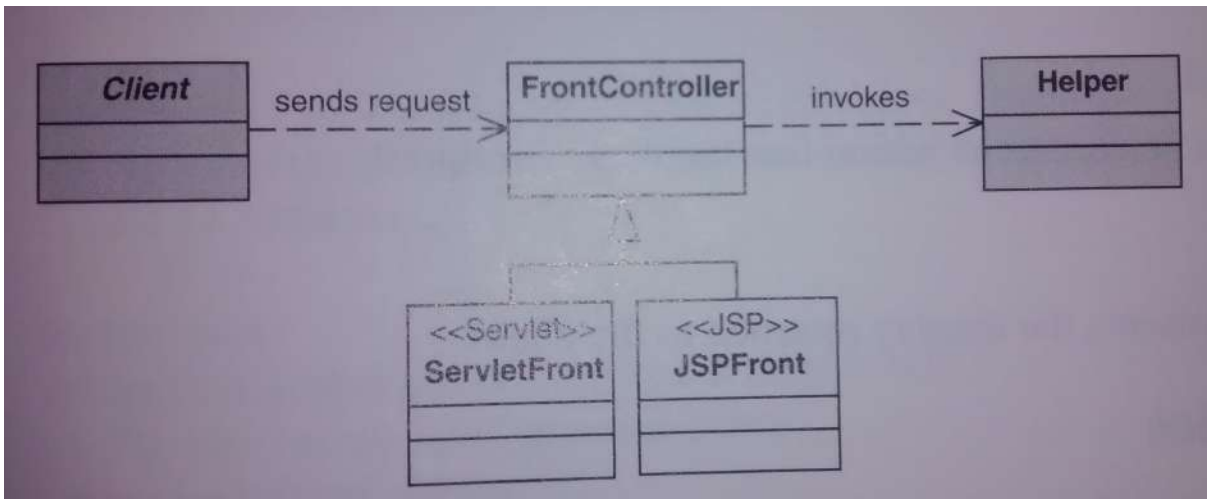
Often used with application controller. Application controller dispatches to view and invokes command (for business processing). In this case, command object executes the request. Depending on the resuly of command object, appropriate view is chosen.



Struts uses a variant of this. MVC2 (Pull based MVC). Spring MVC also follow similar pattern.

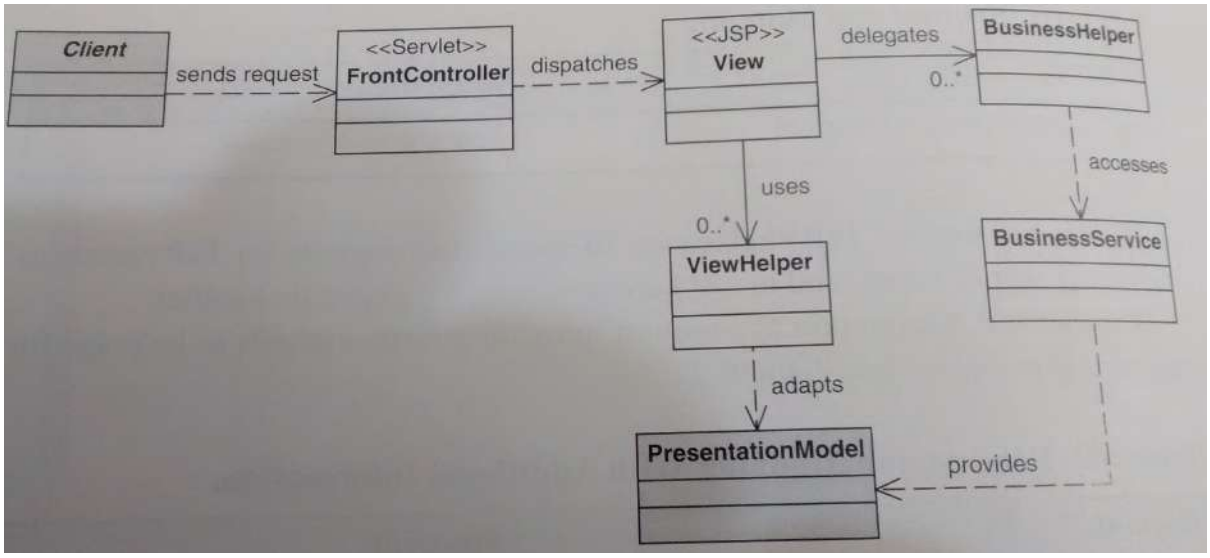
Variant 2

Instead of application controller, it could make use of a helper as well.



