Padrões de Projeto J2EE 3

Padrões da Camada de Negócios (EJB)

Helder da Rocha (helder@acm.org)

argonavis.com.br

Introdução

- *A camada de negócios encapsula a lógica central da aplicação. Considerações de design incluem*
  - *Uso de session beans para modelar ações. Stateless para operações de um único método. Stateful para operações que requerem mais de um método (que retém estado entre chamadas)*
  - *Uso de session beans como fachadas à camada de negócios*
  - *Uso de entity beans para modelar dados persistentes como objetos distribuídos*
  - *Uso de entity beans para implementar lógica de negócio e relacionamentos*
  - *Cache de referências para EJBs em Business Delegates*

*Práticas não recomendadas**

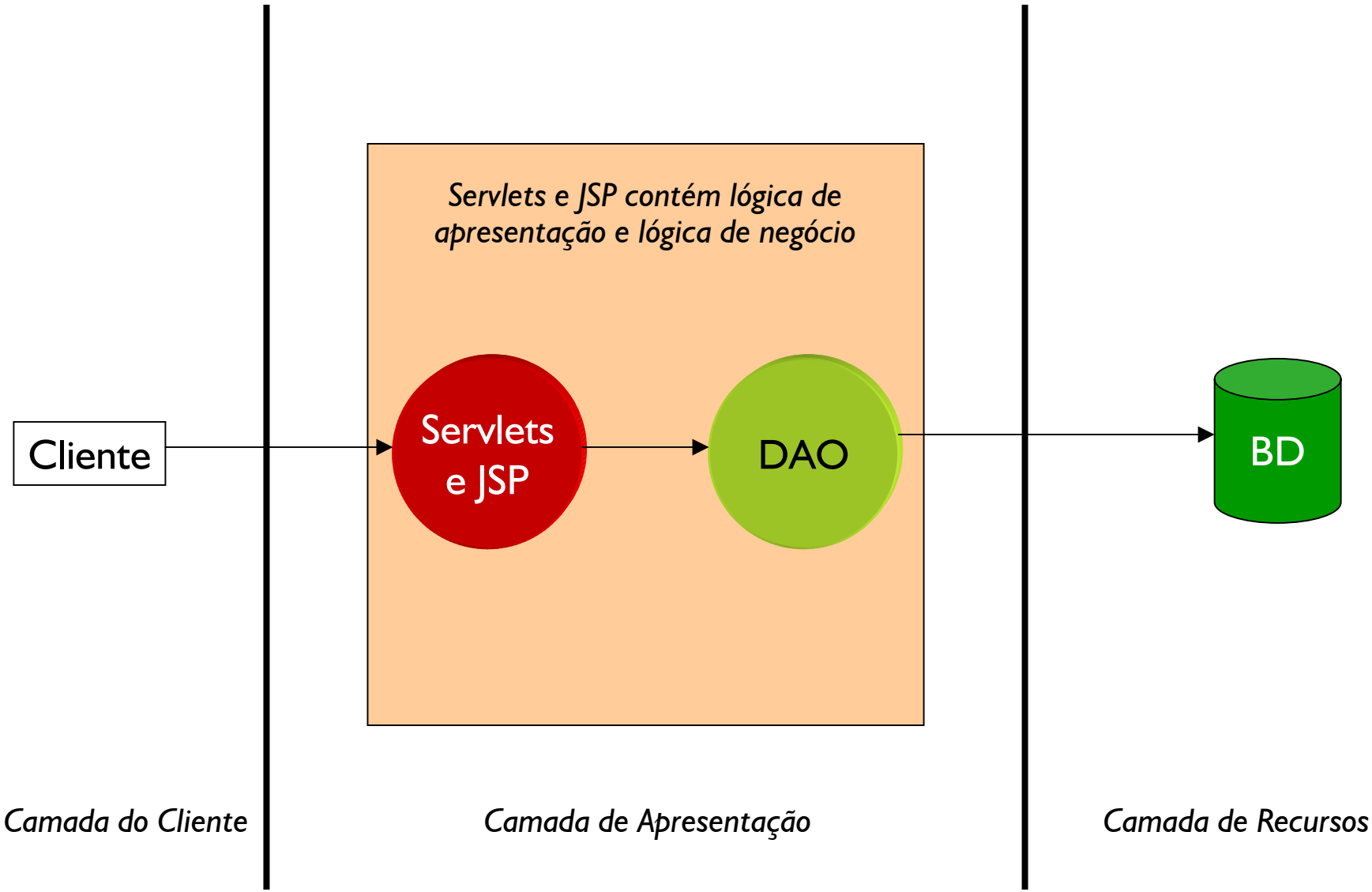
- *Indicam necessidade de aplicação dos padrões*
  1. *Mapeamento direto: modelo de objetos/modelo de Entity Beans*
  2. *Mapeamento direto: modelo relacional/modelo de Entity Beans*
  3. *Mapeamento de cada use case a um Session Bean*
  4. *Exposição de todos os atributos de um EJB via métodos get/set*
  5. *Embutir lookup de serviços nos clientes*
  6. *Usar Entity Beans como objetos read-only*
  7. *Usar Entity Beans como objetos fine-grained*
  8. *Armazenar árvore de objetos dependentes de Entity Beans*
  9. *Expor exceções relacionadas a EJB a clientes não-EJB*
  10. *Usar finder methods para retornar um grande result set*
  11. *Usar EJBs para transações longas*
  12. *Agregar dados de componentes de negócio no cliente*
  13. *Reconstruir estado por chamada em Stateless Session Bean*

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

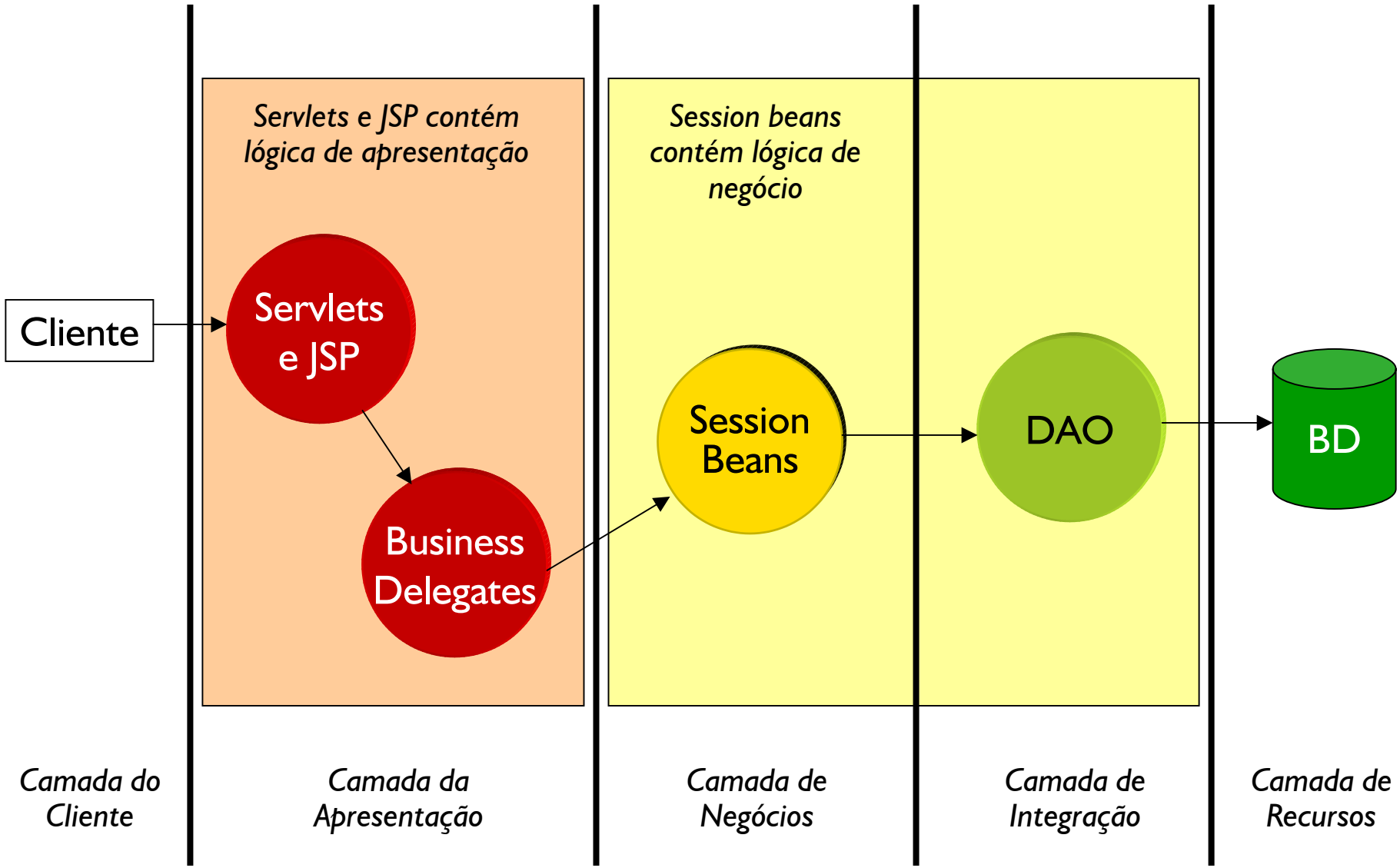
Padrões da Camada de Negócios

- *(7) Business Delegate*
  - *Desacopla camadas de apresentação e de serviços*
- *(8) Value Object ou Transfer Object*
  - *Reduz tráfego e facilita transferência de dados entre camadas*
- *(9) Session Façade*
  - *Oculta complexidade de objetos de negócio e centraliza controle*
- *(10) Composite Entity*
  - *Agrupa composições de entity beans*
- *(11) Value Object Assembler ou Transfer Object Assembler*
  - *Constrói um Value Object composto de múltiplas fontes*
- *(12) Value List Handler*
  - *Lida com execução de queries, caching de resultados, etc.*
- *(13) Service Locator*
  - *Encapsula lógica de consulta e criação de objetos de serviço*

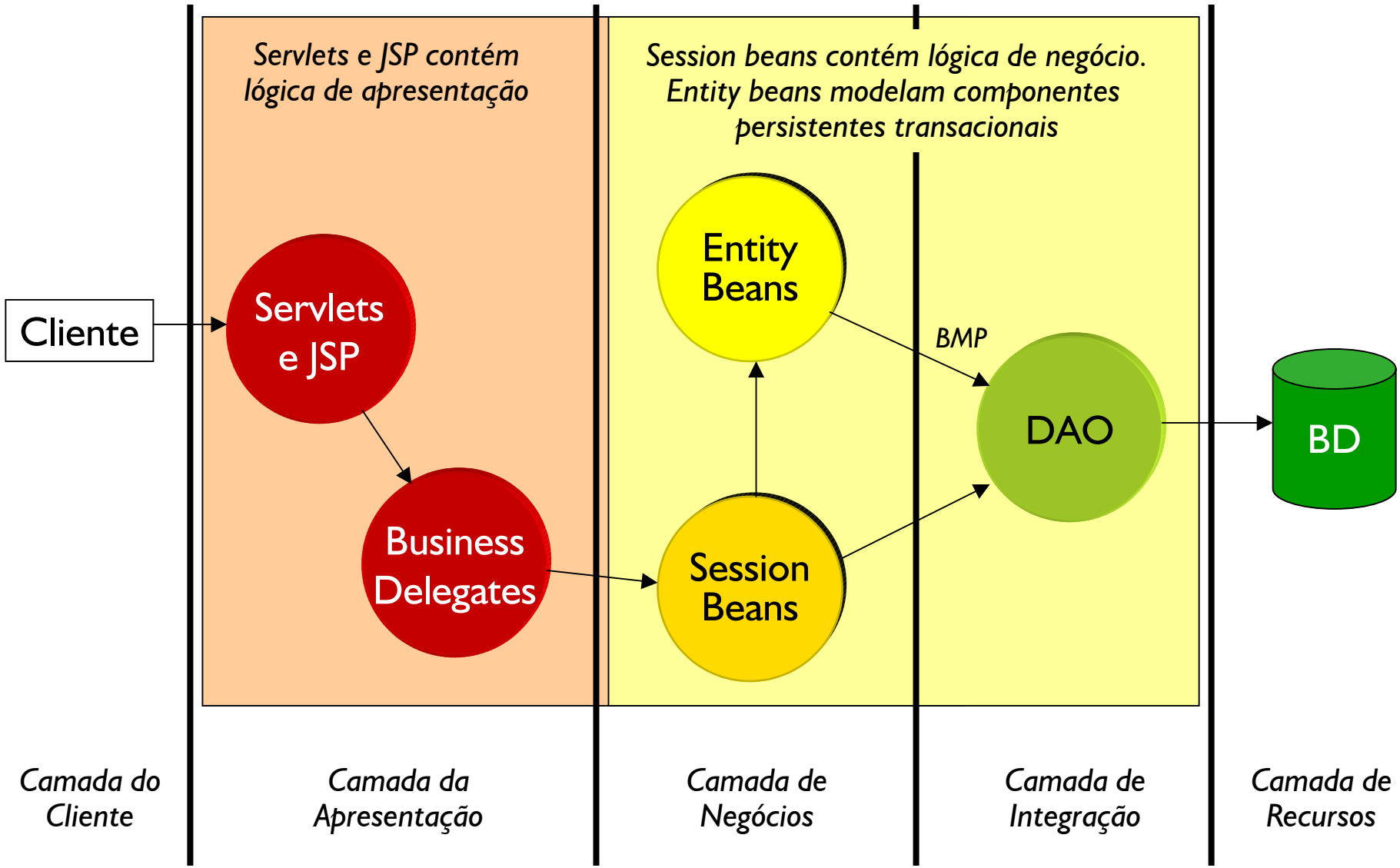
Refatoramento por Camadas (1)



Refatoramento por Camadas (2)



Refatoramento por Camadas (3)



Exercícios

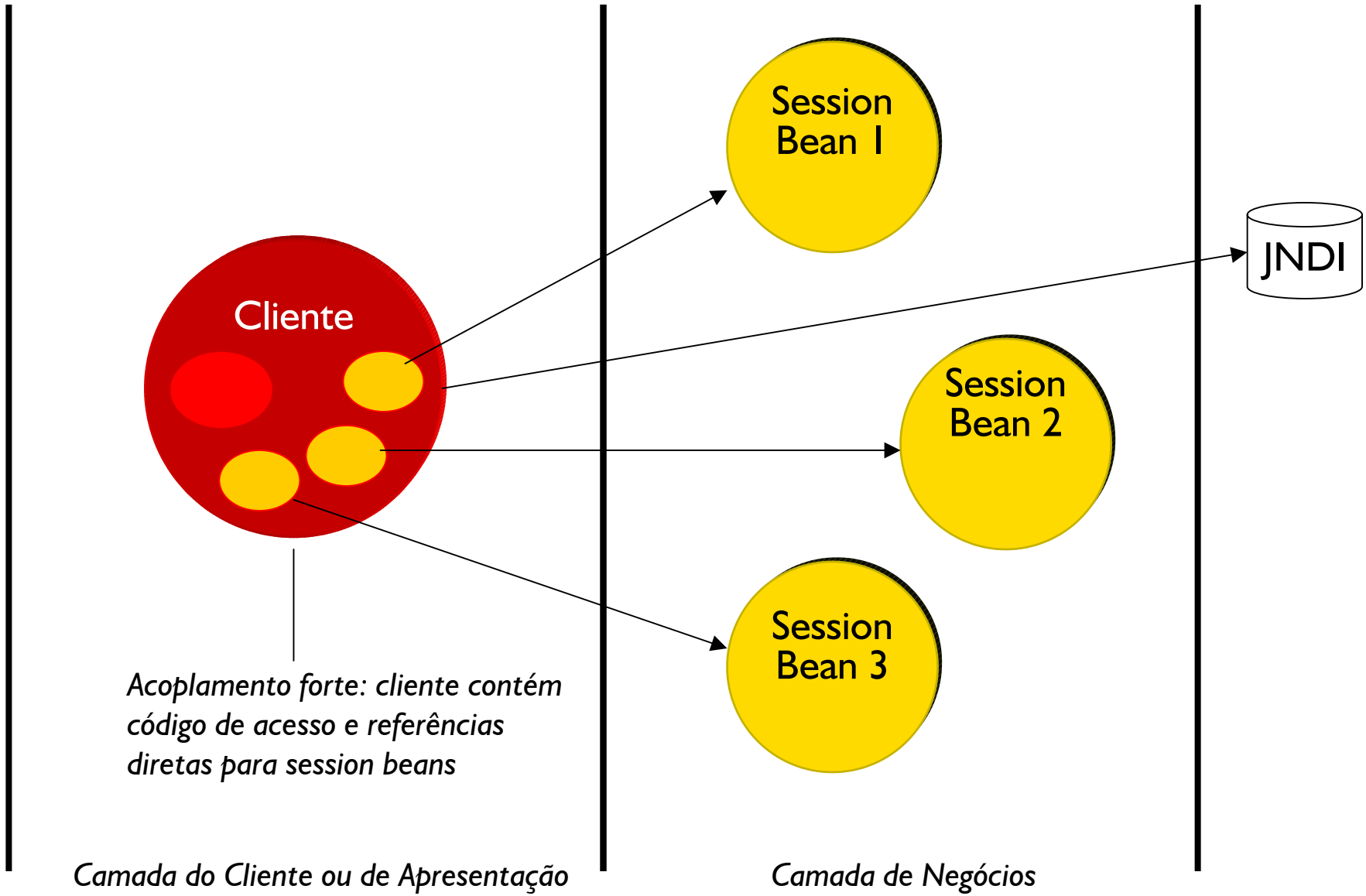
- *1. Converta a aplicação JSP/servlet em ejblayer/camadas para EJB*
  - *a) Crie um Session Façade e um DAO*
  - *b) Crie Business Delegates*
  - *c) Crie Entity Beans*

