Padrões de Projeto J2EE para Aplicações Web

Helder da Rocha (helder@acm.org)
www.argonavis.com.br

Introdução

- Este módulo aborda os principais padrões de projeto J2EE, dentre o catálogo organizado pelo Sun Java Center (SJC) que são aplicáveis a aplicações Web
  - É um módulo de referência. A abordagem, neste curso, será superficial
- Os padrões representam boas práticas e estratégias de implementação para diversos problemas recorrentes no design de aplicações Web
- Conhecer os padrões ajuda a entender melhor sistemas semelhantes, especialmente frameworks
  - Consulte também os padrões GoF, aplicáveis não só à plataforma J2EE mas a OO

Padrões para Aplicações Web

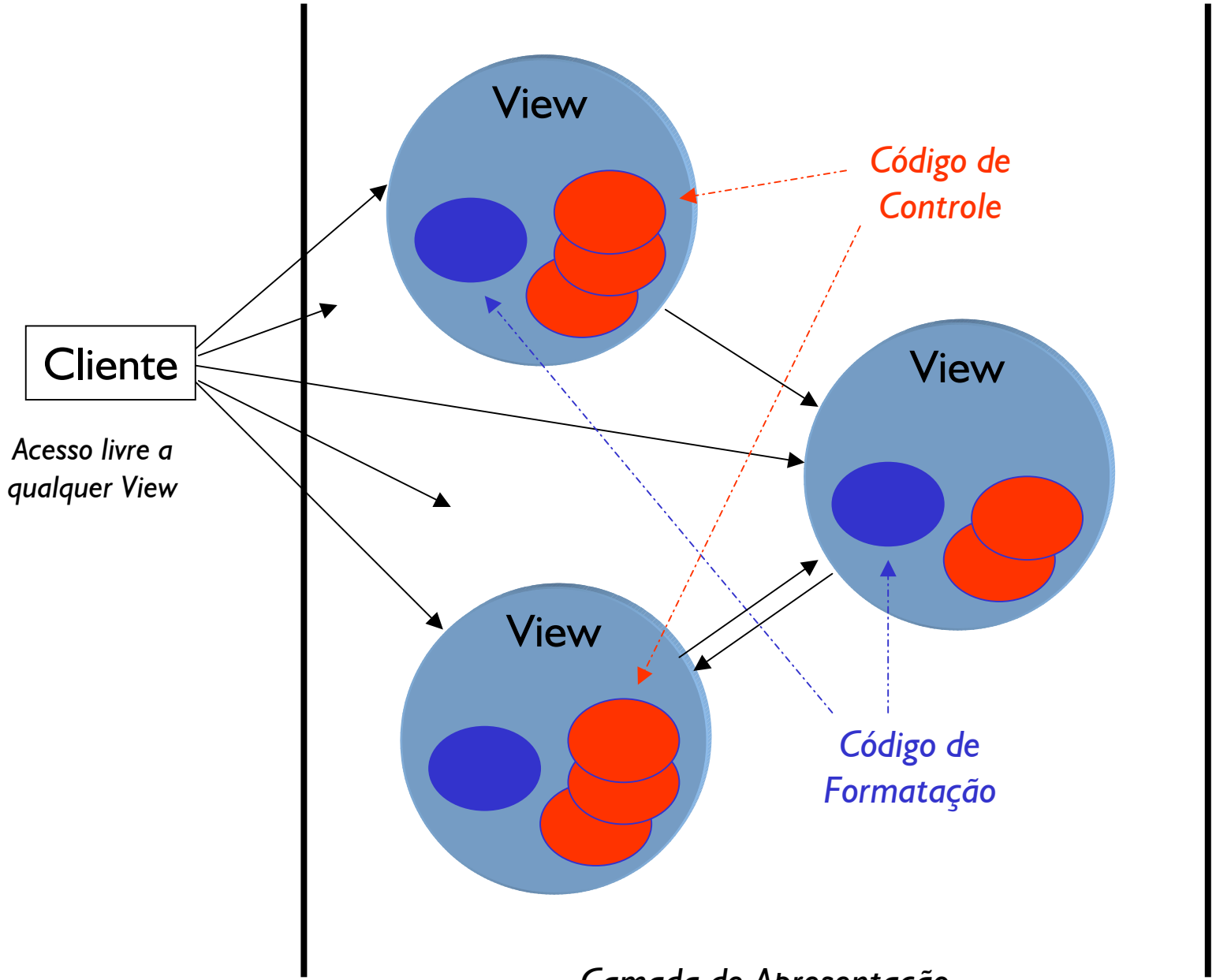
- (1) Front Controller
  - Controlador centralizado para processar de uma requisição
- (2) View Helper
  - Encapsula lógica não-relacionada à formatação
- (3) Composite View
  - Cria uma View composta de componentes menores
- (4) Intercepting Filter
  - Viabiliza pré- e pós-processamento de requisições
- (5) Data Access Object
  - Esconde detalhes do meio de persistência utilizado
- (6) Business Delegate
  - Interface com a camada de negócios
- (7) Transfer Object (ou Value Object)
  - Objeto que é utilizado na comunicação para evitar múltiplas requisições e respostas

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

Solução: Front Controller

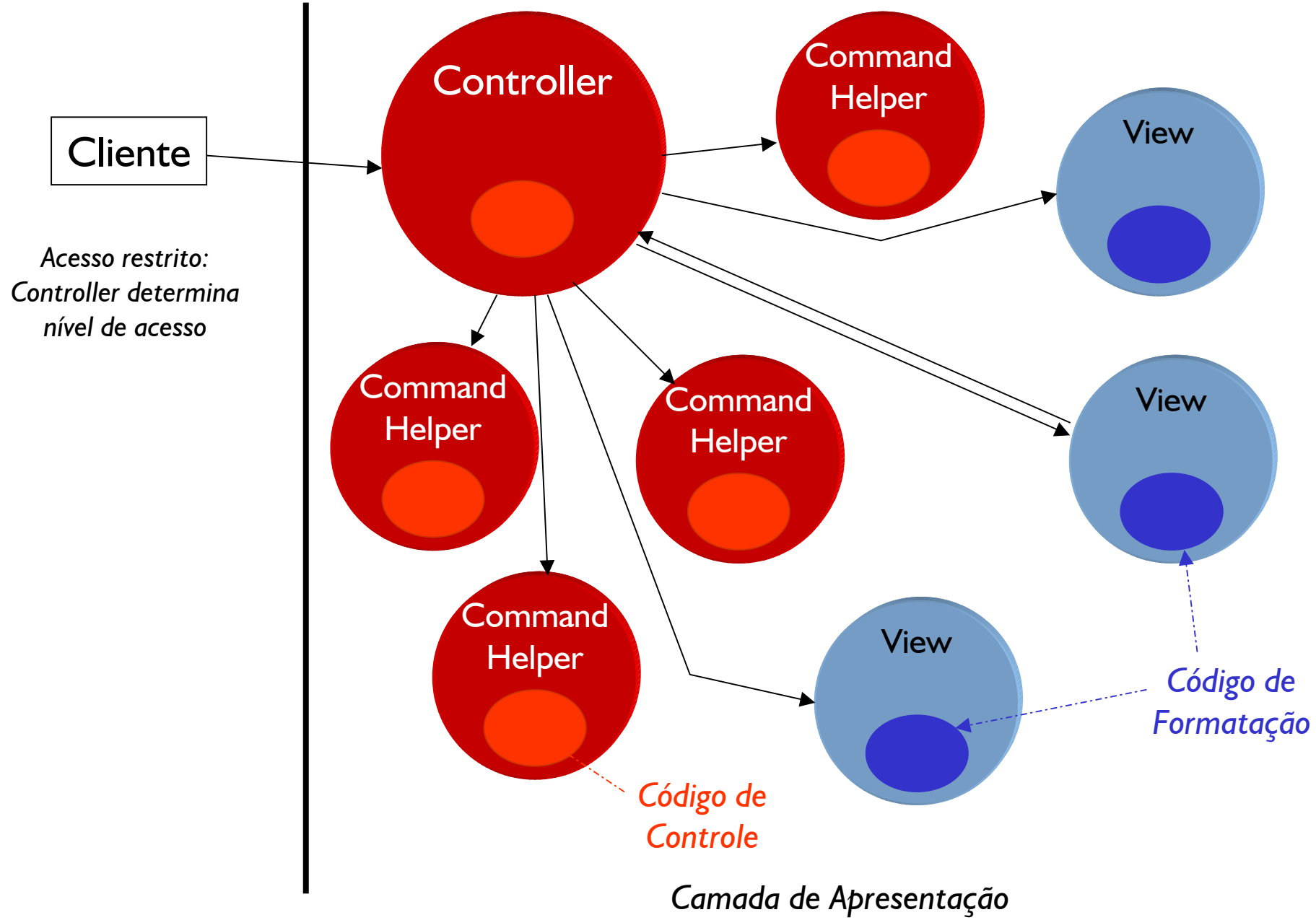
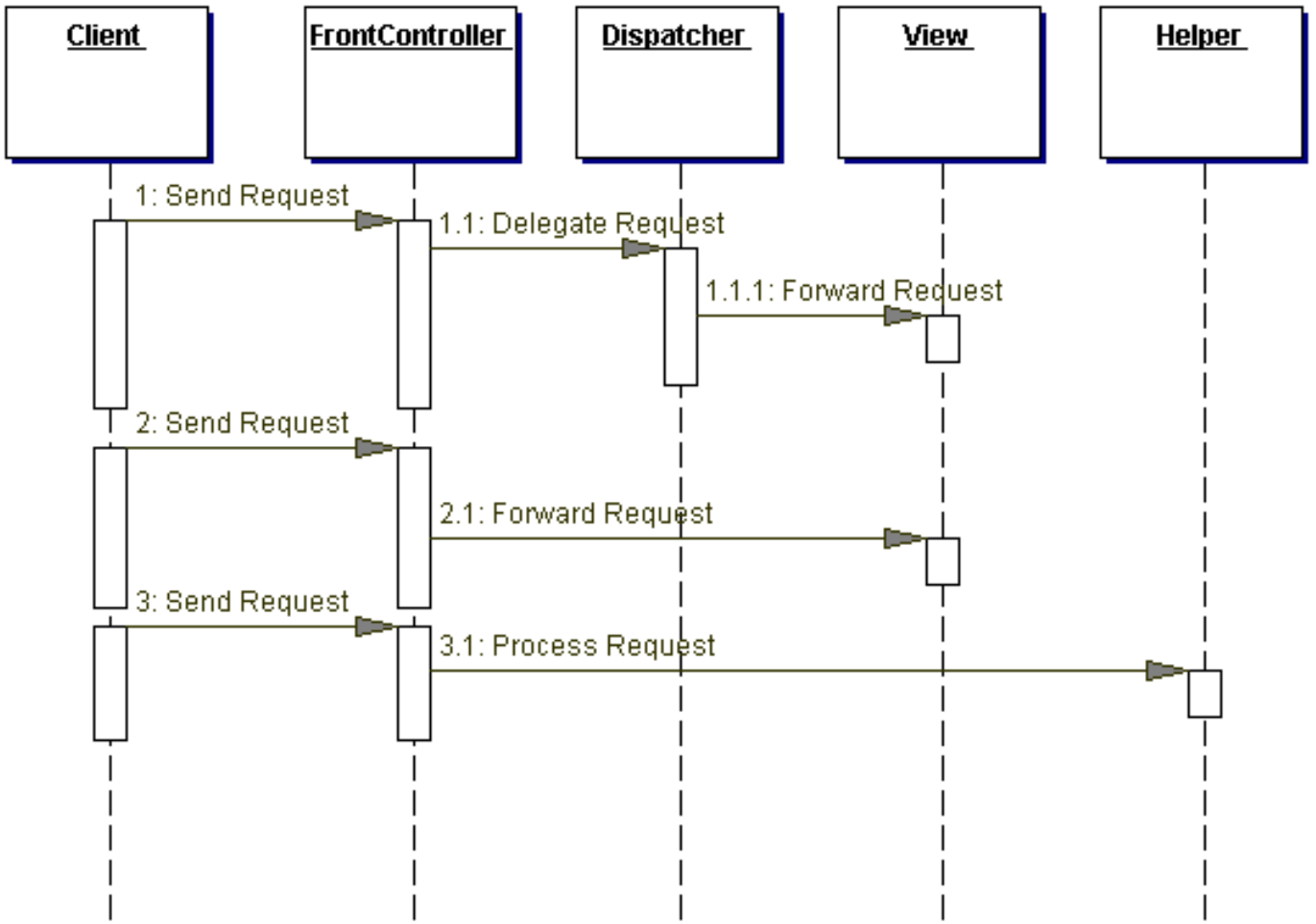


