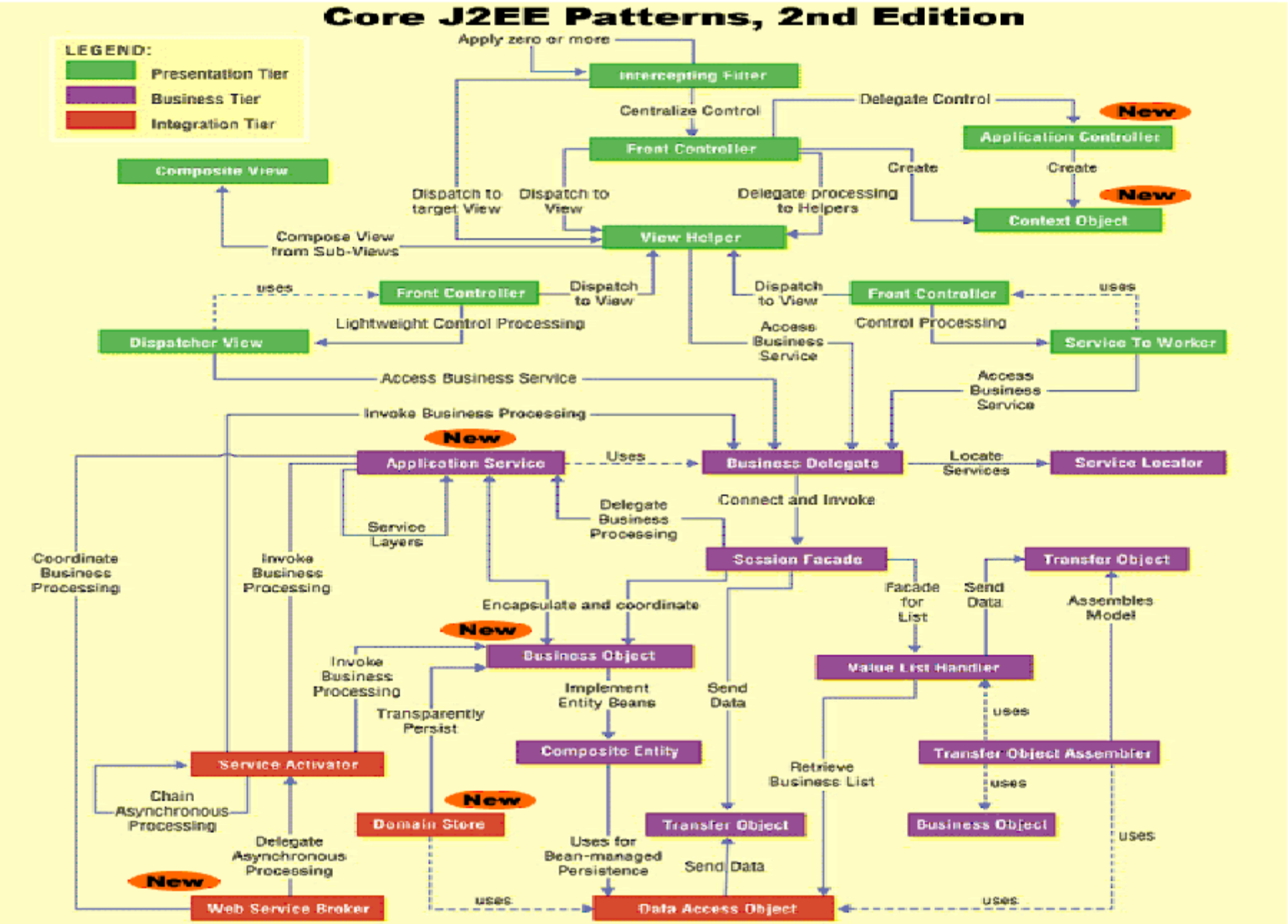
J2EE Presentation Patterns



J2EE Presentation Patterns

1 Intercepting Filter

Problem :

You want to intercept and manipulate a request and a response before and after the request is processed

Forces :

• You want centralized, common processing across requests, such as checking the data-encoding scheme of each request, logging information about each request, or compressing an outgoing response.

• You want pre and postprocessing components loosely coupled with core request-handling services to facilitate unobtrusive addition and removal.

• You want pre and postprocessing components independent of each other and self contained to facilitate reuse.

1 Intercepting Filter

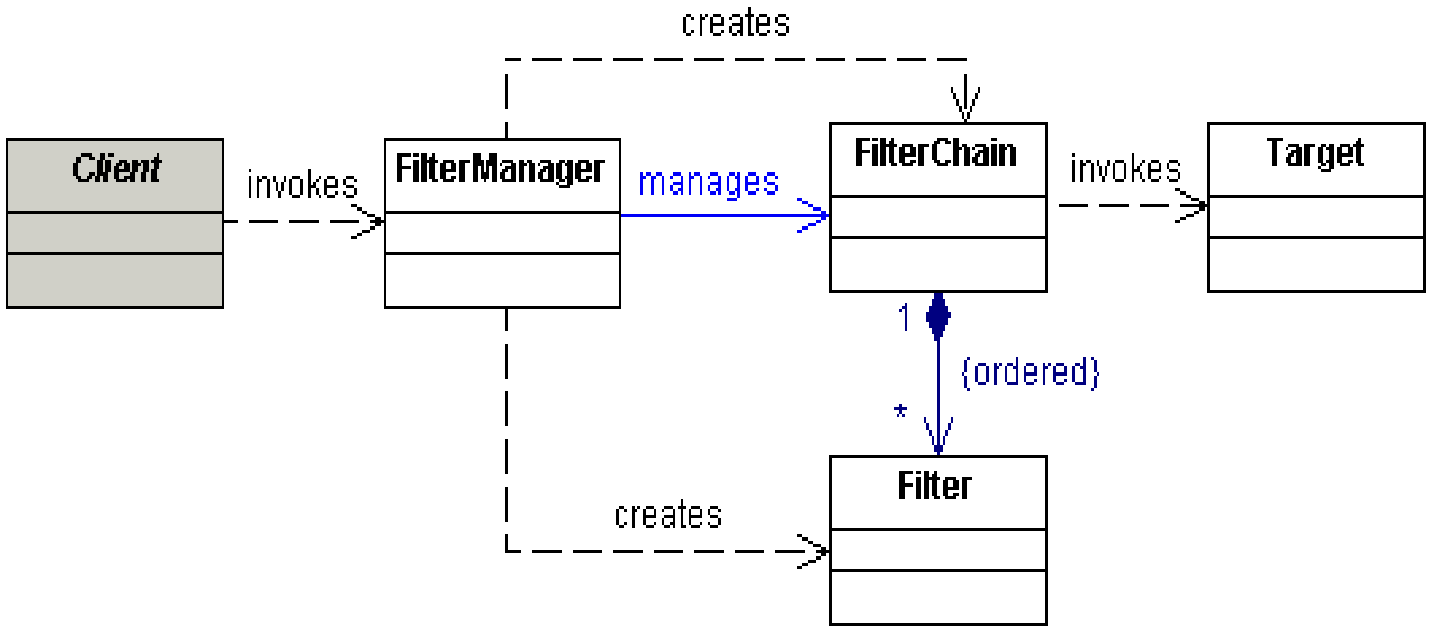
Solution :

Use an Intercepting Filter as a pluggable filter to pre and postprocess requests and responses.

A filter manager combines loosely coupled filters in a chain, delegating control to the appropriate filter. In this way, you can add, remove, and combine these filters in various ways without changing existing code.

