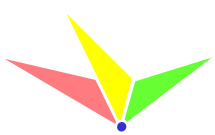
Padrões de Projeto J2EE 2

Padrões da Camada de Apresentação (Web)

Helder da Rocha (helder@acm.org)

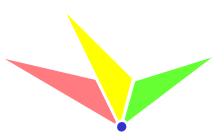
Introdução

- *A camada de apresentação encapsula toda a lógica relacionada com a visualização e comunicação com a interface do usuário*
  - *Requisições e respostas HTTP*
  - *Gerenciamento de sessão HTTP*
  - *Geração de HTML, JavaScript e recursos lado-cliente*
- *Principais componentes: Servlets e JSP*
  - *JSP é indicado para produzir interface do usuário*
  - *Servlets são indicados para processar dados recebidos e concentrar lógica de controle*



Práticas não recomendadas

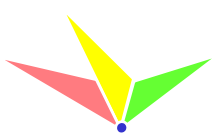
- *Enquanto os padrões promovem as melhores soluções, as práticas não recomendadas indicam necessidade de aplicação dos padrões*
  - *Código de controle em vários Views*
  - *Exposição de estruturas da camada de apresentação a outras camadas e objetos de domínio*
  - *Permissão para requisições de formulário duplicadas*
  - *Permissão de acesso do cliente a recursos sensíveis*
  - *Permissão de inconsistência de propriedades <jsp:property>*
  - *Controladores gordos*
- *Vários padrões e refactorings sugerem soluções que resolvem os problemas acima**



* Veja Presentation Tier Bad Practices e Refactorings em [Core]

Padrões da Camada de Apresentação

- *(1) Front Controller*
  - *Oferece um controlador centralizado para gerenciar o processamento de uma requisição*
- *(2) View Helper*
  - *Encapsula lógica não-relacionada à formatação*
- *(3) Composite View*
  - *Cria uma View composta de componentes menores*
- *(4) Intercepting Filter*
  - *Viabiliza pré- e pós-processamento de requisições*
- *(5) Service To Worker e (6) Dispatcher View*
  - *Combinam Front Controller com um Dispatcher e Helpers. O primeiro concentra mais tarefas antes de despachar a requisição. O segundo realiza mais processamento depois.*

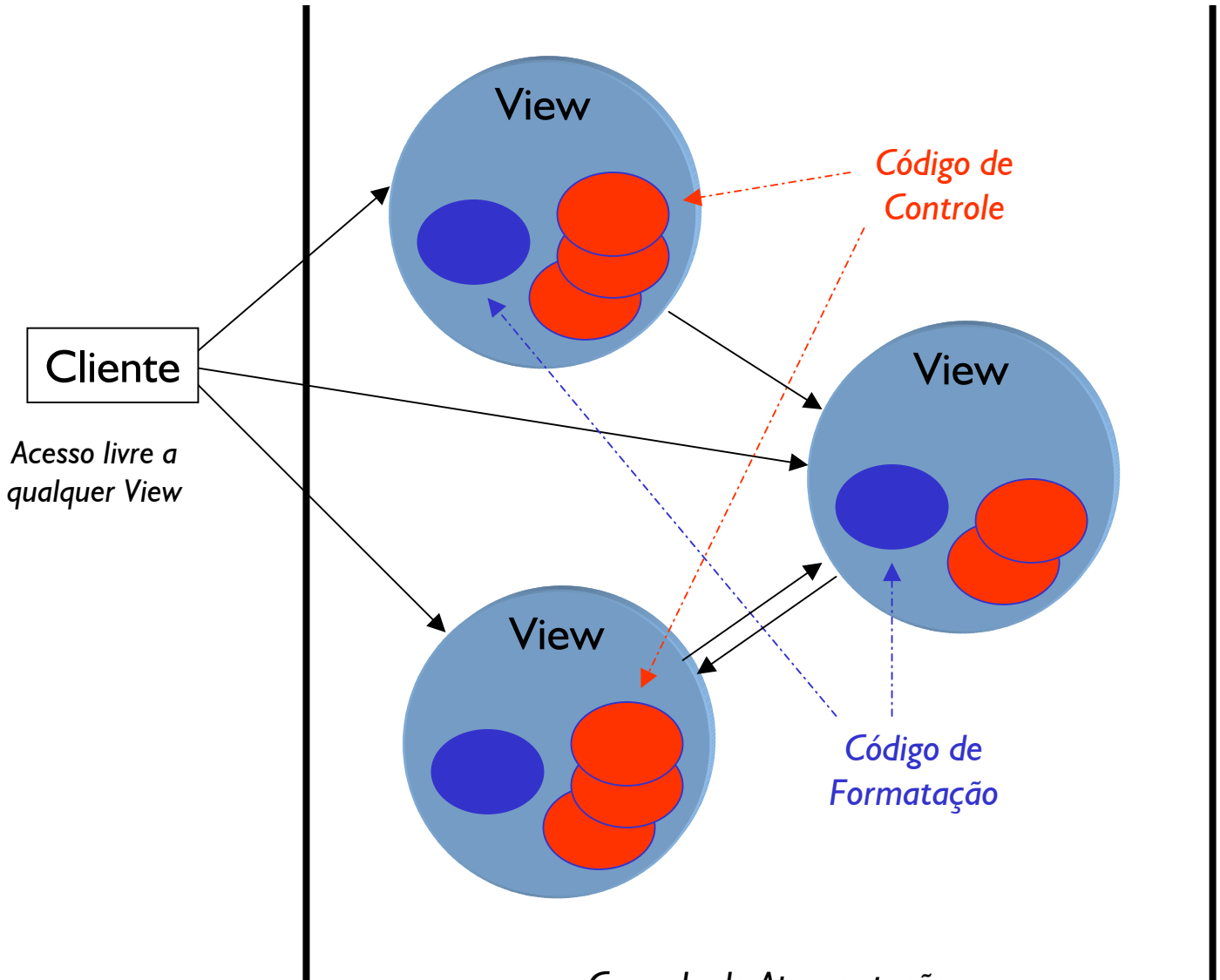


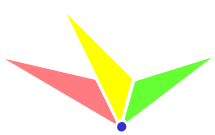
1

Front Controller

Objetivo: centralizar o processamento de requisições em uma única fachada. Front Controller permite criar uma interface genérica para processamento de comandos.

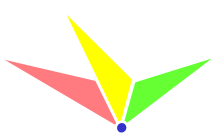
Problema


View
Código de Controle
Cliente
View
Acesso livre a qualquer View
View
Código de Formatação
Camada de Apresentação

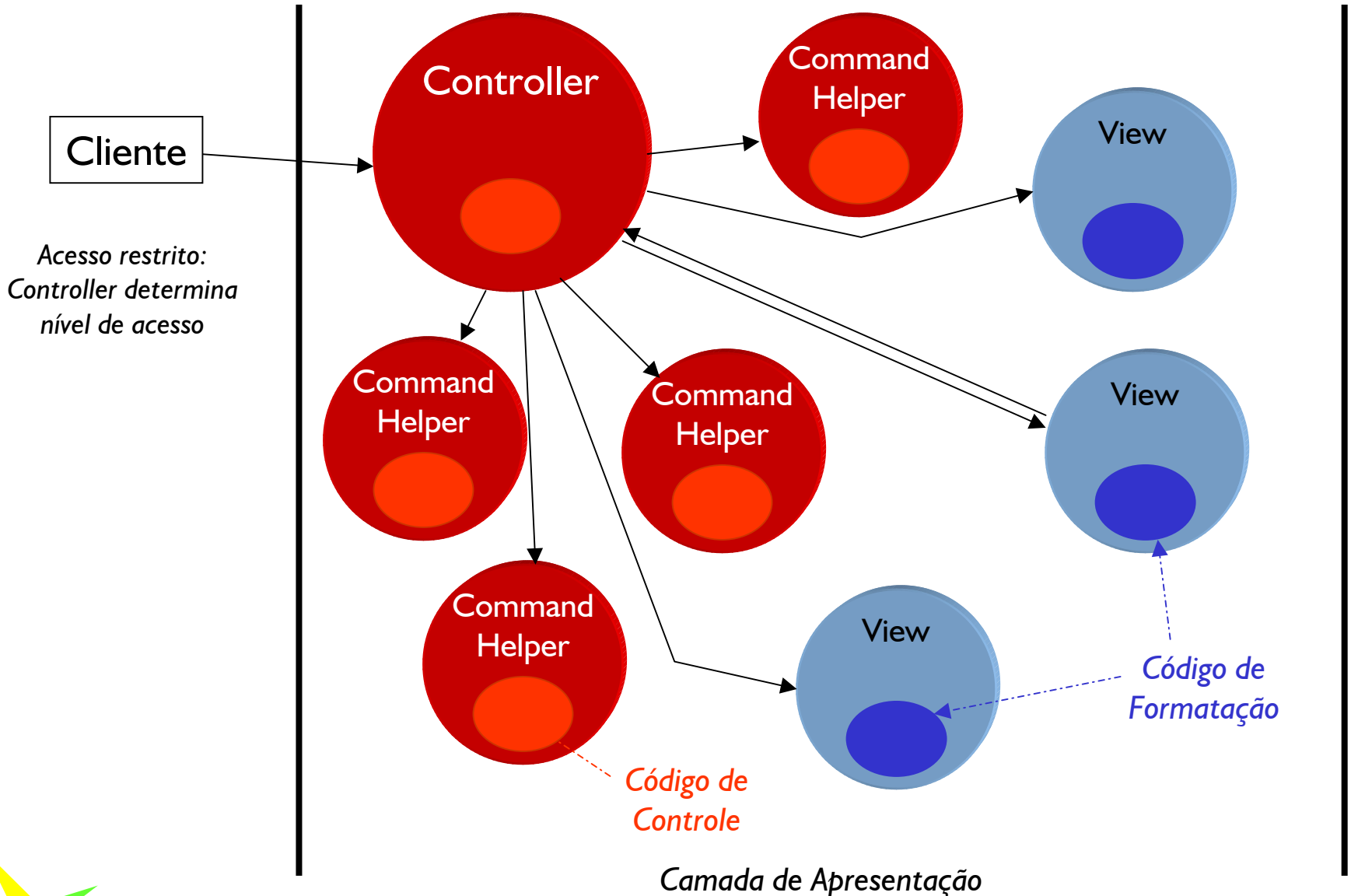


Descrição do problema

- Deseja-se um ponto de acesso centralizado para processamento de todas as requisições recebidas pela camada de apresentação para
  - Controlar a navegação entre os Views
  - Remover duplicação de código
  - Estabelecer responsabilidades mais definidas para cada objeto, facilitando manutenção e extensão: JSP não deve conter código algum ou pelo menos não código de controle
- Se um usuário acessa um View sem passar por um mecanismo centralizado, código de controle é duplicado e misturado ao código de apresentação
  - Também não é possível controlar a navegação: cliente pode iniciar em página que não deveria ter acesso.



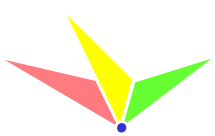
Solução: Front Controller



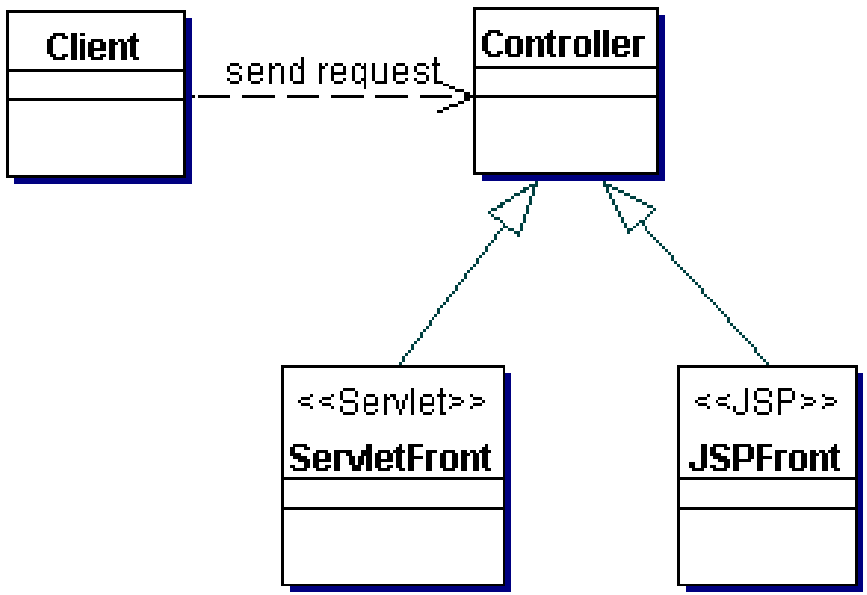
Acesso restrito: Controller determina nível de acesso

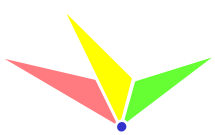
Descrição da solução

- *Controlador é ponto de acesso para processamento de requisições*
  - *chama serviços de segurança (autenticação e autorização)*
  - *delega processamento à camadas de negócio*
  - *define um View apropriado*
  - *realiza tratamento de erros*
  - *define estratégias de geração de conteúdo*
- *O controlador pode delegar processamento a objetos Helper (Comandos, Value Beans, etc.)*
- *Solução depende do uso de um Dispatcher*
  - *Usado para redirecionar para o View correspondente*