Diagrama de Seqüência




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

Fonte: [SJC]

Participantes e responsabilidades

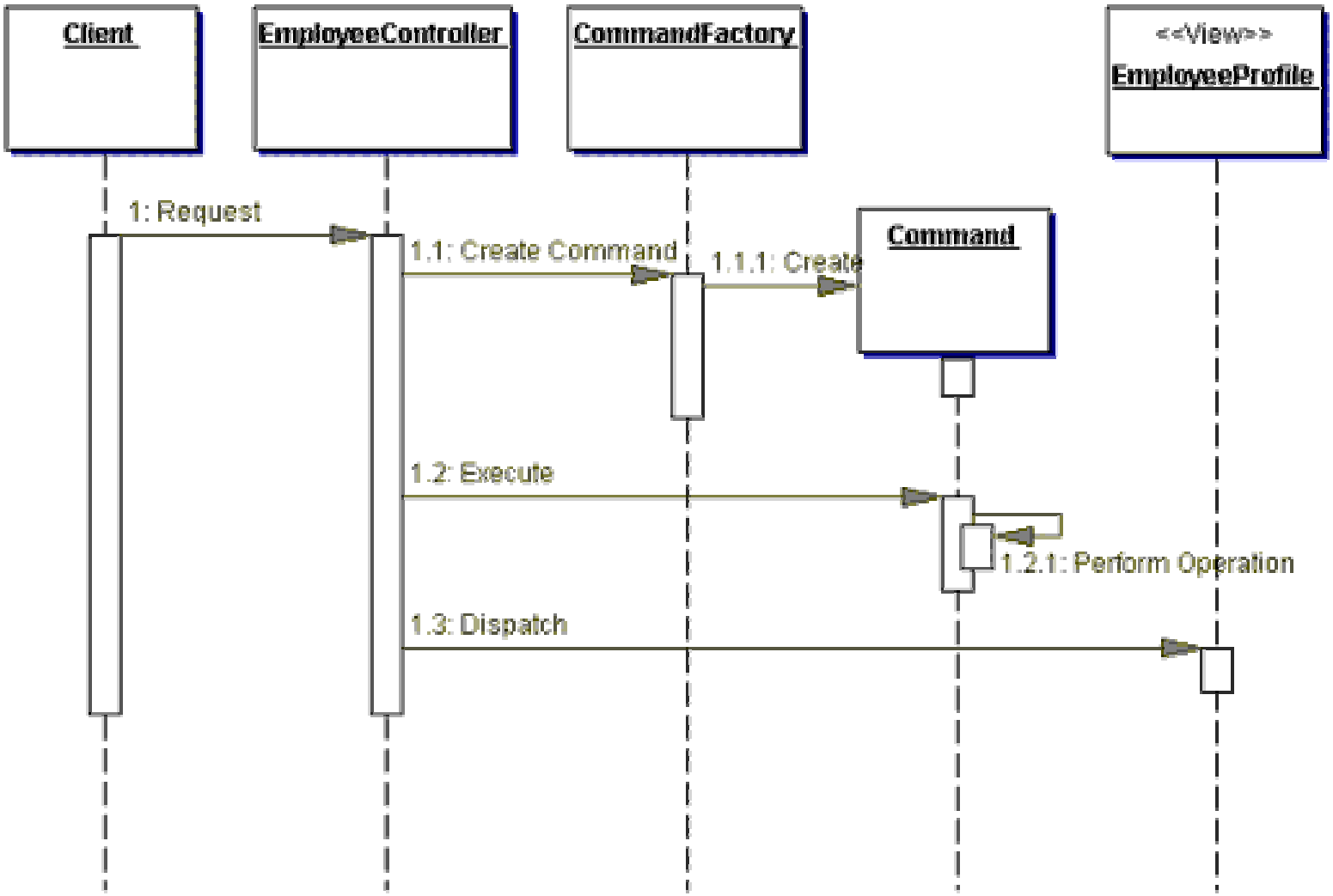
- *Controller*
  - *Ponto de entrada que centraliza todas as requisições*
  - *Pode delegar responsabilidade a Helpers*
- *Dispatcher*
  - *Tipicamente usa ou encapsula objeto javax.servlet.RequestDispatcher*
- *Helper*
  - *Pode ter inúmeras responsabilidades, incluindo a obtenção de dados requerido 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.
  - 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.
- 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

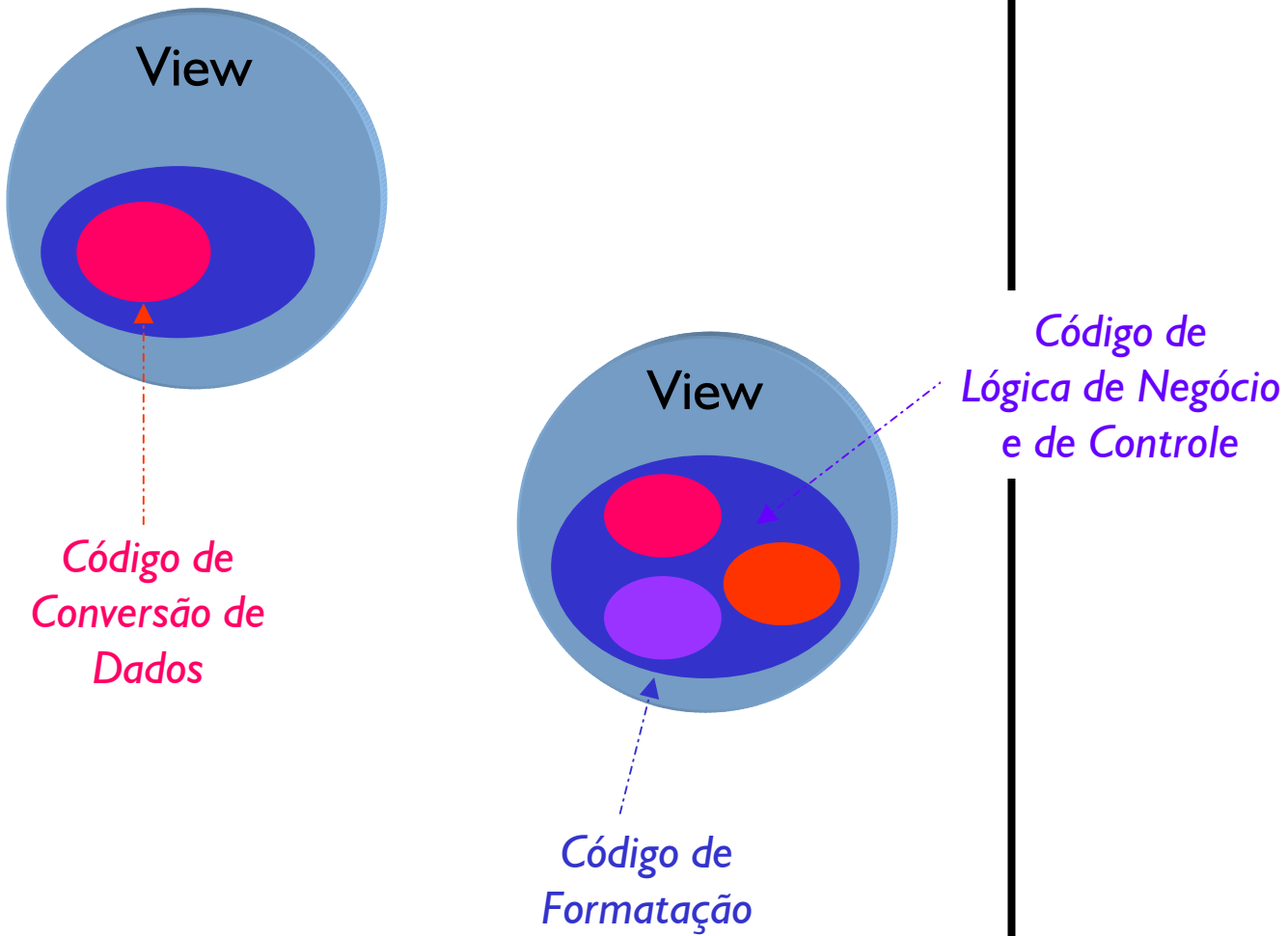
- *1. Altere a aplicação em cap13/fc/ para que utilize FrontController. Empregue as três estratégias de implementação apresentadas:*
  - *a) Implemente o controlador usando um Servlet*
  - *b) Escreva um RequestHelper que mantenha uma tabela de comandos/nomes de classe de objetos Command e receba um 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*

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


View
Código de
Conversão de
Dados
View
Código de
Lógica de Negócio
e de Controle
Código de
Formatação