7

Business Delegate

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

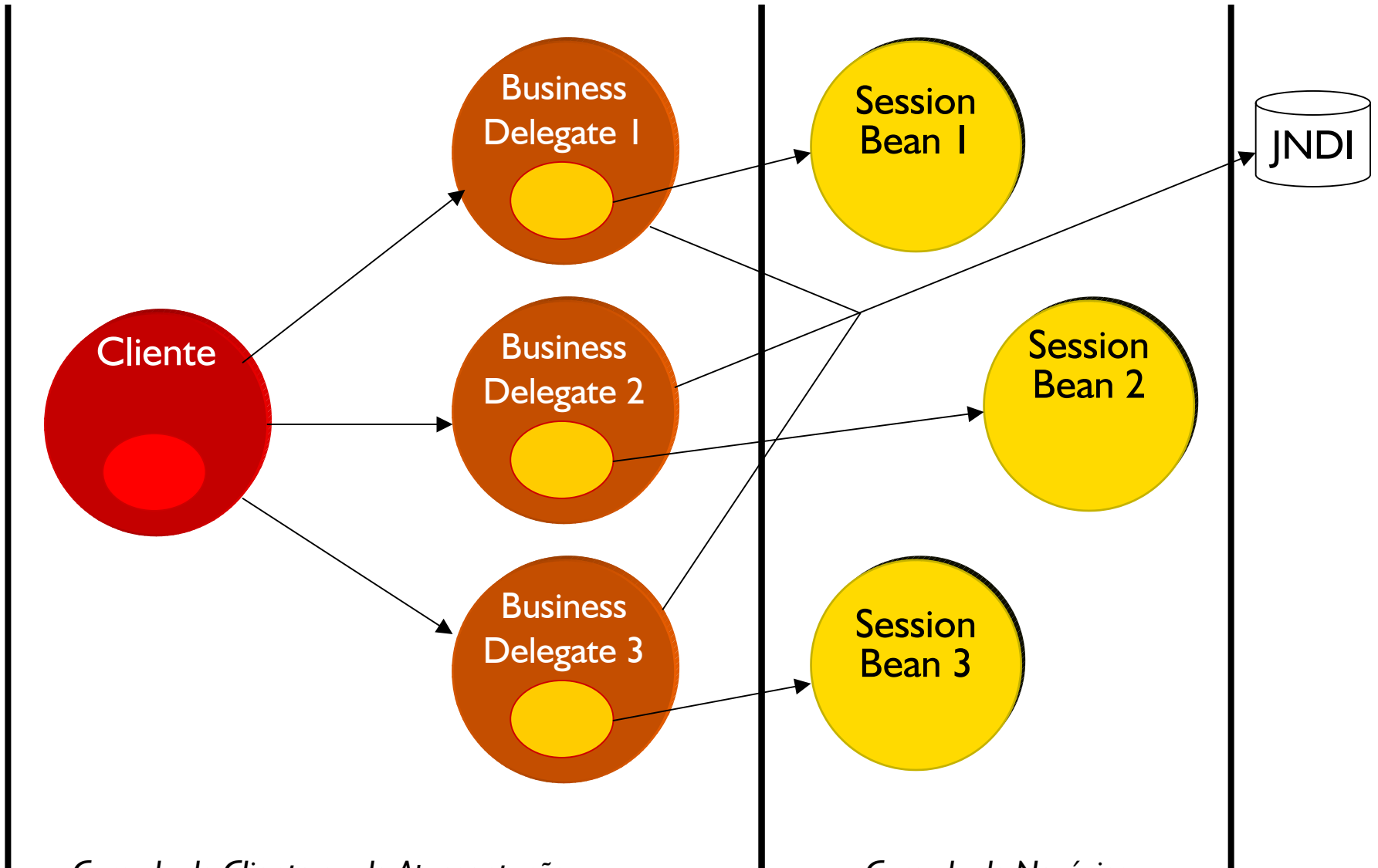
Problema


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

Descrição do problema

- *Cliente contém referências para detalhes da camada de negócios*
  - *Nomes JNDI dos componentes*
  - *Eventuais relacionamentos entre beans*
  - *Exceções exclusivas de EJBs*
- *Acoplamento forte dificulta desenvolvimento da camada de apresentação*
  - *Cliente fica vulnerável a mudanças no modelo de dados modelado pelos objetos*

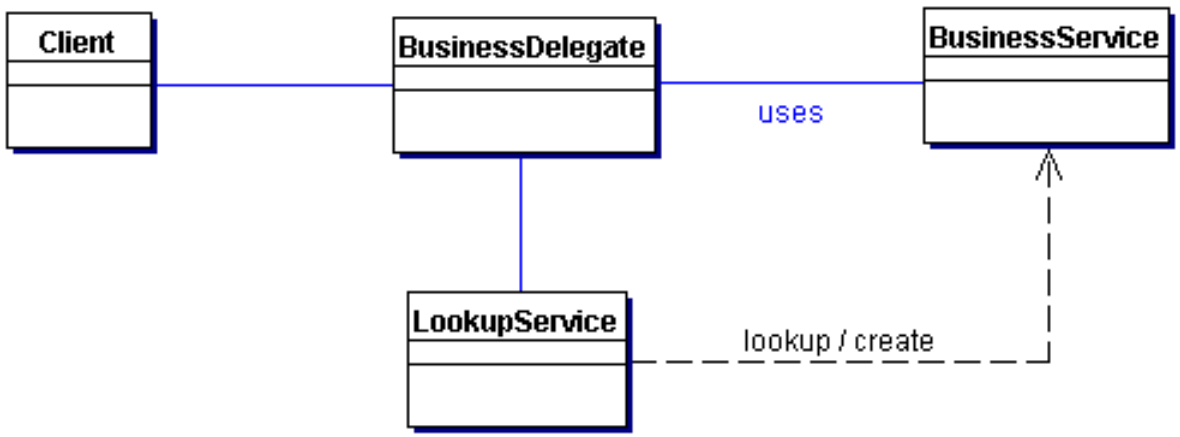
Solução: Business Delegate



Descrição da solução

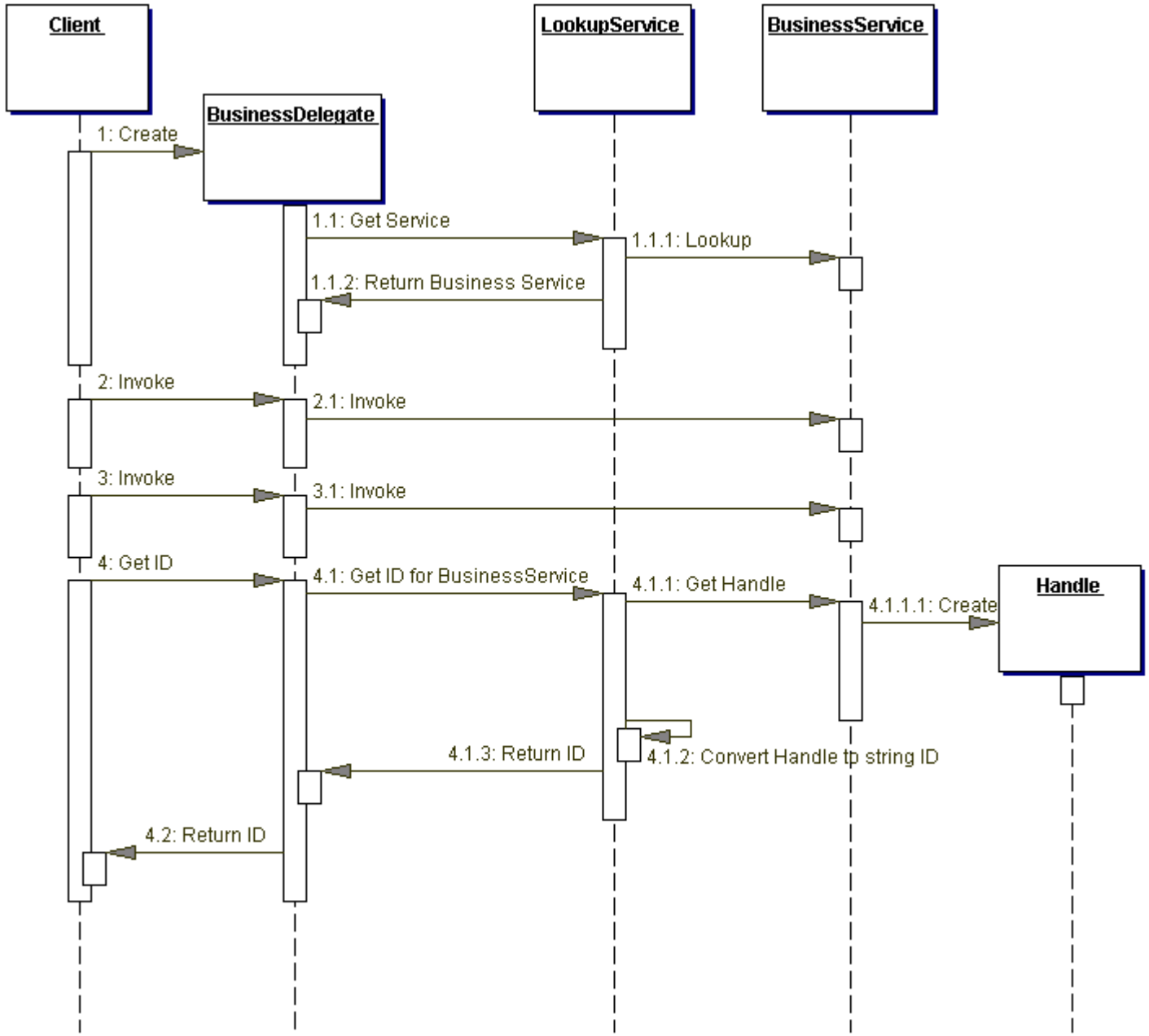
- *Um Business Delegate é uma classe Java comum que encapsula detalhes da camada de negócios e intercepta exceções específicas do EJB isolando-as do cliente*
- *Deve-se introduzir um Business Delegate como fachada para cada Session Bean que for diretamente exposto a clientes*
- *Business Delegates podem usar um Service Locator para localizar os serviços de negócio (JNDI)*

Estrutura UML (1)



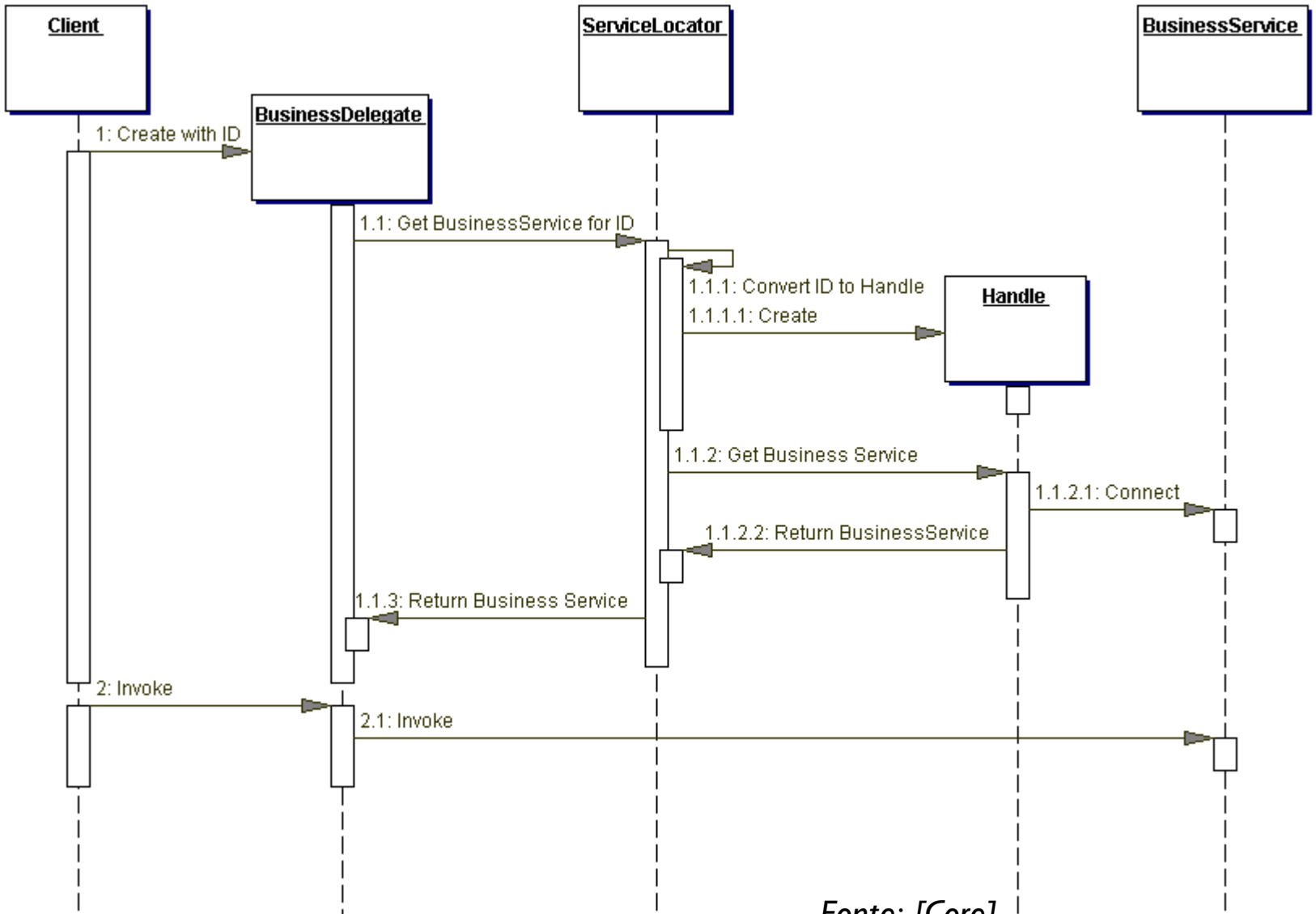
Fonte: [Core]

Diagramas de Seqüência (I)



Fonte: [Core]

Diagramas de Seqüência (2)

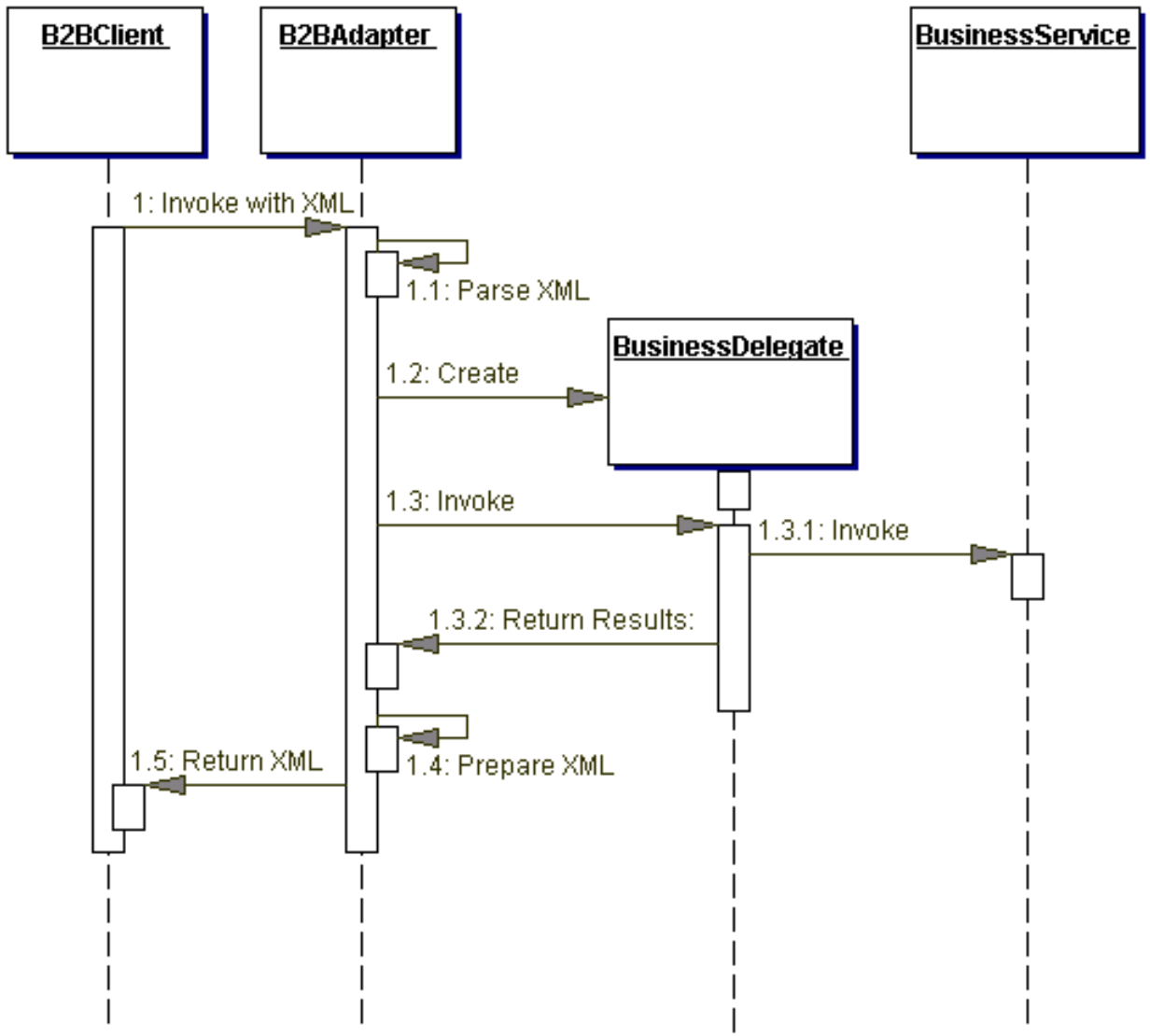


Fonte: [Core]

Melhores estratégias de implementação

- *Delegate Proxy Strategy*
  - *Interface com mesmos métodos que o Session Bean 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)*