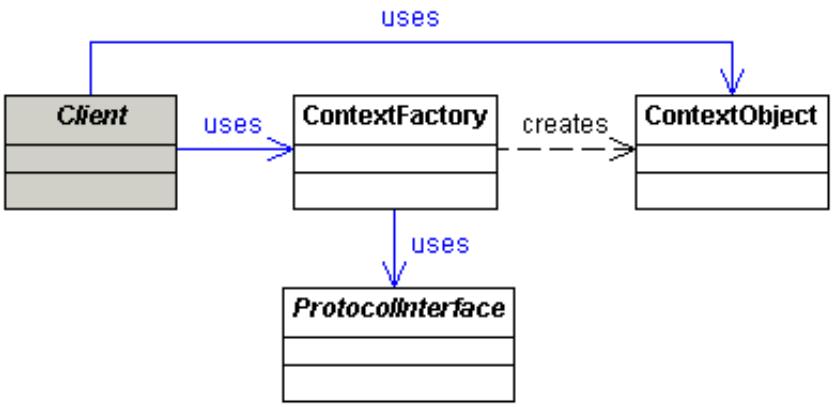
Variant 3

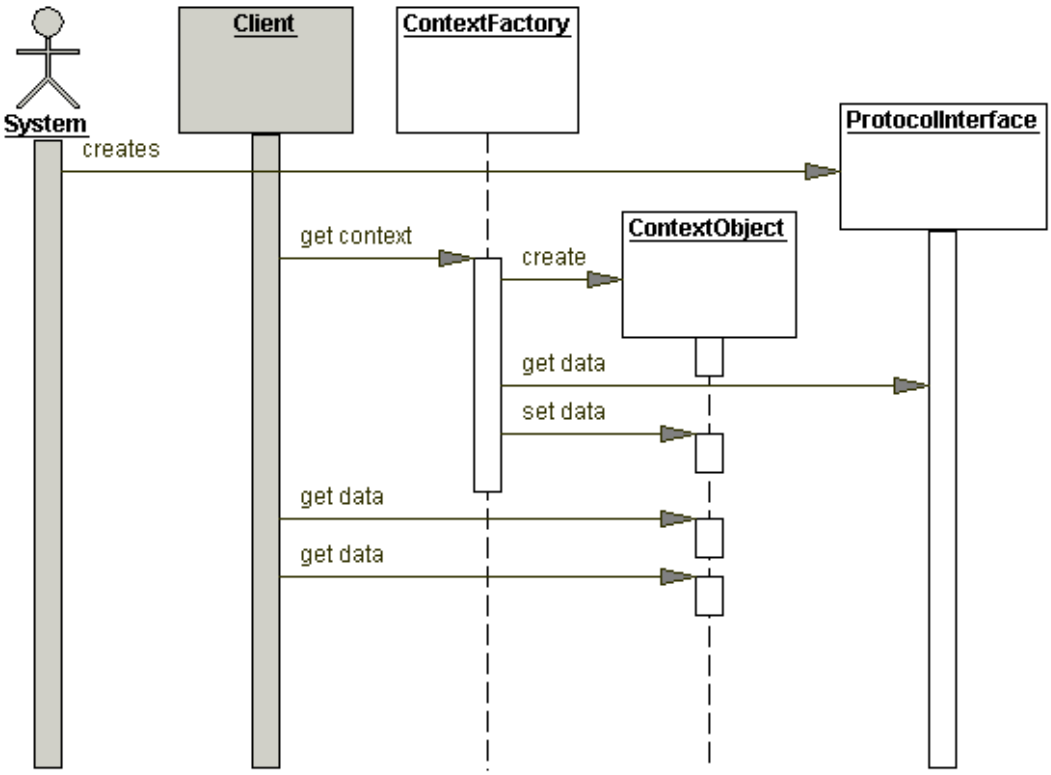
Suitable for apps with few urls. View itself manages everything. Urls are mapped to physical resources directly.



Context object

Context pattern addresses a mechanism to encapsulate environment / protocol specific information in an independent manner.





Typical examples - Http request context, configuration context, security context, etc.

Context object is often using along with a content factory which supplies the context information based on the need.