1 Intercepting Filter

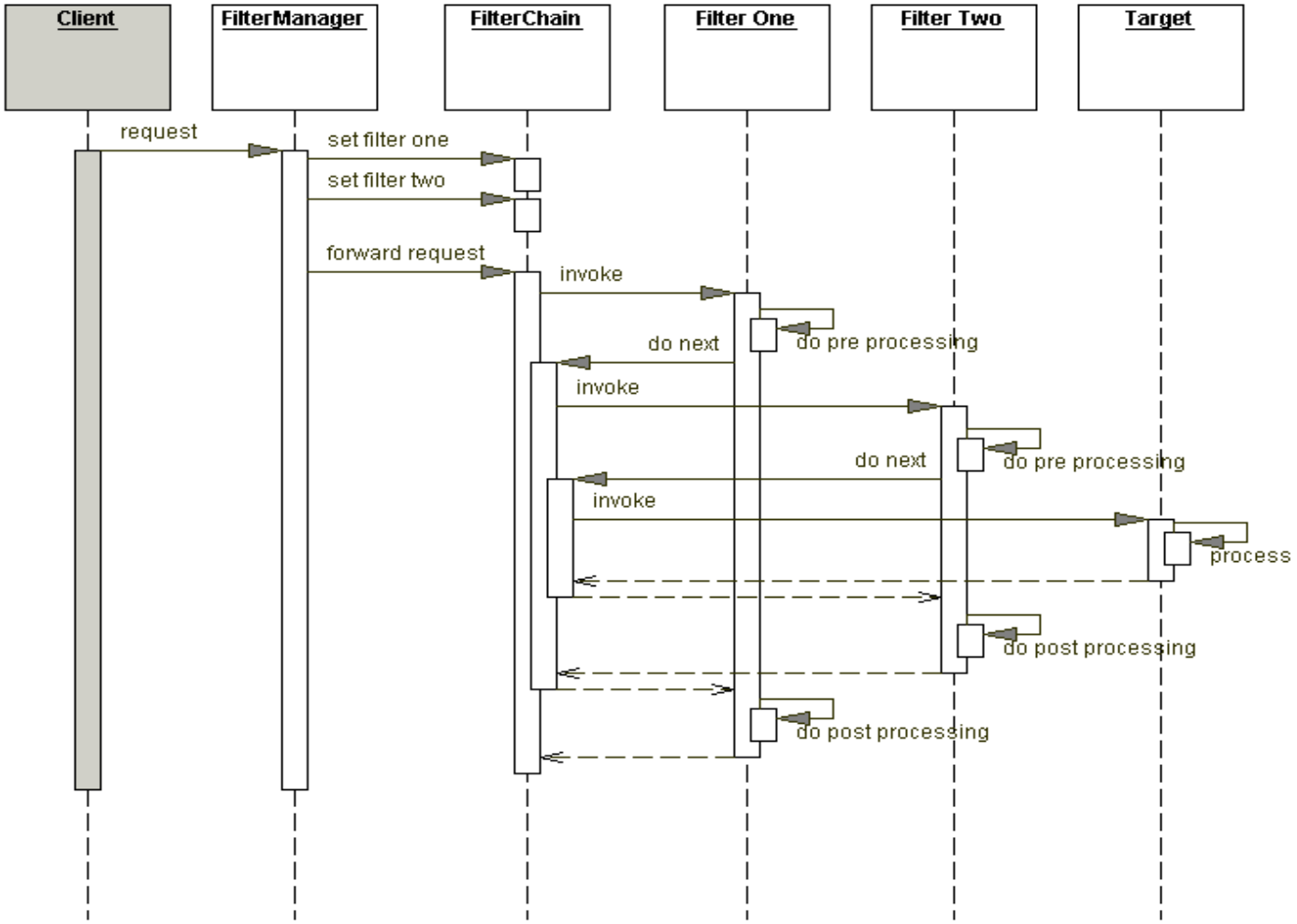
Class Diagram :



J2EE Presentation Patterns

1 Intercepting Filter

Sequence Diagram :



1 Intercepting Filter

Strategies

- Standard Filter Strategy
- Custom Filter Strategy
- Base Filter Strategy
- Template Filter Strategy
- Web Service Message Handling Strategies
  - Custom SOAP Filter Strategy
  - JAX RPC Filter Strategy

Consequences

- Centralizes control with loosely coupled handlers
- Improves reusability
- Declarative and flexible configuration
- Information sharing is inefficient

2 Context Object

Problem :

You want to avoid using protocol-specific system information outside of its relevant context.

Forces :

You have components and services that need access to system information.

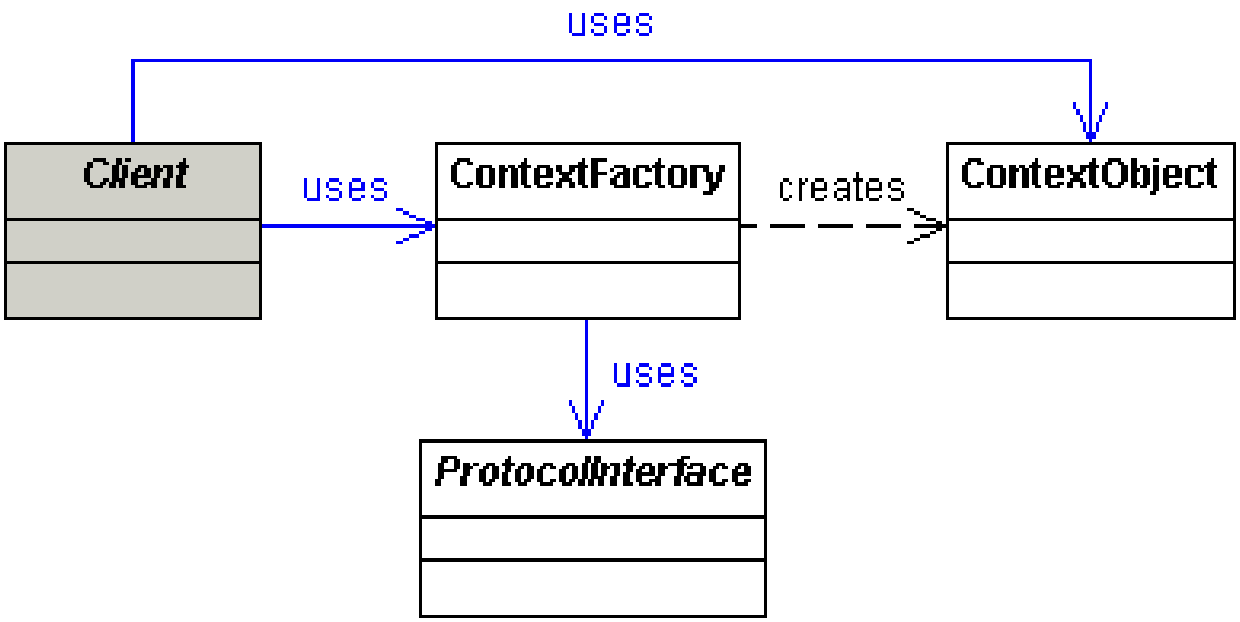
- You want to decouple application components and services from the protocol specifics of system information.
- You want to expose only the relevant APIs within a context.

Solution :

Use a Context Object to encapsulate state in a protocol-independent way to be shared throughout your application.