Solução: View Helpers

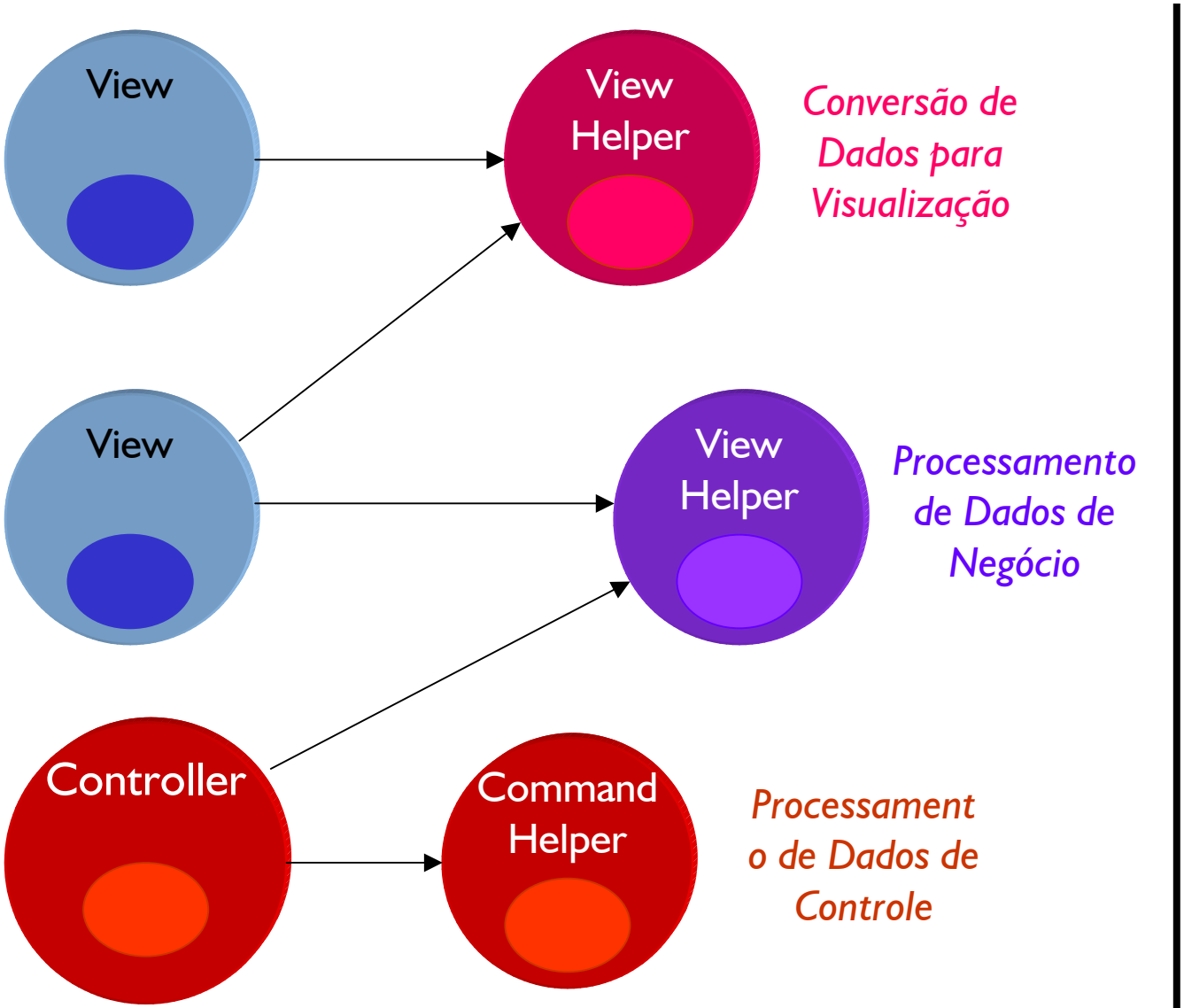
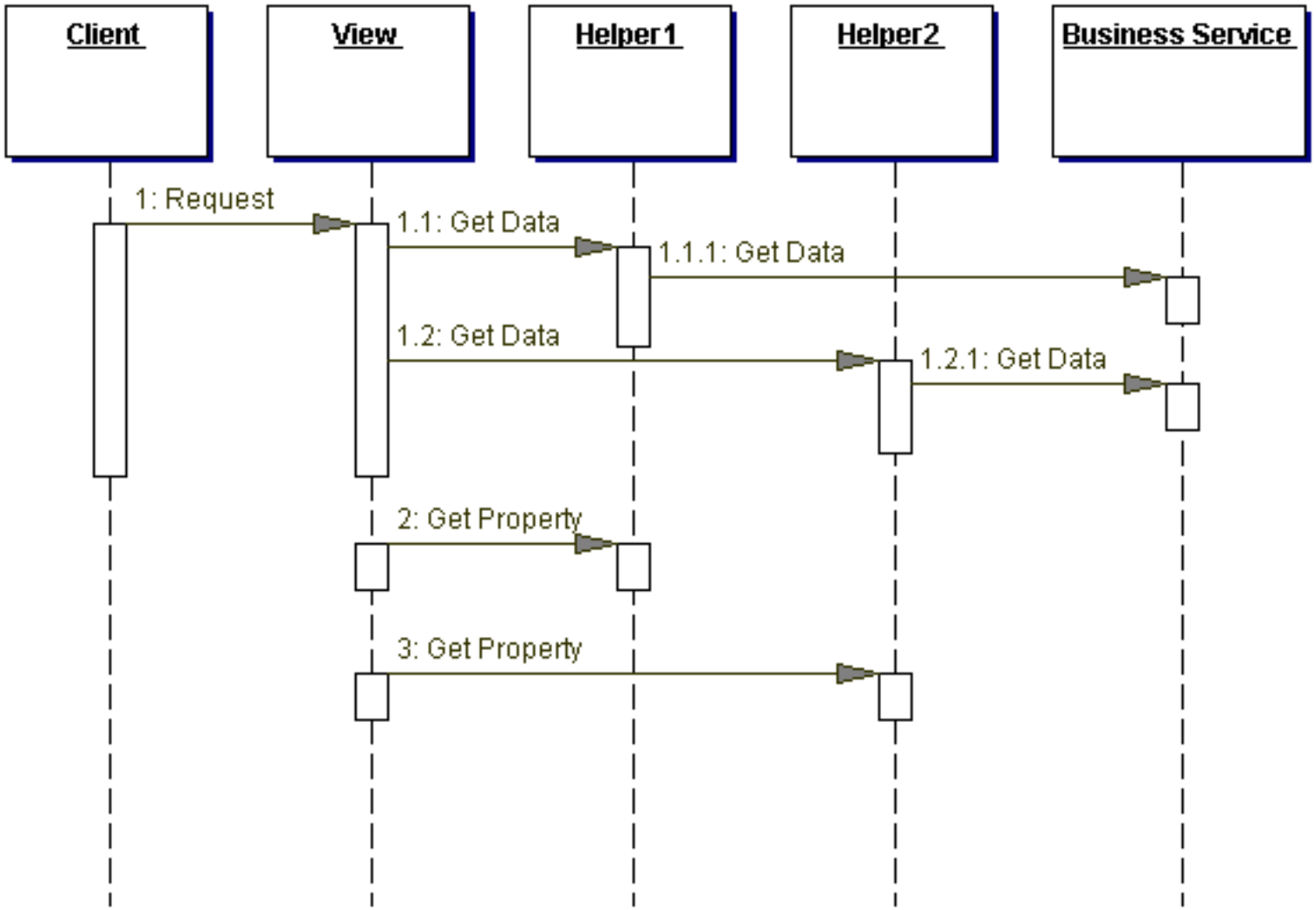


Diagrama de Seqüência



Fonte: [SJC]

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

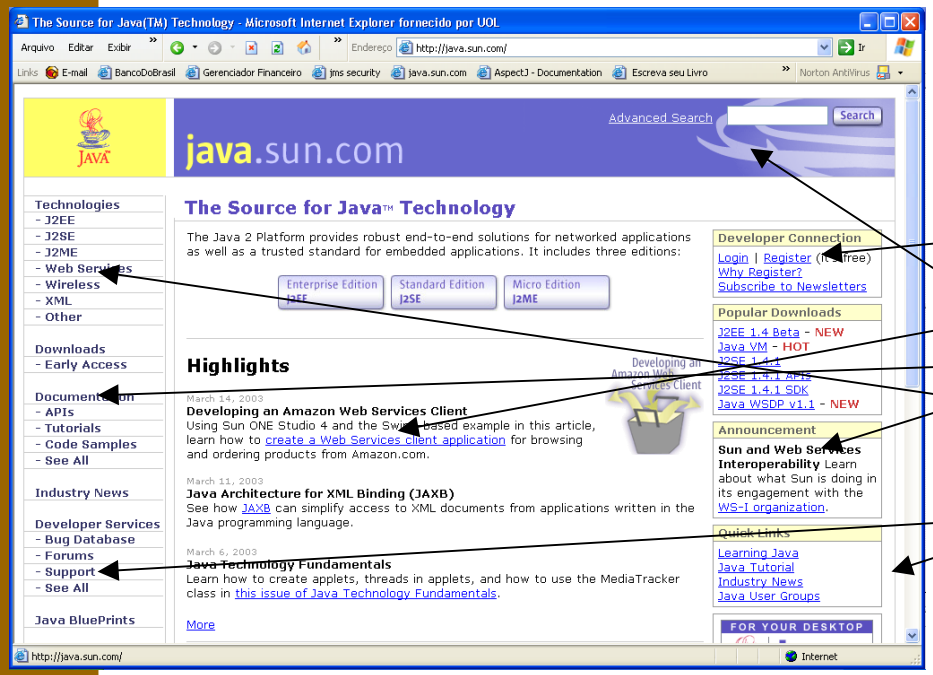
- 2. Altere a aplicação em cap13/vh/ para que utilize ViewHelper. Use JavaBean Helper Strategy:
  - a) Identifique código de conversão de formatos, código de negócio e código de controle (se houver)
  - b) Construa um Helper para cada responsabilidade encontrada
- 3. Use Custom Tag Helper Strategy para encapsular a lógica de repetição
  - Use tags do Struts <logic:iterate> ou JSTL <c:forEach>

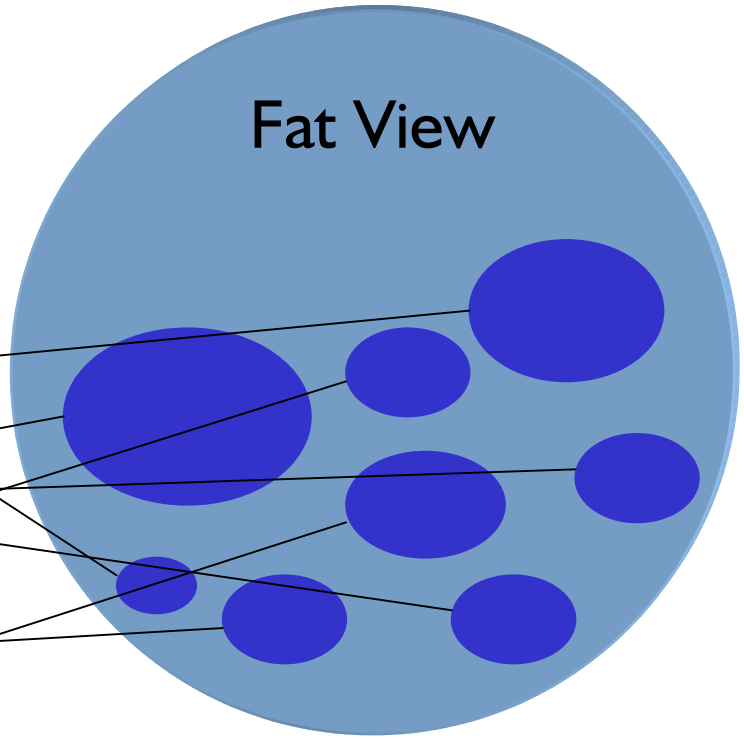
3

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.