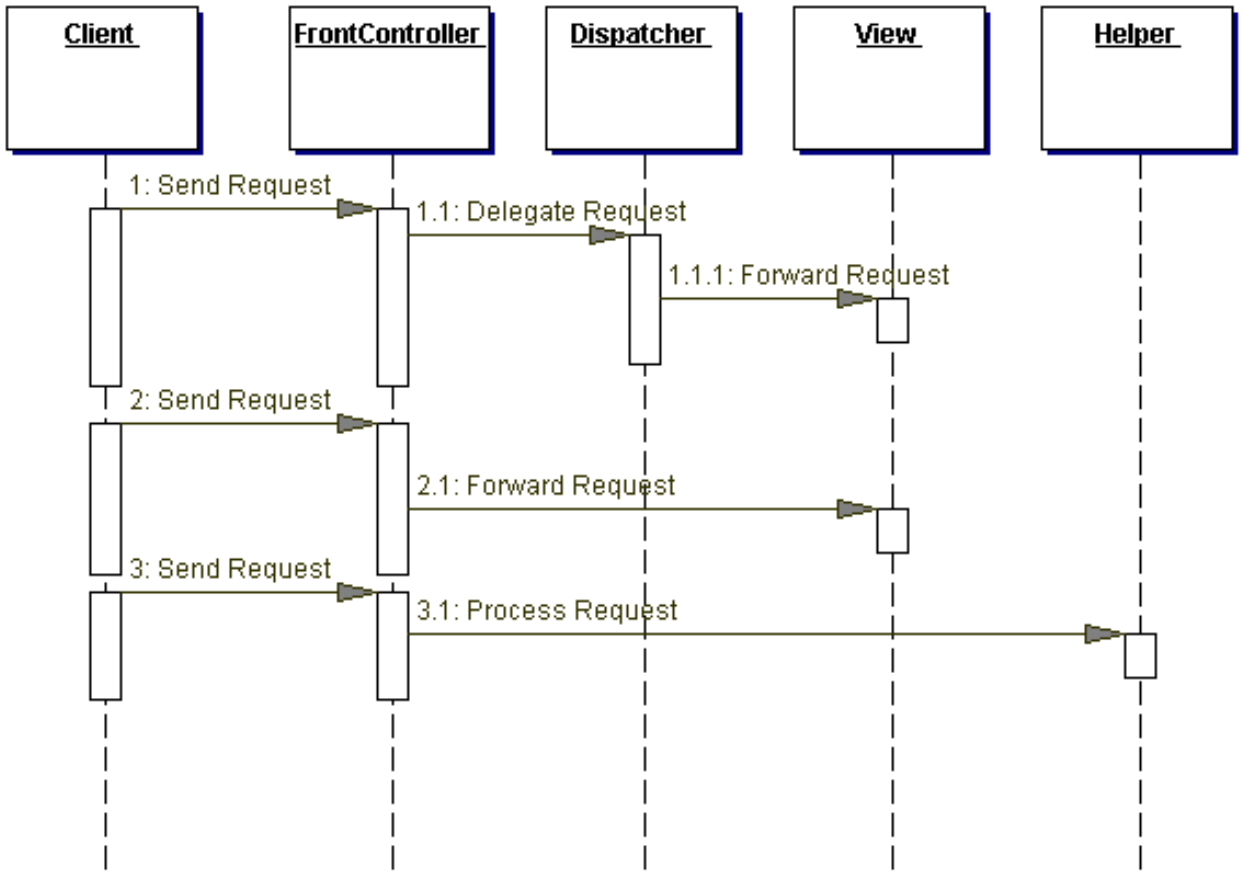
Estrutura UML



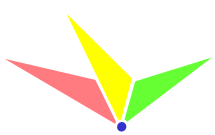


Fonte: [SJC]

Diagrama de Seqüência



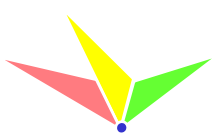

```
1.1    RequestDispatcher rd = request.getRequestDispatcher("View.jsp");
1.1.1  rd.forward(request, response);
```



Fonte: [SJC]

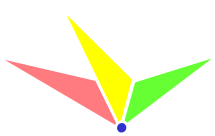
Participantes e responsabilidades

- *Controller*
  - *Ponto de entrada que centraliza todas as requisições*
  - *Pode delegar responsabilidade a Helpers*
- *Dispatcher*
  - *Tipicamente usa ou encapsula objeto javax.servlet.RequestDispatcher*
- *Helper*
  - *Pode ter inúmeras responsabilidades, incluindo a obtenção de dados requerido pelo View*
  - *Pode ser um Value Bean, Business Delegate, Command, ...*
- *View*
  - *Geralmente página JSP*



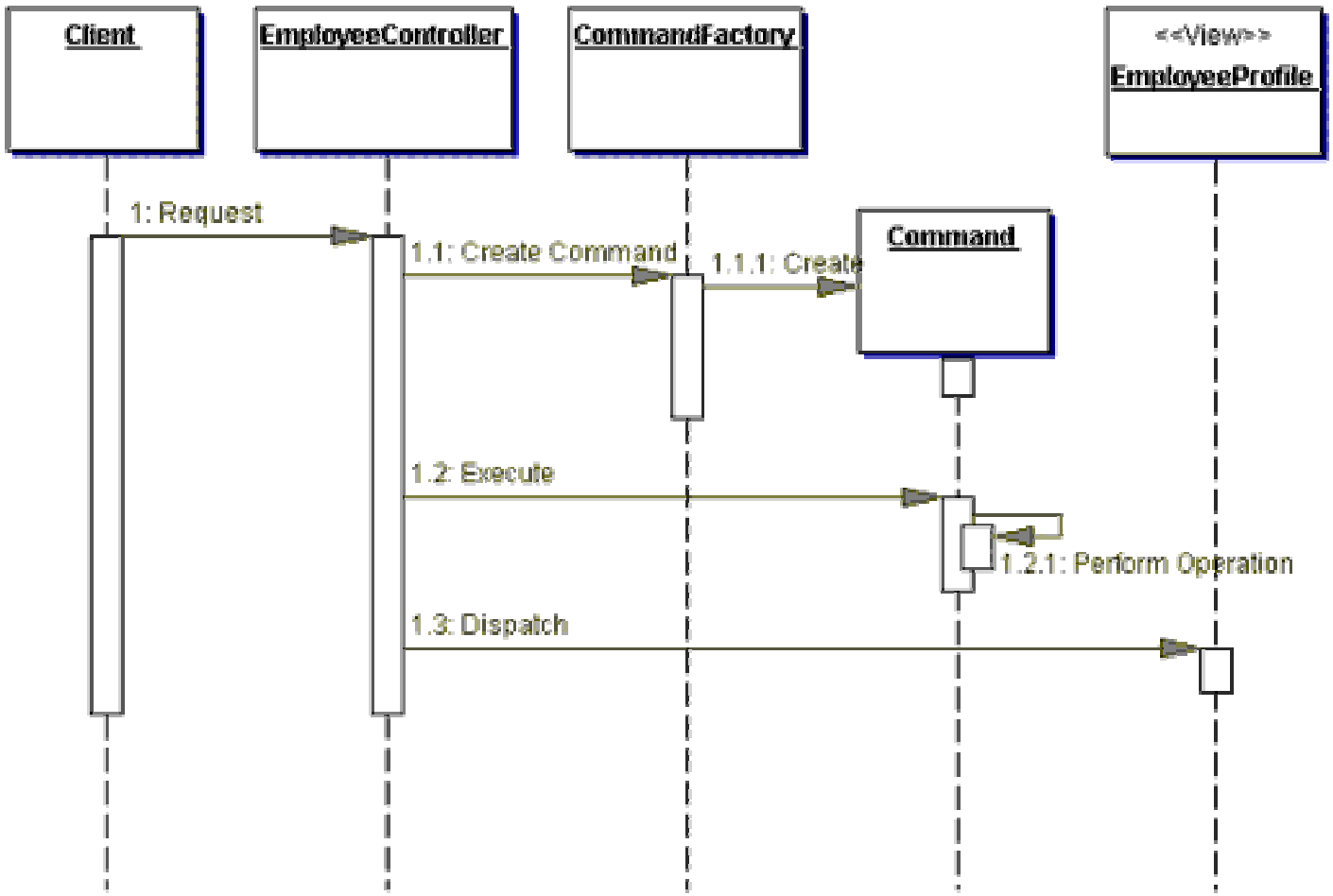
Melhores estratégias de implementação*

- *Servlet Front Strategy*
  - *Implementa o controlador como um servlet. Exemplo 7.14*
  - *Dispatcher and Controller Strategy implementa o Dispatcher dentro do próprio servlet*
- *Command and Controller Strategy*
  - *Interface baseada no padrão Command (GoF) para implementar Helpers para os quais o controlador delega responsabilidades. Exemplo 7.16*
- *Logical Resource Mapping Strategy*
  - *Requisições são feitas para nomes que são mapeados a recursos (páginas JSP, servlets) ou comandos*
  - *Multiplexed Resource Mapping Strategy usa wildcards para selecionar recursos a serem processados*



** Veja exemplos de código em exemplos/preslayer/*

Command and Controller Strategy



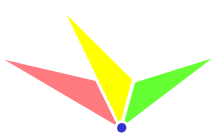
Fonte: [Core]

Conseqüências

- Controle centralizado
  - Facilidade de rastrear e logar requisições
- Melhor gerenciamento de segurança
  - Requer menos recursos. Não é preciso distribuir pontos de verificação em todas as páginas
  - Validação é simplificada
- Melhor possibilidade de reuso
  - Distribui melhor as responsabilidades

Exercícios

- *1. Analise a implementação das estratégias de FrontController (código 7.14, 7.15 e 7.16)*
- *2. Refatore a aplicação em preslayer/fc/ para que utilize FrontController. Empregue as três estratégias de implementação apresentadas:*
  - *a) Implemente o controlador usando um Servlet*
  - *b*) Escreva um RequestHelper que mantenha uma tabela de comandos/nomes de classe de objetos Command e receba um request na construção. Seu método getCommand() deve retornar o comando correspondente recebendo newMessage, lastMessage, allMessages*
  - *c) Configure o web.xml para mapear todas as requisições ao controlador*



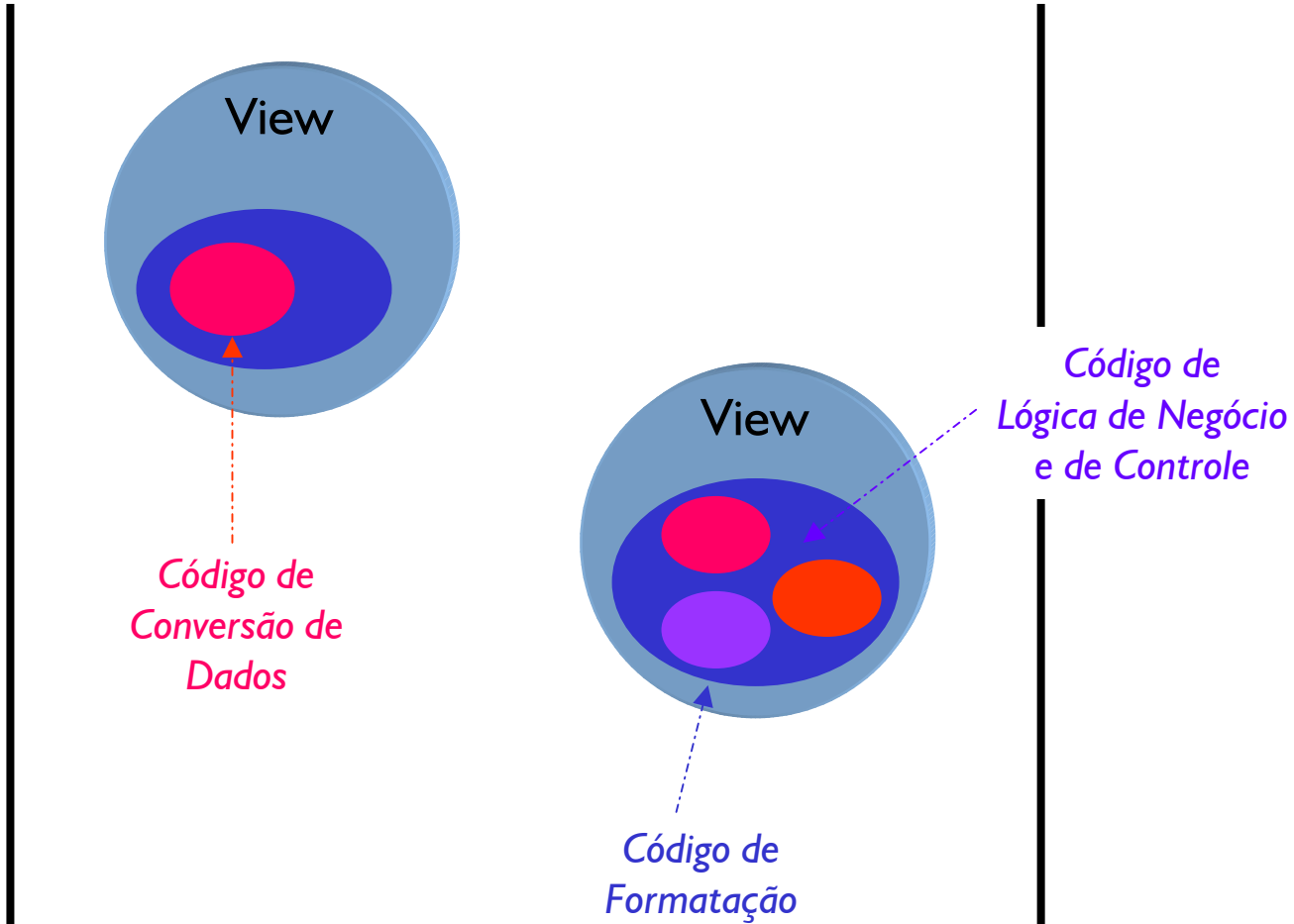
* *Mesmo exercício em Service To Worker*

2

View Helper

Objetivo: separar código e responsabilidades de formatação da interface do usuário do processamento de dados necessários à construção da View. Tipicamente implementados como JavaBeans e Custom Tags.