Delegate Adapter Strategy



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

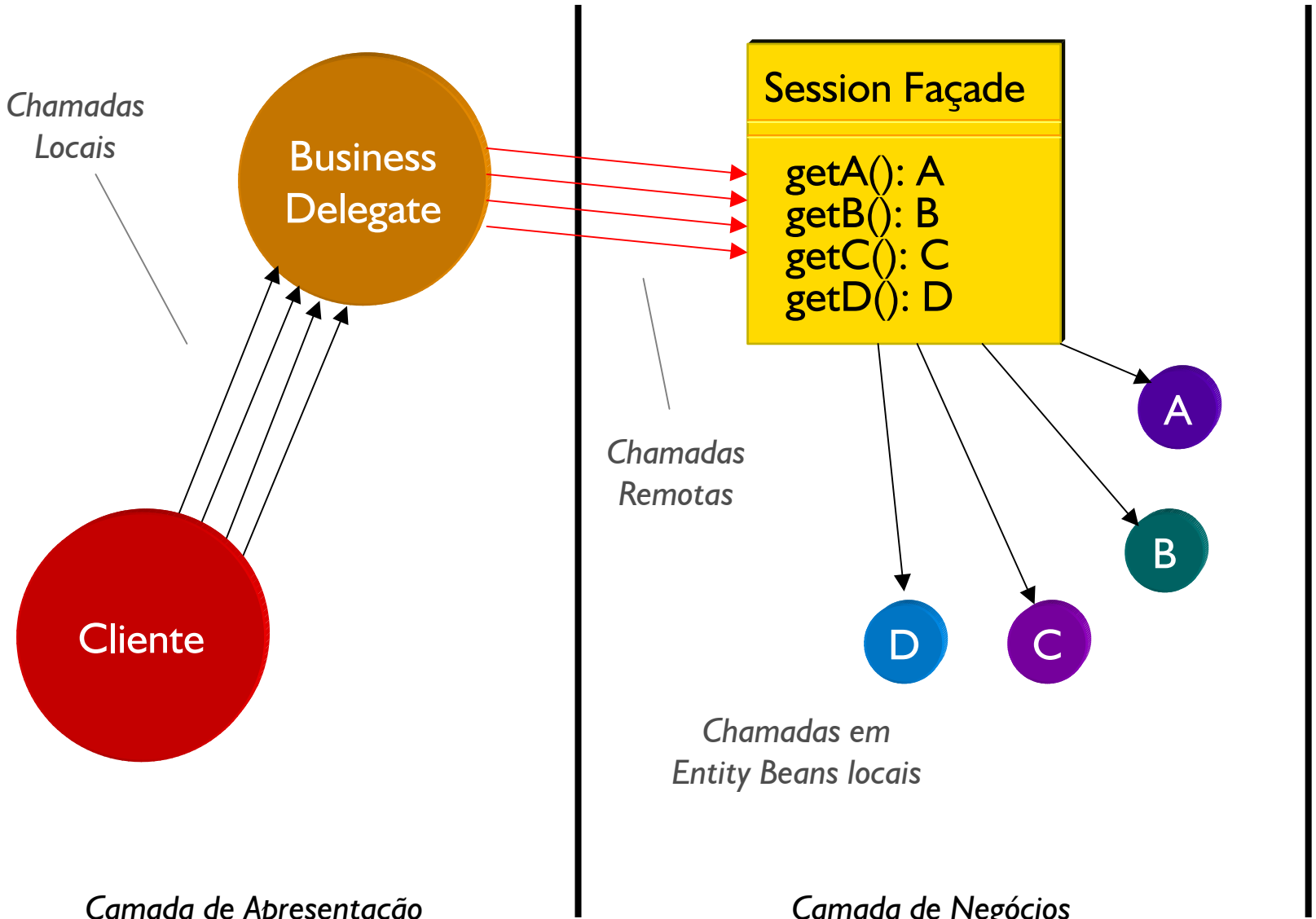
- *1. Analise a implementação das estratégias de Business Delegate (código 8.1 a 8.2)*
- *2. Refatore a aplicação em ejblayer/bd/ para que utilize Business Delegate:*
  - *a) Implemente um Business Delegate para fazer interface com entre o Controller Servlet (ou comandos) e o Session Bean principal.*
  - *b) Trate as exceções e encapsule-as em exceções comuns a todo o sistema*

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.

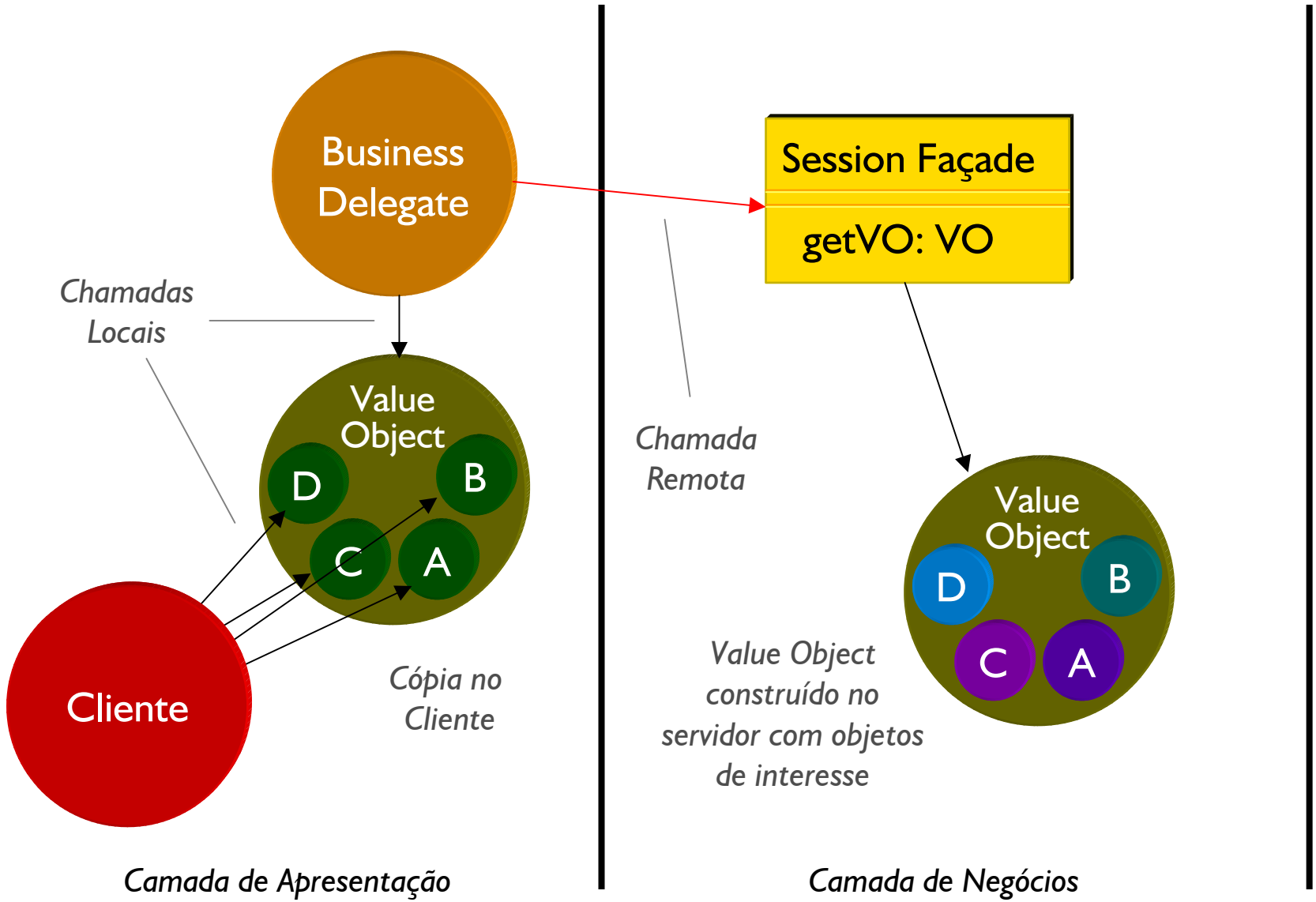
Problema


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

Descrição do problema

- *Cliente precisa obter diversos dados de um Enterprise Bean*
- *Para obter os dados, é preciso realizar diversas chamadas ao bean*
  - *As chamadas são potencialmente remotas*
  - *Fazer múltiplas chamadas através da rede gera tráfego e reduz o desempenho da aplicação*

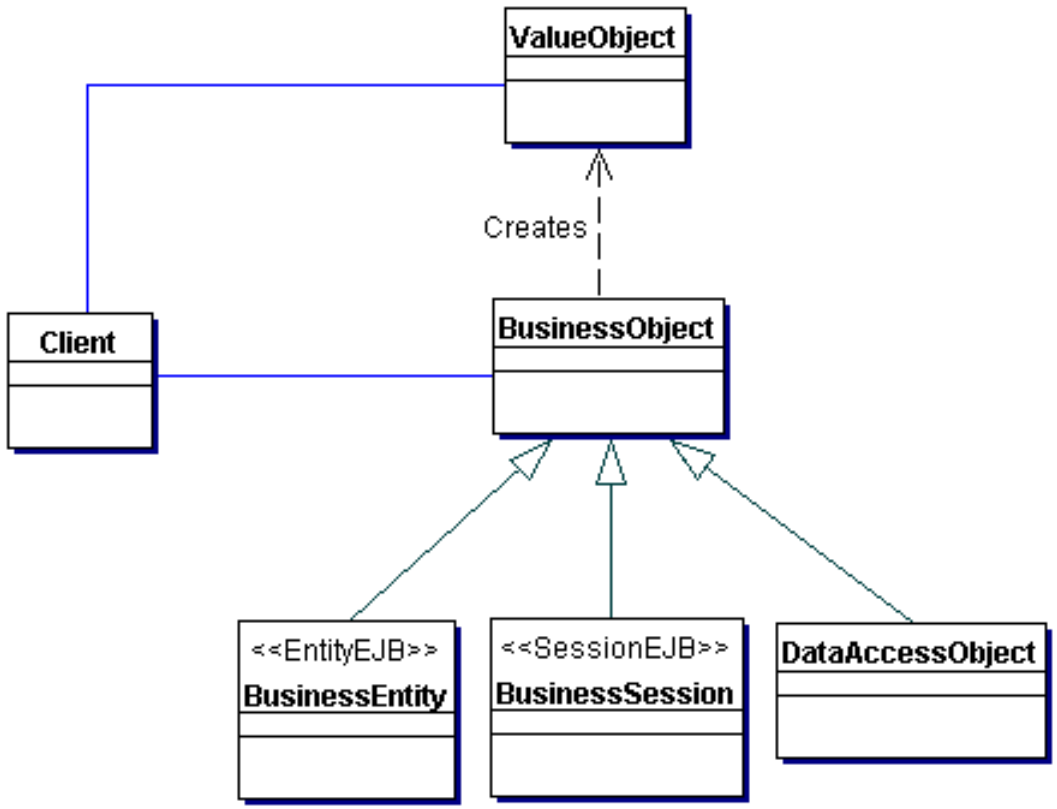
Solução: Value Object



Descrição da solução

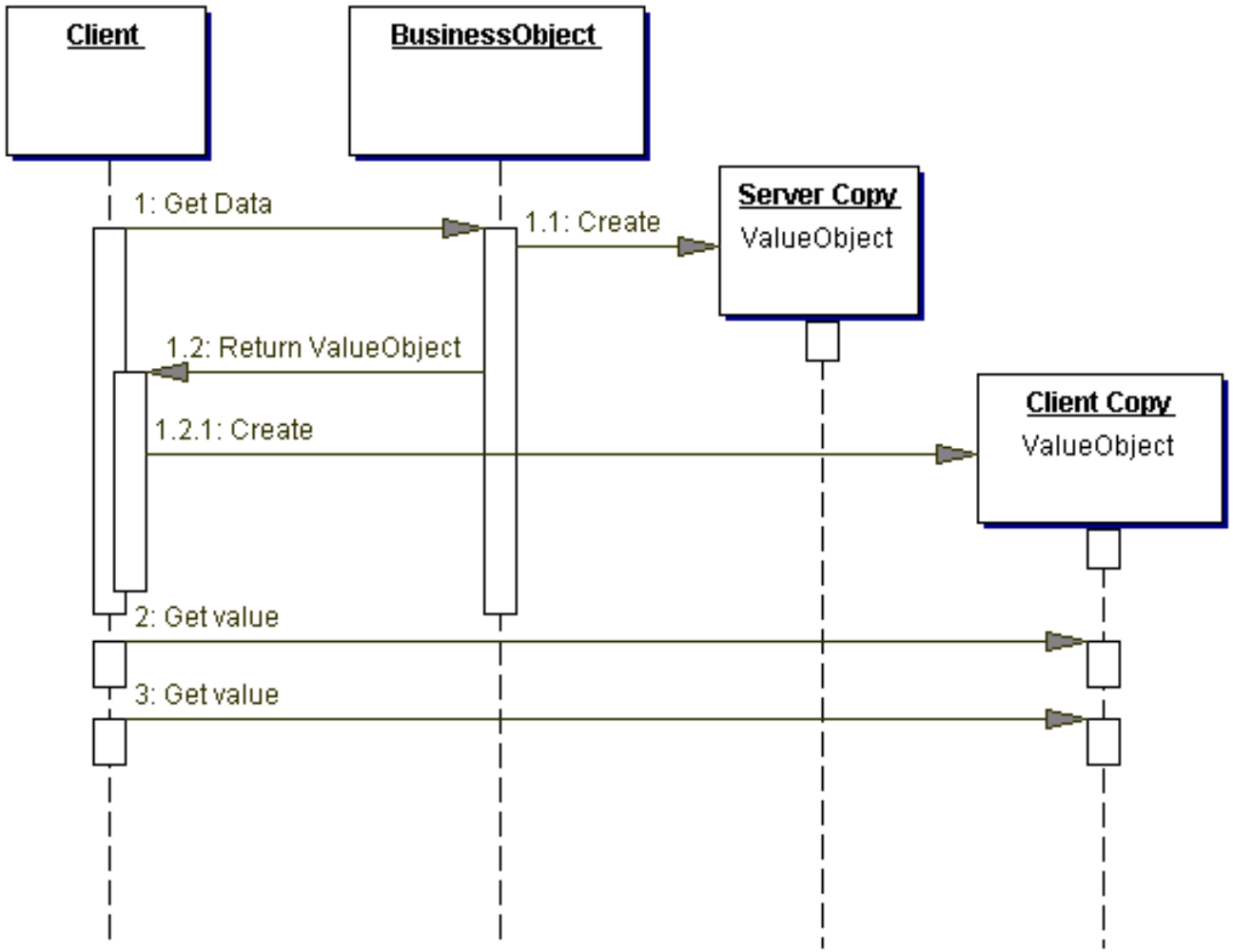
- *Uma única chamada remota é necessária para transferir o Value Object*
  - *O cliente pode extrair as informações de interesse através de chamadas locais*
- *Cópia do cliente pode ficar desatualizada*
  - *Value Object é solução indicada para dados read-only ou informações que não são alteradas com freqüência, ou ainda, quando as alterações não são críticas (não afetam o processo)*
- *Objeto alterado pode ser enviado de volta ao servidor*

Estrutura UML



Fonte: [Core]

Diagrama de Seqüência



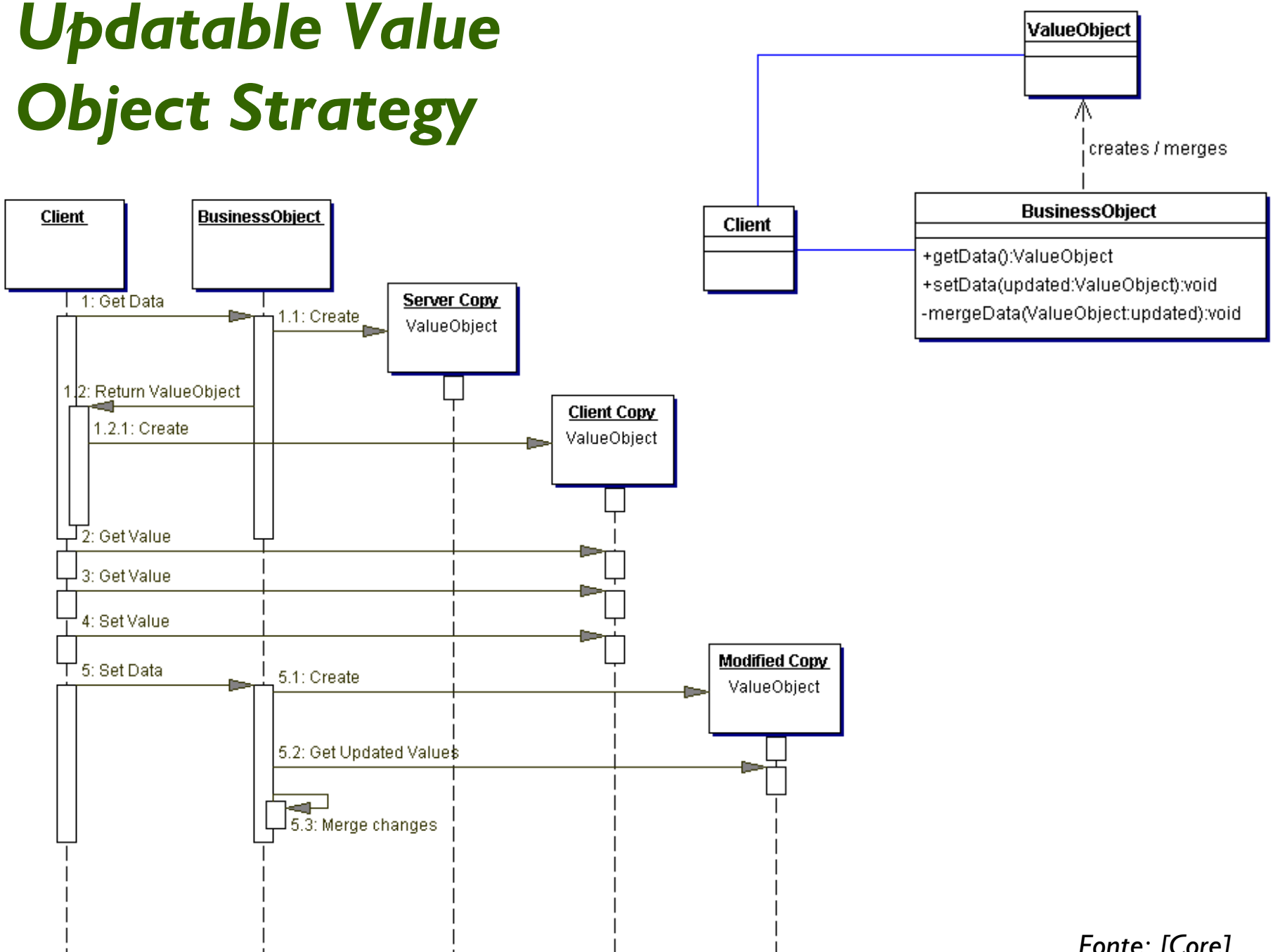
Fonte: [Core]