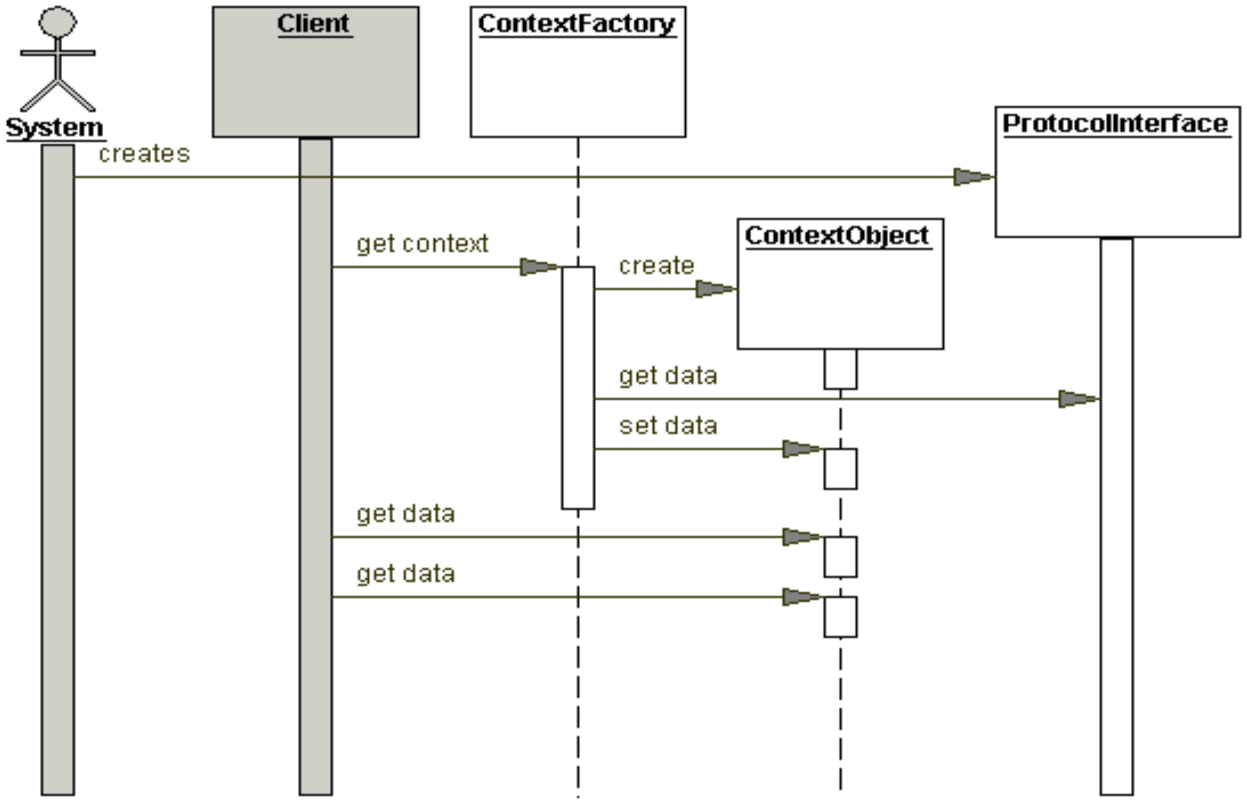
2 Context Object

Class Diagram :



2 Context Object

Sequence Diagram :



2 Context Object

Strategies

Request Context Strategies

- Request Context Map Strategy
- Request Context POJO Strategy
- Request Context Validation Strategy

Configuration Context Strategies

- JSTL Configuration Strategy
- Security Context Strategy

General Context Object Strategies

- Context Object Factory Strategy
- Context Object Auto-Population Strategy

Consequences

Improves reusability and maintainability

Improves testability

Reduces constraints on evolution of interfaces

Reduces performance

3. Front Controller

Problem

You want a centralized access point for presentation-tier request handling.

Forces

You want to avoid duplicate control logic.

You want to apply common logic to multiple requests.

You want to separate system processing logic from the view.

You want to centralize controlled access points into your system.

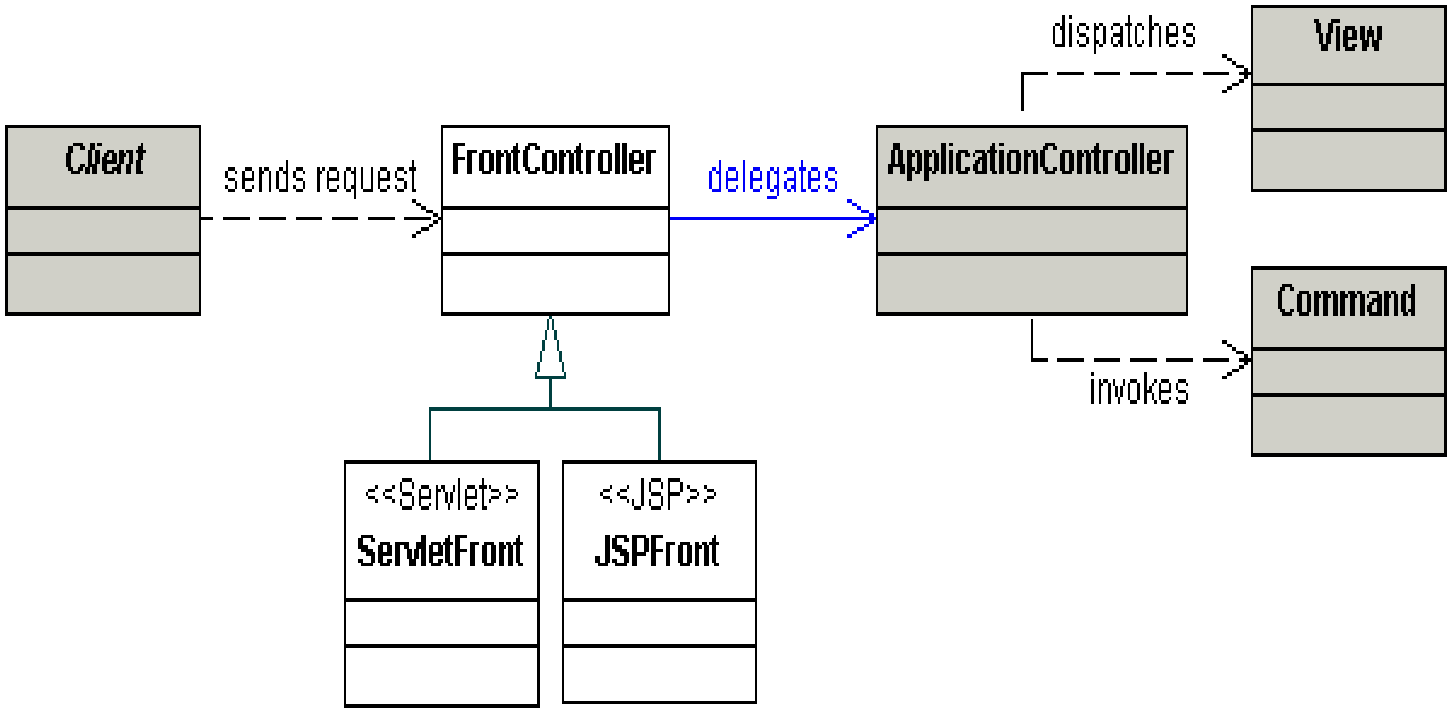
Solution

Use a Front Controller as the initial point of contact for handling all related requests.

The Front Controller centralizes control logic that might otherwise be duplicated, and manages the key request handling activities.

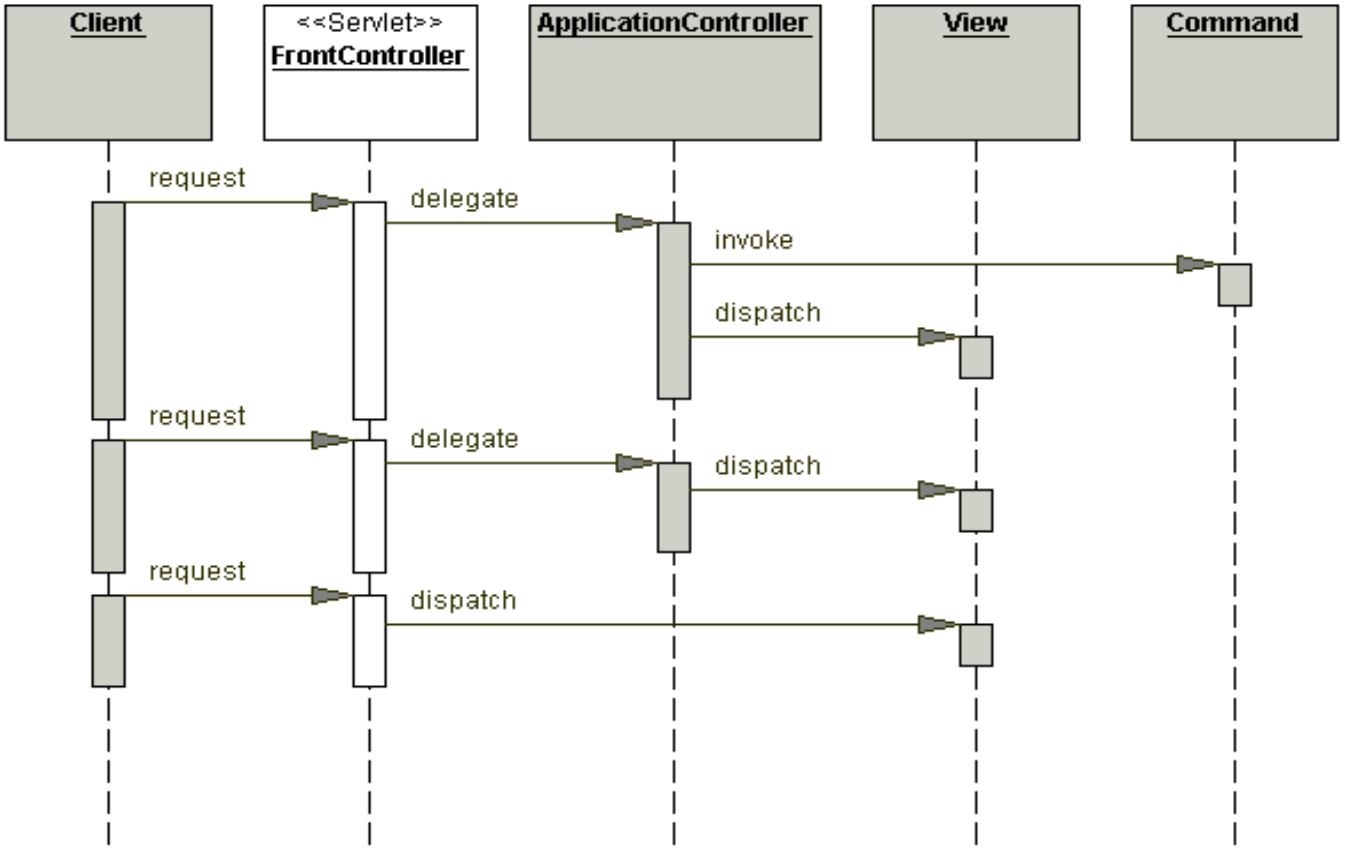
3. Front Controller

Class Diagram :