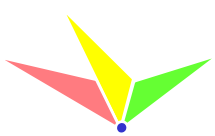
Problema



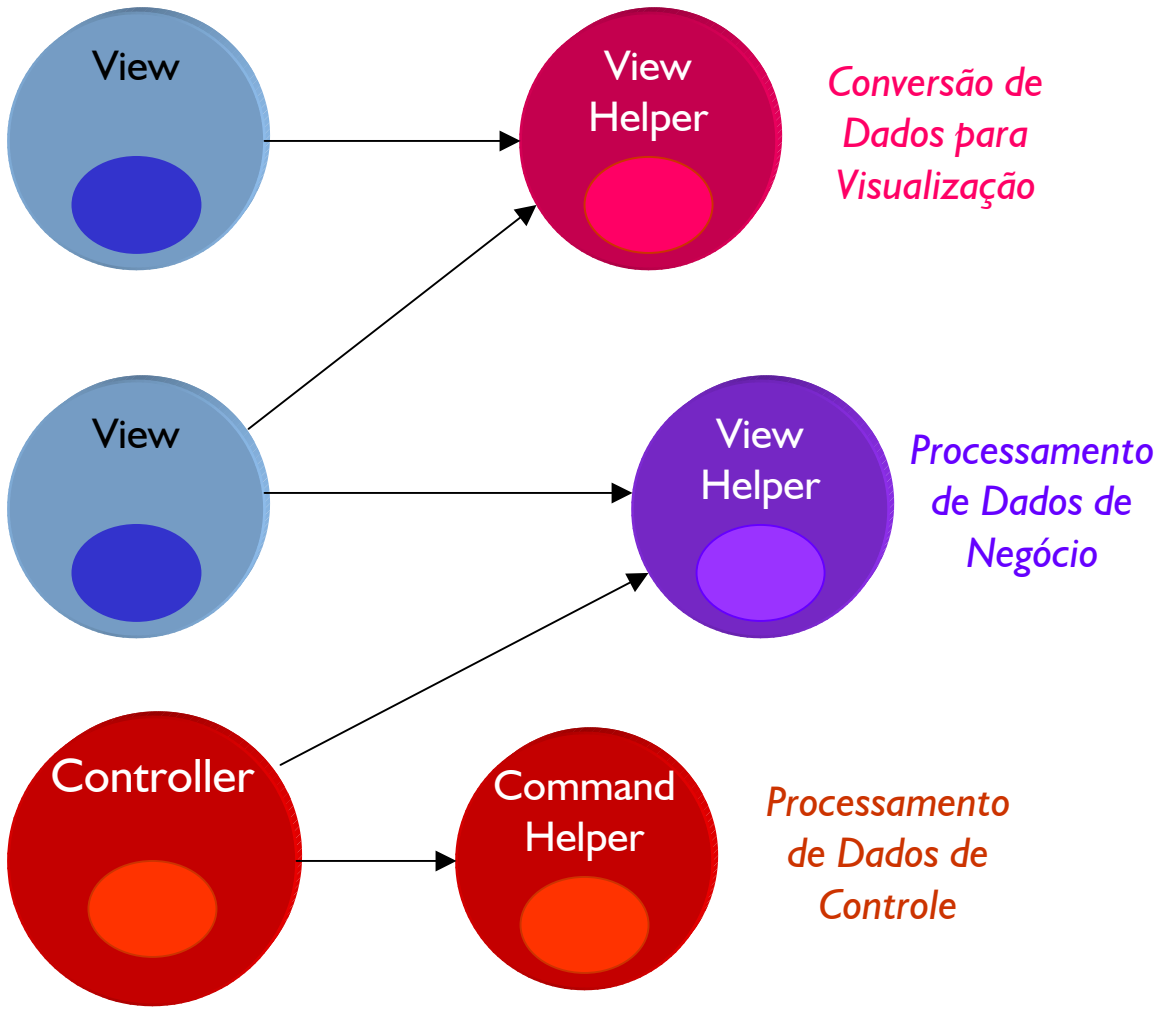
Camada de Apresentação

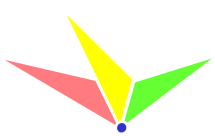
Descrição do problema

- *Mudanças na camada de apresentação são comuns*
  - *Alterações da interface do usuário*
- *Se código de apresentação (HTML, JavaScript) estiver misturado com código de processamento e controle (Java) as mudanças são dificultadas*
  - *Menos flexibilidade, menos reuso, menos modularidade e mistura de papéis em um mesmo componente*
- *É preciso identificar as responsabilidades de cada trecho de código e encapsulá-lo em objetos usados pela camada de apresentação*



Solução: View Helpers

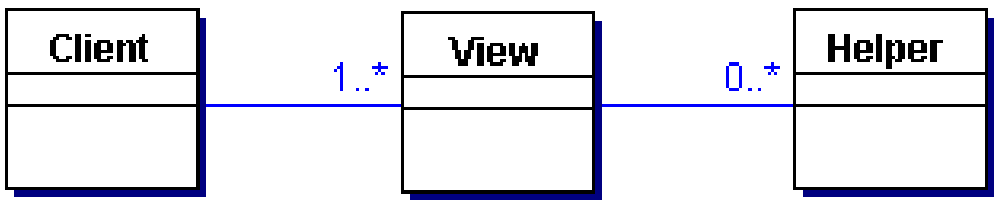




Descrição da solução

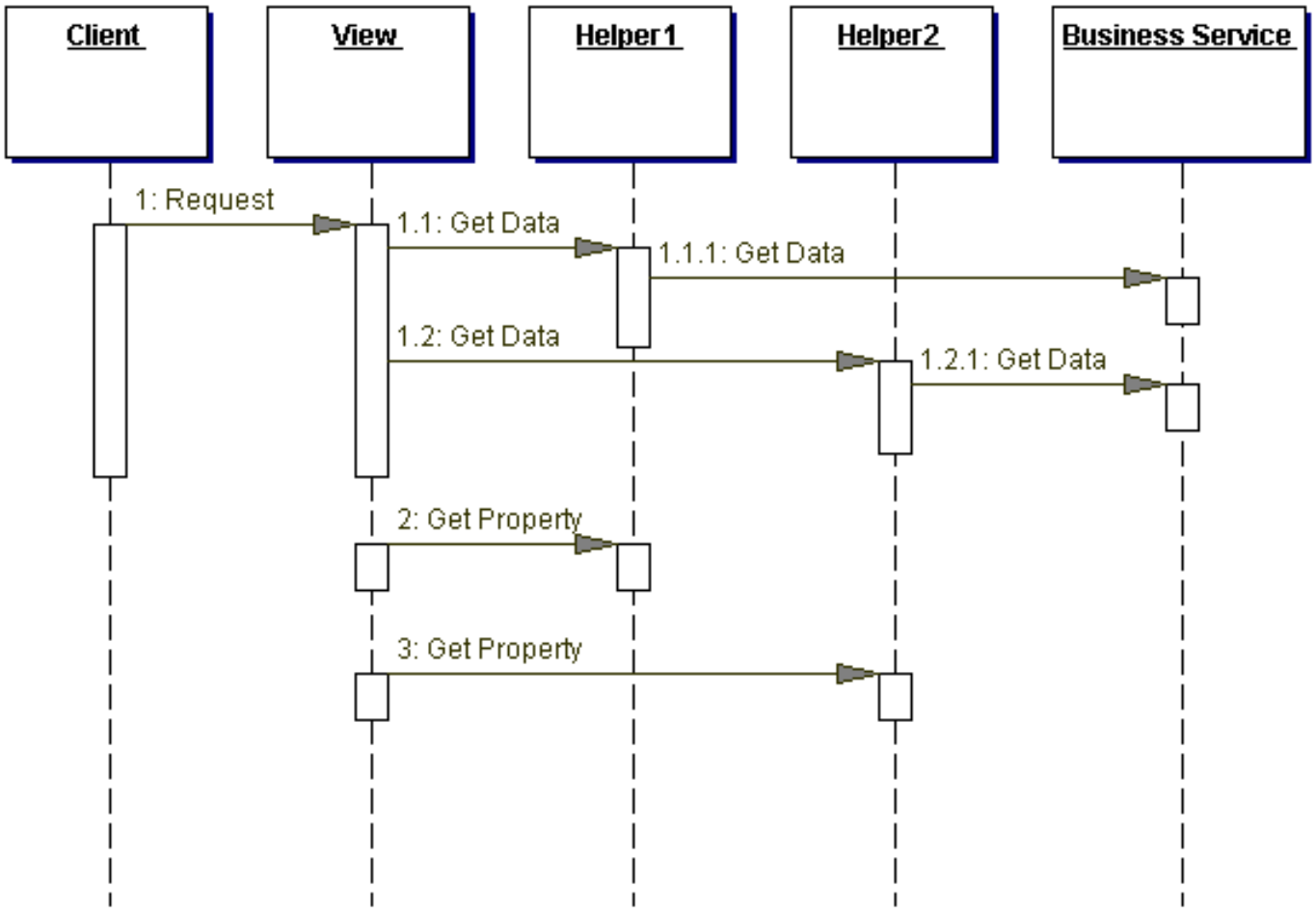
- *O padrão View Helper recomenda soluções para dividir as responsabilidades do View*
- *Uma View contém código relacionado apenas à formatação*
  - *Responsabilidades de processamento são delegados à classes ajudantes: View Helpers (implementadas como JavaBeans ou como Custom Tags)*
  - *Helpers também podem guardar modelo de dados intermediário usado pelo View e servir como adaptadores para dados oriundos da camada de negócios*
  - *Refatoramento pode sugerir separação de determinados trechos de código em objetos Controladores em vez de View Helpers*
- *Há várias estratégias para implementação de Views em associação com View Helpers*

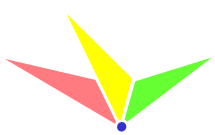
Estrutura UML



Fonte: [SJC]

Diagramas de Seqüência

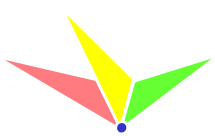




Fonte: [SJC]

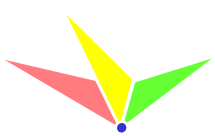
Participantes e responsabilidades

- *Business Service*
  - *Representa um serviço como propriedades, banco de dados, etc. Tipicamente é acessado através de um Business Delegate.*
- *Helper 1*
  - *Ajudam um view ou controller realizar seu processamento*
  - *Têm muitas responsabilidades*
- *Helper 2 (Value Bean)*
  - *Value Bean é um tipo de Helper implementado como JavaBean que tem como objetivo manter estado intermediário de modelo de dados para uso pelo View*
- *View*
  - *Representa e mostra informações no cliente*



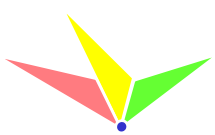
Melhores estratégias de implementação

- JSP View Strategy
  - JSP é componente de View
- JavaBean Helper Strategy
  - Helper implementado como JavaBean
- Custom Tag Helper Strategy
  - Mais complexo que JavaBean Helper
  - Separação de papéis maior (isola a complexidade)
  - Maior índice de reuso (pode-se usar custom tags existentes)
- Business Delegate as Helper Strategy
  - Papéis de View Helper e Business Delegate podem ser combinados para acesso à camada de negócio
  - Pode misturar papéis J2EE



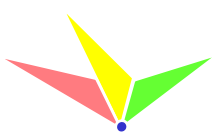
Conseqüências

- Melhora particionamento da aplicação
  - Facilita o reuso
  - Facilita a manutenção
  - Facilita a realização de testes funcionais, de unidade e de integração
- Melhora separação de papéis J2EE
  - Reduz a complexidade para todos os participantes: Web Designer não precisa ver Java e Programador Java não precisa ver JavaScript e HTML



Exercícios

- *1. Analise a implementação das estratégias de ViewHelper (código 7.17 a 7.21)*
- *2. Refatore a aplicação em preslayer/vh/ para que utilize ViewHelper. Use JavaBean Helper Strategy:*
  - *a) Identifique código de conversão de formatos, código de negócio e código de controle (se houver)*
  - *b) Construa um Helper para cada responsabilidade encontrada*
- *3. Use Custom Tag Helper Strategy para encapsular a lógica de repetição*
  - *Use tags do Struts <logic:iterate> ou JSTL <c:forEach>*

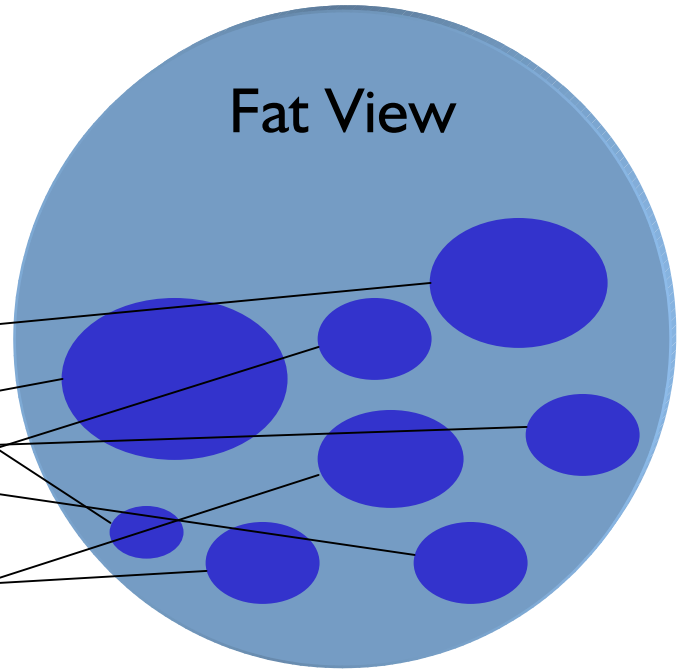


3

Composite View

Objetivo: criar um componente de View a partir de Views menores para dividir as responsabilidades, simplificar a construção da interface e permitir o reuso de componentes da View.