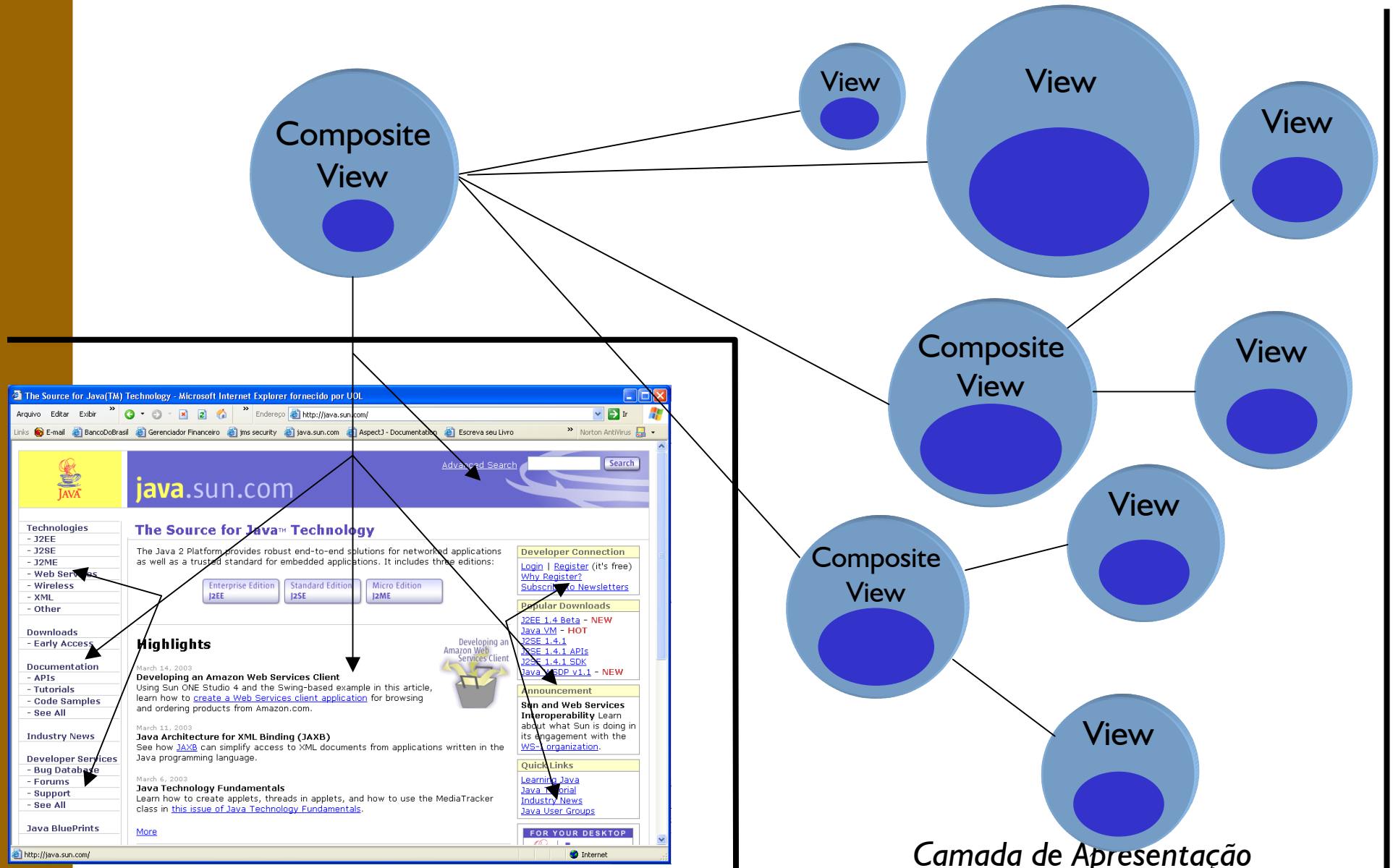
Problema



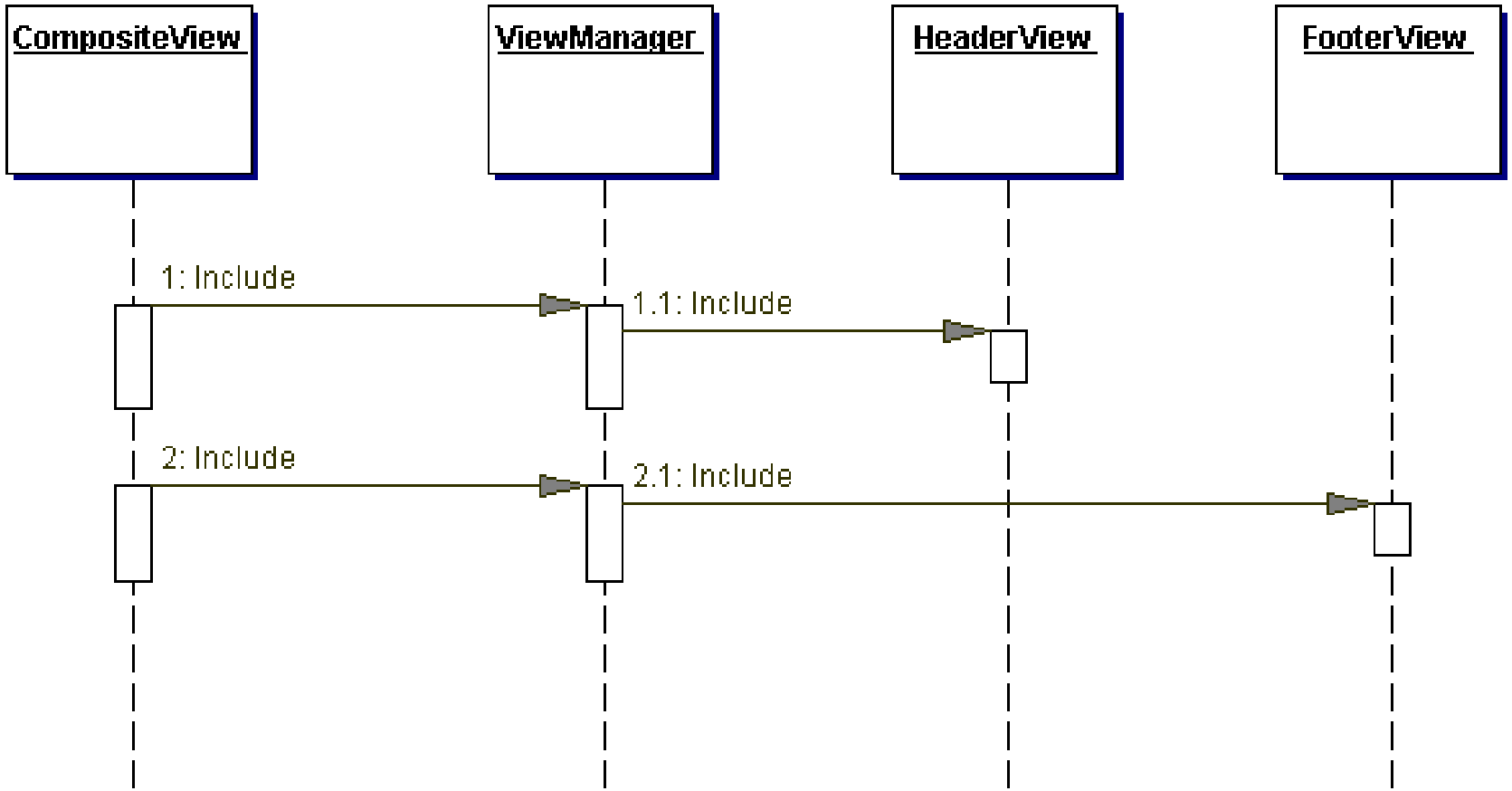


Camada de Apresentação

Solução: Composite View



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*
  - *Utiliza JavaBeans para incluir outros views na página*
  - *Mais simples que solução com Custom Tags*
- *Early Binding Resource Strategy (Translation-time)*
  - *Usa tags padrão: <%@ include %> e <%@ file %>*
  - *Carga é feita em tempo de compilação: alterações só são vistas quando página for recompilada*
- *Late Binding Resource Strategy (Run-time)*
  - *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*

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

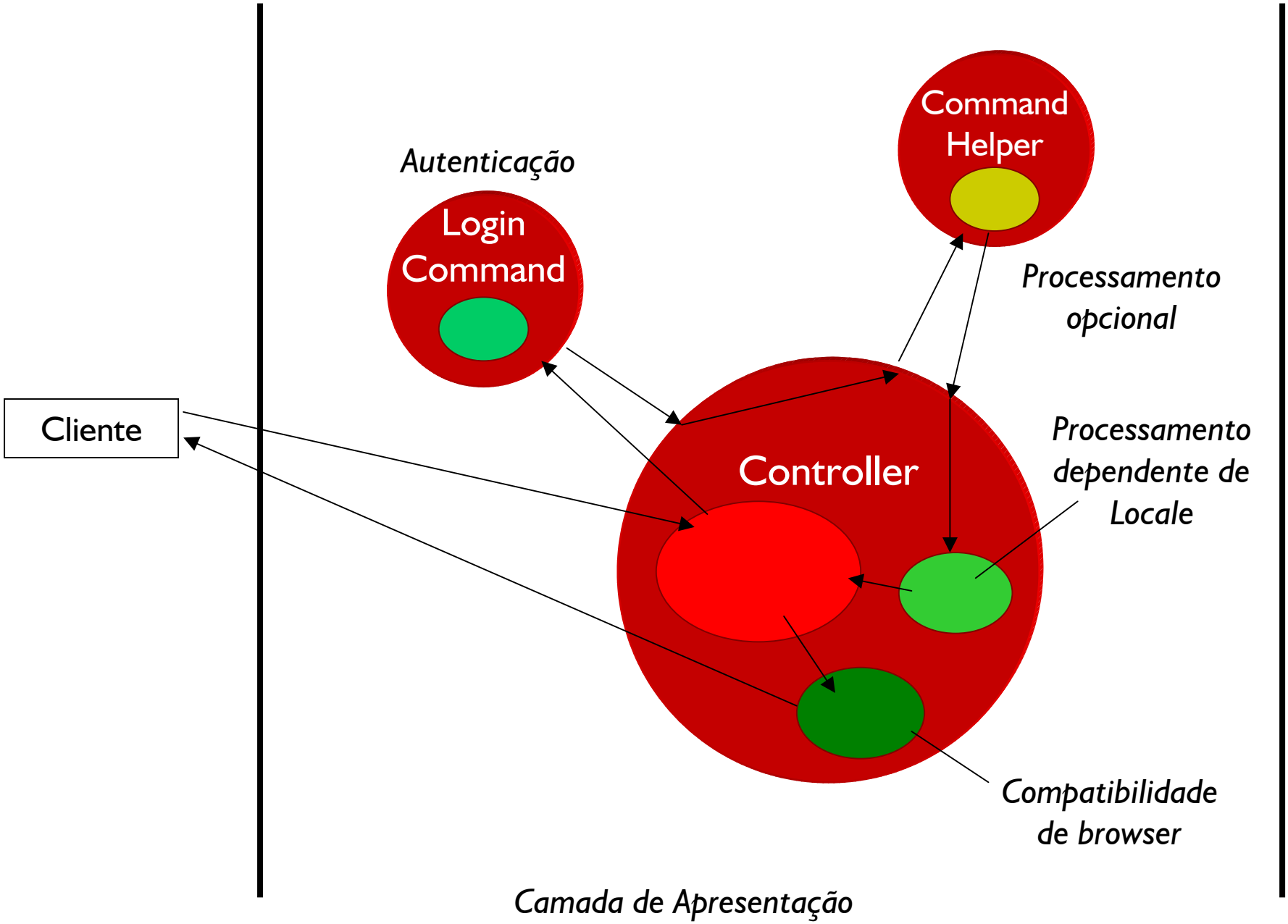
- *4. Refatore a aplicação em cap13/cv/ para que utilize CompositeView (os blocos estão identificados com comentários no HTML em messages.jsp). Escolha as melhores estratégias entre Translation-time e Run-time Stategies*
  - *a) Qual a melhor estratégia para o navbar (raramente muda)?*
  - *b) E para o bloco principal?*
- *5. Implemente o menu usando Custom Tag View Management Strategy*
- *6. 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