Melhores estratégias de implementação*

- *Updatable Value Objects Strategy*
  - *Permite a transferência de um objeto para o cliente, a alteração do objeto pelo cliente e sua devolução ao servidor*
- *Multiple Value Objects Strategy*
  - *Permite a criação de Value Objects diferentes a partir de uma mesma fonte*
- *Entity Inherits Value Object Strategy*
  - *Entity Bean herda de uma classe de Value Object*
- *Value Object Factory Strategy*
  - *Suporta a criação dinâmica de Value Objects*

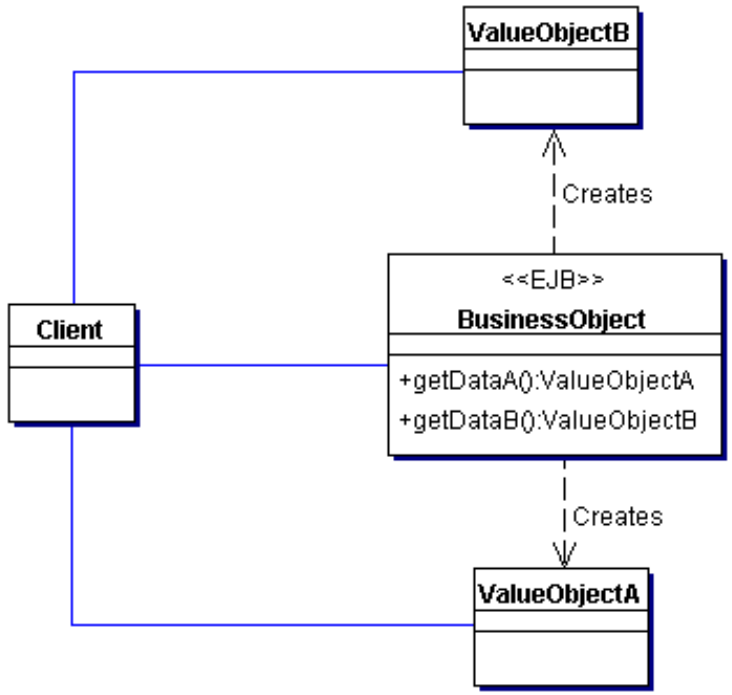
** Veja exemplos de código em exemplos/ejblayer/*

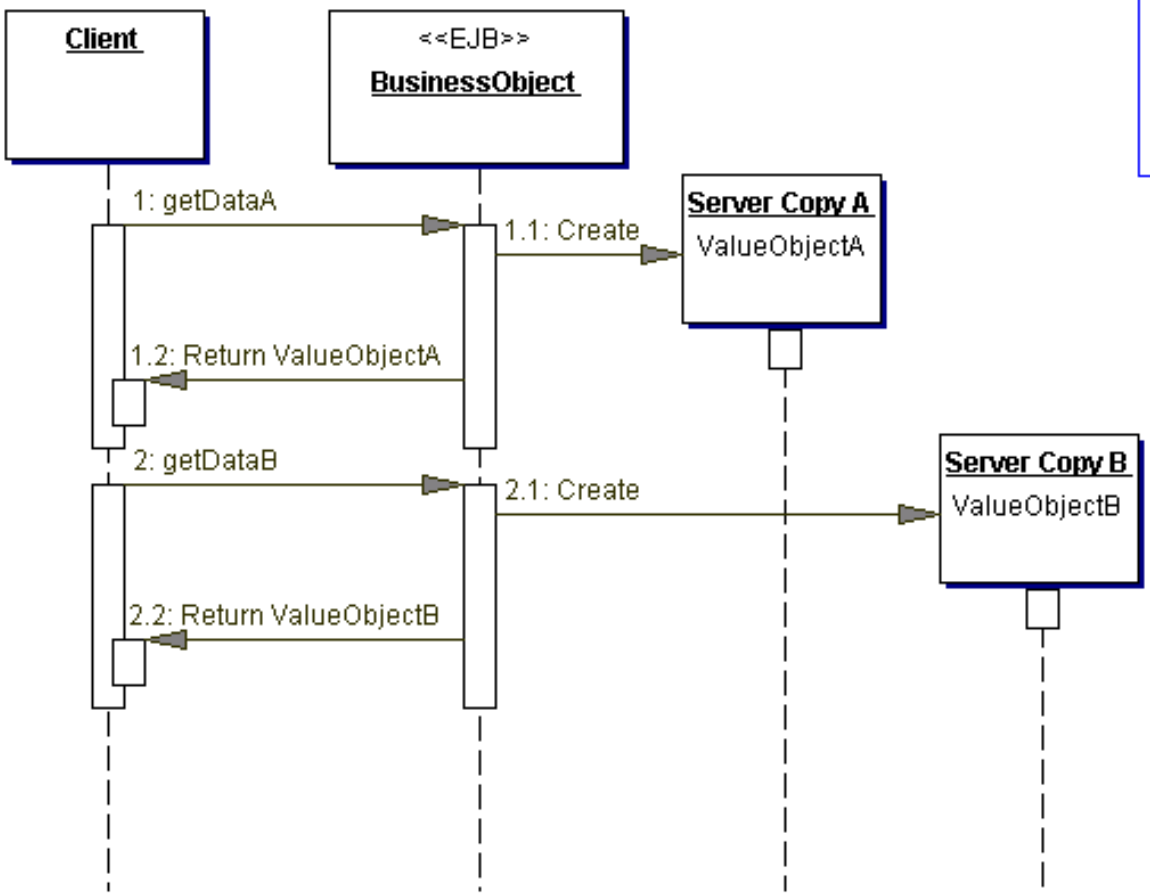
Updatable Value Object Strategy



Fonte: [Core]

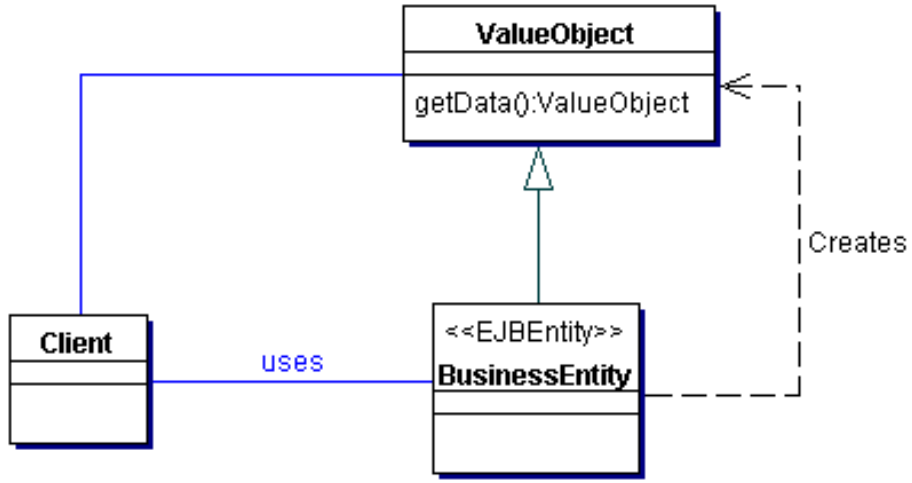
Multiple Value Object Strategy

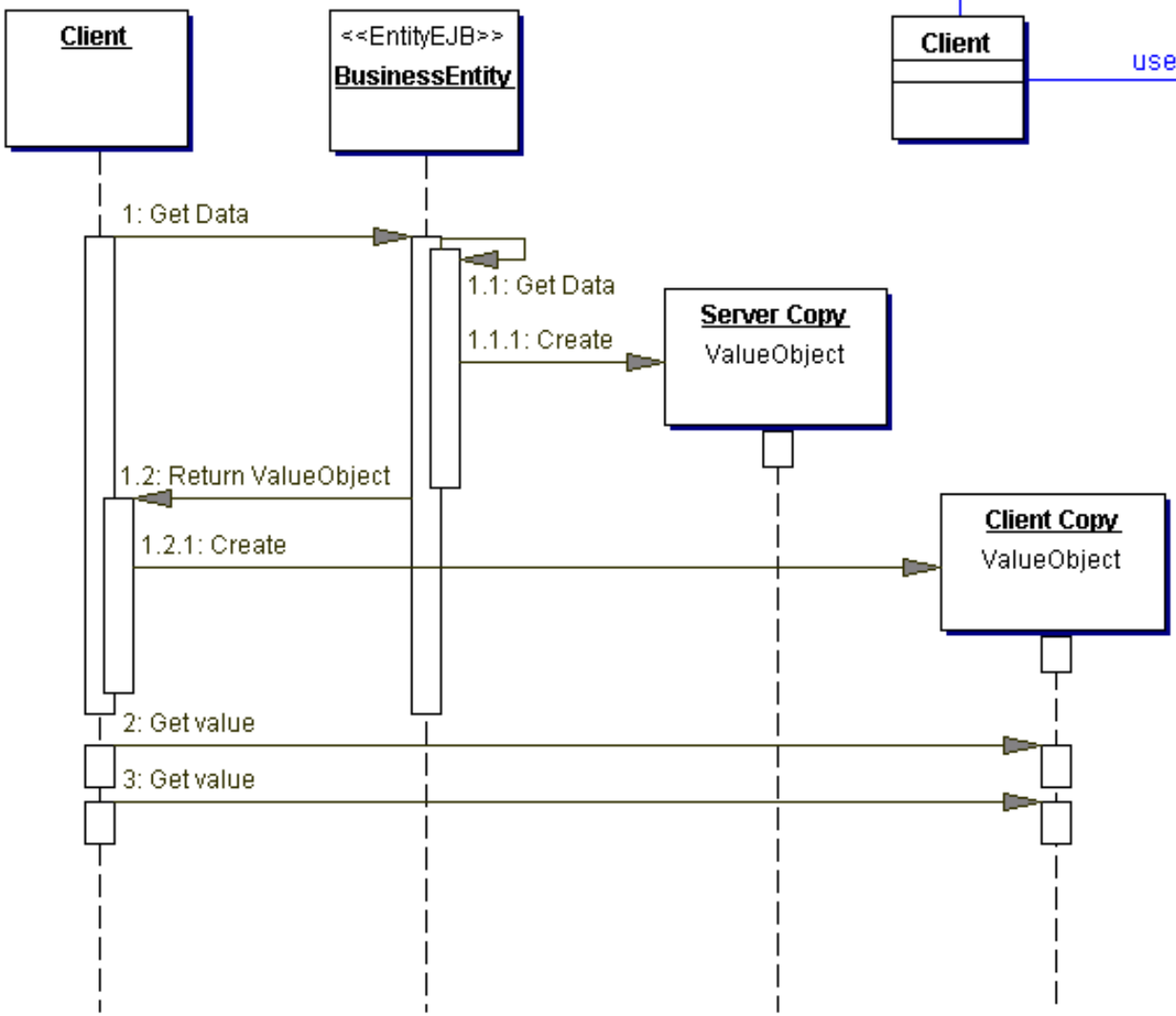




Fonte: [Core]

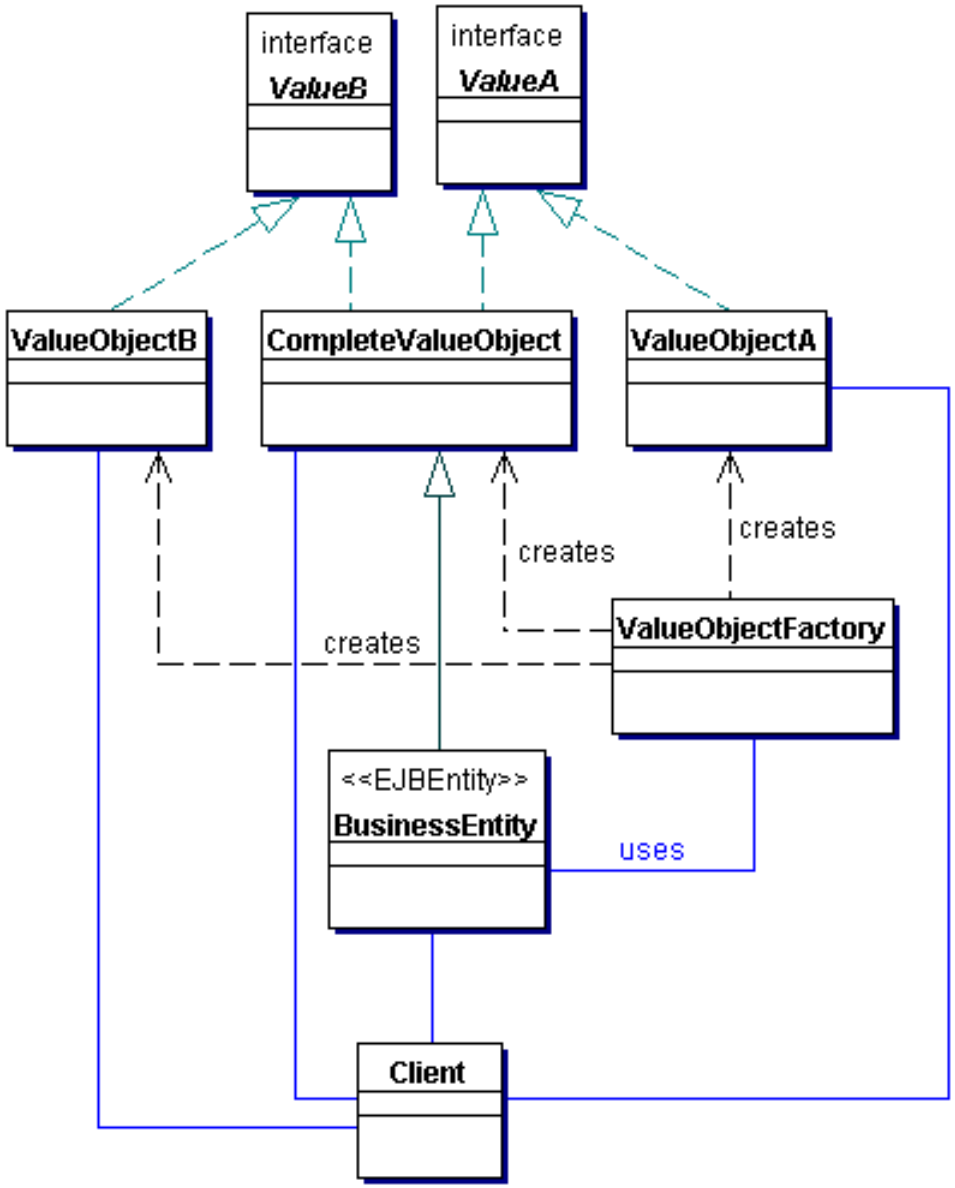
Entity Inherits Value Object Strategy





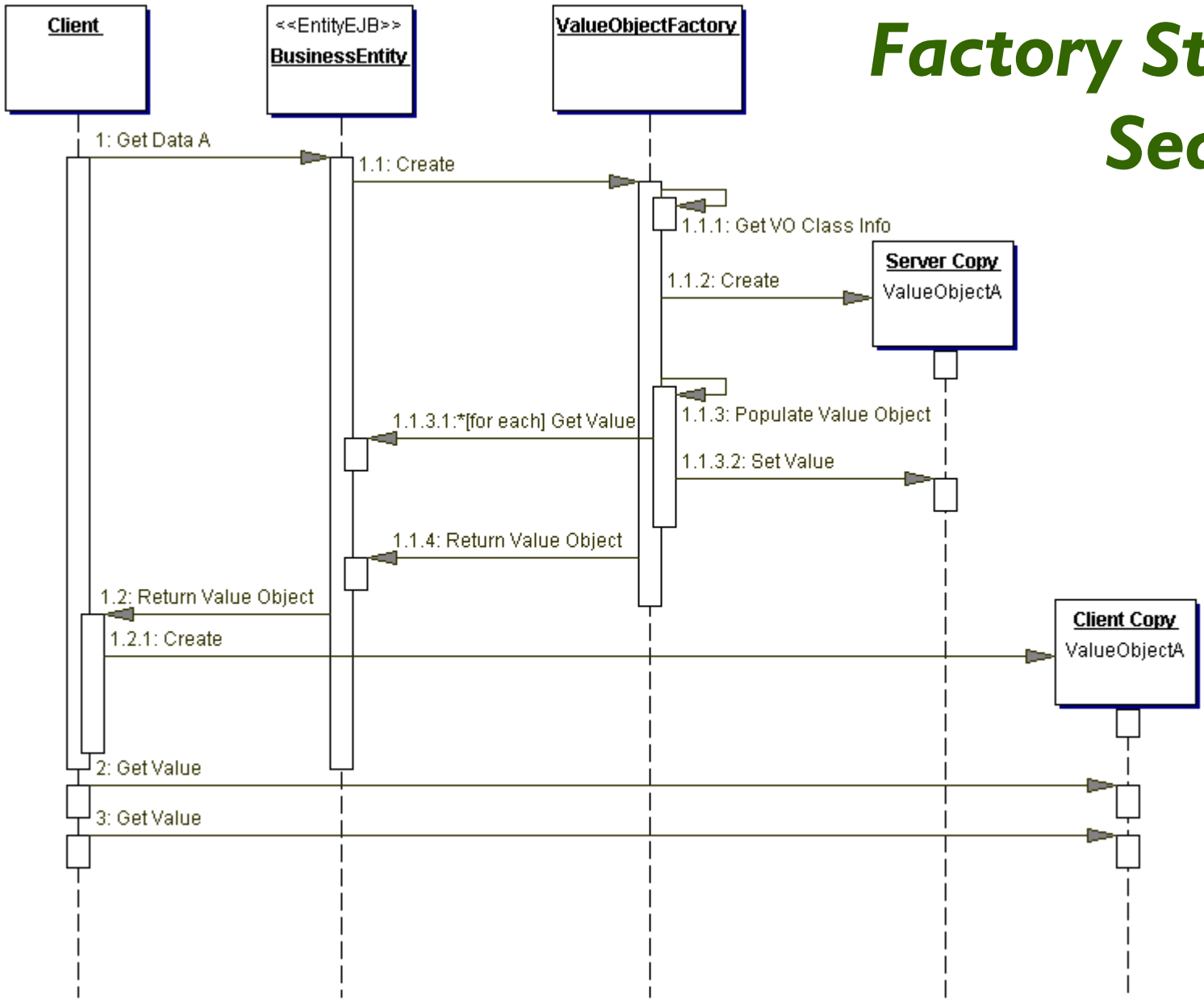
Fonte: [Core]