Different strategies:

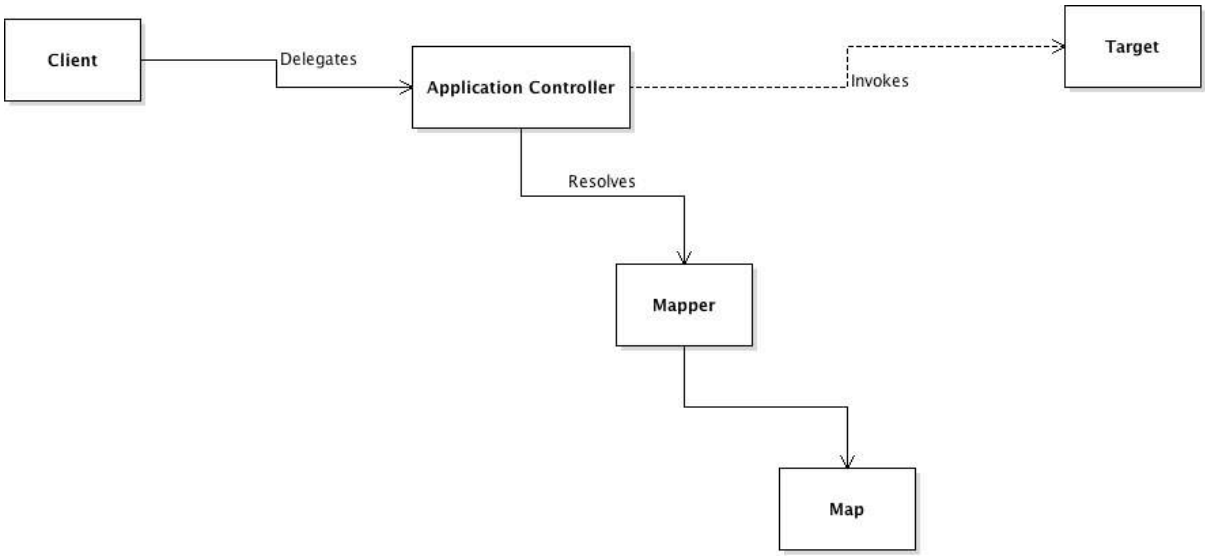
- Request strategy: Encapsulating HTTP request in a context object
- Configuration strategy: Encapsulating config information in a context object.
- Security strategy: Encapsulating security information in a context object.

Application controller

Purpose: Action management for views. It works with view management such as front controller.

Example: Struts action management.

Implementation detail: It could even work with Command object.



View Helper

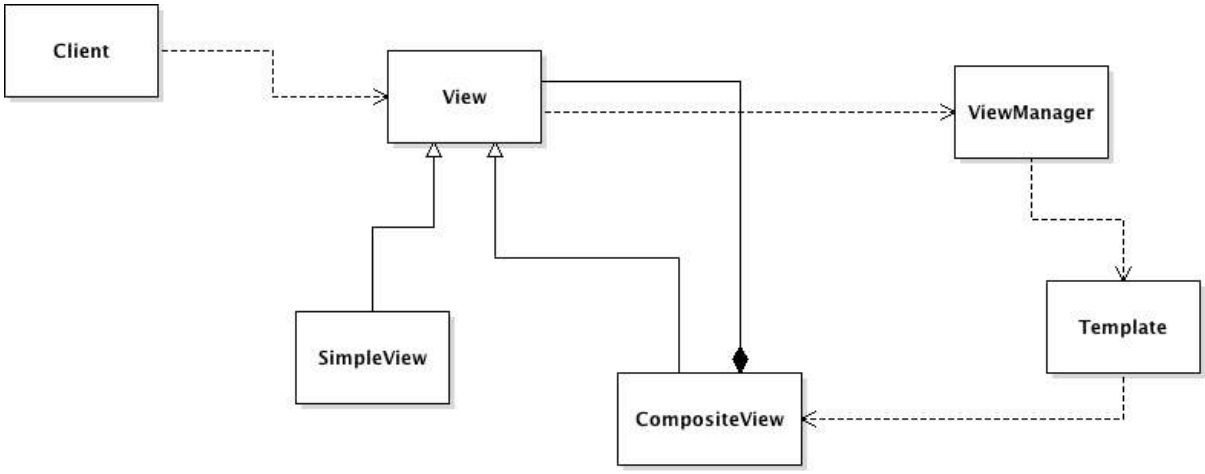
Purpose: Separate processing logic from the view markup.

POJO (Java bean) could be used as view helper. Custom tags can be used as view helper - Implementation as custom tag or tag file.

Composite View

Purpose: Have sub-views of a page independent of the layout.

Similar to portlets.