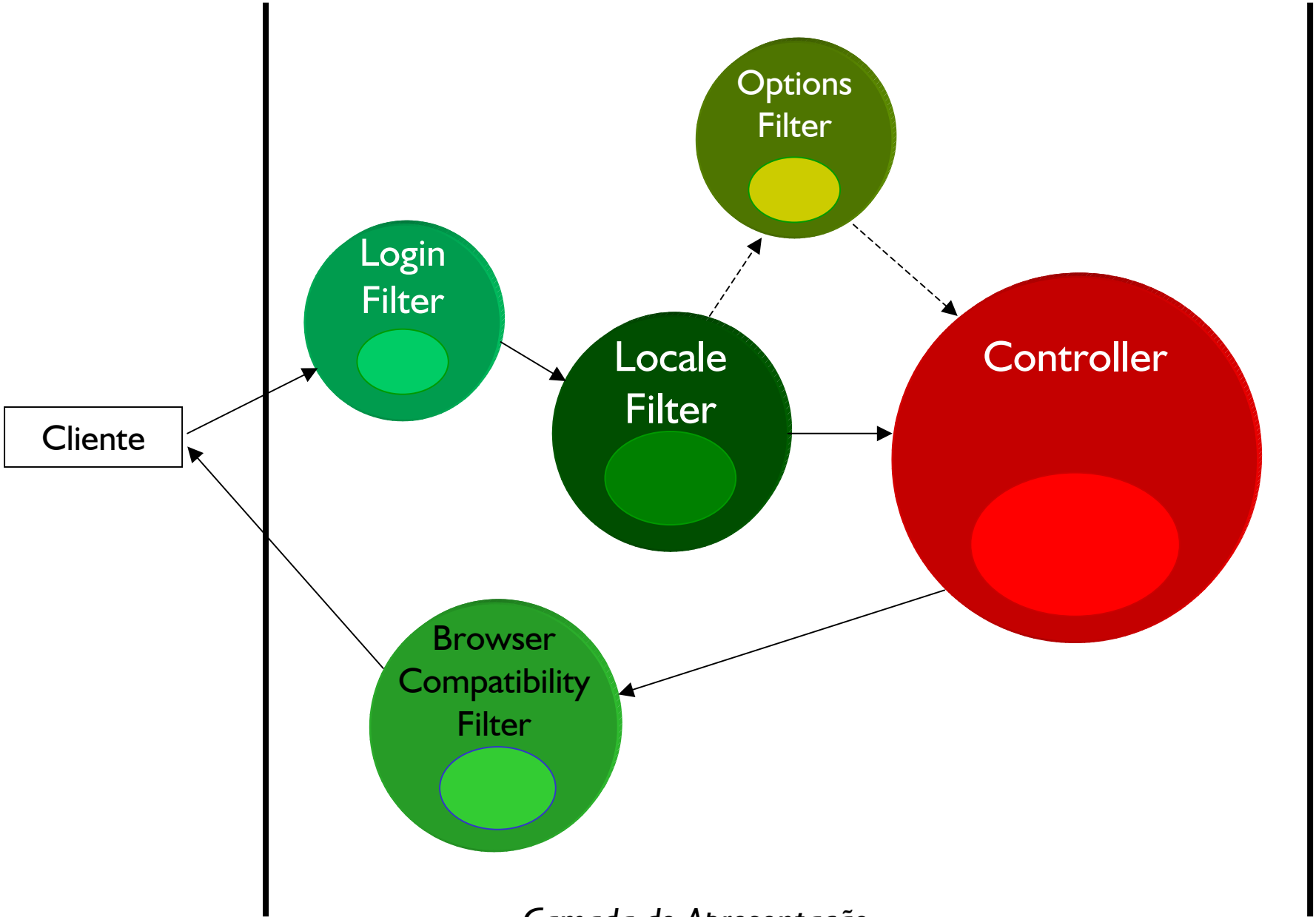
Problema


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

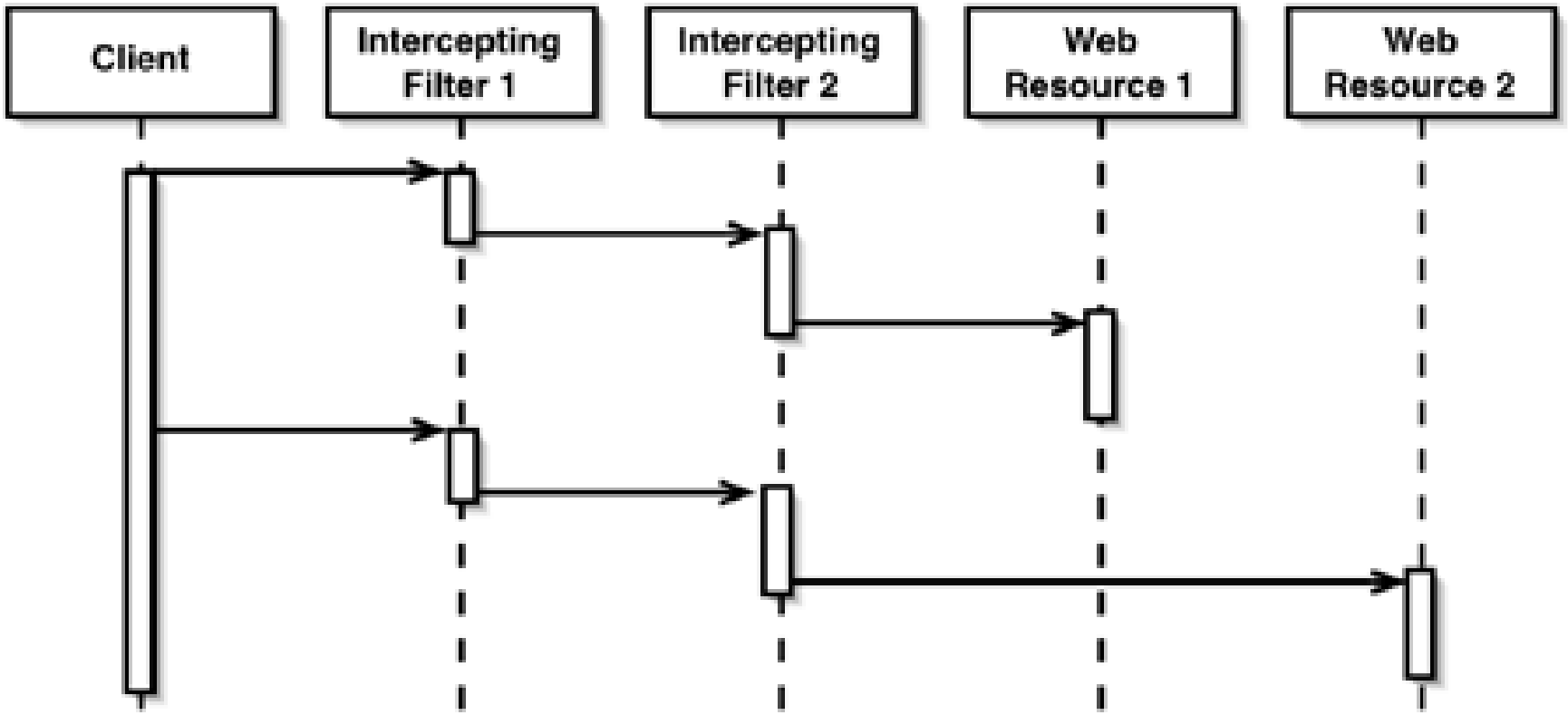
Solução: Intercepting Filter



Descrição da solução

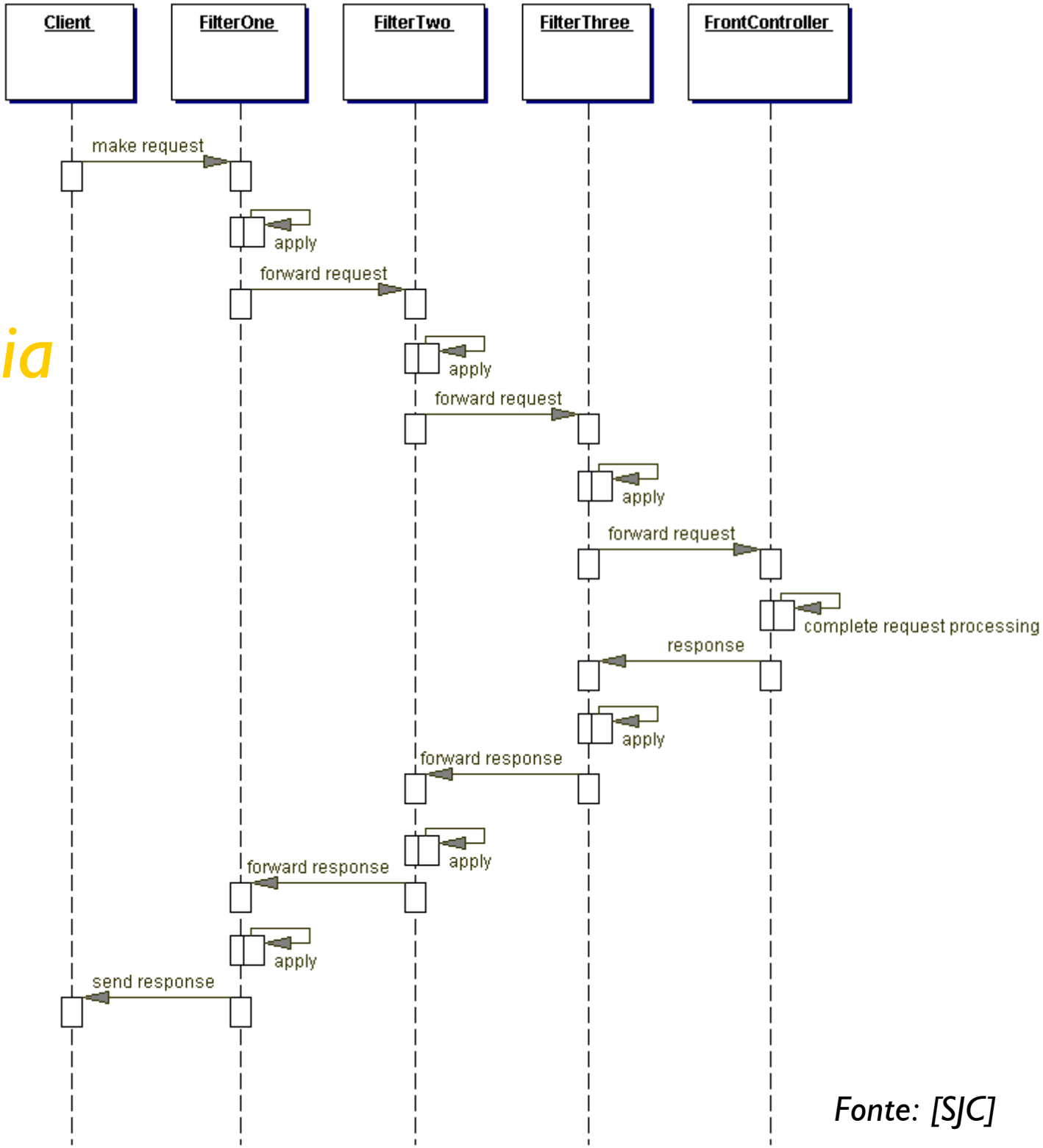
- *Criar filtros plugáveis para processar serviços comuns de forma padrão, sem requerer mudanças no código de processamento*
  - *Filtros interceptam requisições entrantes e respostas, viabilizando pré- e pós-processamento*
  - *Filtros podem ser incluídos dinamicamente e sua composição pode ser alterada*
  - *Filtros são uma estrutura implementada na API Servlet 2.3 (veja cap. 6)*

Exemplo de solução



Fonte: [Blueprints]

Diagrama de Seqüência



Fonte: [SJC]