3. Front Controller

Sequence Diagram :



3. Front Controller

Strategies

Standard Filter Strategy

Custom Filter Strategy

Base Filter Strategy

Template Filter Strategy

Web Service Message Handling Strategies

- Custom SOAP Filter Strategy
- JAX RPC Filter Strategy

Consequences

Centralizes control with loosely coupled handlers

Improves reusability

Declarative and flexible configuration

Information sharing is inefficient

4. Application Controller

Problem

You want to centralize and modularize action and view management.

Forces

You want to reuse action and view-management code.

You want to improve request-handling extensibility, such as adding use case functionality to an application incrementally.

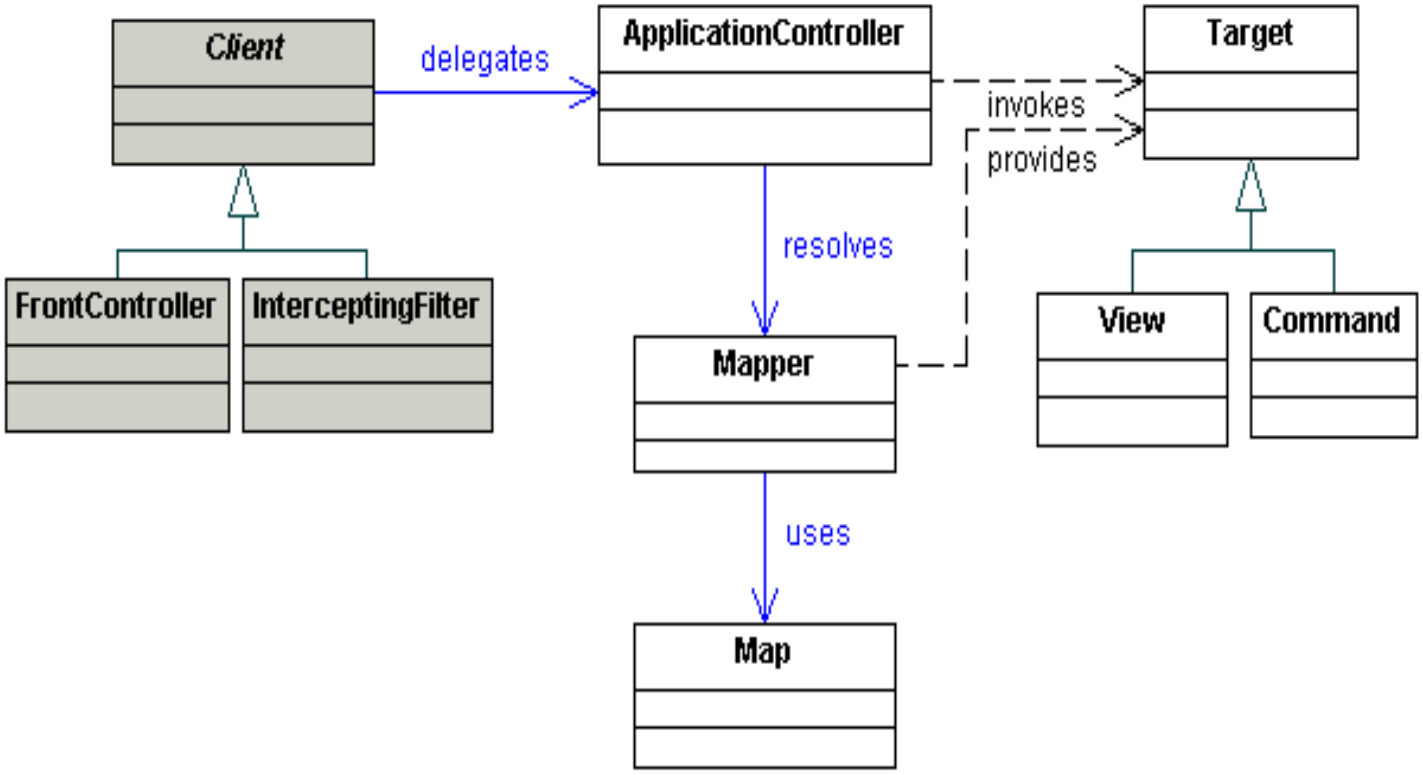
You want to improve code modularity and maintainability, making it easier to extend the application and easier to test discrete parts of your request-handling code independent of a web container.

Solution

Use an Application Controller to centralize retrieval and invocation of request-processing components, such as commands and views.

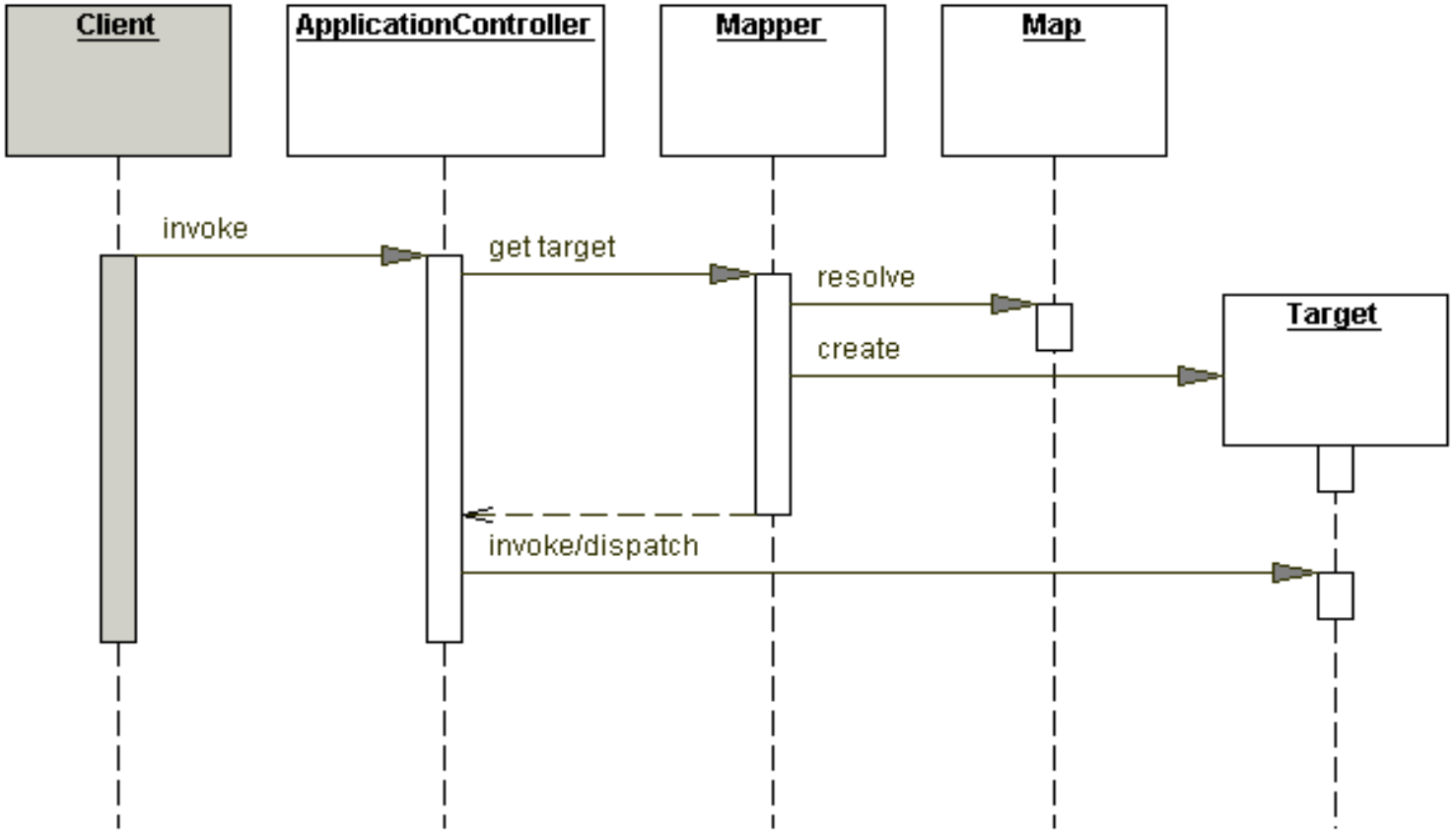
4. Application Controller

Class Diagram :