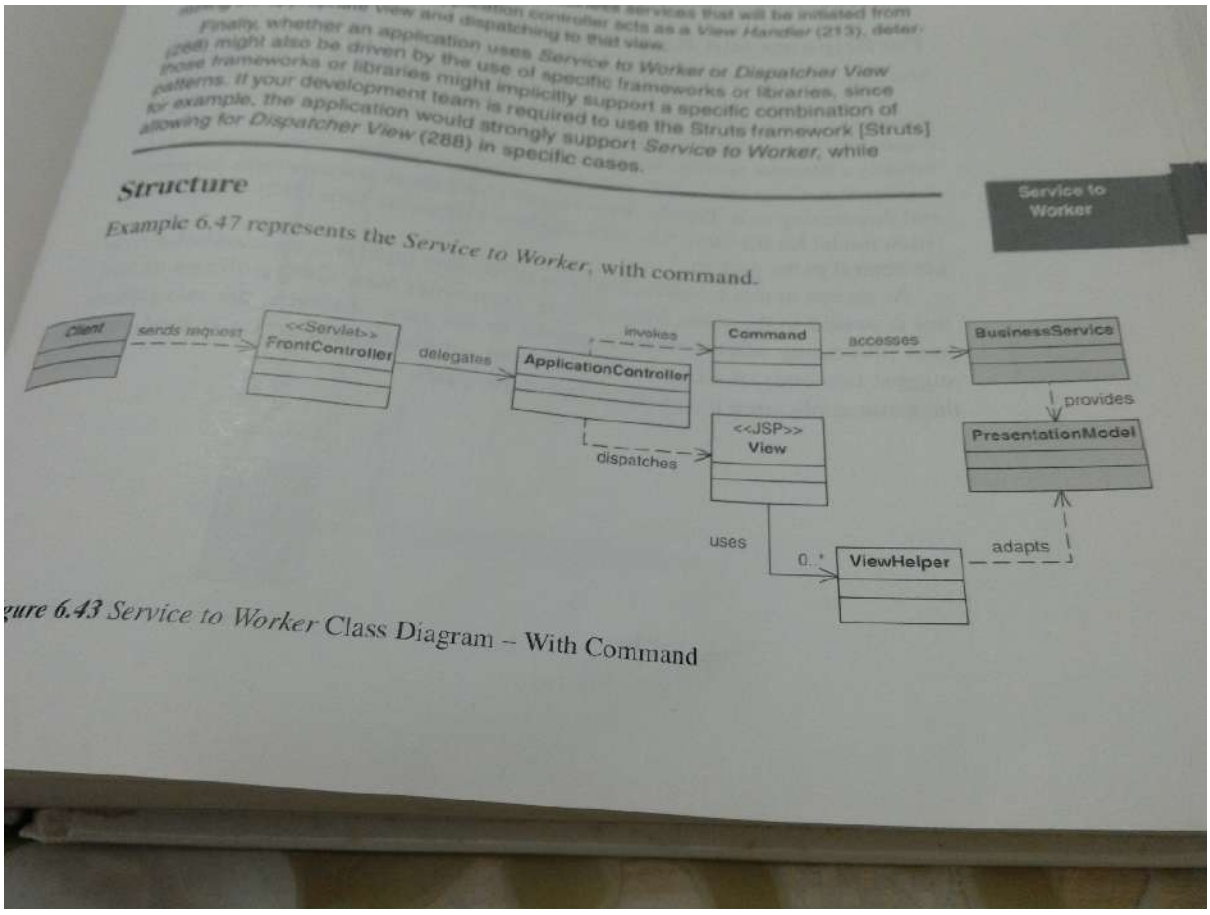


Service to woker

This pattern puts together most of the other presentation logic patterns for controller based strategy.

Purpose

- View requires data from business services. Data will be made available in the form of presentation model.
- Controller based strategy is in place for presentation layer.



Structure

Example 6.47 represents the Service to Worker, with command.

ure 6.43 Service to Worker Class Diagram – With Command

Dispatcher view

This pattern put together other presentation tear patterns for view based strategy (request handling primarily done through views).

Purpose

- Leveraging business logic to prepare presentation model.

- Works for view based strategy.

troller typically does nothing more than forward to a view. Such cases are examples of a *Dispatcher in Controller* strategy (178). A client might submit a request that includes a query parameter describing an action to be performed. For example:

```
http://some.server.com/Controller?action=showaccount.jsp
```

The sole responsibility of the controller in this case is to dispatch to the view showaccount.jsp.