Value Object Factory Strategy: Estrutura



Fonte: [Core]

Value Object Factory Strategy: Seqüência



Fonte: [Core]

Conseqüências

- *Simplifica Entity Bean 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

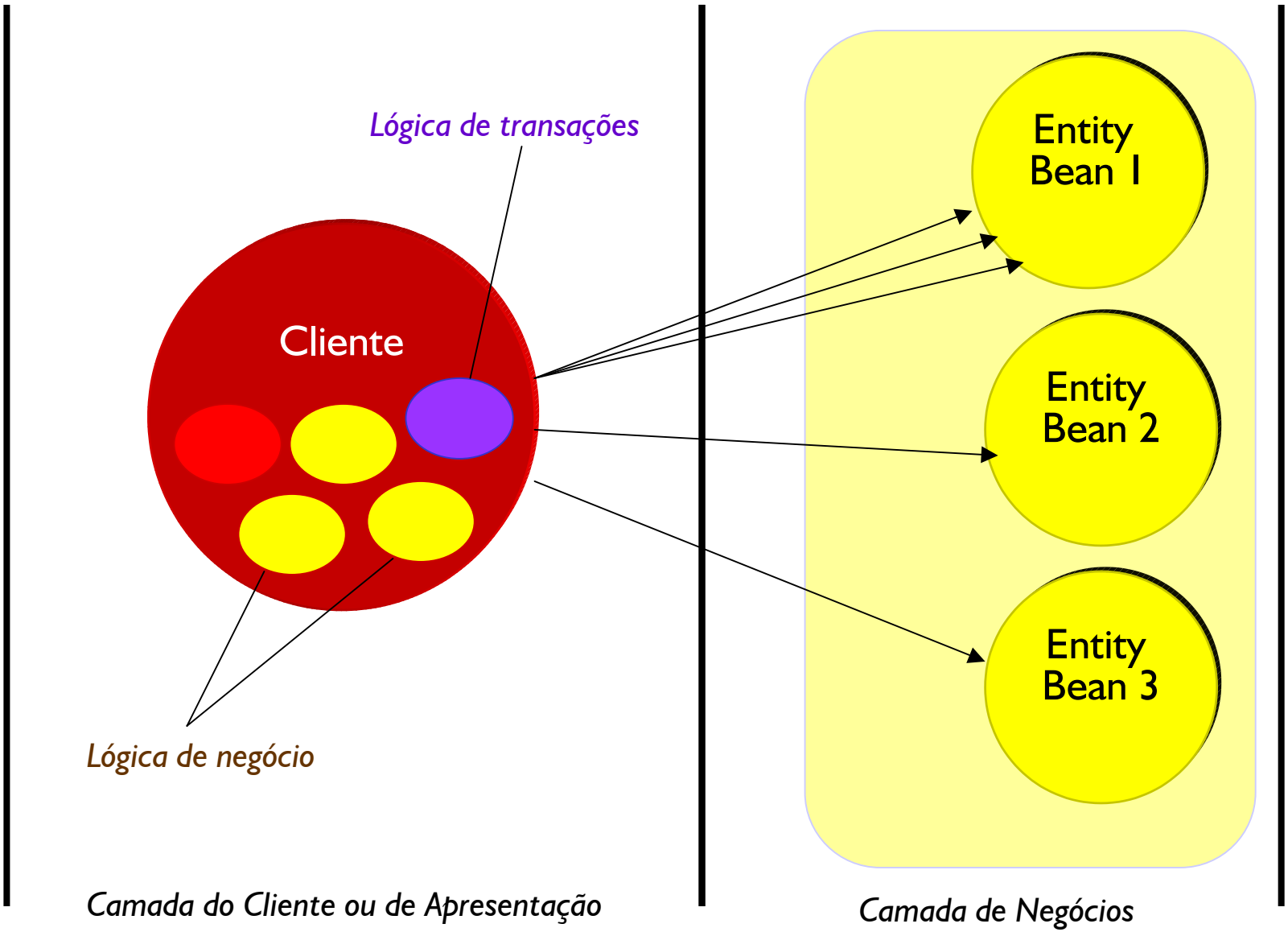
- *1. Analise a implementação das estratégias de Value Object (código 8.3 a 8.14)*
- *2. Refatore a aplicação em ejblayer/vo/ para que utilize Value Object*
  - *a) Crie um Value Object que representa uma cópia do objeto MensagemEntityBean*
  - *b) Implemente no Façade um método que retorne o Value Object 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.*

9

Session Façade

Objetivo: Simplificar a interface do cliente de enterprise beans e controlar o acesso e a comunicação entre entity beans. Session Façade representa uma função ou várias funções exercidas por um sistema.

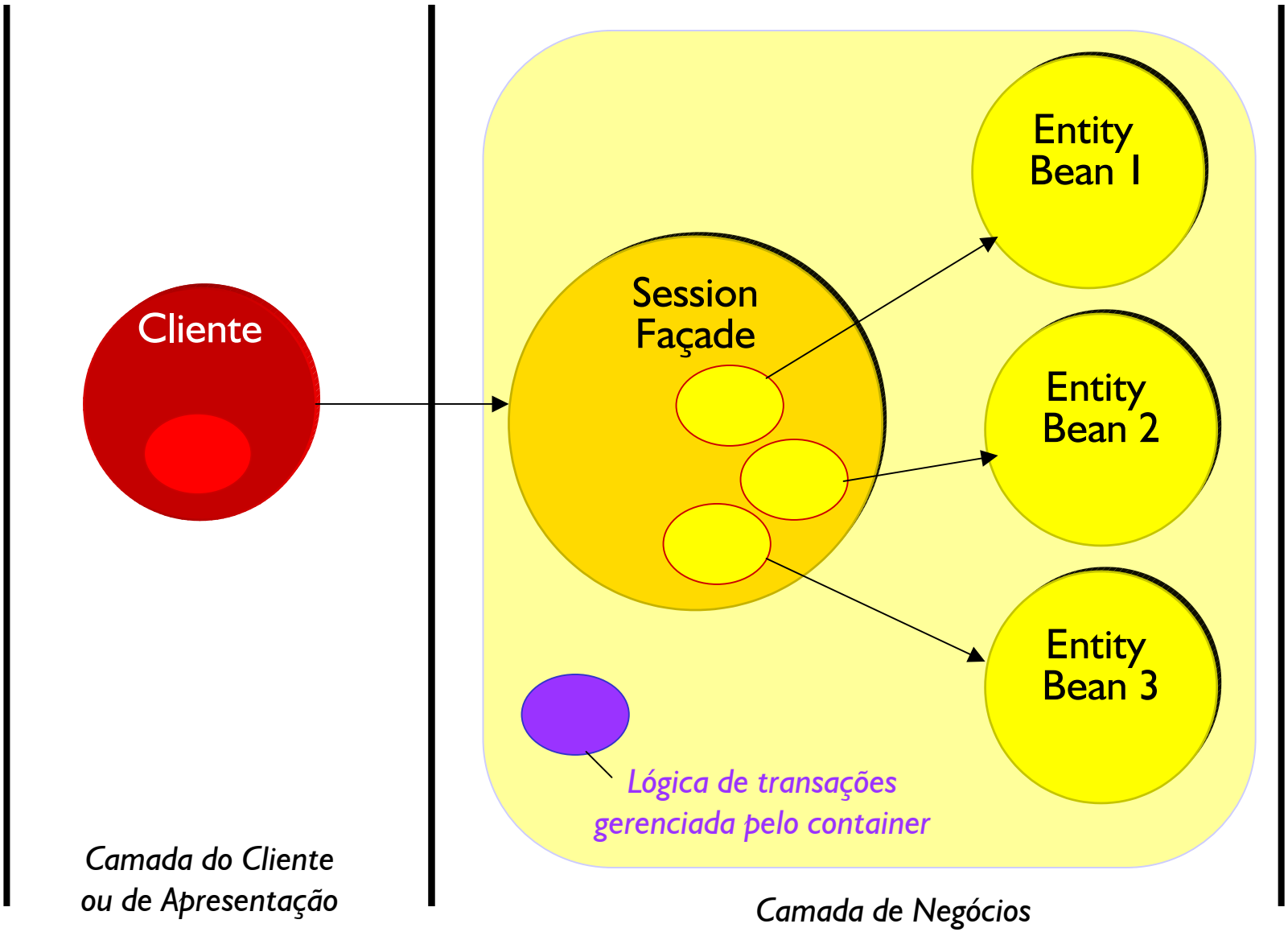
Problema


Lógica de transações
Cliente
Lógica de negócio
Entity Bean 1
Entity Bean 2
Entity Bean 3
Camada do Cliente ou de Apresentação
Camada de Negócios

Descrição do problema

- *Cliente precisa de serviços prestados pela camada de negócio*
- *A camada de negócios está publicamente exposta através da interface de Entity Beans*
- *O cliente precisa descobrir quais beans utilizar, precisa localizá-los (JNDI), tratar eventuais exceções, controlar seus relacionamentos e controlar sua lógica de exceções para cada requisição*
  - *Interface complexa e difícil de usar*
  - *Fortemente acoplada ao modelo de objetos (vulnerável)*

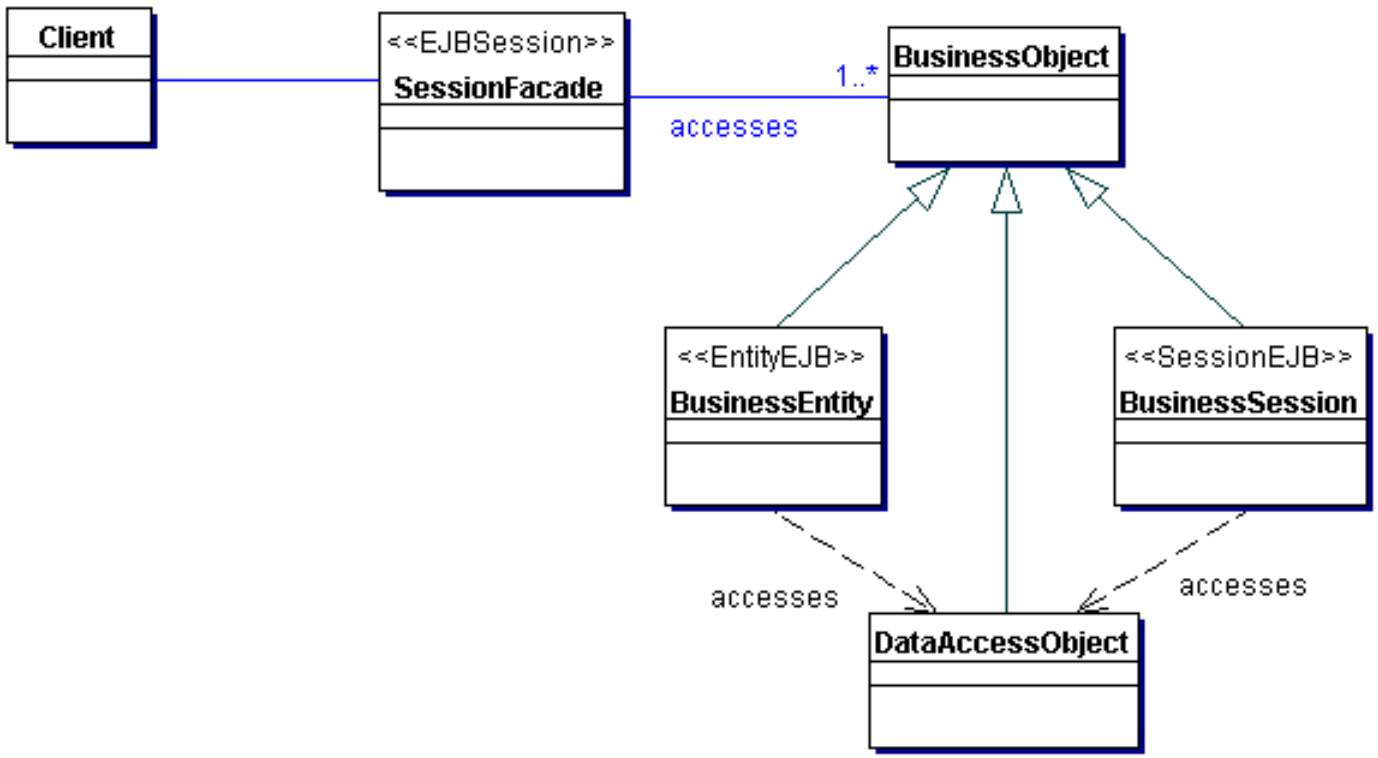
Solução: Session Façade



Descrição da solução

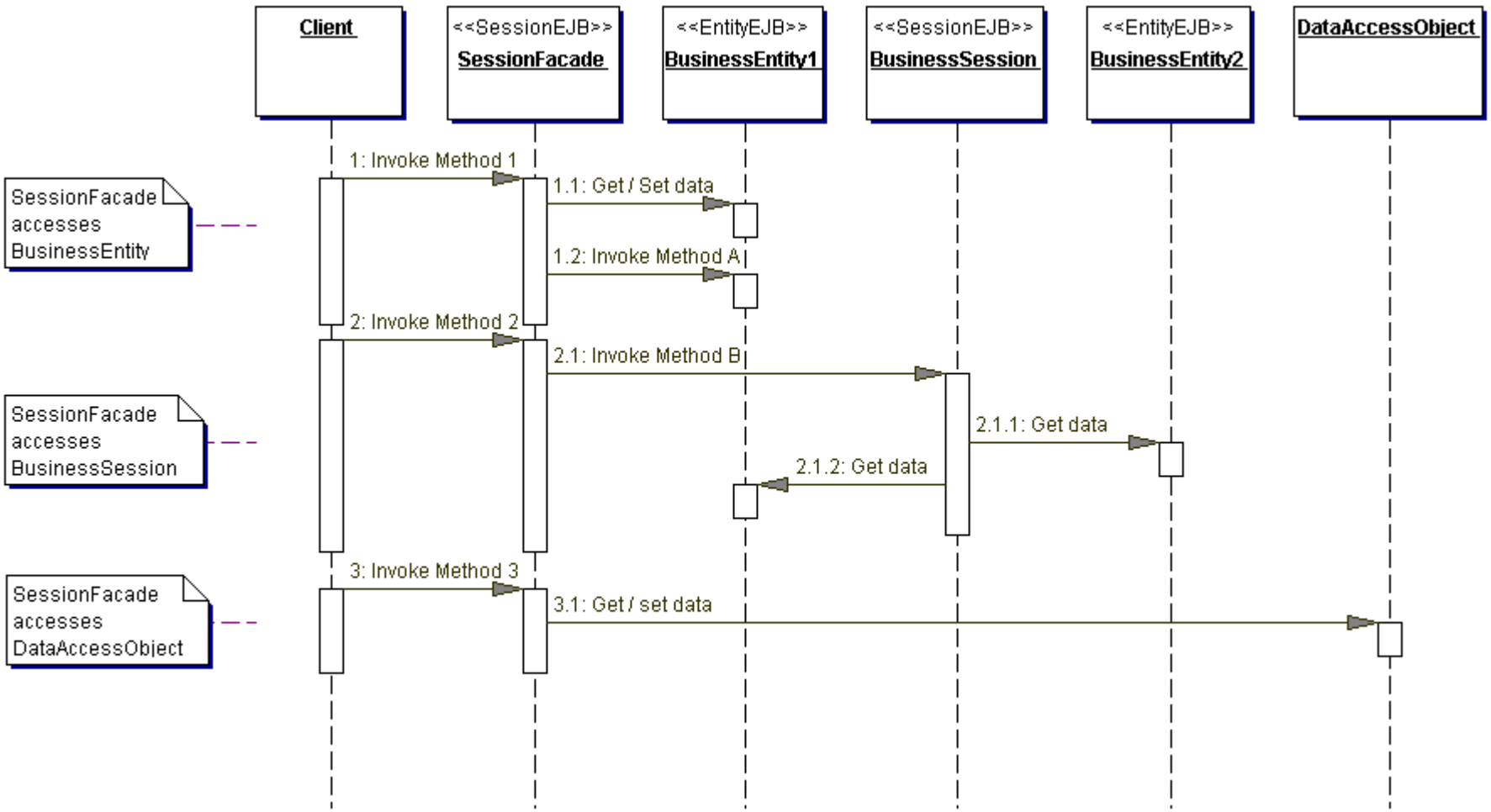
- *Mover a lógica de negócio de interação com Entity Beans para fora do cliente*
  - *Implementar a fachada do cliente como um Session Bean*
- *Reduz-se o número de chamadas remotas do cliente*
  - *As chamadas estão concentradas na fachada do Session Bean*
- *Lógica de transações pode ser implementada no Session Bean ou gerenciada pelo Container (CMT)*
  - *O cliente não é mais responsável pelo controle de transações*

Estrutura UML



Fonte: [Core]

Diagrama de Seqüência



Fonte: [Core]

Melhores estratégias de implementação

- Stateless Session Façade Strategy
  - Implementada com Stateless Session Bean
  - Ideal se métodos não tem relação alguma entre si (no que se refere a utilização de estado anterior)
- Stateful Session Façade Strategy
  - Implementada com Stateful Session Bean
  - Necessária se uma operação precisar de mais de uma chamada de método para completar
- Data Access Object Strategy
  - Usa DAO como meio de persistência
  - Ideal para sistemas simples sem Entity Beans

Conseqüências

- *Introduz camada controladora entre camada de negócios e clientes*
- *Expõe interface uniforme*
- *Reduz acoplamento*
- *Melhora a performance*
  - *Seleciona apenas métodos de interesse aos clientes*
  - *Reduz o número de chamadas*
- *Centraliza controle de segurança e transações*
- *Reduz a interface visível aos clientes*