4. Application Controller

Sequence Diagram :



4. Application Controller

Strategies

Command Handler Strategy

View Handler Strategy

Transform Handler Strategy

Navigation and Flow Control Strategy

Message Handling Strategies

Custom SOAP Message Handling Strategy

JAX RPC Message Handling Strategy

Consequences

Improves modularity

Improves reusability

Improves extensibility

5. View Helper

Problem

You want to separate a view from its processing logic.

Forces

You want to use template-based views, such as JSP.

You want to avoid embedding program logic in the view.

You want to separate programming logic from the view to facilitate division of labor between software developers and web page designers.

Solution

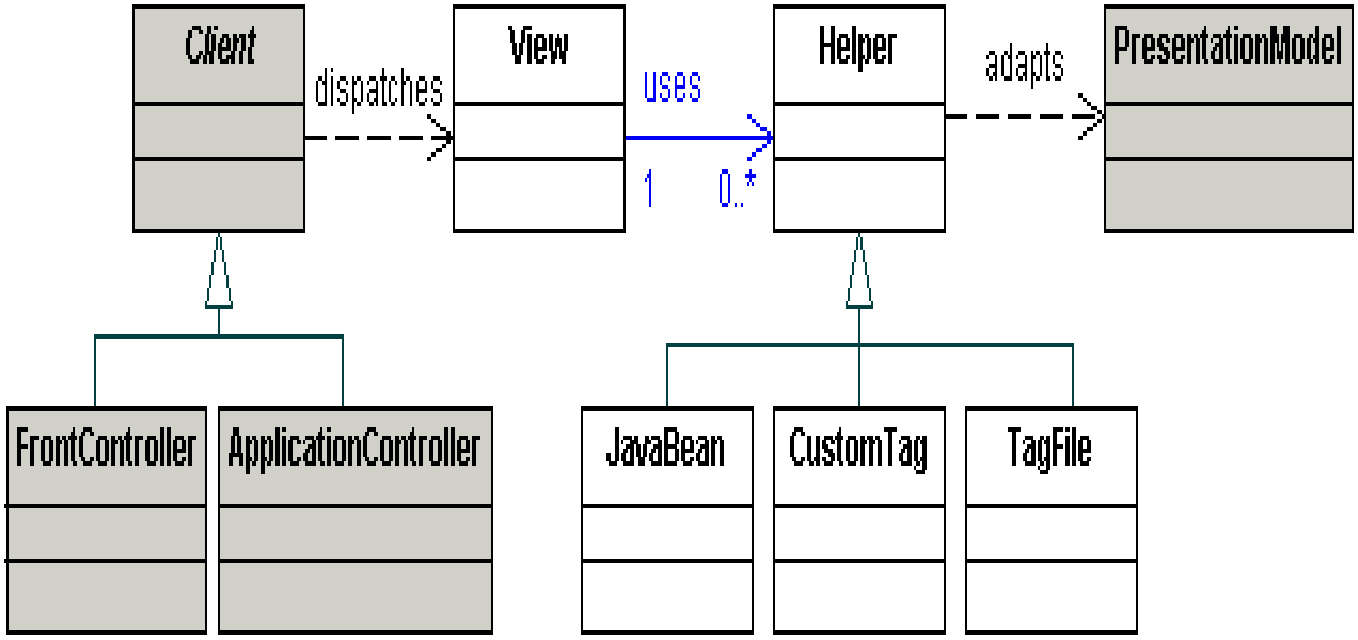
Use Views to encapsulate formatting code and Helpers to encapsulate view-processing logic.

A View delegates its processing responsibilities to its helper classes, implemented as POJOs, custom tags, or tag files.

Helpers serve as adapters between the view and the model, and perform processing related to formatting logic, such as generating an HTML table.

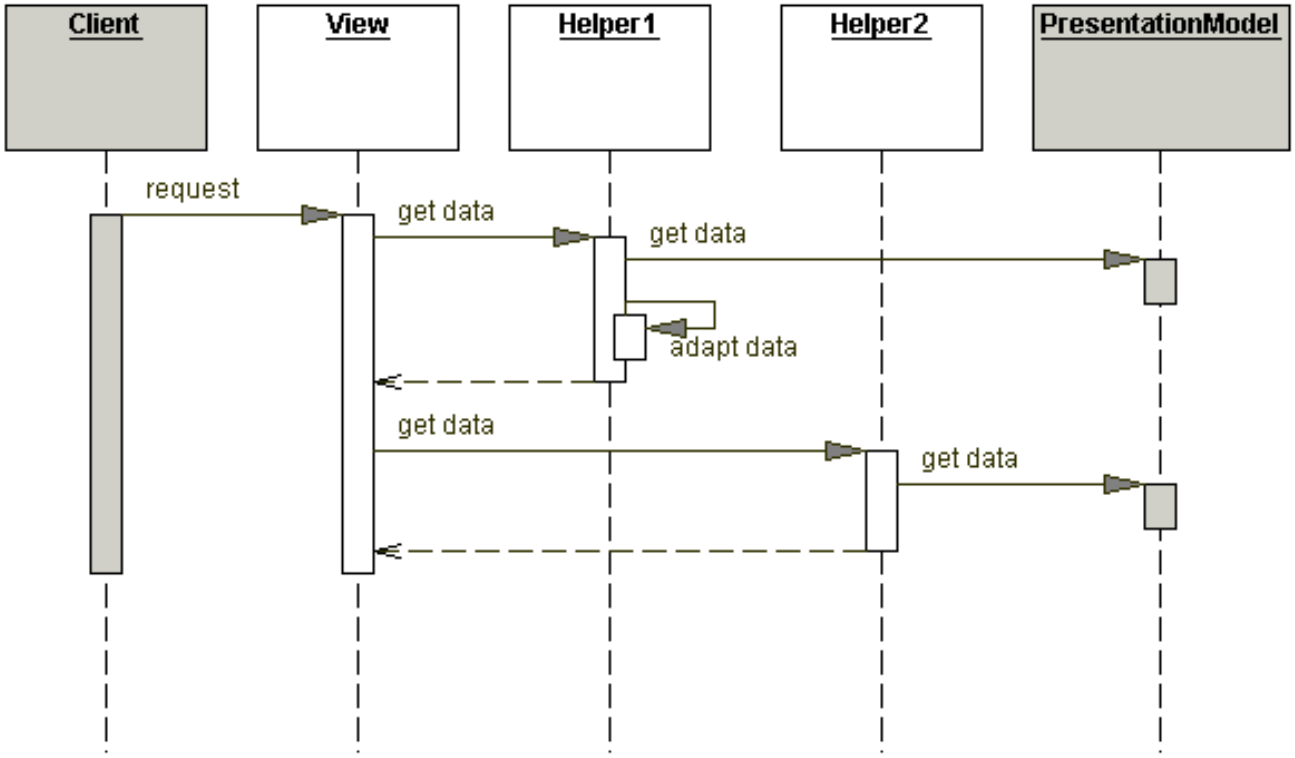
5. View Helper

Class Diagram :



5. View Helper

Sequence Diagram :



5. View Helper

Strategies

Template-Based View Strategy

Controller-Based View Strategy

JavaBean Helper Strategy

Custom Tag Helper Strategy

Tag File Helper Strategy

Business Delegate as Helper Strategy

Consequences

Improves application partitioning, reuse, and maintainability

Improves role separation

Eases testing

Helper usage mirrors scriptlets

6. Composite View

Problem

You want to build a view from modular, atomic component parts that are combined to create a composite whole, while managing the content and the layout independently.