An *Application Controller* (205) can also be used in a very limited form as part of *Dispatcher View*, performing basic view management. If a client submits a request that includes a logical reference to an action, then the *Application Controller* (205) resolves that logical name to a concrete view. For example, consider the following request:

```
http://some.server.com/Controller?action=showaccount
```

The *Application Controller* (205) acts as a *View Handler* (213), resolving the logical name showaccount to an actual view name, which might be showaccount.jsp. The *Application Controller* (205) then dispatches to that view.

Structure

Figure 6.46 shows the class diagram that represents the *Dispatcher View*.

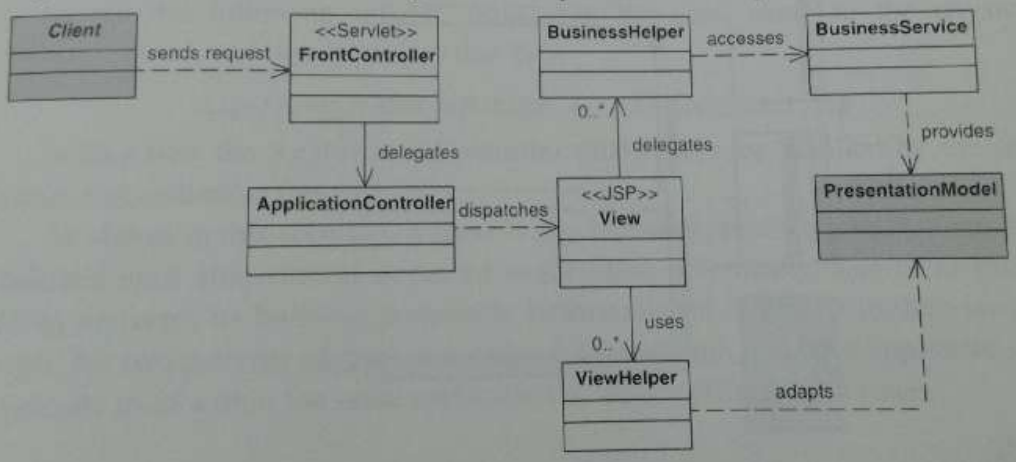
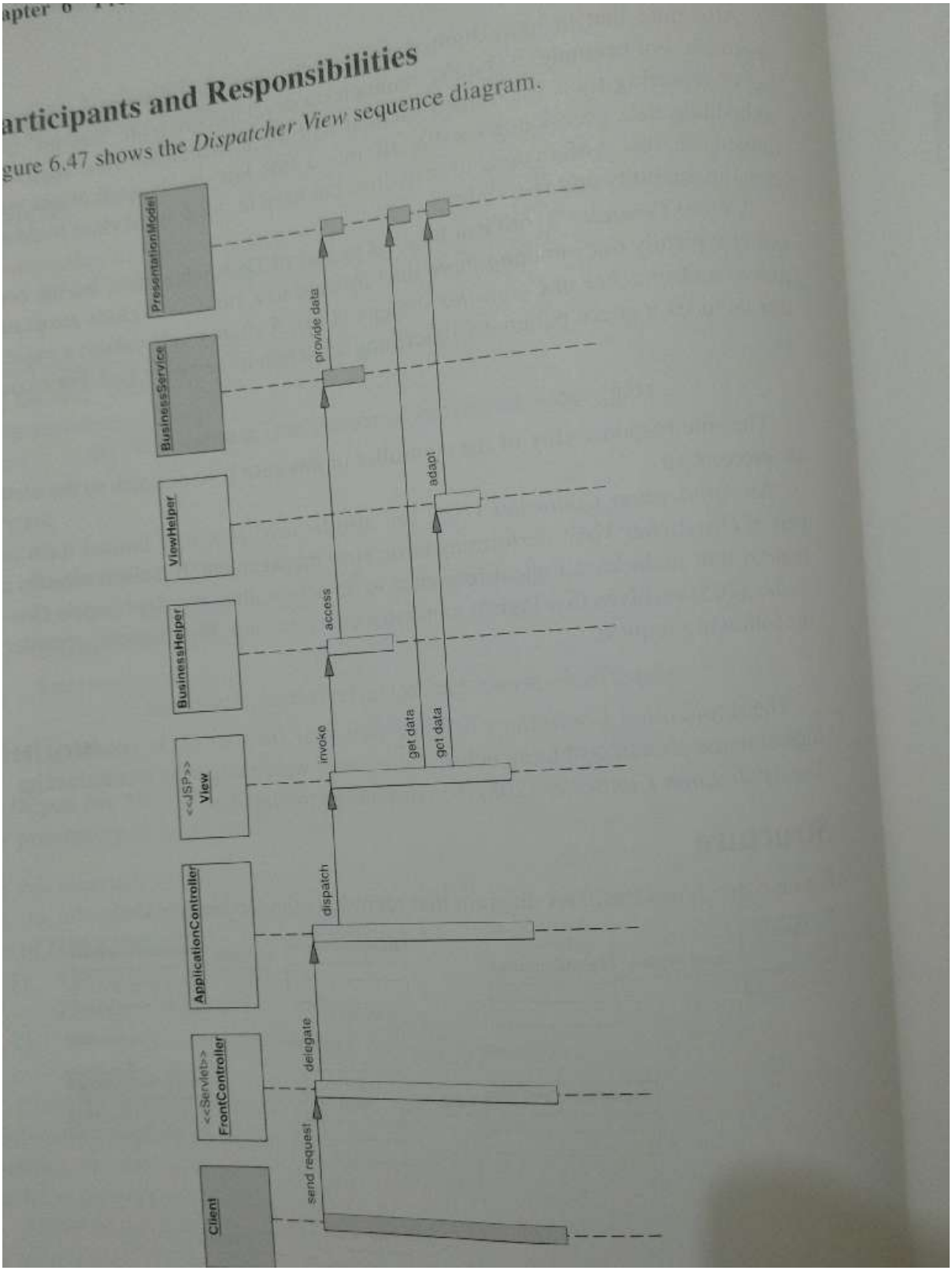