Exercícios

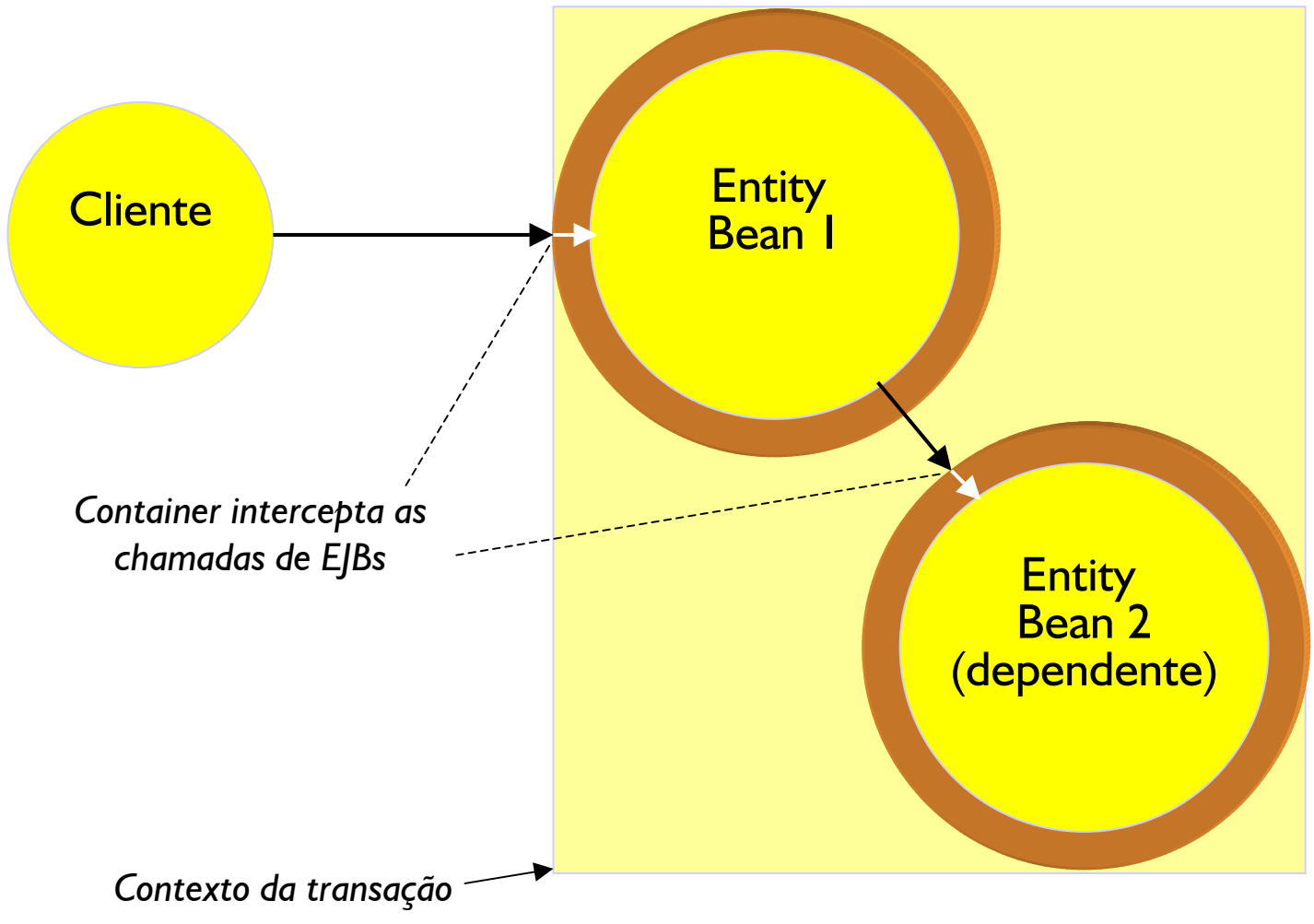
- *1. Analise a implementação das estratégias de Session Façade (código 8.15 e 8.16)*
- *2. Refatore a aplicação em ejblayer/sf/ para que utilize Session Façade:*
  - *a) Crie uma Session Façade que implante toda a funcionalidade do Servlet existente*
  - *b) Faça o servlet se comunicar com o Façade*
  - *c) Faça o Façade chamar os métodos utilitários*
- *3. Use um DAO entre o Session Façade e os dados*
- *4. Use um Business Delegate entre o Session Façade e o cliente (Servlet)*

10

Composite Entity

Objetivo: Modelar, representar e gerenciar um conjunto de objetos persistentes relacionados em vez de representá-los como entity beans fine-grained. Um Composite Entity representa um grafo de objetos.

Problema

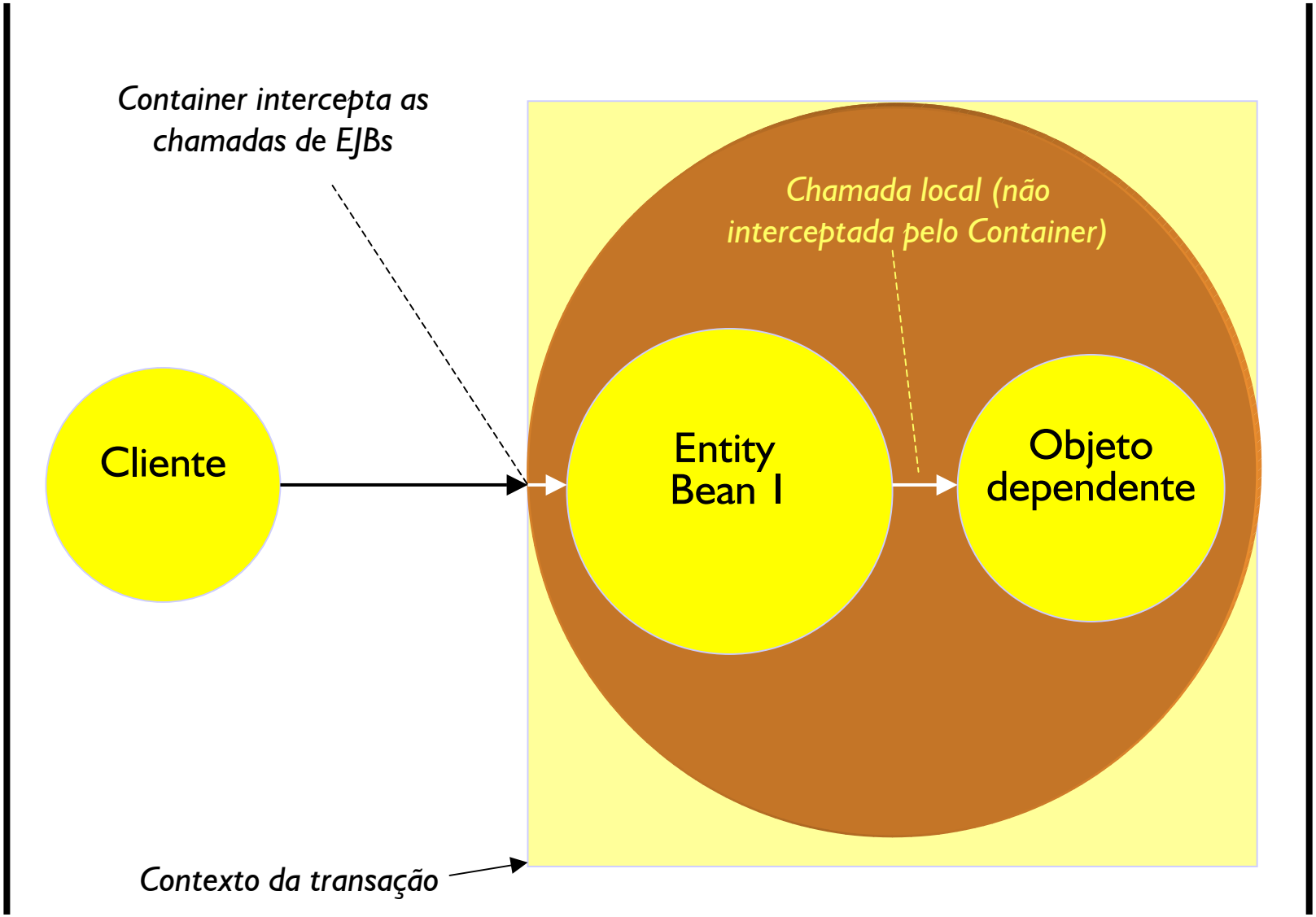

Cliente
Entity
Bean 1
Entity
Bean 2
(dependente)
Container intercepta as
chamadas de EJBs
Contexto da transação

Descrição do problema

- *Relacionamentos entity-to-entity devem ser evitados*
  - *Difíceis de manter: acoplamento forte*
  - *Quaisquer chamadas são interceptadas pelo container, trazendo overhead desnecessário*
  - *Container assume que Entity Bean é objeto remoto e faz marshalling e unmarshalling de todas as chamadas**
  - *Contexto da transação inclui toda a corrente de Entity Beans dependentes, causando problemas de performance e possível deadlock*
- *É preciso eliminar o relacionamento entity-entity*

* *Container pode otimizar as chamadas, mas isto depende do fabricante*

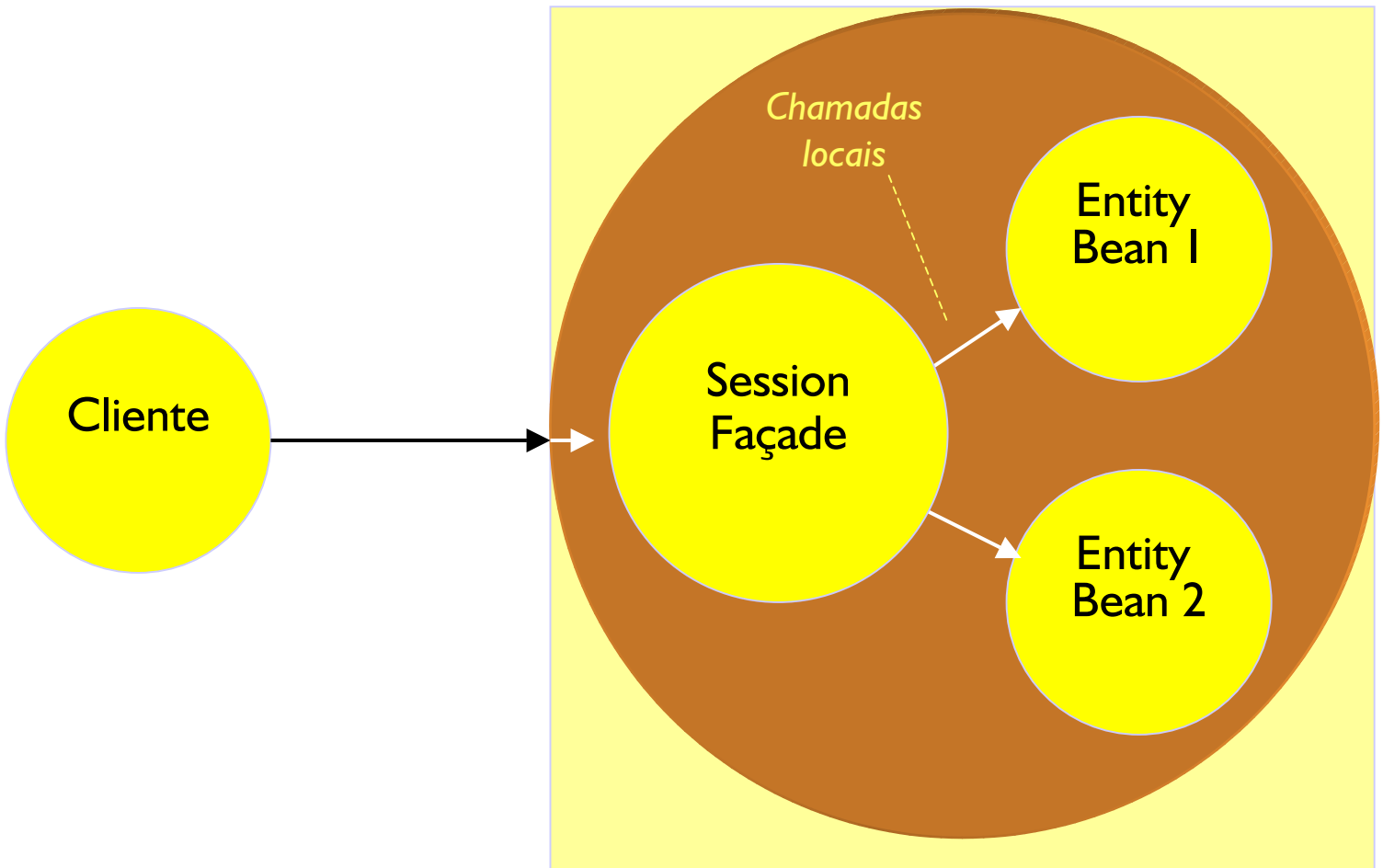
Solução: Composite Entity



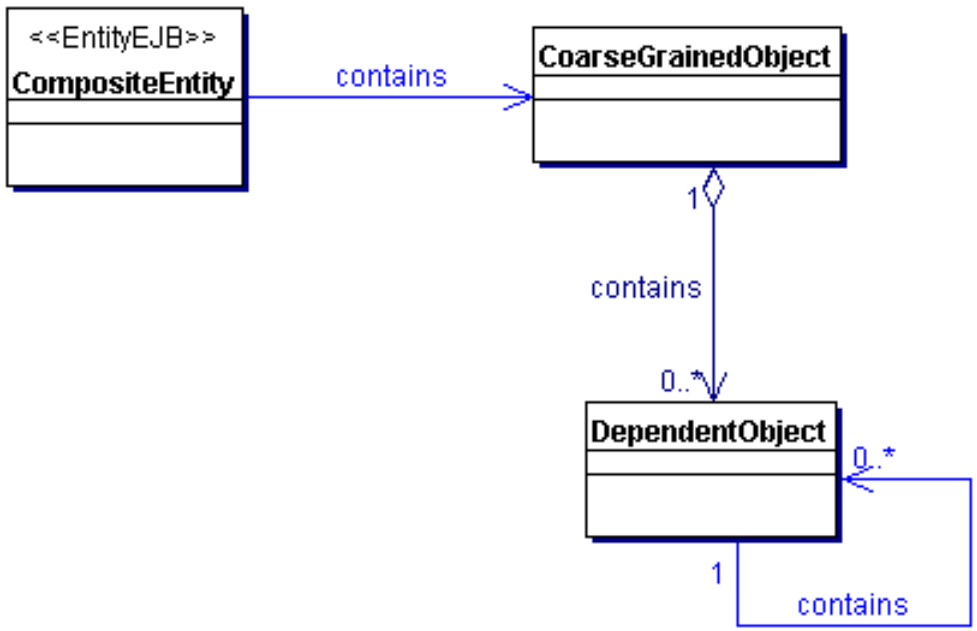
Descrição da solução

- *Transformar o Entity Bean dependente em um objeto Java comum*
- *Entity beans não devem modelar todos os objetos (fine-grained), apenas os principais*
  - *Dependências devem ser objetos comuns*
  - *Chamadas de Entity Beans a seus objetos dependentes não são interceptadas pelo container, resultando em melhor performance*
- *Se houver necessidade que outro objeto seja mesmo um Entity Bean, mover a lógica de comunicação para um Session Bean (solução 2)*

Solução 2: Session Façade

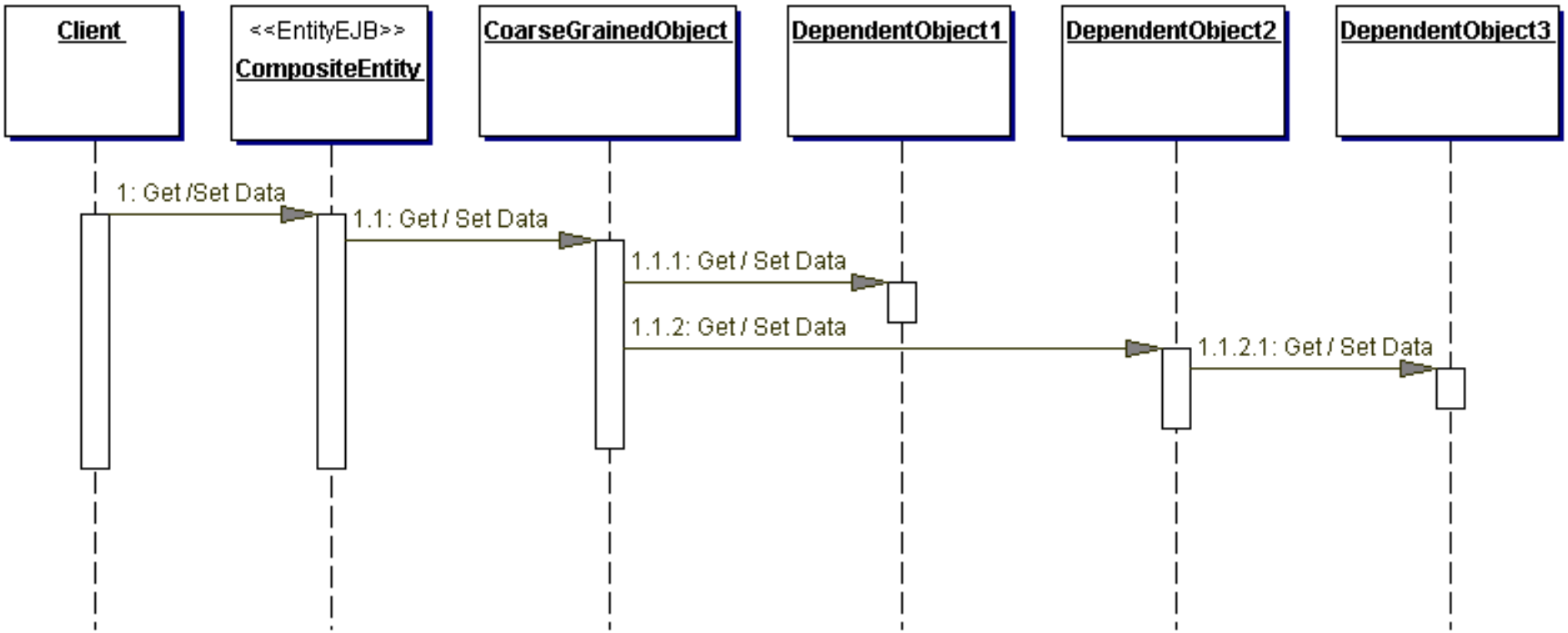


Estrutura UML



Fonte: [Core]