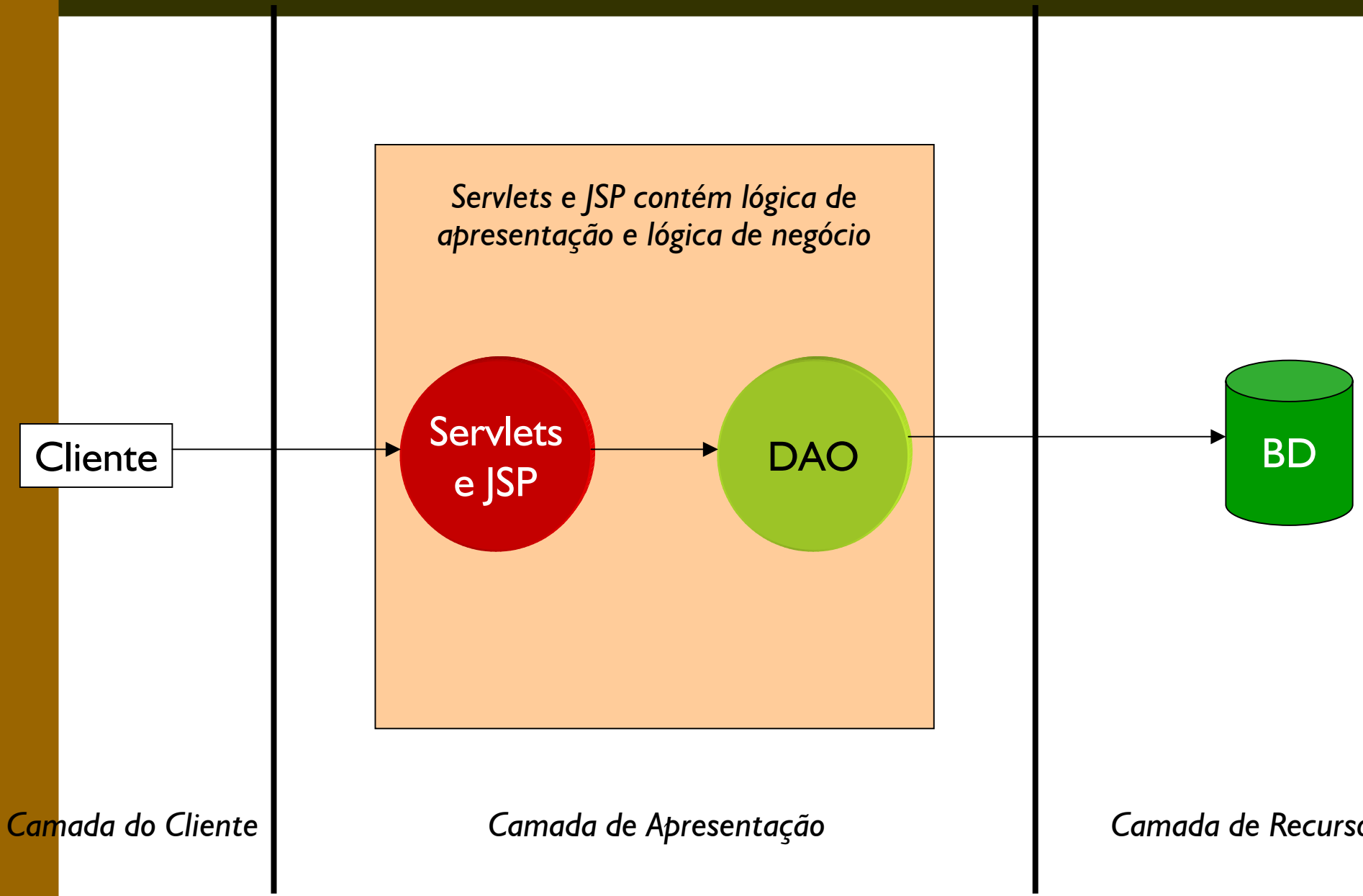
Conseqüências

- 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

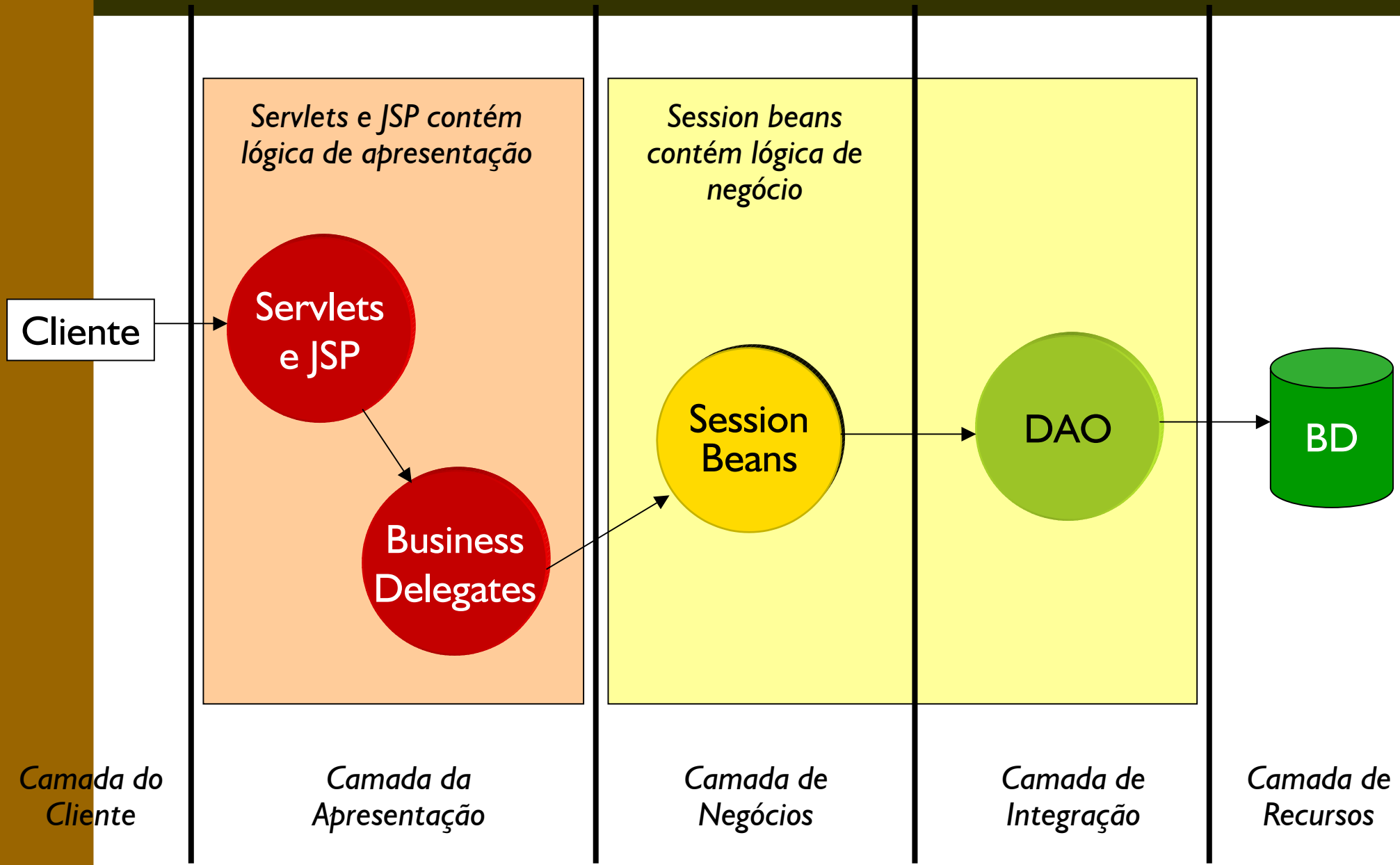
Exercícios

- 7. Refatore a aplicação em cap13/if/ para que utilize Intercepting Filter:
  - 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

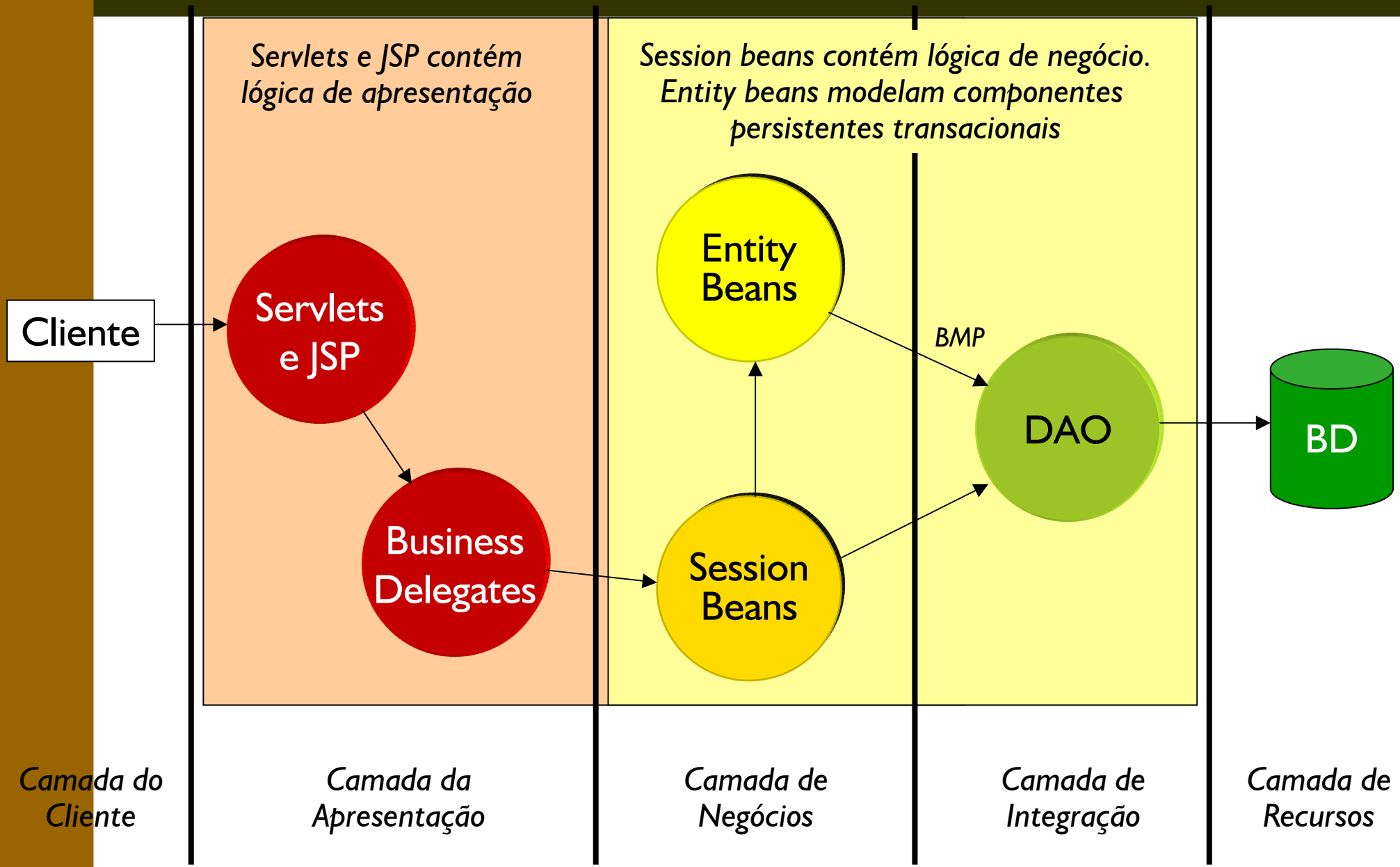
Refatoramento por Camadas (I)



Refatoramento por Camadas (2)



Refatoramento por Camadas (3)