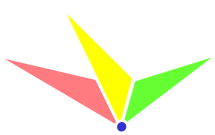
Problema



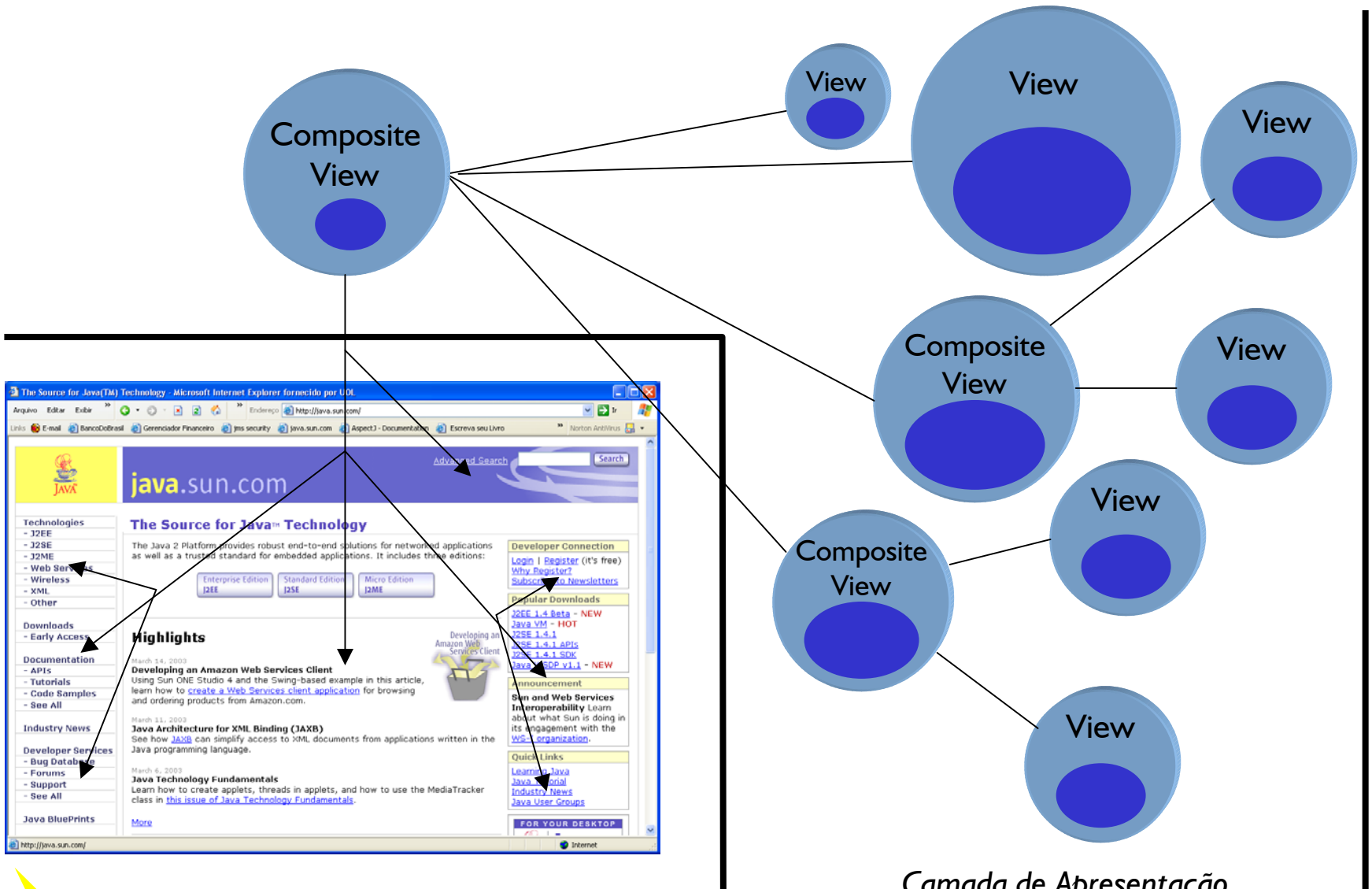
Camada de Apresentação

Descrição do problema

- *Páginas Web sofisticadas dividem seu conteúdo em várias partes, que tem função e tempo de vida diferentes*
  - *Podem ser manipuladas por pessoas diferentes já que cada seção tem finalidade, escopo e duração diferentes*
- *Se a página for gerada a partir de uma única View que concentra todo o código, a atualização da página é dificultada*
  - *É difícil identificar cada parte dentro de um documento*
  - *Há risco, quando se atualiza uma seção, de se invadir outra seção e afetar os dados*



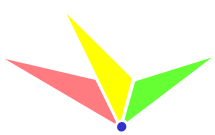
Solução: Composite View



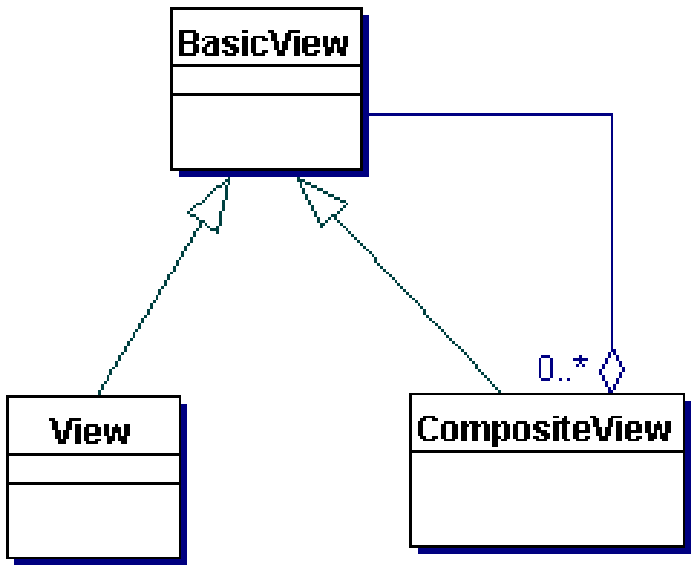
Camada de Apresentação

Descrição da solução

- *Usar views que consistem de composições de views menores*
  - *Os componentes do template podem ser incluídos dinamicamente e gerenciados separadamente*
  - *O layout geral da página pode ser manipulado independente do conteúdo*
  - *Componentes podem conter views ou coleções de views*

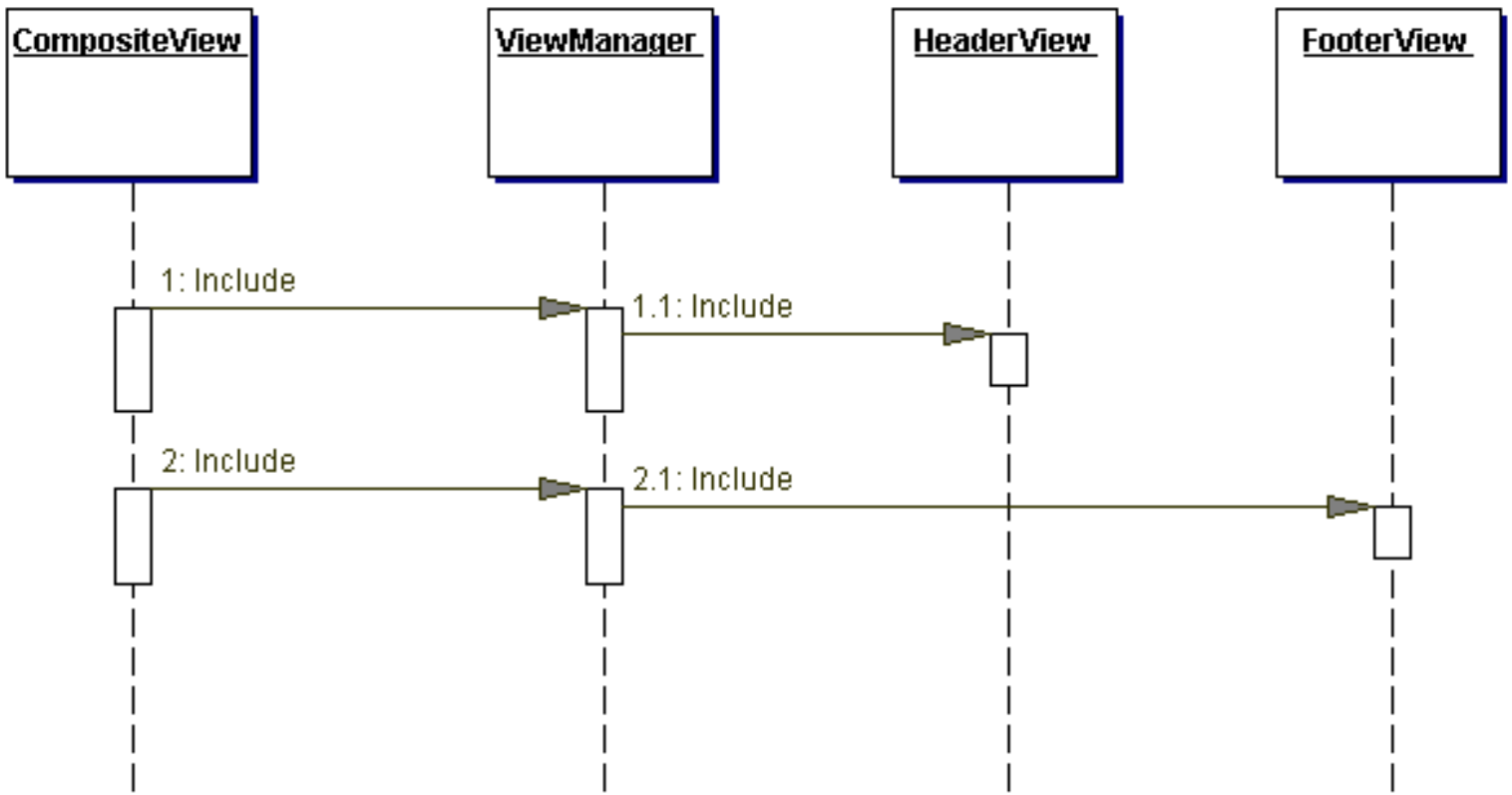


Estrutura UML



Fonte: [SJC]

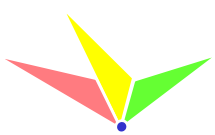
Diagramas de Seqüência



Fonte: [SJC]

Participantes e responsabilidades

- *Composite View*
  - *Agregado composto de sub-views*
- *View Manager*
  - *Gerencia a inclusão de porções de fragmentos de template no Composite View*
  - *Geralmente parte do processador JSP mas pode ser implementado também como JavaBean*
- *Included View*
  - *Sub view que pode ser uma view final ou uma composição de views menores*



Melhores estratégias de implementação

- *JavaBean View Management Strategy (7.22)**
  - *Utiliza JavaBeans para incluir outros views na página*
  - *Mais simples que solução com Custom Tags*
- *Early Binding Resource Strategy (Translation-time; 7.24)*
  - *Usa tags padrão do JSP:* `<%@ include %>` e `<%@ file %>`
  - *Carga é feita em tempo de compilação: alterações só são vistas quando página for recompilada*
- *Late Binding Resource Strategy (Run-time; 7.25)*
  - *Usa tag padrão do JSP: <jsp:include>*
  - *Carga é feita quando página é carregada: alterações são visíveis a cada atualização*
- *Custom Tag View Management Strategy (7.23)*
  - *Utiliza Custom Tags: solução mais elegante e reutilizável*

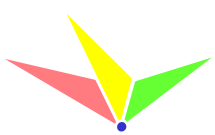
* *Exemplos de código de [Core]*

Conseqüências

- *Promove design modular*
  - *Permite maior reuso e reduz duplicação*
- *Melhora flexibilidade*
  - *Suporta inclusão de dados com base em decisões de tempo de execução*
- *Melhora facilidade de manutenção e gerenciamento*
  - *Separação da página em pedaços menores permite que sejam modificados e mantidos separadamente*
- *Reduz facilidade de gerenciamento*
  - *Possibilidade de erros na apresentação devido à composição incorreta das partes*
- *Impacto na performance*
  - *Inclusões dinâmicas fazem página demorar mais para ser processada*

Exercícios

- *1. Analise a implementação das estratégias de Composite View (código 7.22 a 7.28)*
- *2. Refatore a aplicação em preslayer/cv/ para que utilize CompositeView (os blocos estão identificados com comentários no HTML em messages.jsp). Escolha as melhores estratégias entre Translation-time e Run-time Stategies*
  - *a) Qual a melhor estratégia para o navbar (raramente muda)?*
  - *b) E para o bloco principal?*
- *3. Implemente o menu usando Custom Tag View Management Strategy*
- *4. Implemente o bloco de mensagens usando JavaBean View Management Strategy (já está implementado)*

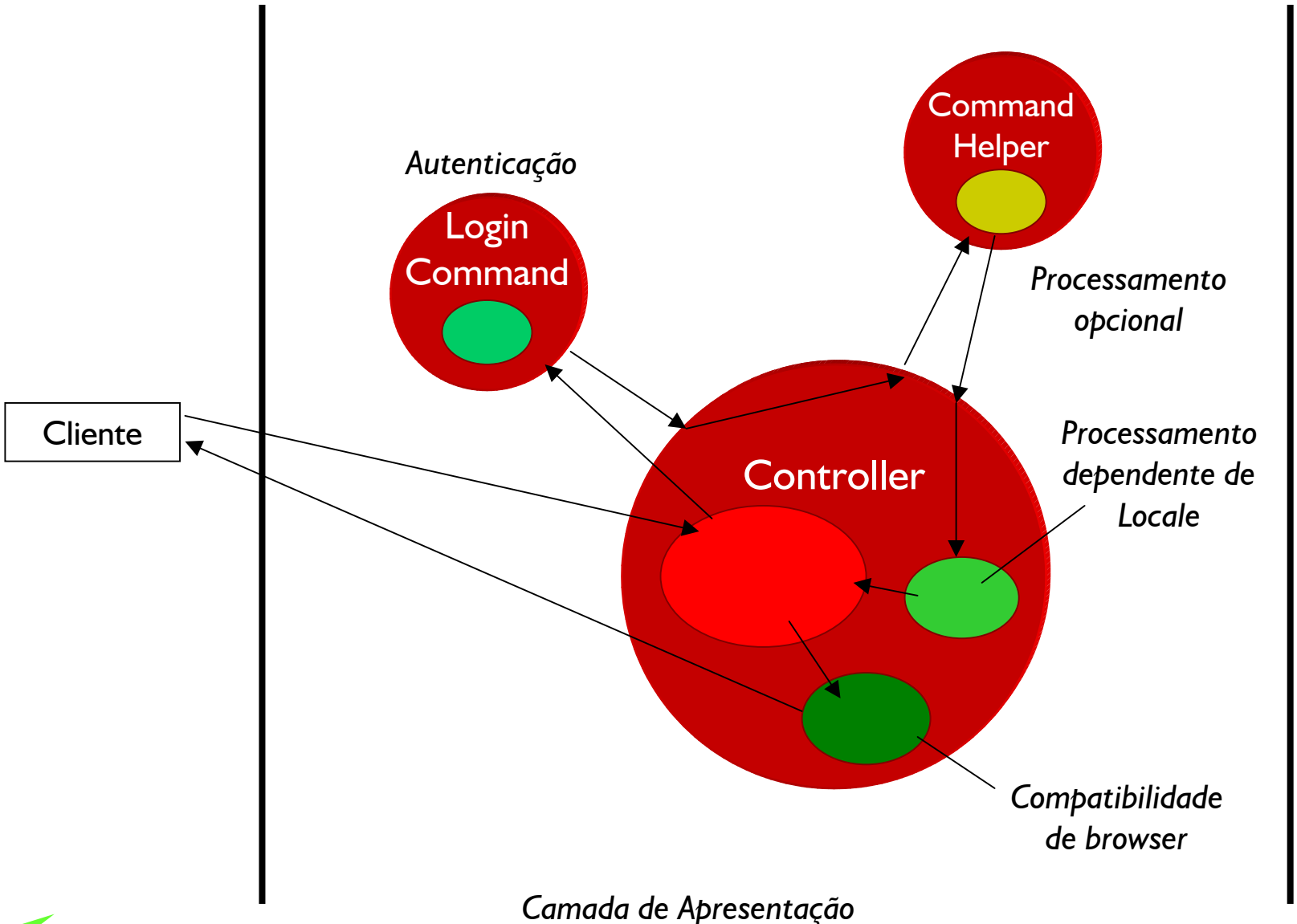


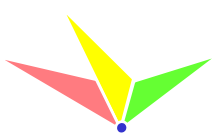
4

Intercepting Filter

Objetivo: permitir o pré- e pós processamento de uma requisição. Intercepting Filter permite encaixar filtros decoradores sobre a requisição ou resposta e remover código de transformação da requisição do controlador