Diagrama de Seqüência



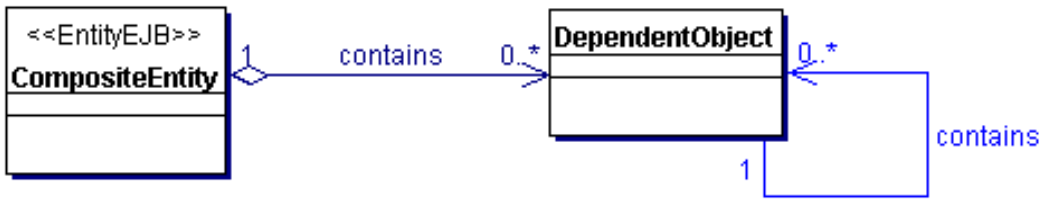
Fonte: [Core]

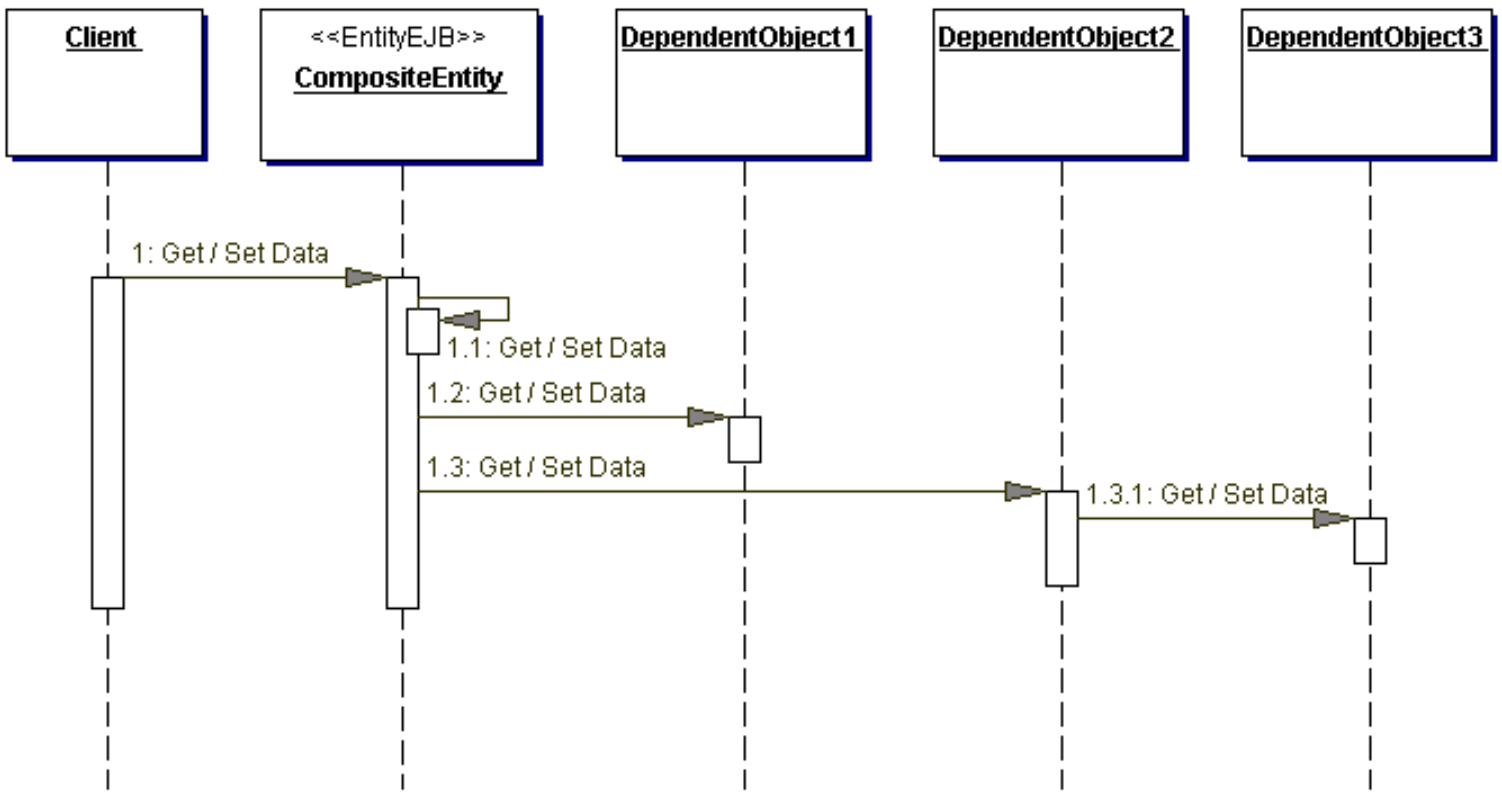
Melhores estratégias de implementação

- *Composite Entity Contains Coarse-Grained Object*
  - *O objeto é contido no bean. Relacionamentos entre o objeto e outros objetos é preservado: principal estratégia*
- *Composite Entity Implements Coarse-Grained Object*
  - *O próprio bean é o objeto. Os objetos dependentes são os atributos do bean.*
- *Composite Value Object Strategy*
- *Para containers pré-EJB 2.0*
  - *Lazy Loading Strategy*
  - *Store Optimization (Dirty Marker) Strategy*

** Exemplos de código de [Core]*

CE Implements Coarse-grained Object





Conseqüências

- *Elimina relacionamento entre Entity Beans*
- *Reduz a quantidade de Entity Beans*
- *Melhora a performance da rede*
- *Reduz dependência em esquema de BD*
- *Aumenta granularidade dos objetos*
- *Facilita a criação de VOs compostos*

Exercícios

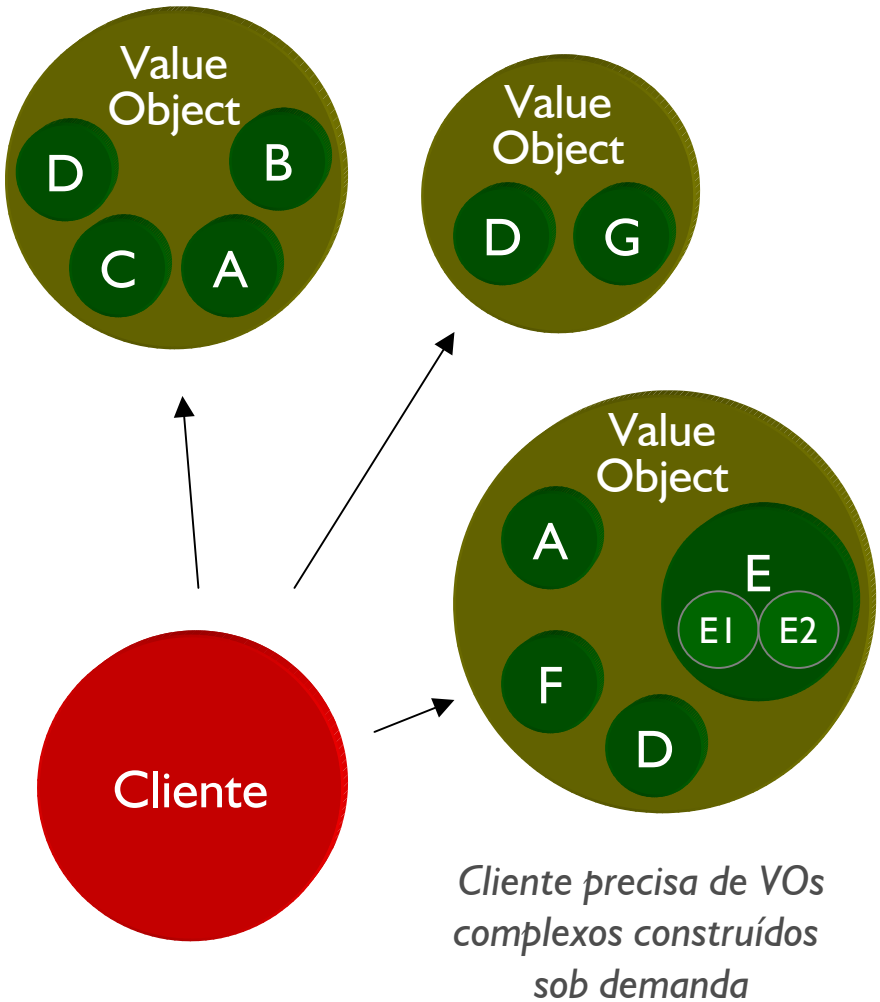
- *1. Analise a implementação das estratégias de Composite Entity (código 8.18 a 8.23)*
- *2. Refatore a aplicação em ejblayer/ce/ para que utilize Composite Entity*
  - *a) Desenhe o relacionamento atual entre os entity beans. Será que todos os objetos deveriam ser Entity Beans? Justifique sua resposta.*
  - *b) Utilize a estratégia mais simples para implementar um VO que possa ser enviado pelo cliente, lido, alterado e retornado.*

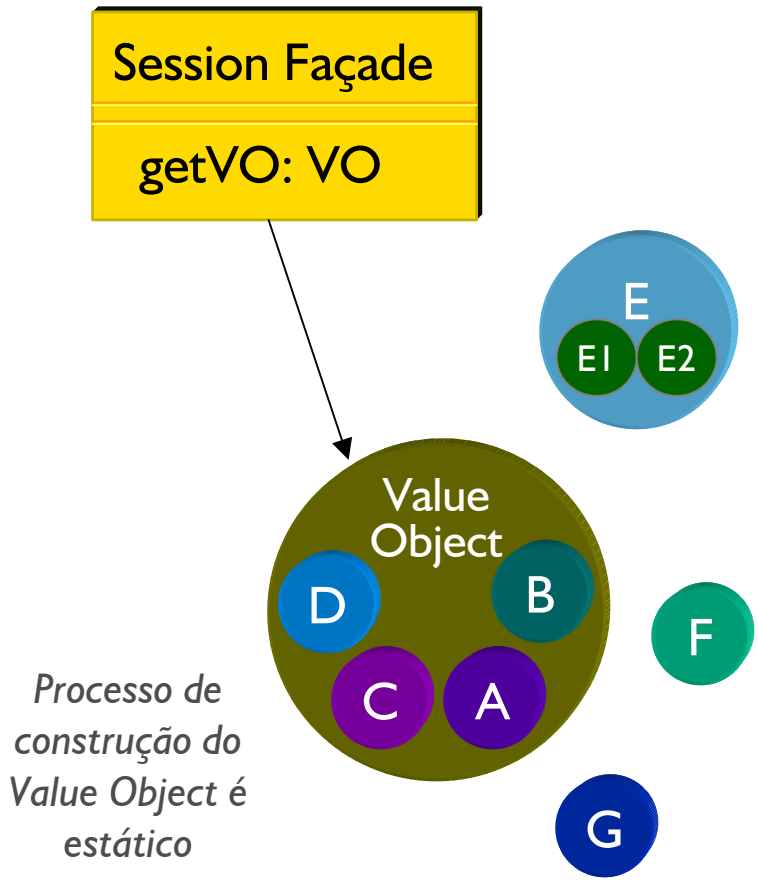
11

Value Object Assembler
ou
Transfer Object Assembler

Objetivo: Construir um modelo ou submodelo de dados requerido pelo cliente. O Value Object Assembler usa Value Objects para recuperar dados de vários objetos de negócio e outros objetos que definem o modelo ou parte dele.

Problema

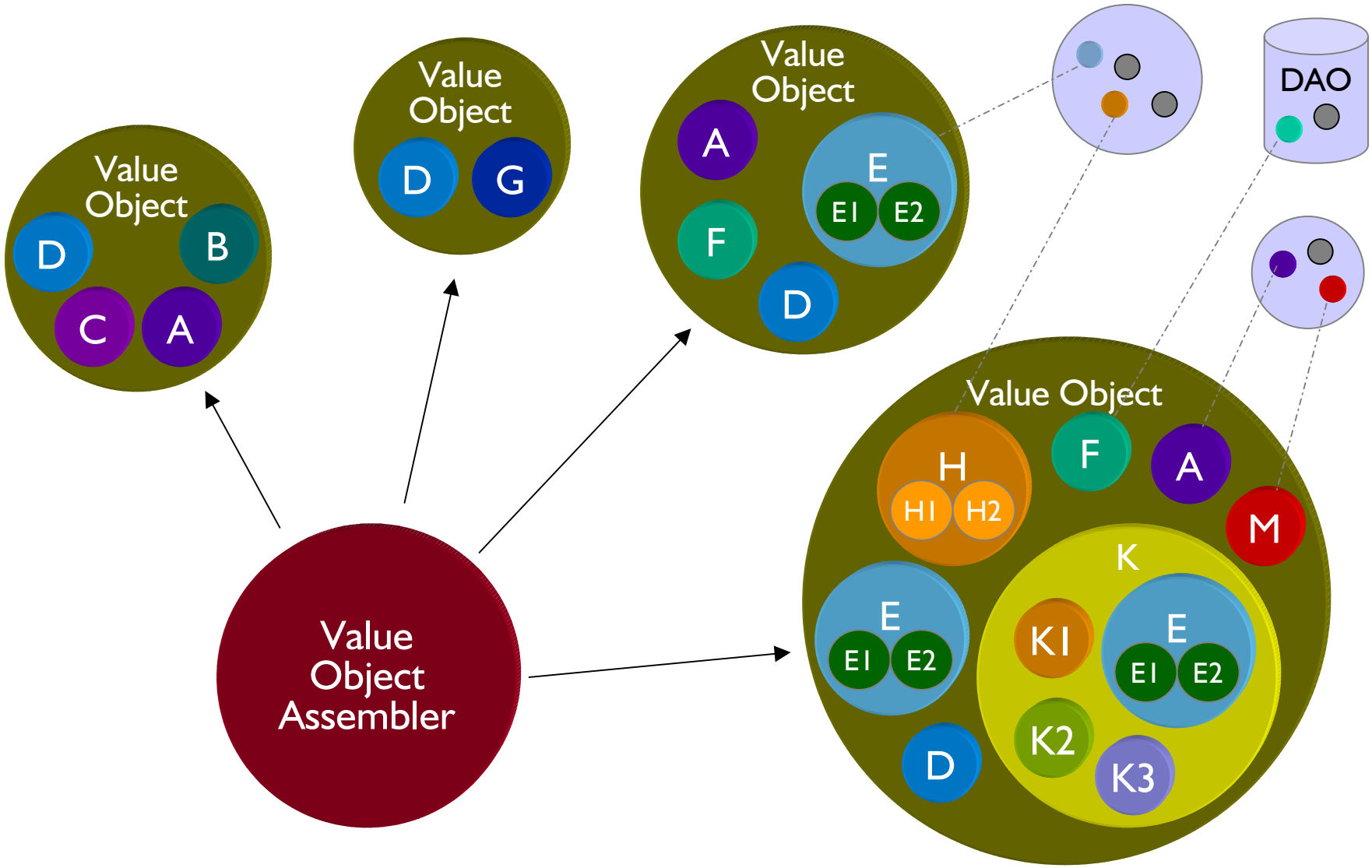

Value Object
D
B
C
A
Value Object
D
G
Value Object
A
E
E1
E2
F
D
Cliente
Cliente precisa de VOs complexos construídos sob demanda
Camada de Apresentação


Session Façade
getVO: VO
E
E1
E2
Value Object
D
B
C
A
F
G
Processo de construção do Value Object é estático
Camada de Negócios

Descrição do problema

- *O cliente precisa interagir com múltiplos componentes para obter alguns dados de cada um deles*
  - *Chamar vários métodos ou vários Value Objects menores aumenta o tráfego na rede*
  - *Cliente se torna fortemente acoplado à interface*
- *O servidor precisa saber montar componentes "on-the-fly" de acordo com as necessidades do cliente.*