Figure 6.46 *Dispatcher View* Class Diagram

articipants and Responsibilities

gure 6.47 shows the *Dispatcher View* sequence diagram.



Presentation layer considerations

Session management

Session state on the client

a. Store sessions on hidden fields.
    i. Advantages
        1. No worry about server sessions, hence scale out is easy.
    ii. Disadvantages
        1. Network round trip.
        2. Security - expose data. Needs encryption.
        3. Session has to be "stringified" (type limitation)
b. Store in cookies
    i. Advantages
        1. No server session, scale out easy.
    ii. Disadvantages
        1. Network round trip
        2. Security - expose data. Needs encryption.
        3. Size limitation with cookies.
        4. Data has to be "Stringified" (type limitation).
c. Store in URLs.
    i. Advantages
        1. No server session.
    ii. Disadvantages
        1. Lengthy URLs.
        2. Security issue.
        3. Type limitation.

Session state on the server

a. Store sessions on hidden fields.
    i. Advantages
        1. No size limitation.
        2. No type limitation.
        3. No security issues.
        4. No network round trip.
    ii. Potential issue
        1. Session sharing among machines in a cluster.

a. Solution - "Sticky" load balancers.
b. Store session in business tier as EJB bean.
c. Store session in resource tier (rdbms)
d. Most app servers support session replication using cache.

Form Validations

- Must validate at client side & server side. Client side validation is not trustable.
- Use validator frameworks.
    - Struts & Spring has its own validation framework.
        - Or, use external frameworks like Apache commons Validator.
    -

Preventing duplicate form submission

- Create a synchronizer token, put it in form & in session.
- On form submission, check the validity of the form by comparing one from session & in the form.

Controlling client access

1. Protect server resources through guarded configuration.
    a. Define role based security constraints in web.xml
        i. Form based authentication
            1. Define a form to capture user id/password.
        ii. Basic authentication
            1. Browser will show a dialog to capture the user id / password.
    b. Put jsps in /WEB-INF
        i. No client can access it.
        ii. Only servlets can redirect to it.
2. Securing a section of view.

Helpers - Always initialize the state variables.

Presentation layer - Bad practice

Control code in multiple views

Solution:

- Use servlets as controllers.
- Use java beans for accessing/storing data
- Use custom tag helpers for formatting & displaying data
- Let the controller servlet create beans & forward it to the view (jsp).

Sharing presentation tier data structures such as HttpServletRequest to domain objects or business tier

Solution:

- Extract the required parameters and send only the parameters.

Exposing sensitive resources such as properties file to direct client access

Solution:

- Put them in /WEB-INF directory
- Put them in an access controlled directory.

Java backing beans as view helpers - Assuming that it is initialized using <jsp:set/>

Solution:

- Initialize the variables of the beans in bean itself.

Fatty controllers

Solution:

- Use command helper java bean to delegate certain processing.f

http://www.computepatterns.com