Forces

You want common subviews, such as headers, footers and tables reused in multiple views, which may appear in different locations within each page layout.

You have content in subviews which might which frequently change or might be subject to certain access controls, such as limiting access to users in certain roles.

You want to avoid directly embedding and duplicating subviews in multiple views which makes layout changes difficult to manage and maintain.

6. Composite View

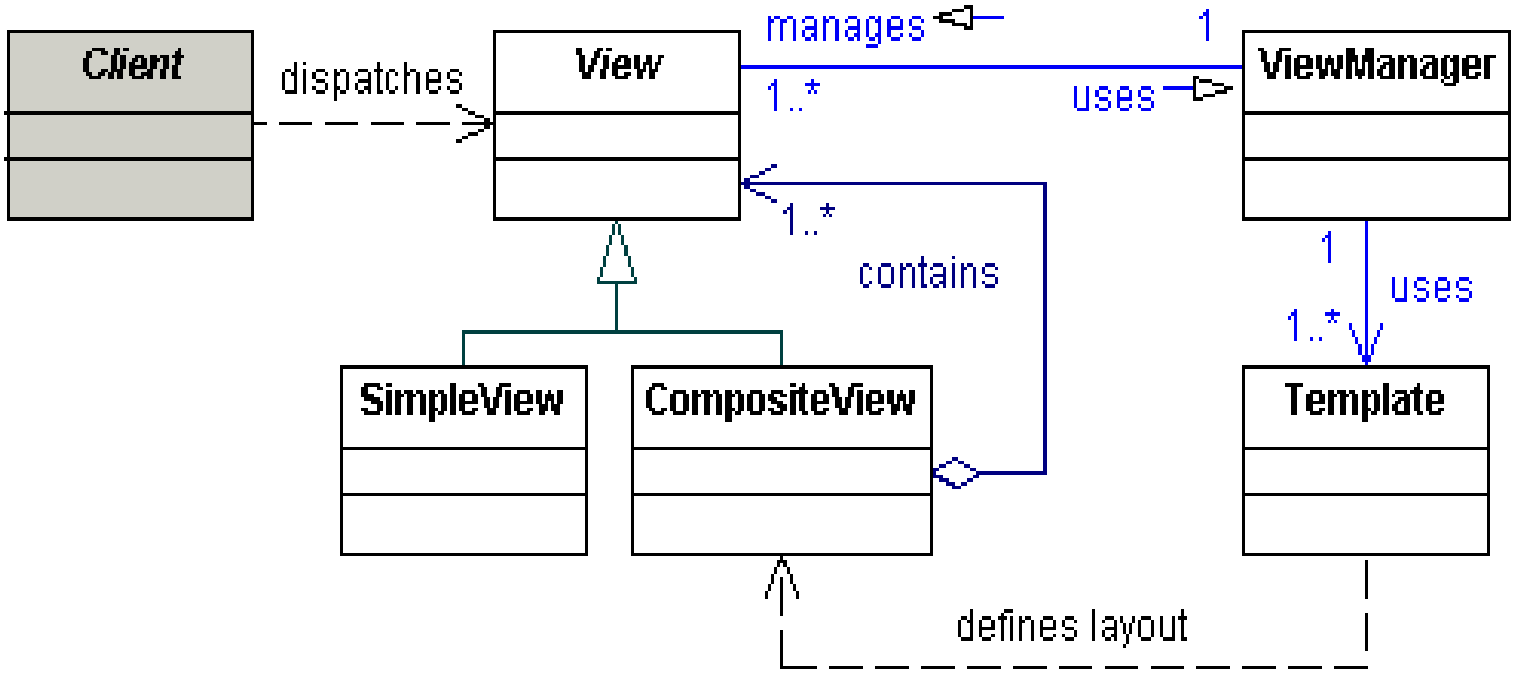
Solution

Use Composite Views that are composed of multiple atomic subviews.

Each subview of the overall template can be included dynamically in the whole, and the layout of the page can be managed independently of the content.

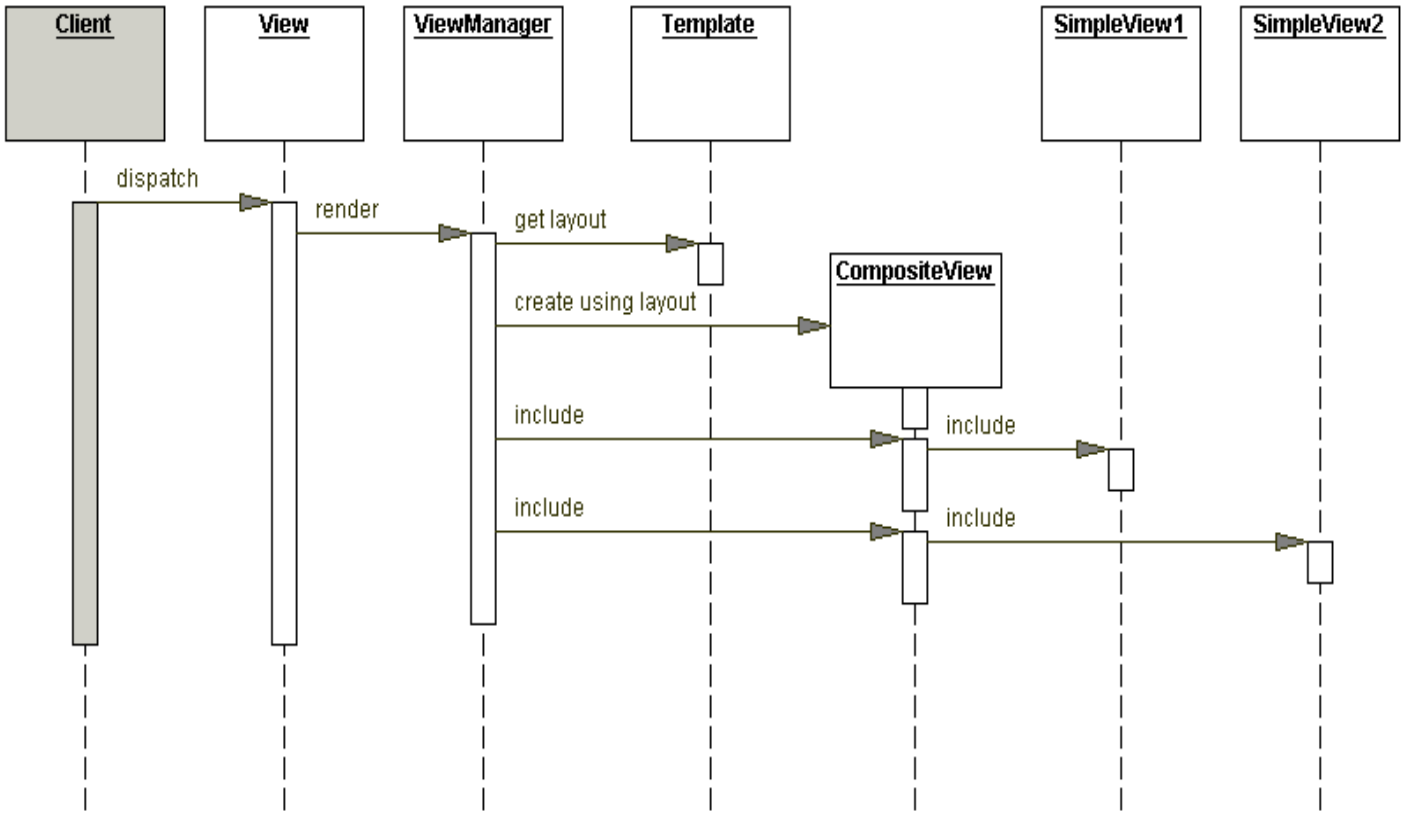
6. Composite View

Class Diagram :



6. Composite View

Sequence Diagram :



6. Composite View

Strategies

JavaBean View Management Strategy

Standard Tag View Management Strategy

Custom Tag View Management Strategy

Transformer View Management Strategy

Early-Binding Resource Strategy

Late-Binding Resource Strategy

Consequences

Improves modularity and reuse

Adds role-based or policy-based control

Enhances maintainability

Reduces maintainability

Reduces performance

7. Dispatcher View

Problem

You want a view to handle a request and generate a response, while managing limited amounts of business processing.