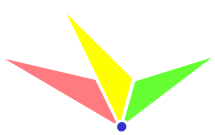
Problema


Command
Helper
Autenticação
Login
Command
Processamento
opcional
Cliente
Controller
Processamento
dependente de
Locale
Compatibilidade
de browser
Camada de Apresentação

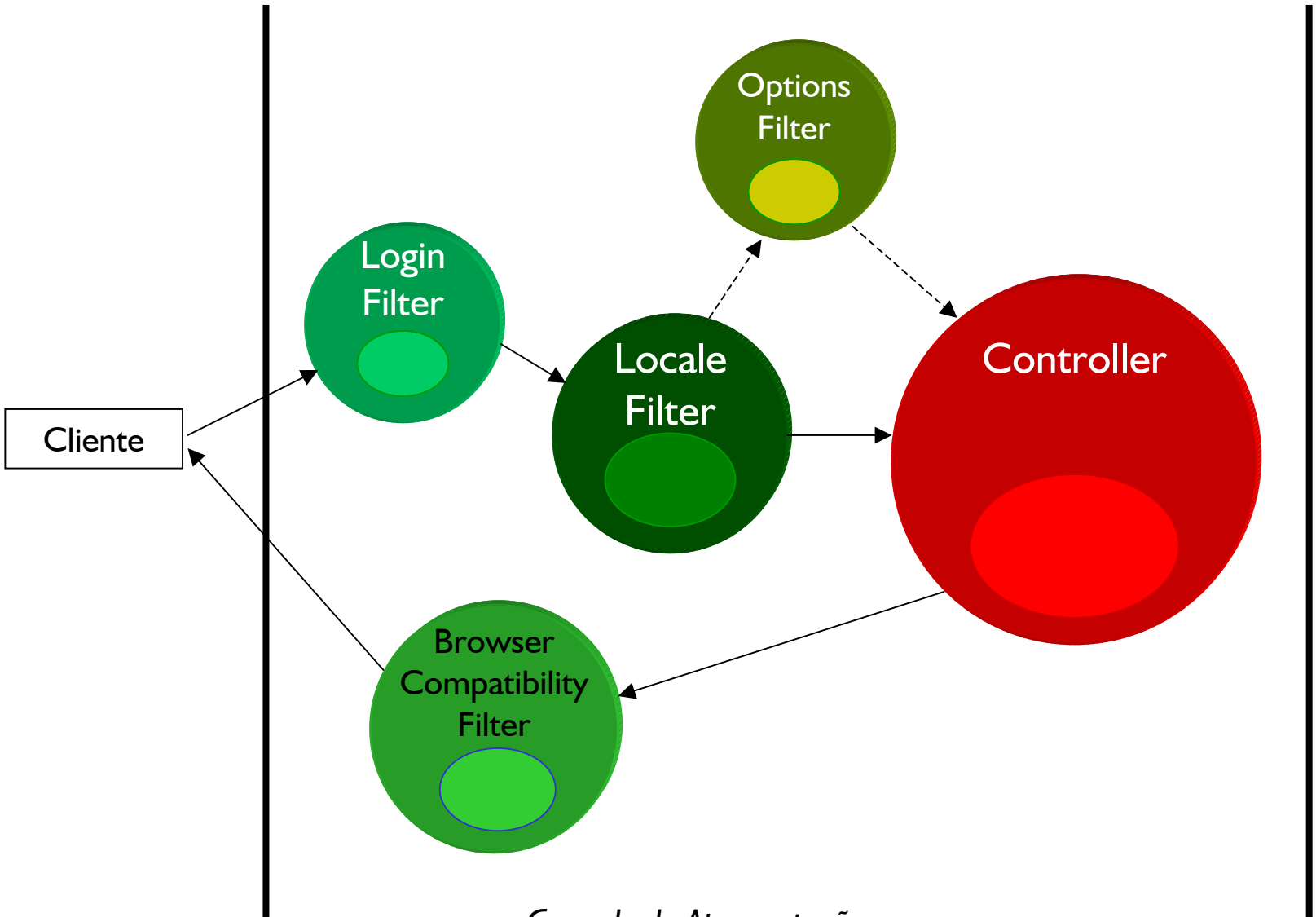


Descrição do problema

- *A camada de apresentação recebe vários diferentes tipos de requisições, que requerem processamento diferenciado*
- *No recebimento de uma requisição, várias decisões precisam ser tomadas para selecionar a forma de realização do processamento*
  - *Isto pode ser feito diretamente no controlador via estruturas if/else. Desvantagem: embute fluxo da filtragem no código compilado, dificultando a sua remoção ou adição*
  - *Incluir tratamento de serviços no próprio controlador impede que esse código possa ser reutilizado em outros contextos*



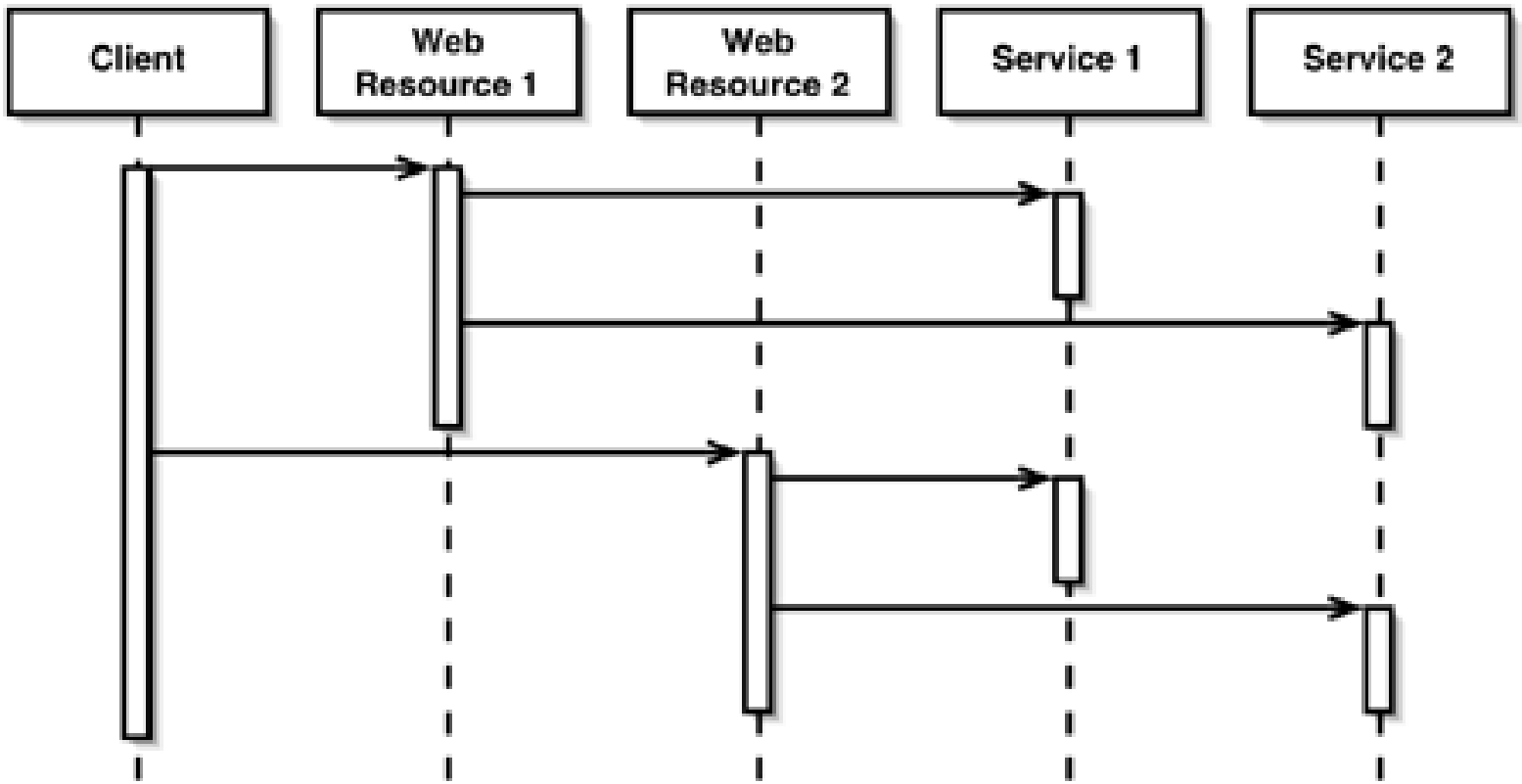
Solução: Intercepting Filter



Descrição da solução

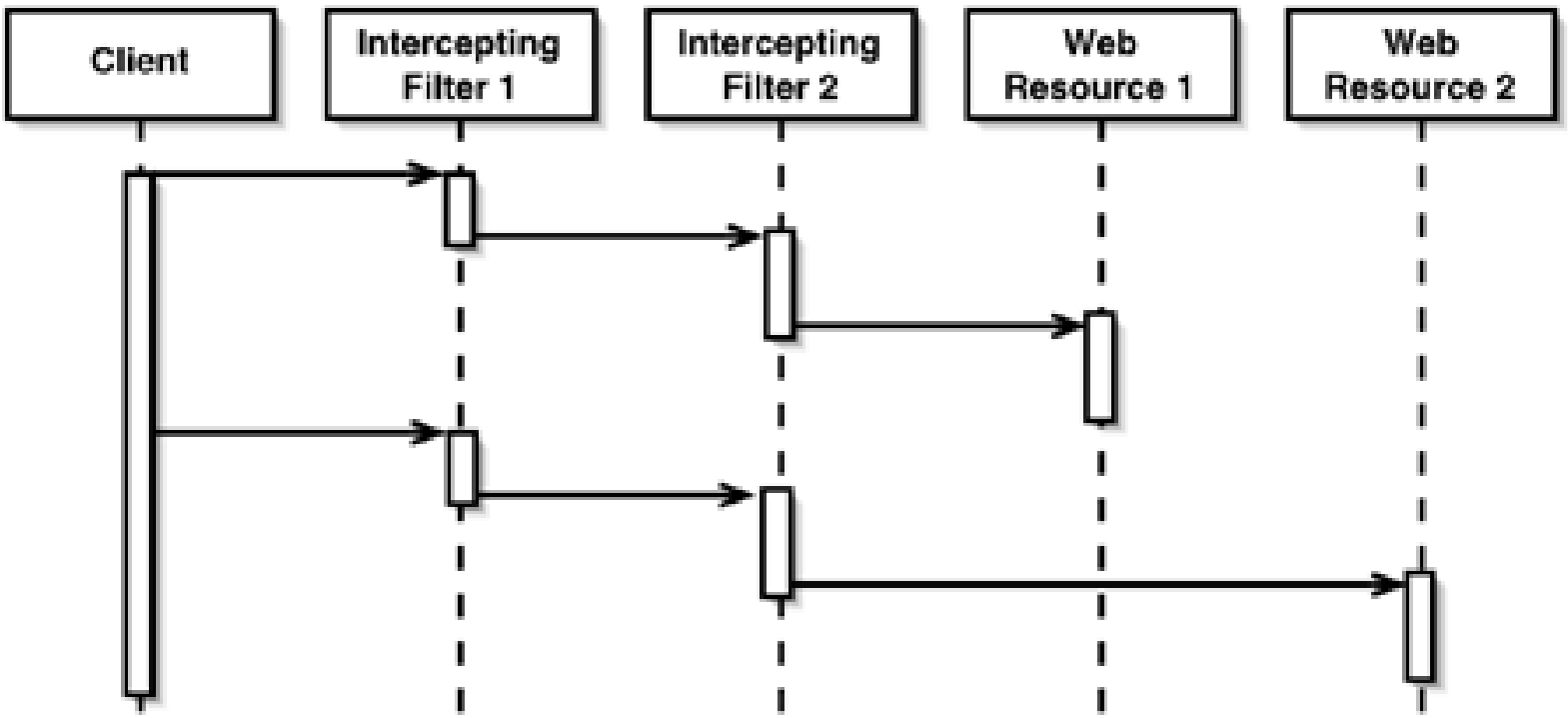
- *Criar filtros plugáveis para processar serviços comuns de forma padrão, sem requerer mudanças no código de processamento*
  - *Filtros interceptam requisições entrantes e respostas, viabilizando pré- e pós-processamento*
  - *Filtros podem ser incluídos dinamicamente e sua composição pode ser alterada*
  - *Filtros são uma estrutura implementada na API Servlet 2.3*

Exemplo de problema



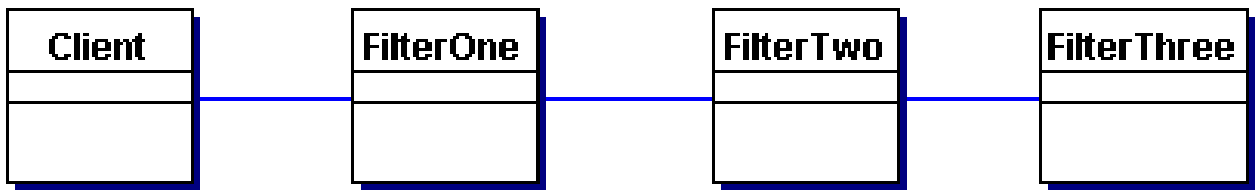
Fonte: [Blueprints]