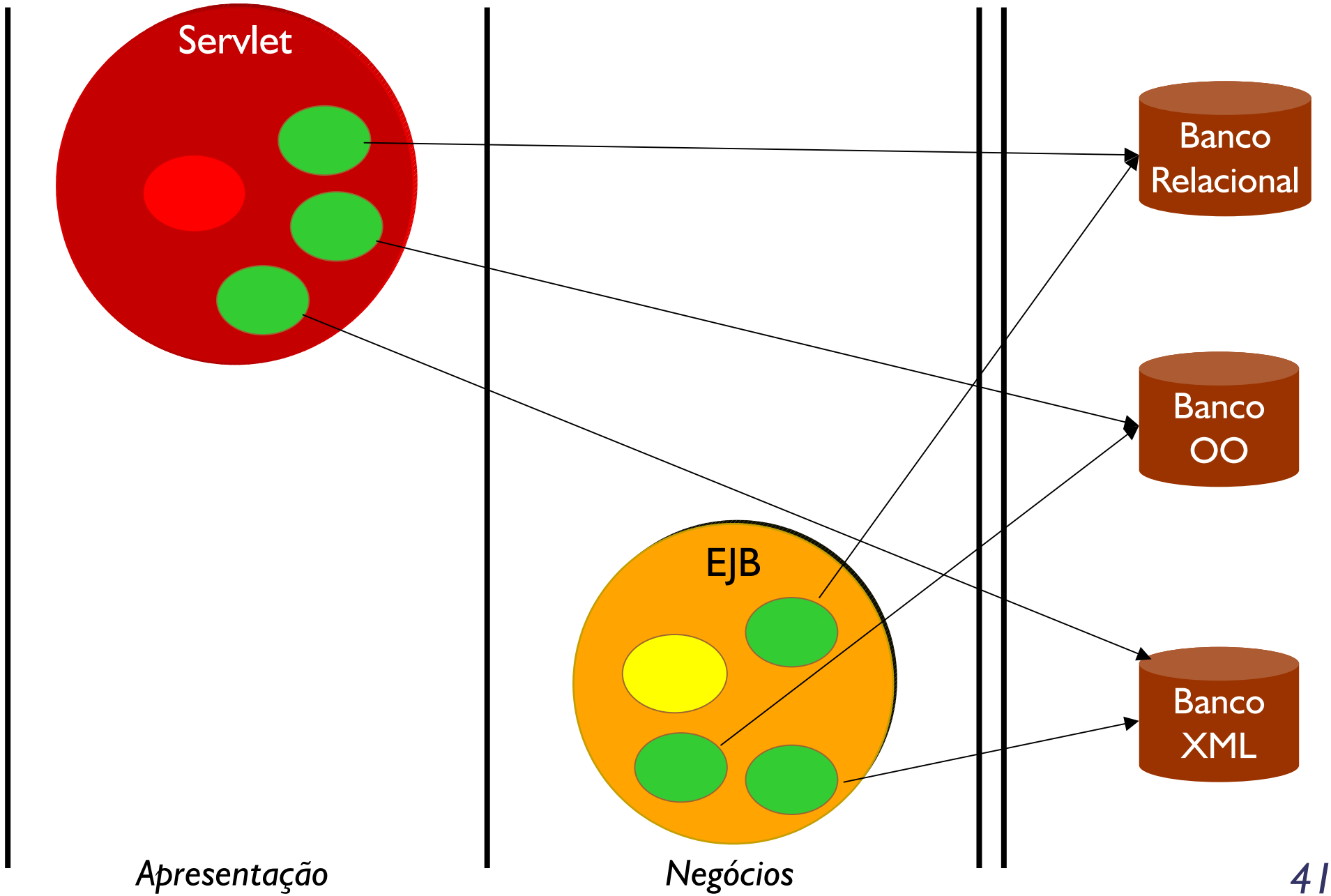


5

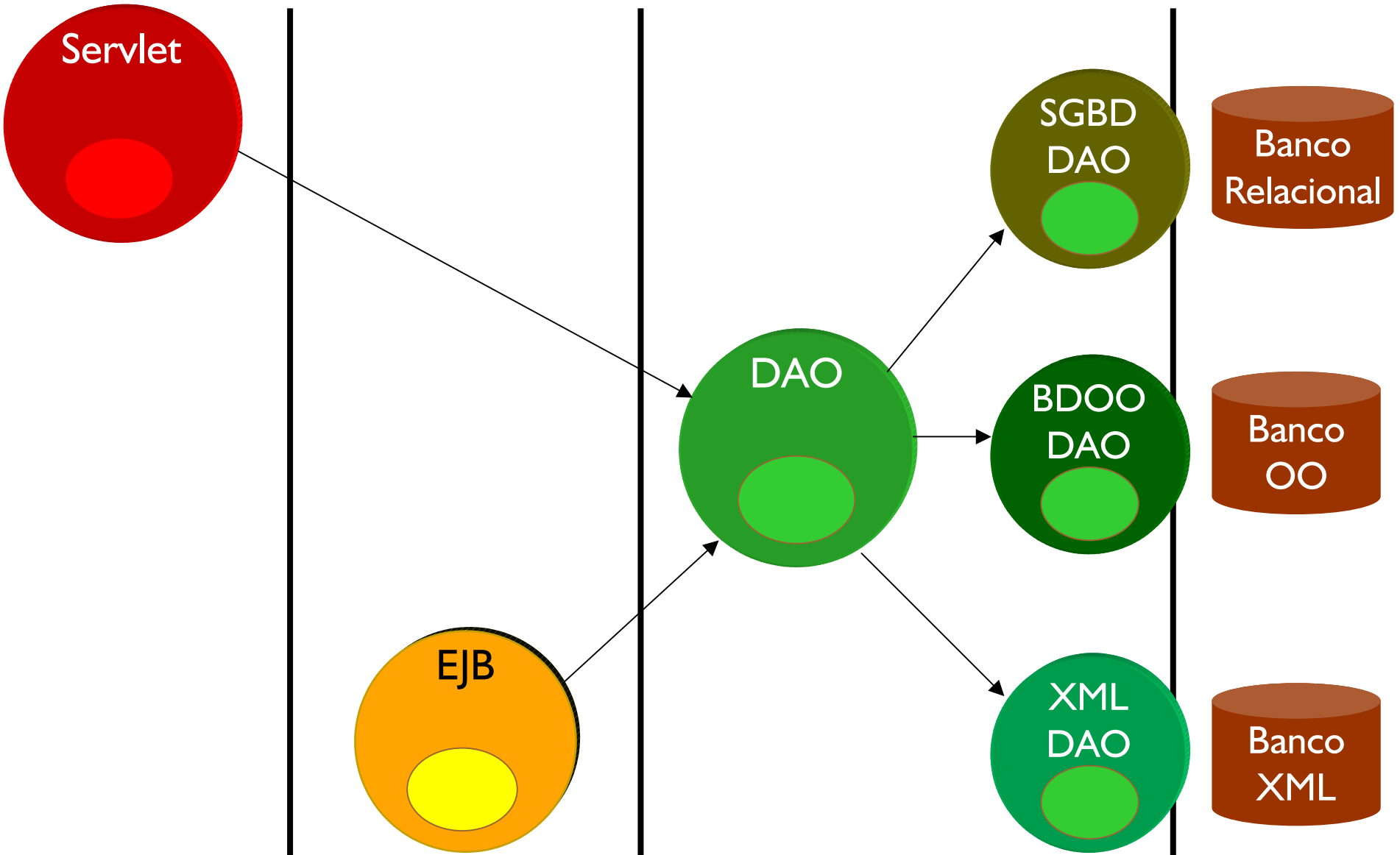
Data Access Object (DAO)

Objetivo: Abstrair e encapsular todo o acesso a uma fonte de dados. O DAO gerencia a conexão com a fonte de dados para obter e armazenar os dados.

Problema


Servlet
Banco Relacional
Banco OO
EJB
Banco XML
Apresentação
Negócios

Solução: Data Access Object



Conseqüências

- *Transparência quanto à fonte de dados*
- *Facilita migração para outras implementações*
- *Reduz complexidade do código nos objetos de negócio (ex: Entity Beans BMP e servlets)*
- *Centraliza todo acesso aos dados em camada separada*
- *Requer design de hierarquia de classes (Factory)*

Exercícios

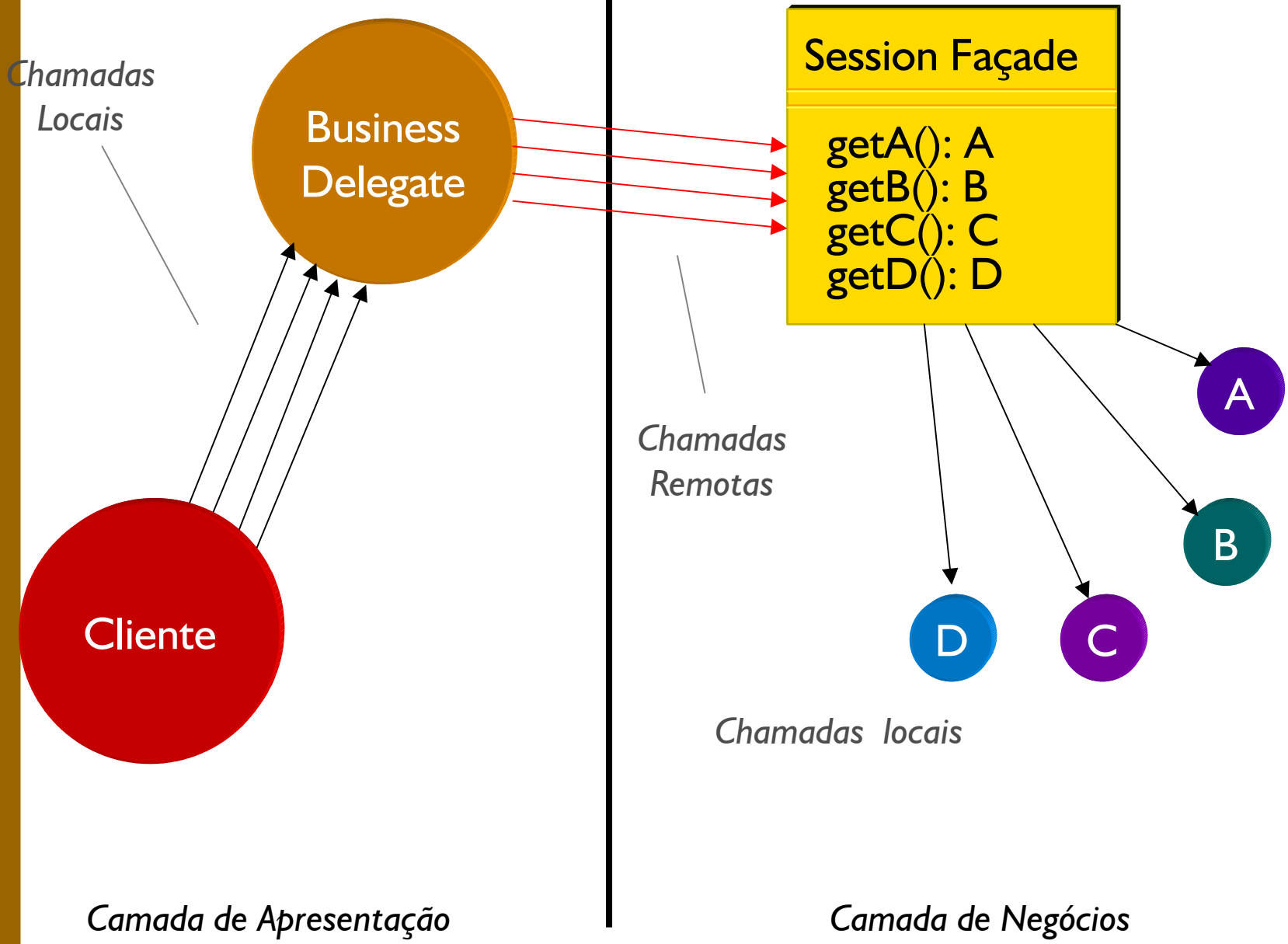
- *8. Analise o código do DAO existente (XML) e implemente um DAO e código para armazenar as mensagens no banco de dados Cloudscape:*
  - *a) Implemente a interface MessageBeanDAO*
  - *b) Implemente um mecanismo de seleção do meio de persistência escolhido através do web.xml e um Factory Method através do qual a aplicação possa selecionar o DAO desejado*