Forces

You have static views.

You have views generated from an existing presentation model.

You have views which are independent of any business service response.

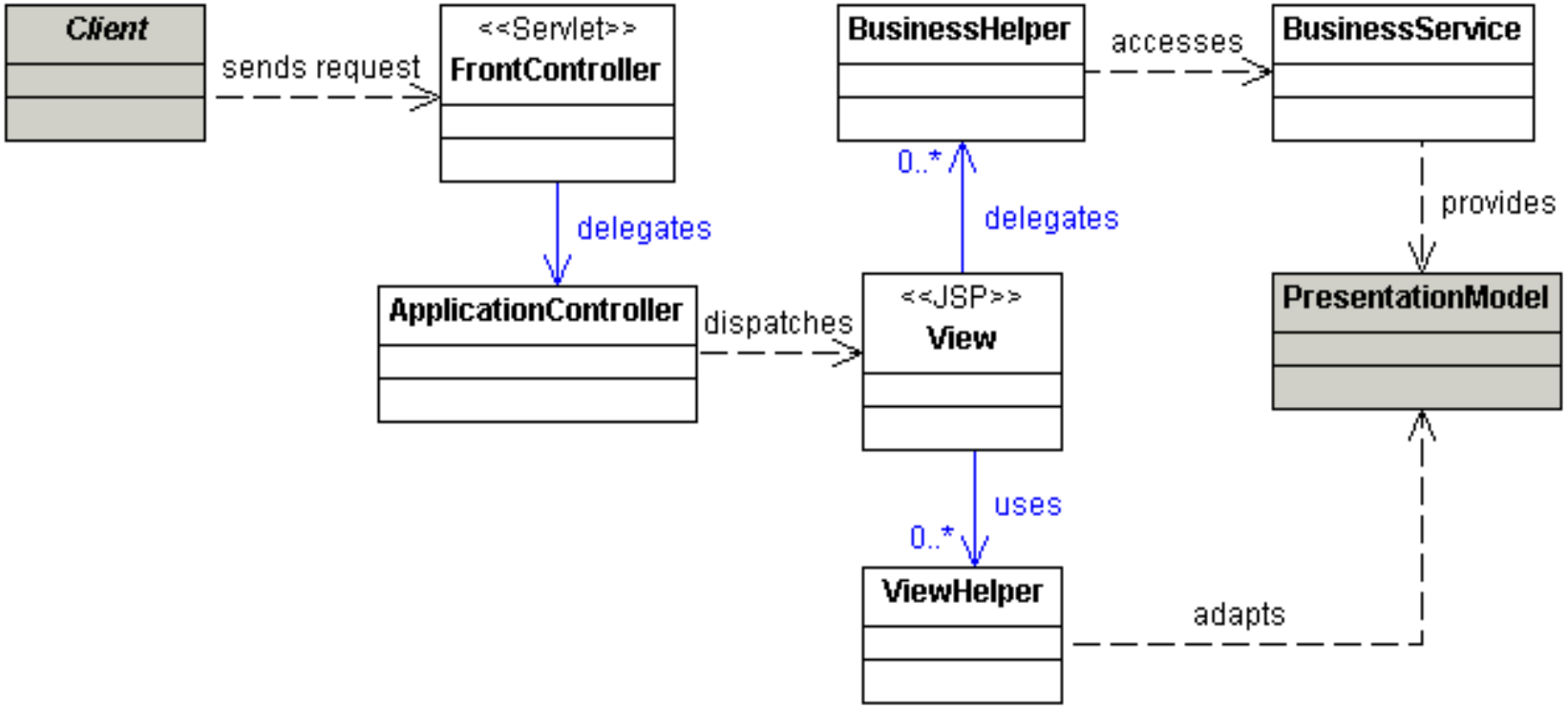
You have limited business processing.

Solution

Use Dispatcher View with views as the initial access point for a request. Business processing, if necessary in limited form, is managed by the views.

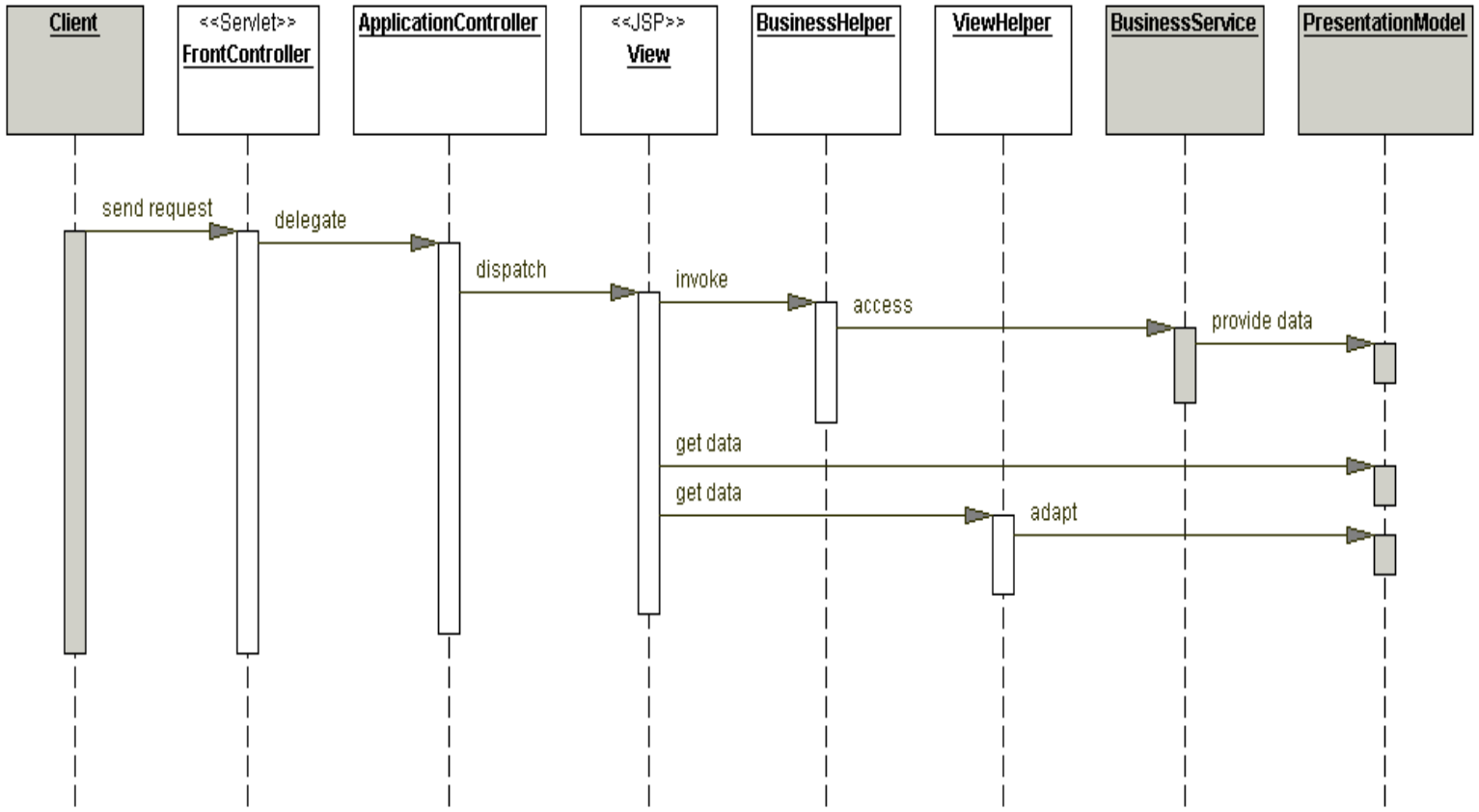
7. Dispatcher View (JSF page-centric)

Class Diagram :



7. Dispatcher View

Sequence Diagram :



7. Dispatcher View

Strategies

Servlet Front Strategy

JSP Front Strategy

Template-Based View Strategy

Controller-Based View Strategy

JavaBean Helper Strategy

Custom Tag Helper Strategy

Dispatcher in Controller Strategy

Consequences

Leverages frameworks and libraries.