Exemplo de solução



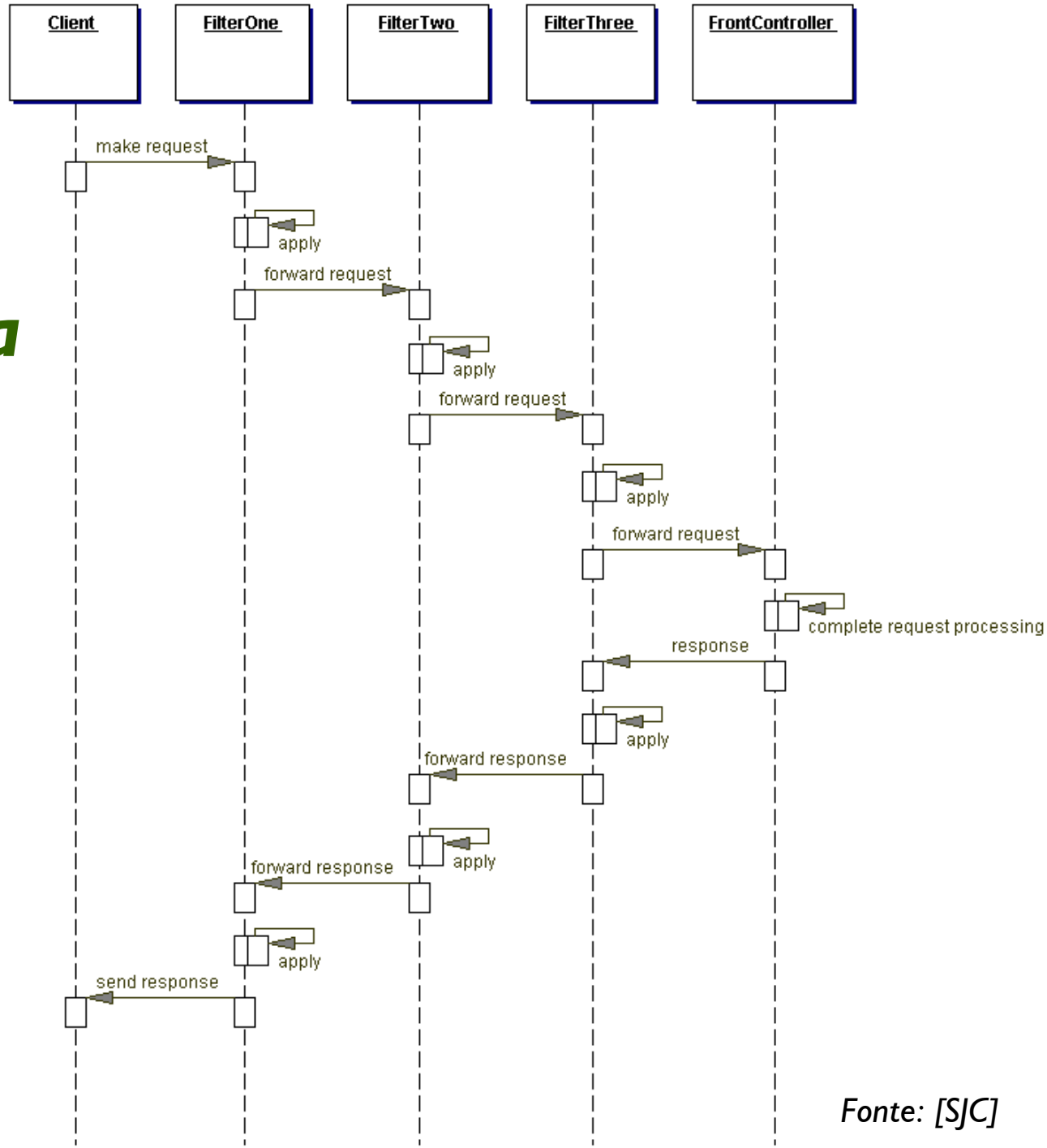
Fonte: [Blueprints]

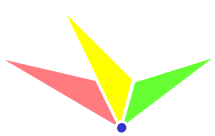
Estrutura UML (I)



Fonte: [SJC]

Diagramas de Seqüência (I)

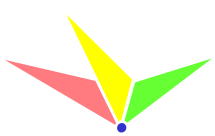




Fonte: [SJC]

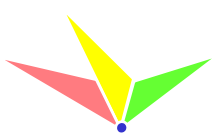
Melhores estratégias de implementação

- *Intercepting Filter é implementado na API Servlet 2.3 através da classe Filter*
  - *As melhores estratégias utilizam a classe Filter*
- *Declared Filter Strategy (Exemplo_1)*
  - *Estratégia que oferece maior flexibilidade*
  - *Filtros são conectados ao fluxo de controle através de declarações no deployment descriptor*
- *Dynamic Filter Strategy (Exemplo_2)*
  - *Implementação desenvolvida pelo programador usando o padrão Decorator [GoF]*



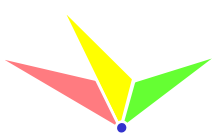
Conseqüências

- *Centraliza controle com processadores fracamente acoplados*
  - *Como um controlador, fornecem um ponto centralizado para processamento de requisições*
  - *Podem ser removidos, adicionados, combinados em cascata*
- *Melhora reuso*
  - *Filtros são destacados do controlador e podem ser usados em outros contextos*
- *Configuração declarativa e flexível*
  - *Serviços podem ser reorganizados sem recompilação*
- *Compartilhamento ineficiente de informações*
  - *Se for necessário compartilhar informações entre filtros, esta solução não é recomendada*



Exercícios

- *1. Analise a implementação das estratégias de Intercepting Filter (código exemplos_if.txt)*
- *2. Refatore a aplicação em preslayer/if/ para que utilize Intercepting Filter. Use Declared Filter Strategy:*
  - *a) A página login.jsp é chamada se o LoginBean for null. Implemente esta funcionalidade usando um filtro*
  - *b) Implemente um filtro que coloque os parâmetros de entrada em caixa-alta*
  - *c) Experimente com composição de filtros no deployment descriptor*

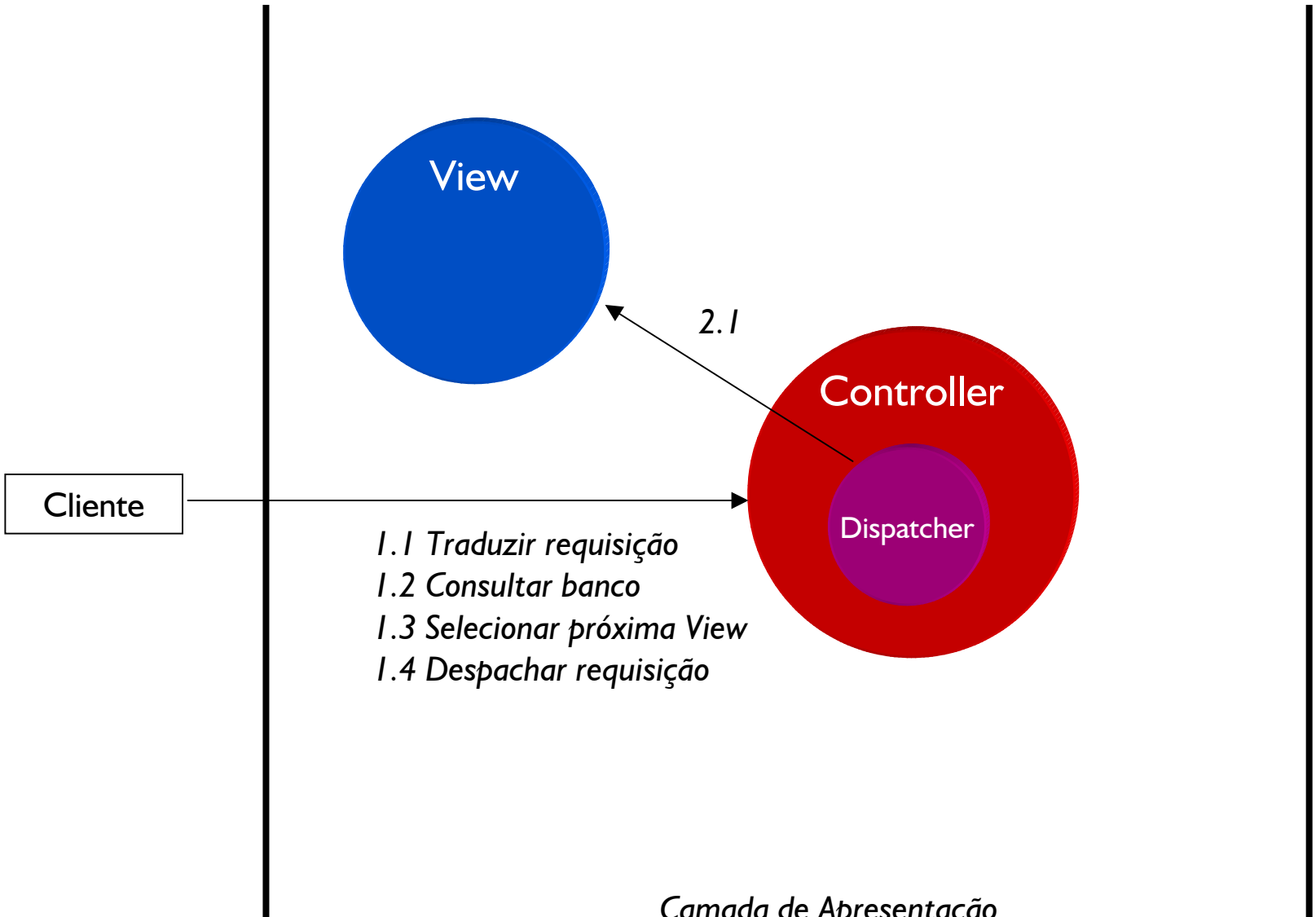


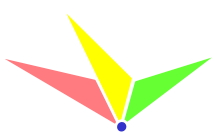
5

Service To Worker

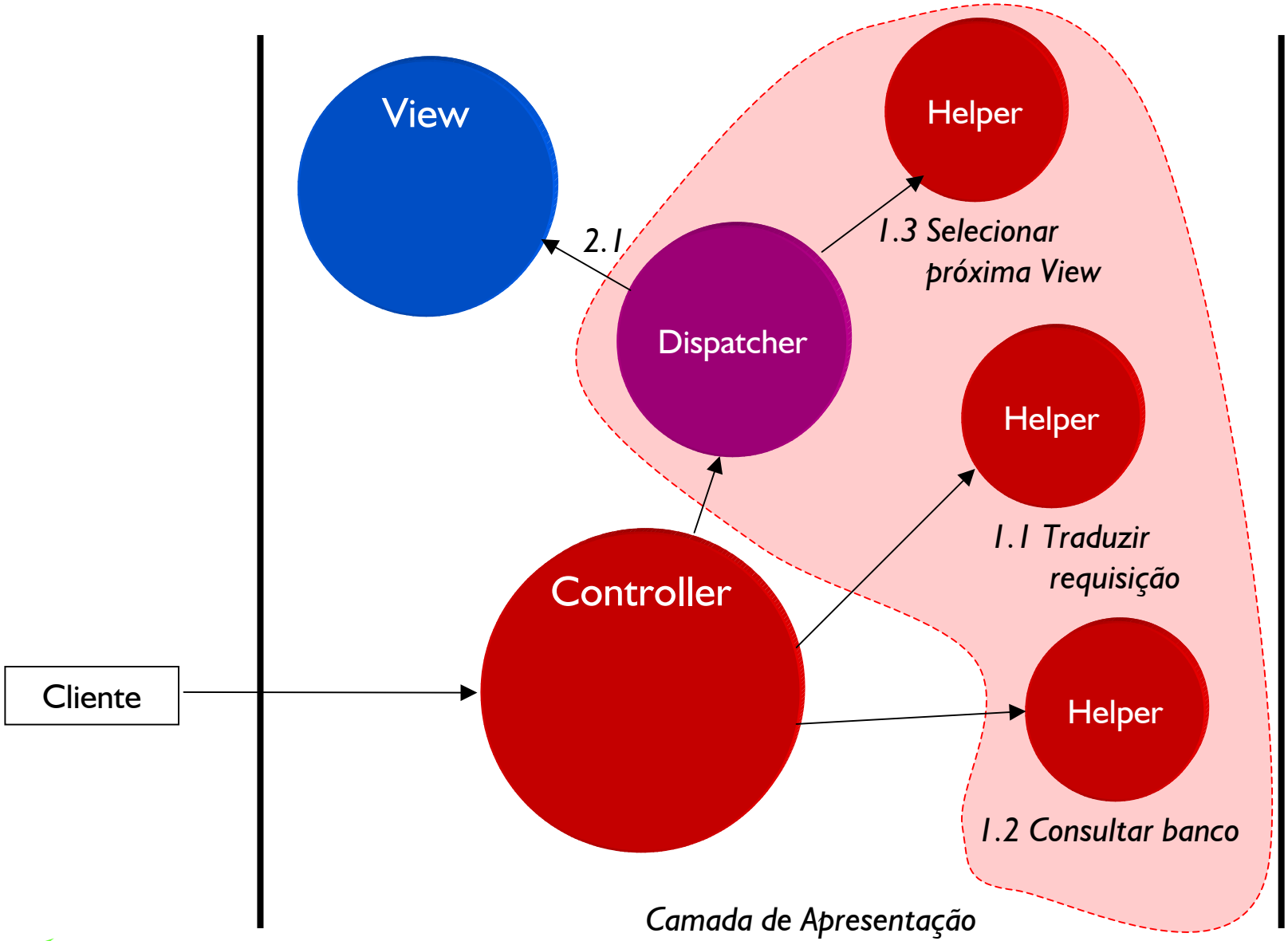
Objetivo: combinar padrões de apresentação e encapsular lógica de navegação, que consiste em escolher uma View e despachar a requisição para ela. Service To Worker realiza mais processamento antes de despachar a requisição.