8

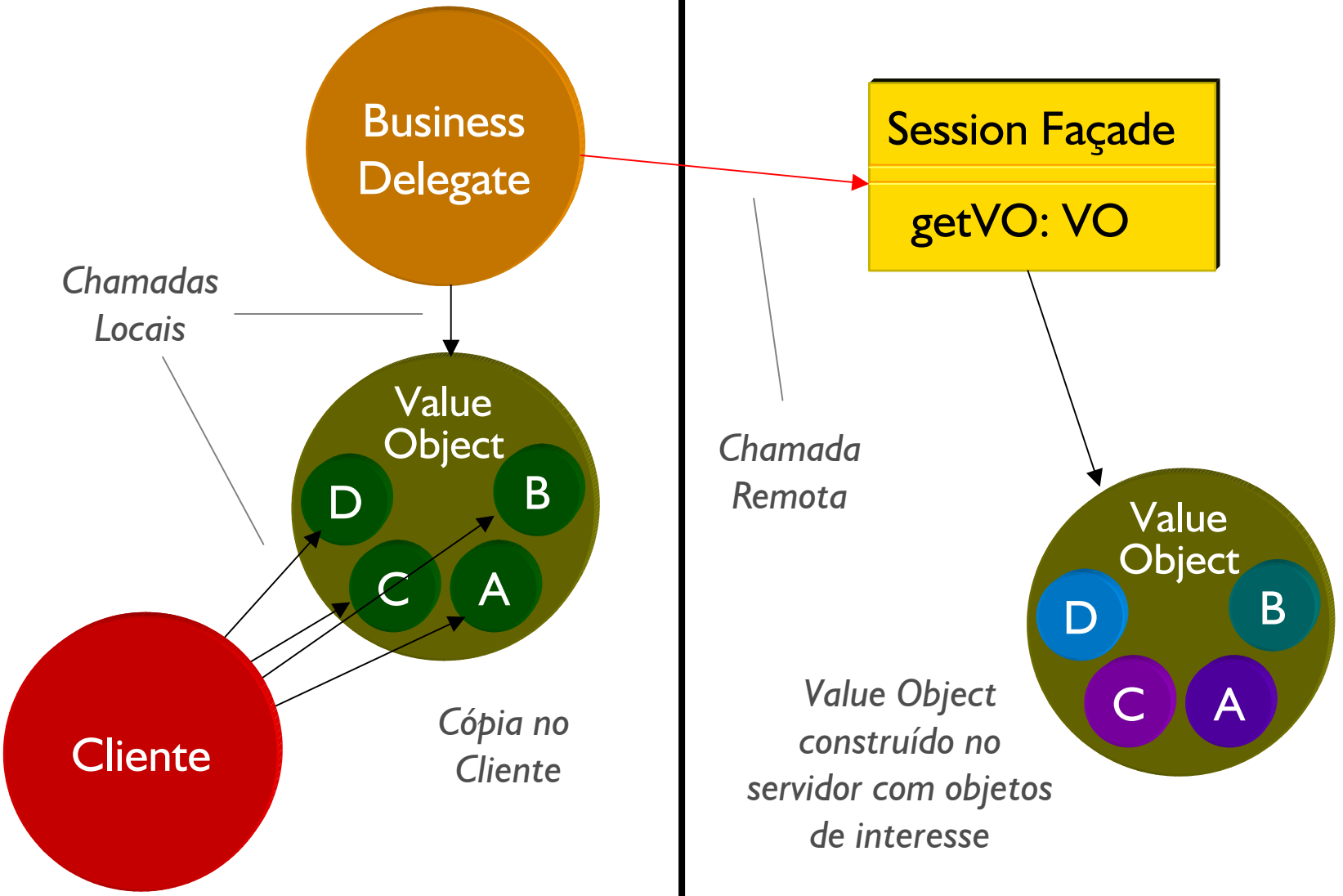
Value Object ou Transfer Object

Objetivo: Reduzir a quantidade de requisições necessárias para recuperar um objeto. Value Object permite encapsular em um objeto um subconjunto de dados utilizável pelo cliente e utilizar apenas uma requisição para transferi-lo.

Problema


Chamadas
Locais
Business
Delegate
Session Façade
getA(): A
getB(): B
getC(): C
getD(): D
A
B
C
D
Chamadas
Remotas
Chamadas locais
Cliente
Camada de Apresentação
Camada de Negócios

Solução: Value Object



Conseqüências

- Simplifica DAO, EJBs e interface remota
- Transfere mais dados em menos chamadas
- Reduz tráfego de rede
- Reduz duplicação de código
- Pode introduzir objetos obsoletos
- Pode aumentar a complexidade do sistema
  - Sincronização
  - Controle de versões para objetos serializados

Exercícios

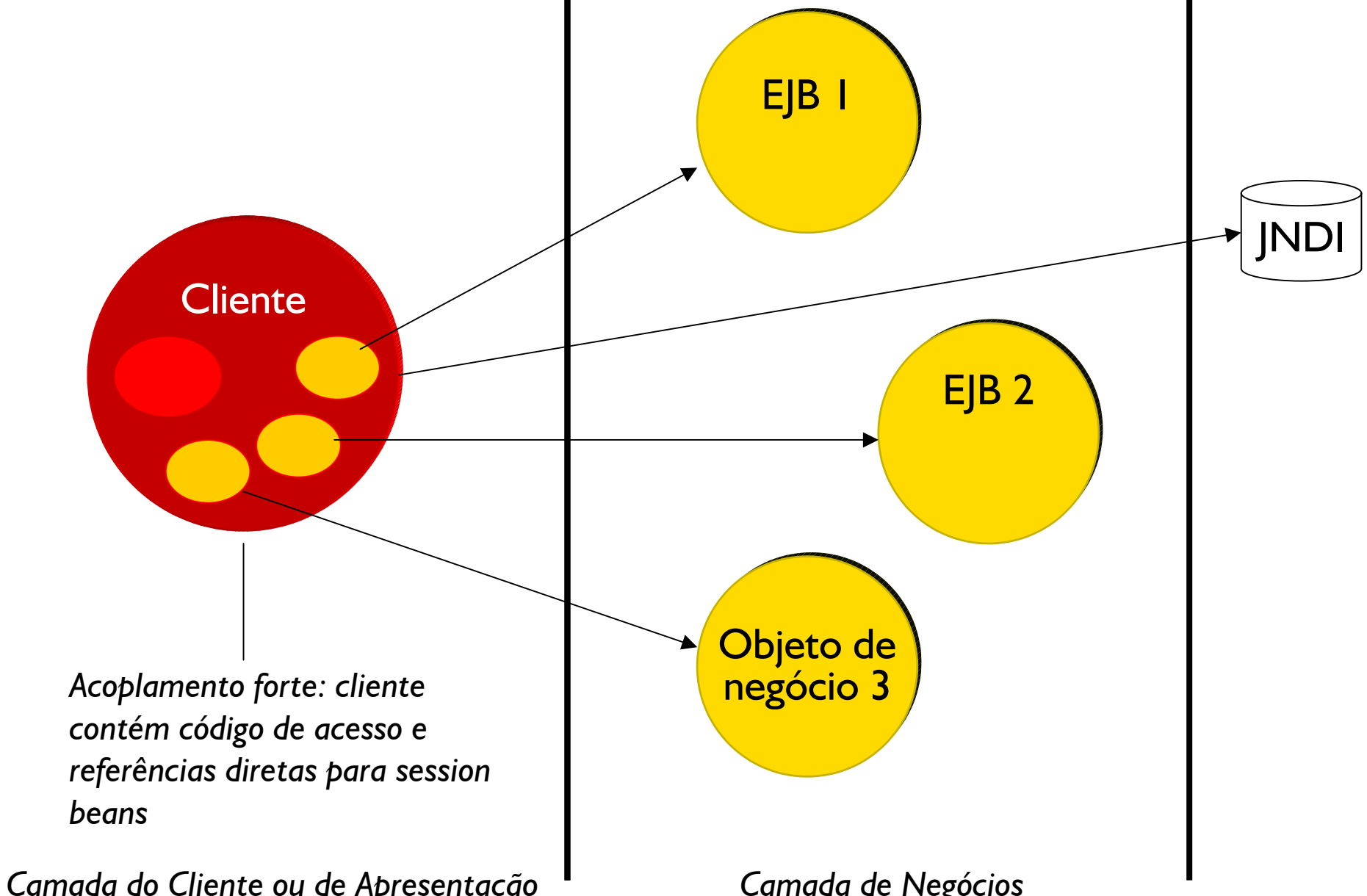
- 9. Refatore a aplicação em cap13/vo/ para que utilize Value Object
  - a) Crie um Value Object que representa uma cópia do objeto MensagemBean
  - b) Implemente no Façade um método que retorne o Value Object para o cliente, e outro que o receba de volta a atualize os dados corretamente.
  - b) Refatore o cliente para que ele use esse objeto e extraia os dados corretamente.

7

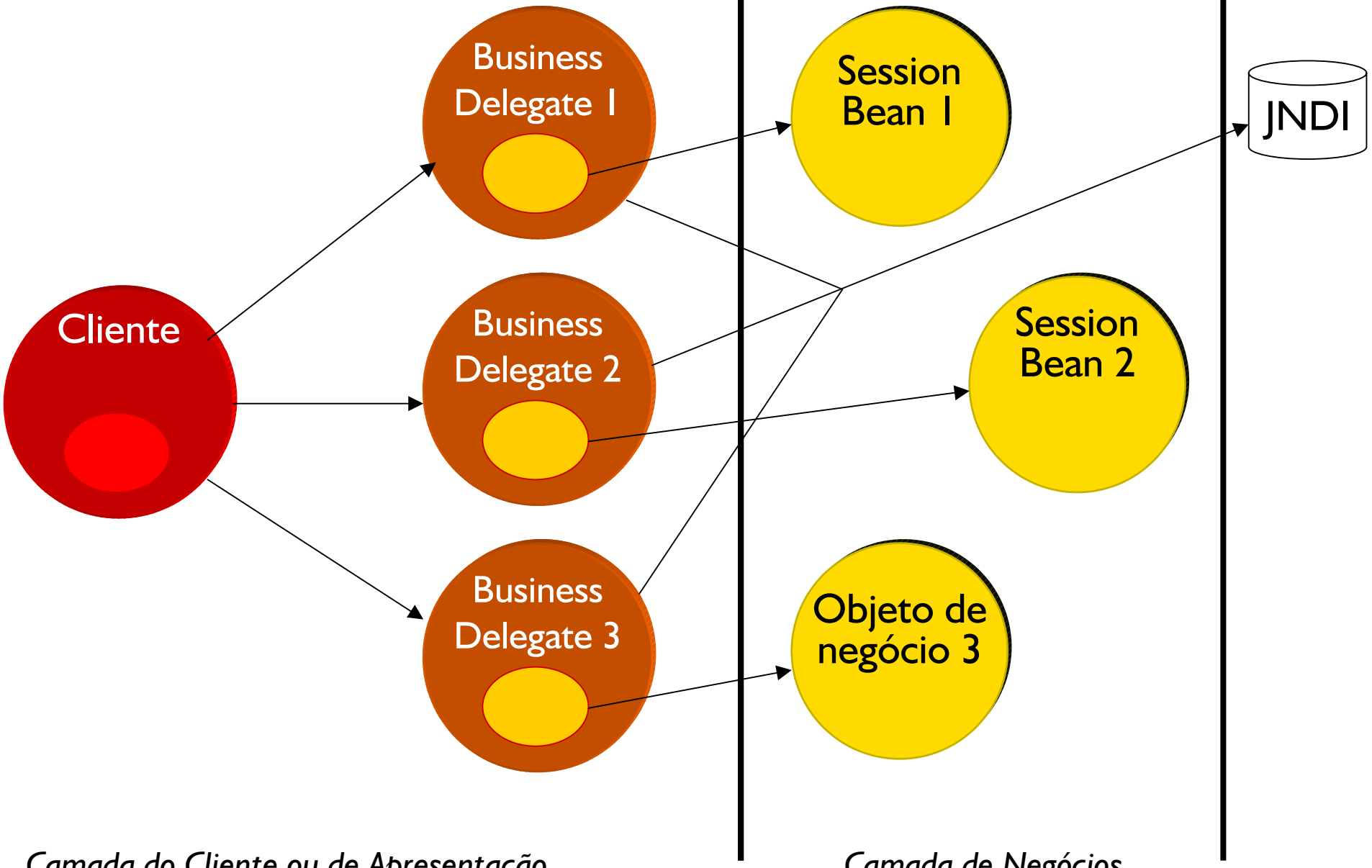
Business Delegate

Objetivo: isolar cliente de detalhes acerca da camada de negócios. Business delegates funcionam como proxies ou fachadas para cada session bean.

Problema


EJB 1
JNDI
Cliente
EJB 2
Objeto de negócio 3
Acoplamento forte: cliente contém código de acesso e referências diretas para session beans
Camada do Cliente ou de Apresentação
Camada de Negócios

Solução: Business Delegate



Melhores estratégias de implementação

- *Delegate Proxy Strategy*
  - *Interface com mesmos métodos que o objeto de negócio que está intermediando*
  - *Pode realizar cache e outros controles*
- *Delegate Adapter Strategy*
  - *Permite integração de um sistema com outro (sistemas podem usar XML como linguagem de integração)*

Conseqüências

- Reduz acoplamento
- Traduz exceções de serviço de negócio
- Implementa recuperação de falhas
- Expõe interface mais simples
- Pode melhorar a performance com caches
- Introduz camada adicional
- Transparência de localidade
  - Oculta o fato dos objetos estarem remotos

Exercícios

- *10. Refatore a aplicação em cap13/bd/ para que utilize Business Delegate:*
  - *a) Implemente um Business Delegate para fazer interface com entre o Controller Servlet (ou comandos) e o DAO.*
  - *b) Trate as exceções específicas de cada camada e encapsule-as em exceções comuns a todo o sistema*

helder@acm.org

argonavis.com.br