Introduces potential for poor separation of the view from the model and control logic.

Separates processing logic from view and improves reusability.

8. Service To Worker (Struts & Web Flow)

Problem

You want to perform core request handling and invoke business logic before control is passed to the view.

Forces

You want specific business logic executed to service a request in order to retrieve content that will be used to generate a dynamic response.

You have view selections which may depend on responses from business service invocations.

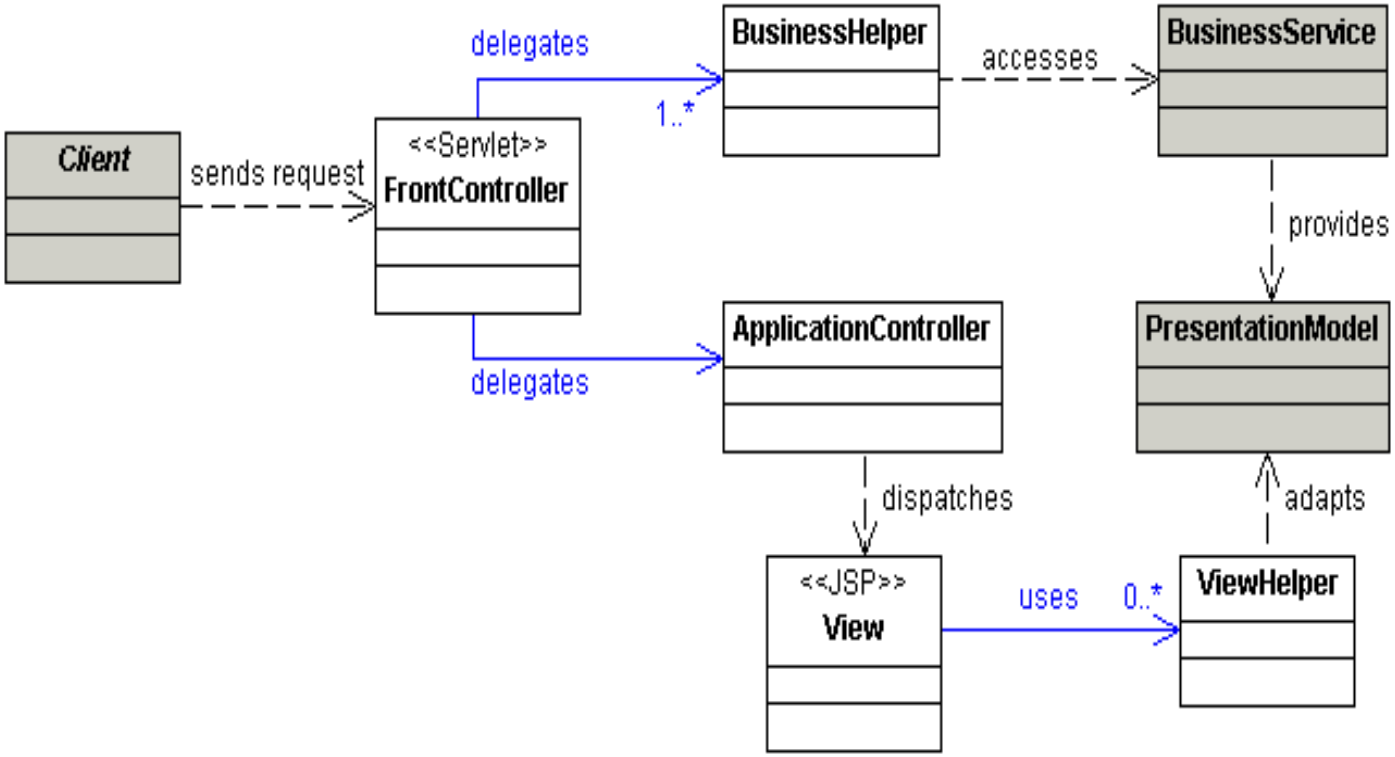
You may have to use a framework or library in the application.

Solution

Use Service to Worker to centralize control and request handling to retrieve a presentation model before turning control over to the view. The view generates a dynamic response based on the presentation model.

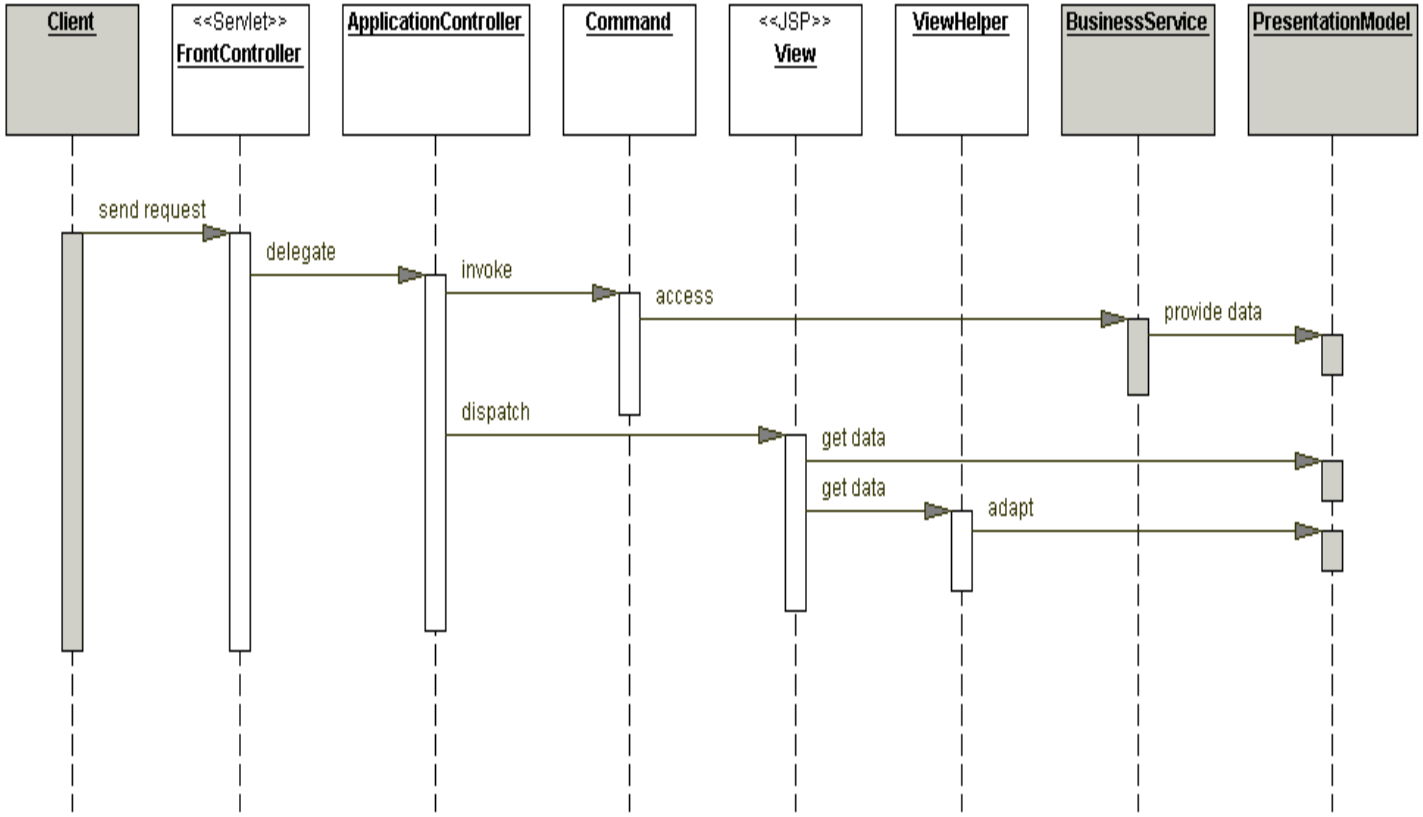
8. Service To Worker

Class Diagram :



8. Service To Worker

Sequence Diagram :



8. Service To Worker

Strategies

Servlet Front Strategy

JSP Front Strategy

Template-Based View Strategy

Controller-Based View Strategy

JavaBean Helper Strategy

Custom Tag Helper Strategy

Dispatcher in Controller Strategy

Consequences

Leverages frameworks and libraries.

Introduces potential for poor separation of the view from the model and control logic.

Separates processing logic from view and improves reusability.