Problema


View
2.1
Controller
Dispatcher
Cliente
1.1 Traduzir requisição
1.2 Consultar banco
1.3 Selecionar próxima View
1.4 Despachar requisição
Camada de Apresentação



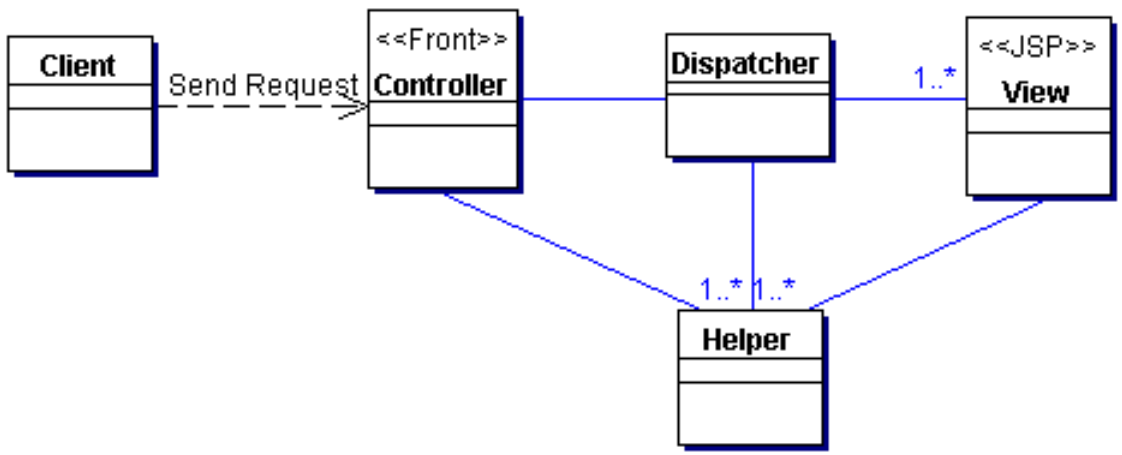
Solução: Service To Worker

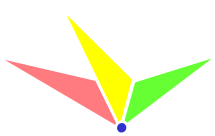


Descrição

- *Service To Worker combina Front Controller e View Helper com o objetivo de separar a lógica que tem responsabilidades diferentes*
  - *Maior parte das responsabilidades estão acumuladas entre os controladores e Dispatcher*
  - *Recuperação do conteúdo necessário para compor a View é obtido antes de despachar a requisição*

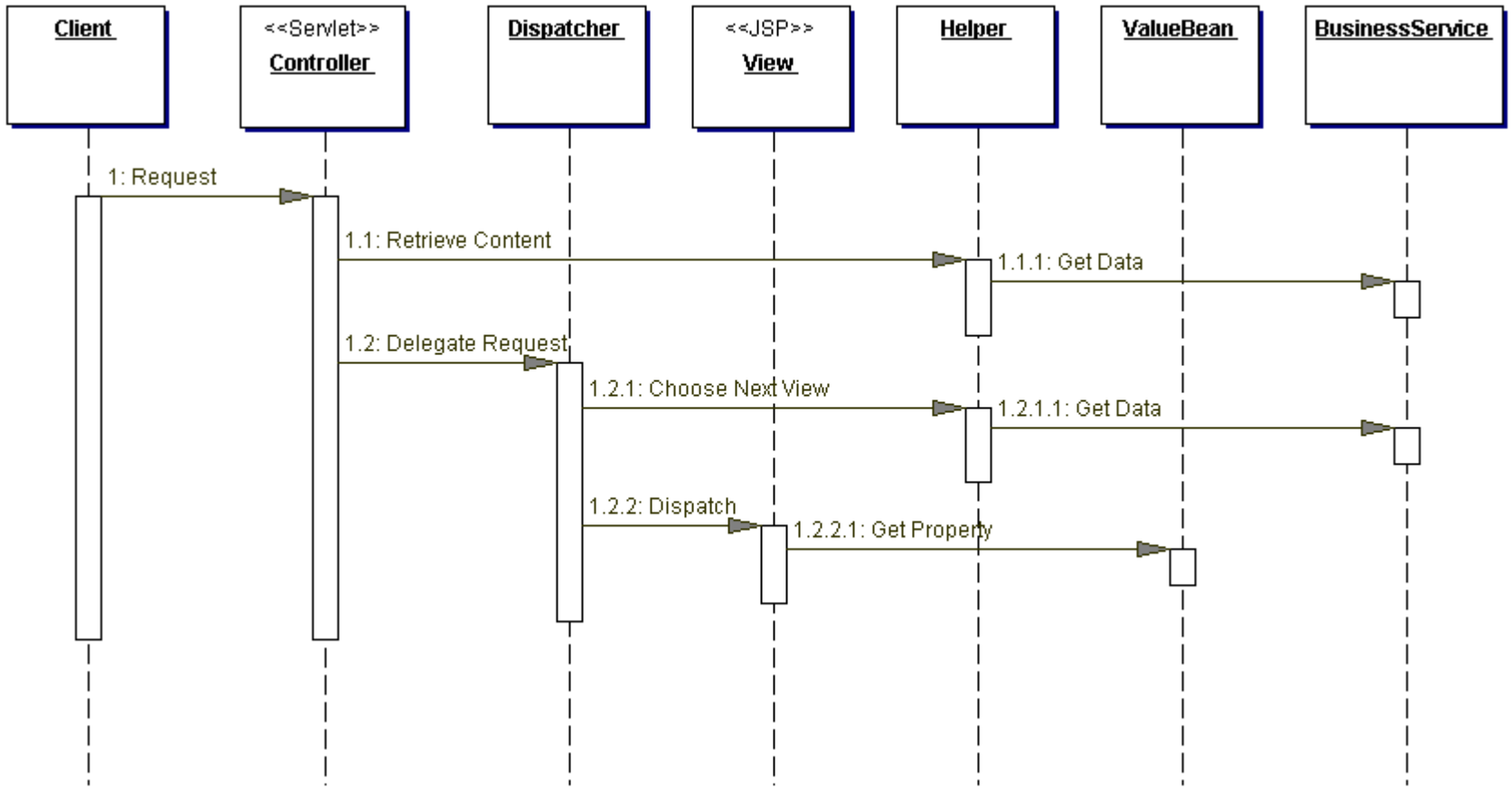
Estrutura UML (1)

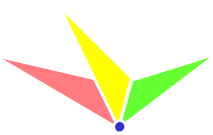




Fonte: [1]

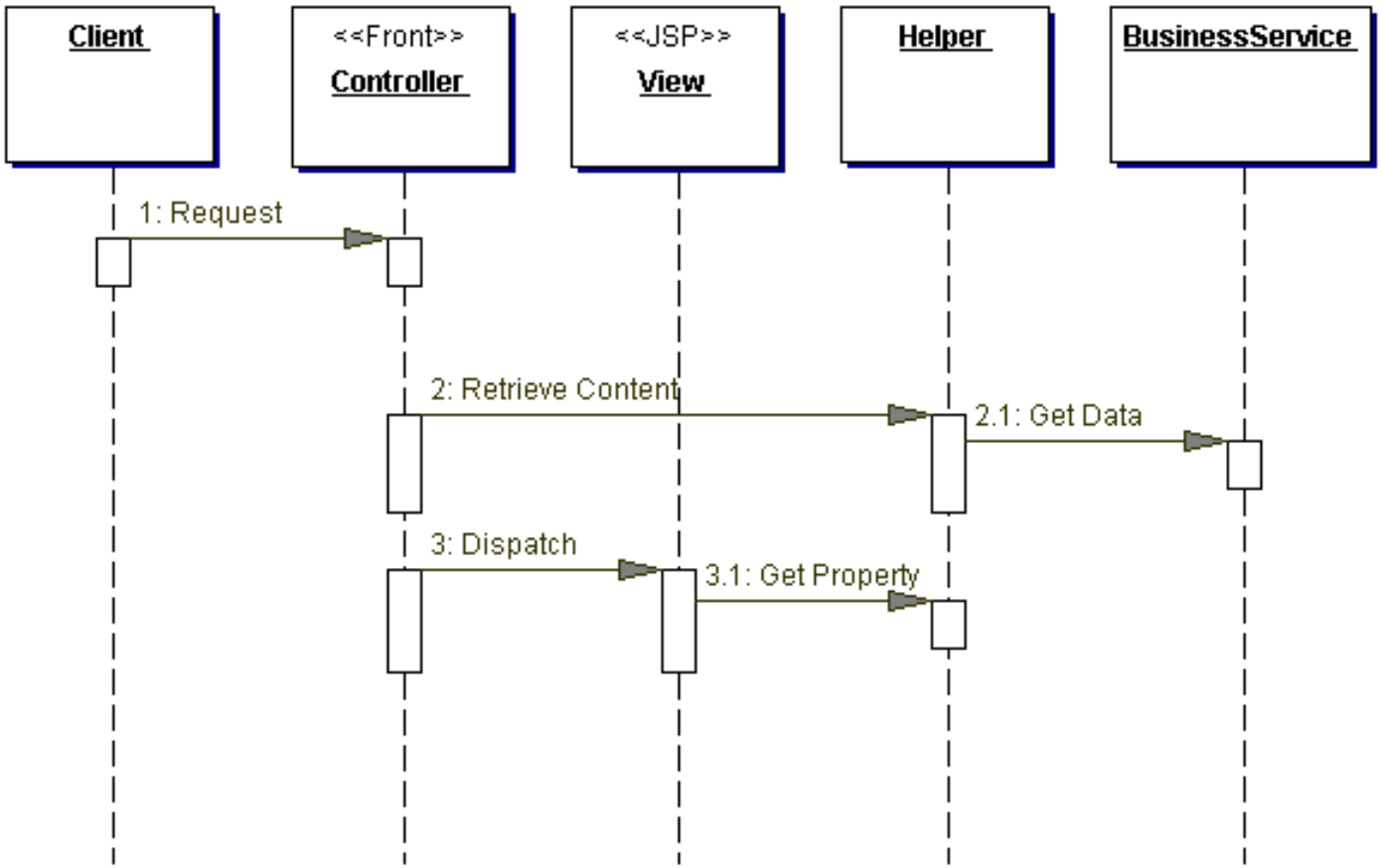
Diagramas de Seqüência





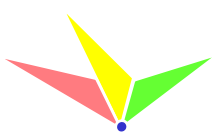
Fonte: [1]

Dispatcher in Controller Strategy



Exercícios

- *1. Analise a implementação das estratégias de Service to Worker (código 7.29 a 7.33)*
- *2. Refatore a aplicação em preslayer/fc/ para que utilize Service to Worker:*
  - *a) Use Dispatcher in Controller Strategy*
  - *b) Distribua o controle em Comandos (veja exercício do FrontController)*
  - *b) Use getCommand() no Business Service para traduzir a requisição em um comando*

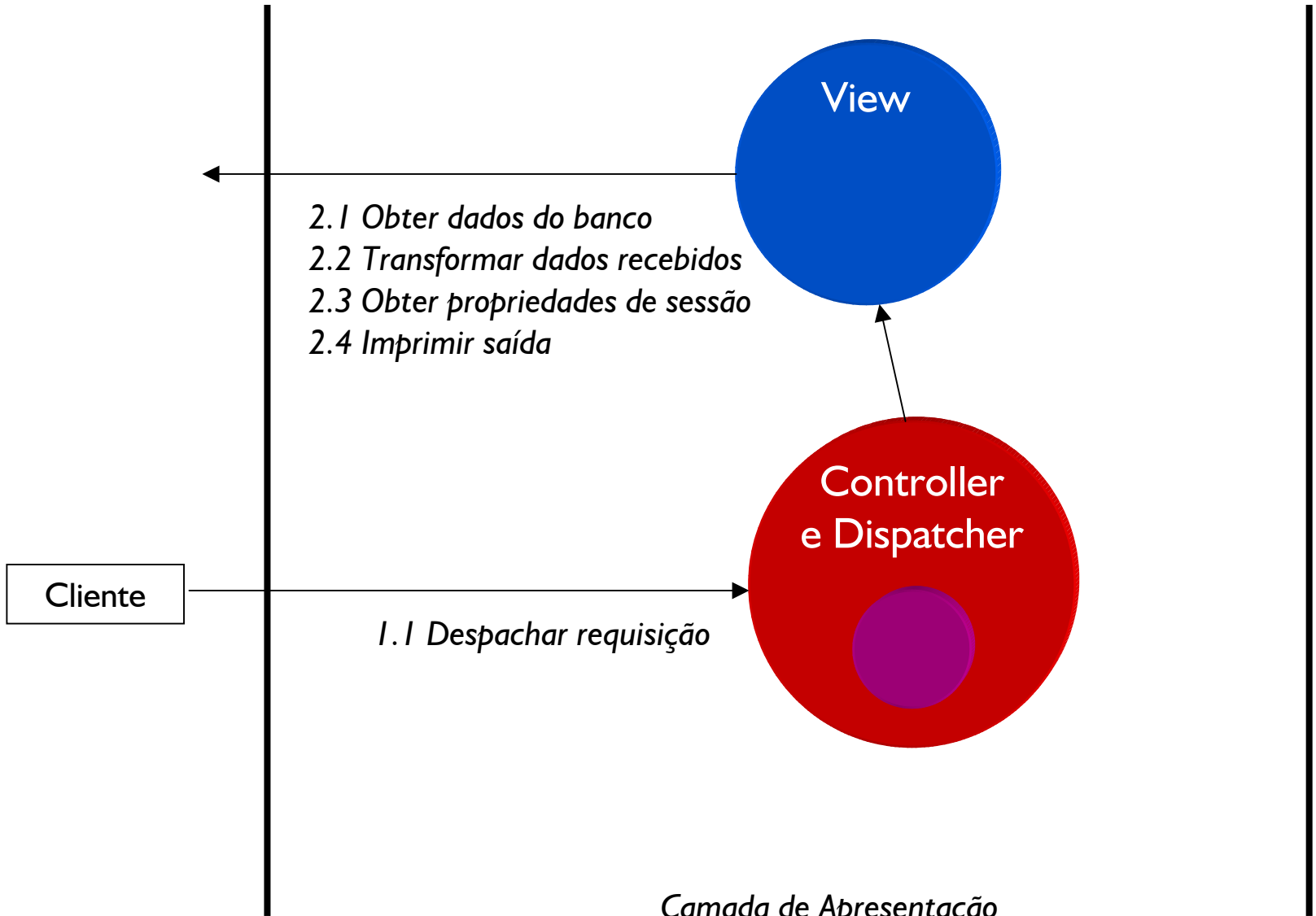


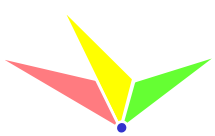
6

Dispatcher View

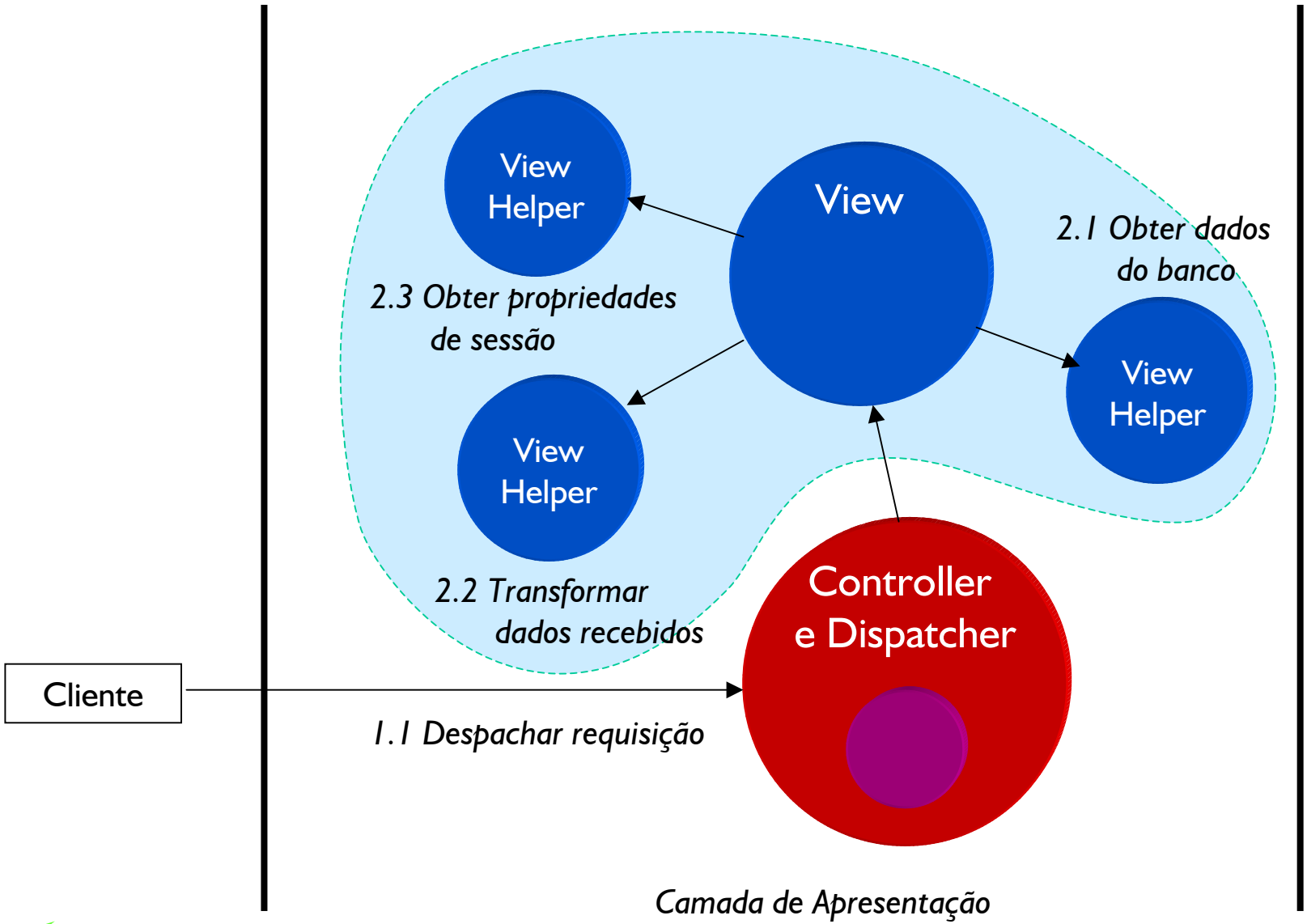
Objetivo: combinar padrões de apresentação e encapsular lógica de navegação, que consiste em escolher uma View e despachar a requisição para ela. Dispatcher View realiza mais processamento depois de despachar a requisição.

Problema


View
2.1 Obter dados do banco
2.2 Transformar dados recebidos
2.3 Obter propriedades de sessão
2.4 Imprimir saída
Controller
e Dispatcher
Cliente
1.1 Despachar requisição
Camada de Apresentação



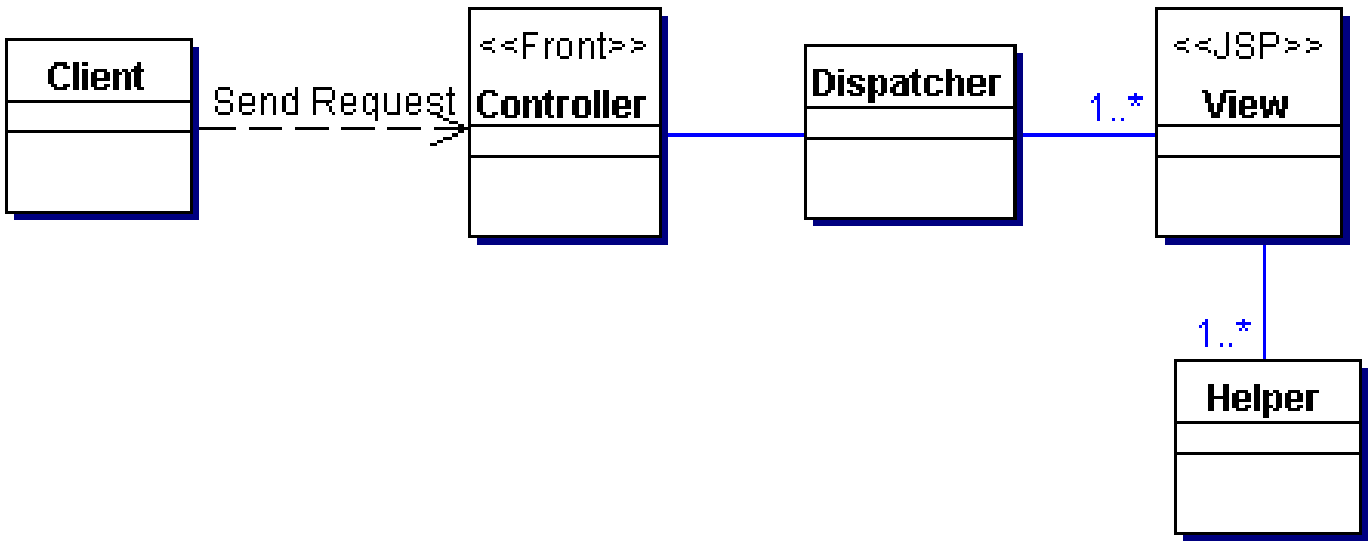
Solução: Dispatcher View



Descrição

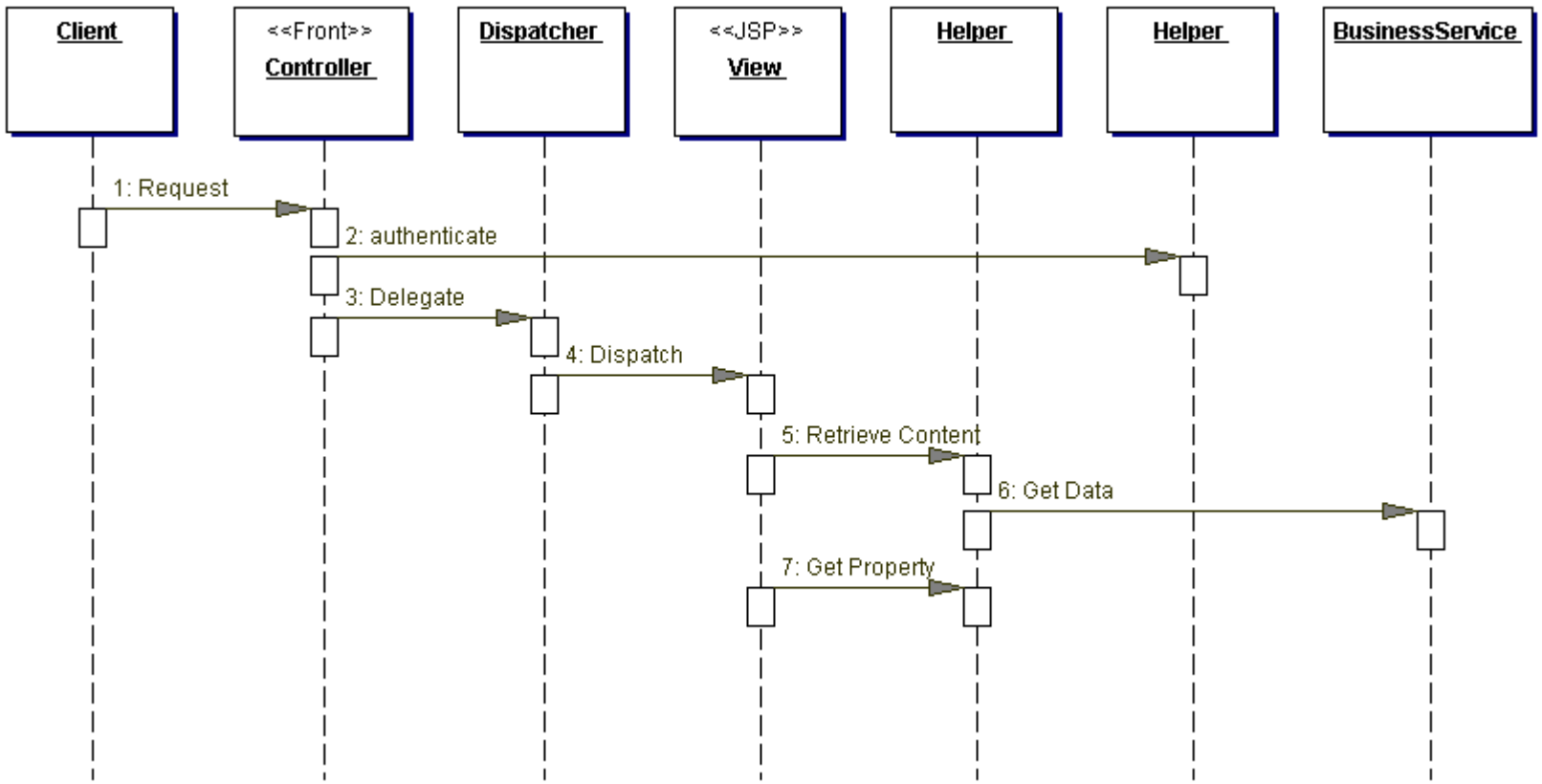
- *Dispatcher View combina Front Controller e View Helper com o objetivo de separar a lógica que tem responsabilidades diferentes*
  - *Maior parte das responsabilidades estão acumuladas entre o Dispatcher e View*
  - *Recuperação do conteúdo necessário para compor a View é obtido após despachar a requisição*

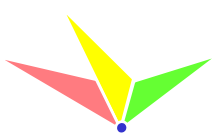
Estrutura UML



Fonte: [1]

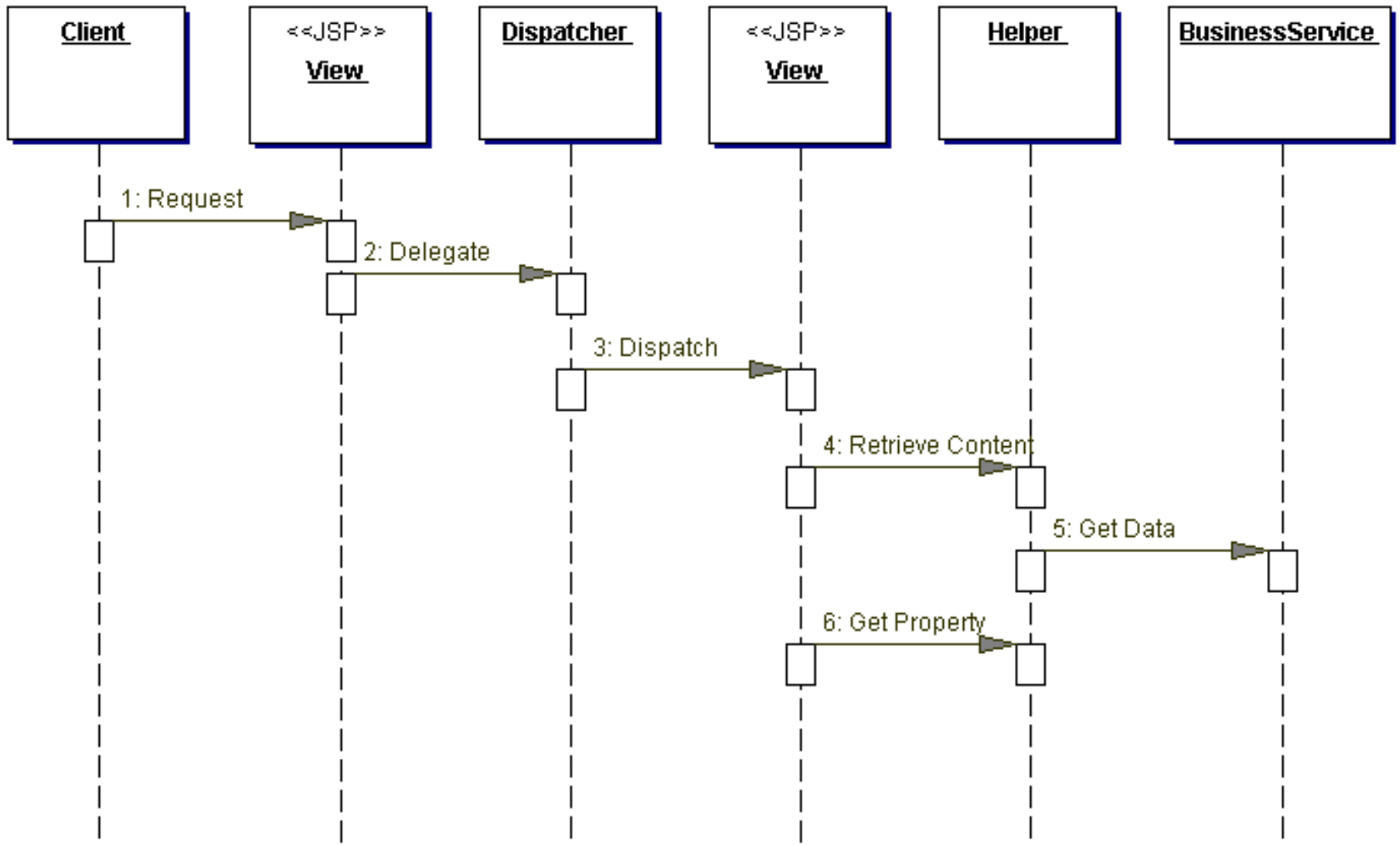
Diagramas de Seqüência

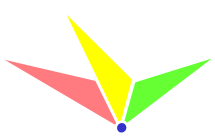




Fonte: [1]

Dispatcher in View Strategy





Exercícios

- *1. Analise a implementação das estratégias de Dispatcher View (código 7.34 a 7.35)*
- *2. Refatore a aplicação em preslayer/fc/ para que utilize Dispatcher View:*
  - *a) Use JavaBean Helper Strategy para converter as mensagens em caixa-alta.*

Fontes

[SJC] SJC Sun Java Center J2EE Patterns Catalog. http://developer.java.sun.com/developer/restricted/patterns/J2EEPatternsAtAGlance.html.

[Blueprints] J2EE Blueprints patterns Catalog. http://java.sun.com/blueprints/patterns/catalog.htm. Contém padrões extras usados na aplicação Pet Store.

[Core] Deepak Alur, John Crupi, Dan Malks. Core J2EE Patterns: Best Practices and Design Strategies. Prentice-Hall, 2001. http://java.sun.com/blueprints/corej2eepatterns/index.html.

Curso J931: J2EE Design Patterns

Versão 1.0

www.argonavis.com.br

(helder@acm.org)