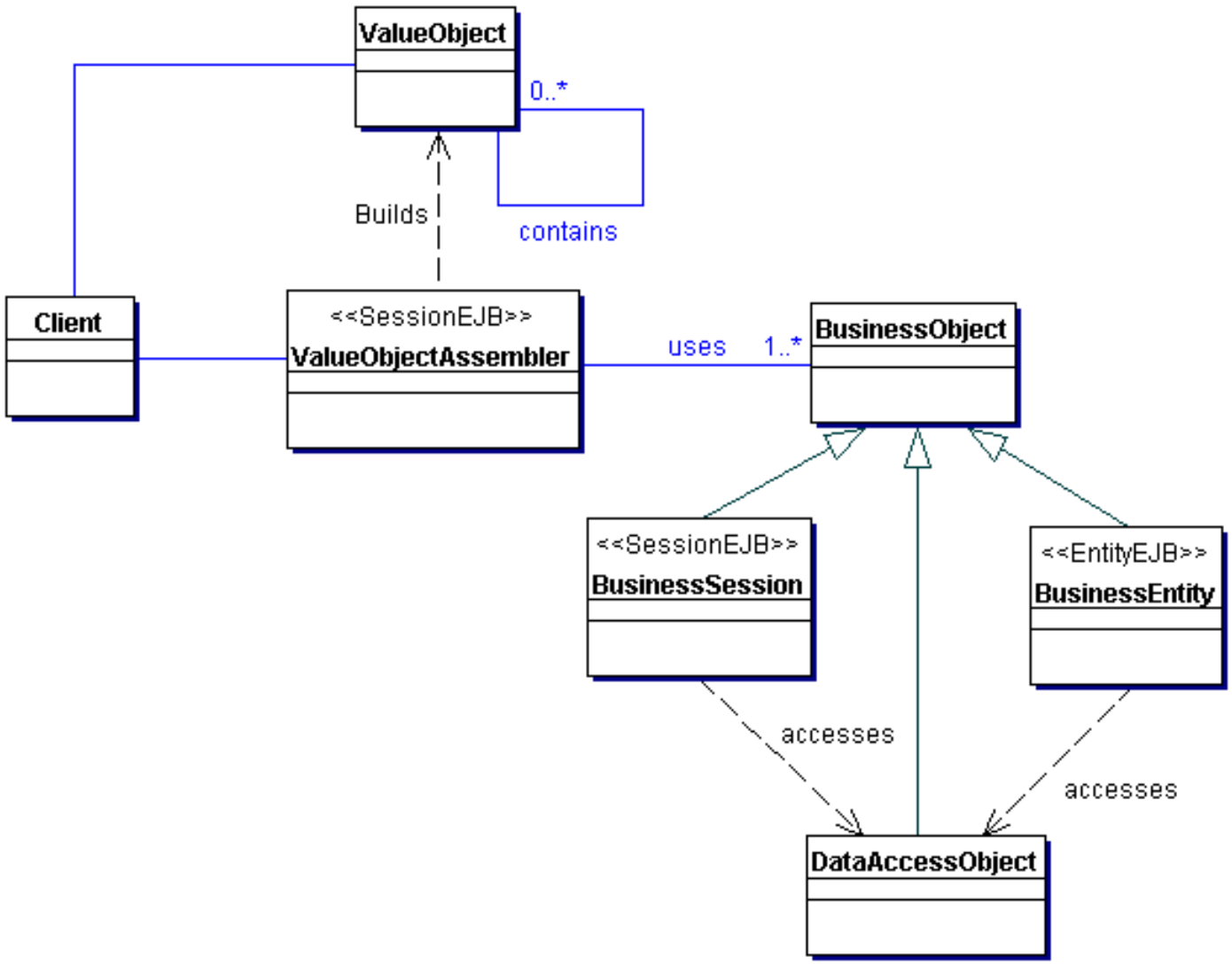
Solução: Value Object Assembler



Descrição da solução

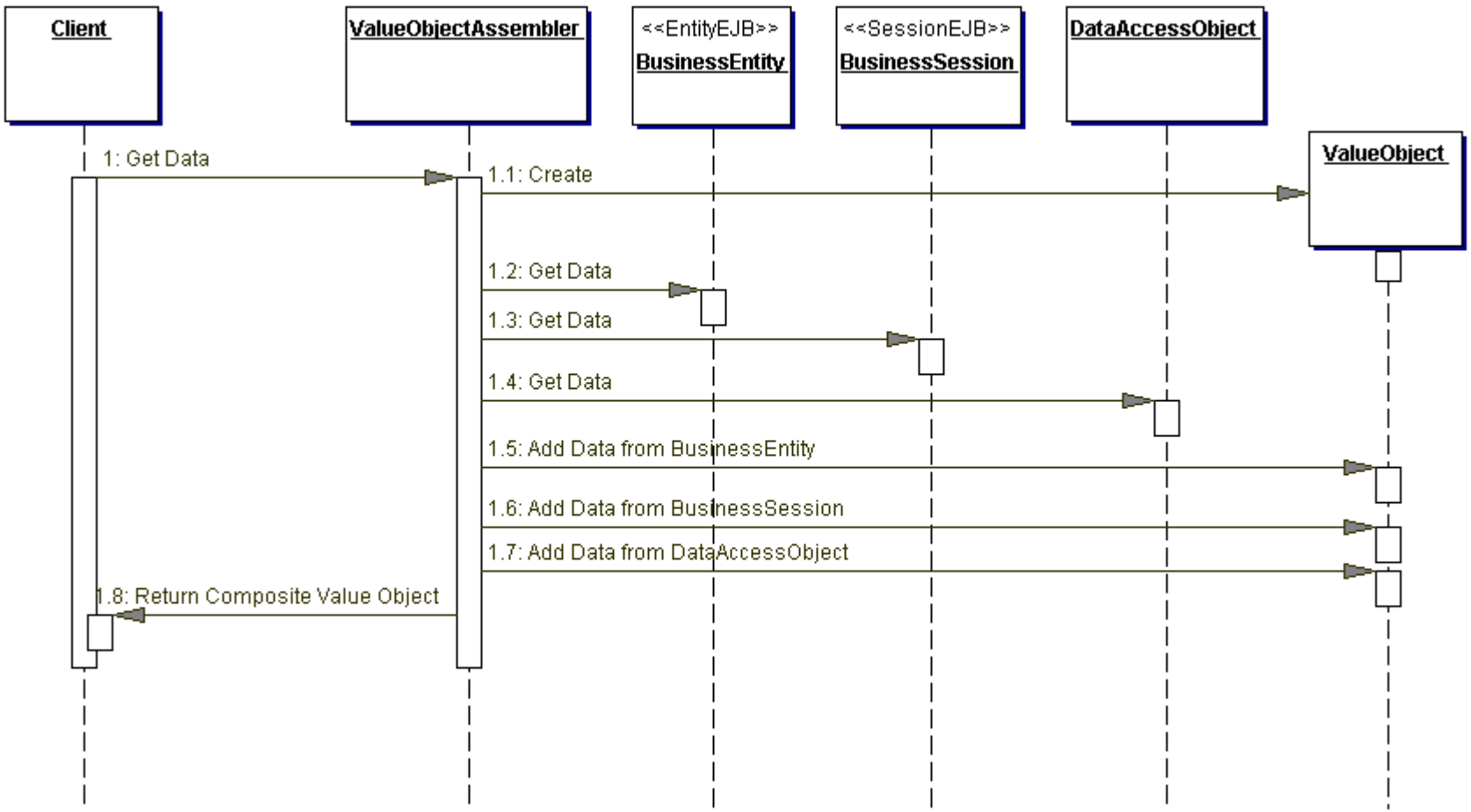
- *O Value Object Assembler constrói um Value Object composto que representa dados de diferentes componentes de negócio*
  - *Os dados são transportados para o cliente através de uma única chamada*
  - *Este Value Object deve ser imutável, devido à sua complexidade: clientes os utilizam apenas para processamento read-only*
- *O servidor obtém value objects de vários componentes e utiliza-os para construir um novo Value Object composto*

Estrutura UML



Fonte: [Core]

Diagramas de Seqüência



Fonte: [Core]

Melhores estratégias de implementação

- *Java Object Strategy*
  - *Implementado como um objeto Java comum*
  - *Servido por um Session Bean*
- *Session Bean Strategy*
  - *Implementado por um Session Bean Stateless*
- *Business Object Strategy*
  - *Pode ser implementado como session bean, entity bean, DAO, objeto Java arbitrário ou outro Value Object Assembler*

Conseqüências

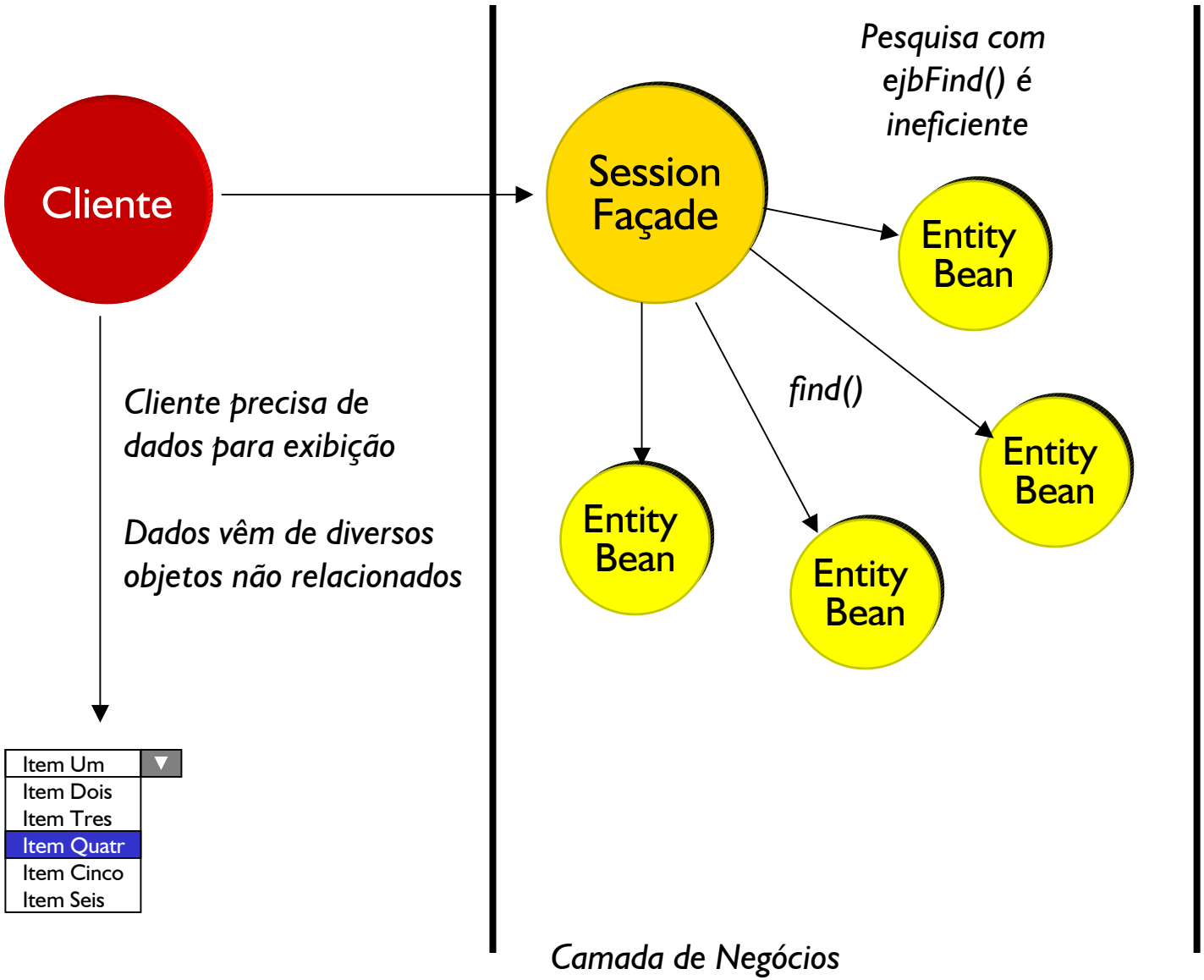
- *Separa lógica de negócio*
- *Reduz acoplamento entre clientes e o modelo da aplicação*
- *Melhora performance de rede*
- *Melhora performance do cliente*
- *Melhora performance das transações*
- *Pode introduzir VOs obsoletos*

12

Value List Handler

Objetivo: Controlar a busca, fazer um cache dos resultados, e oferecer os resultados ao cliente em um cursor (iterator) cujo tamanho e dados adequam-se às requisições do cliente.

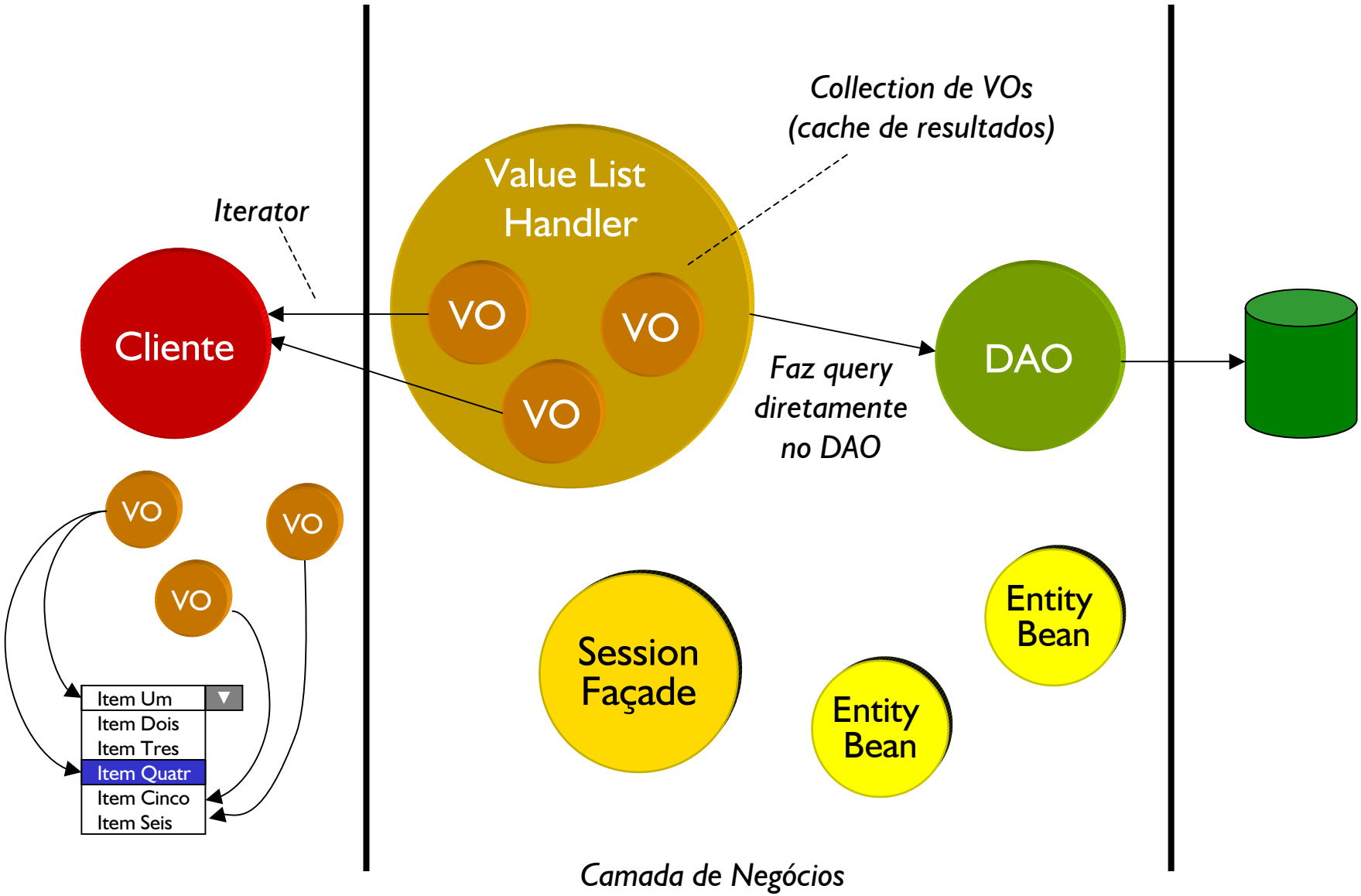
Problema


Cliente
Session Façade
Pesquisa com ejbFind() é ineficiente
Entity Bean
Entity Bean
Entity Bean
Entity Bean
find()
Cliente precisa de dados para exibição
Dados vêm de diversos objetos não relacionados
Item Um
Item Dois
Item Tres
Item Quatr
Item Cinco
Item Seis
Camada de Negócios

Descrição do problema

- *Cliente precisa de dados de diversas fontes (diversos objetos diferentes)*
- *Dados são somente para leitura*
- *Solução: Value Object ou VO Assembler*
- *Mas, dados não são conhecidos de antemão*
  - *Cliente quer fazer um query para obter os dados*
  - *Fazer query com EJBs é ineficiente (métodos finder) ou limitado (EJB-QL)*

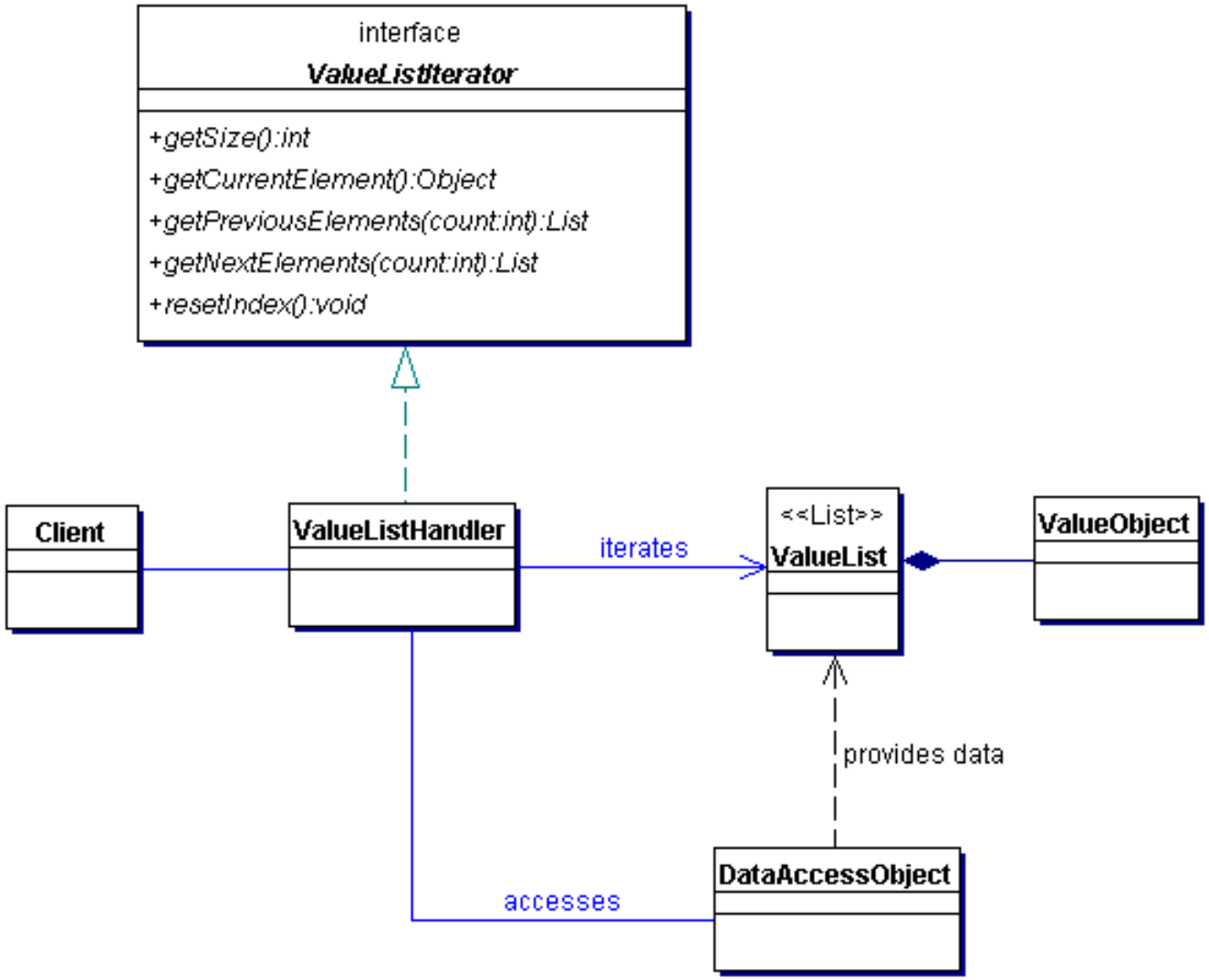
Solução: Value List Handler



Descrição da solução

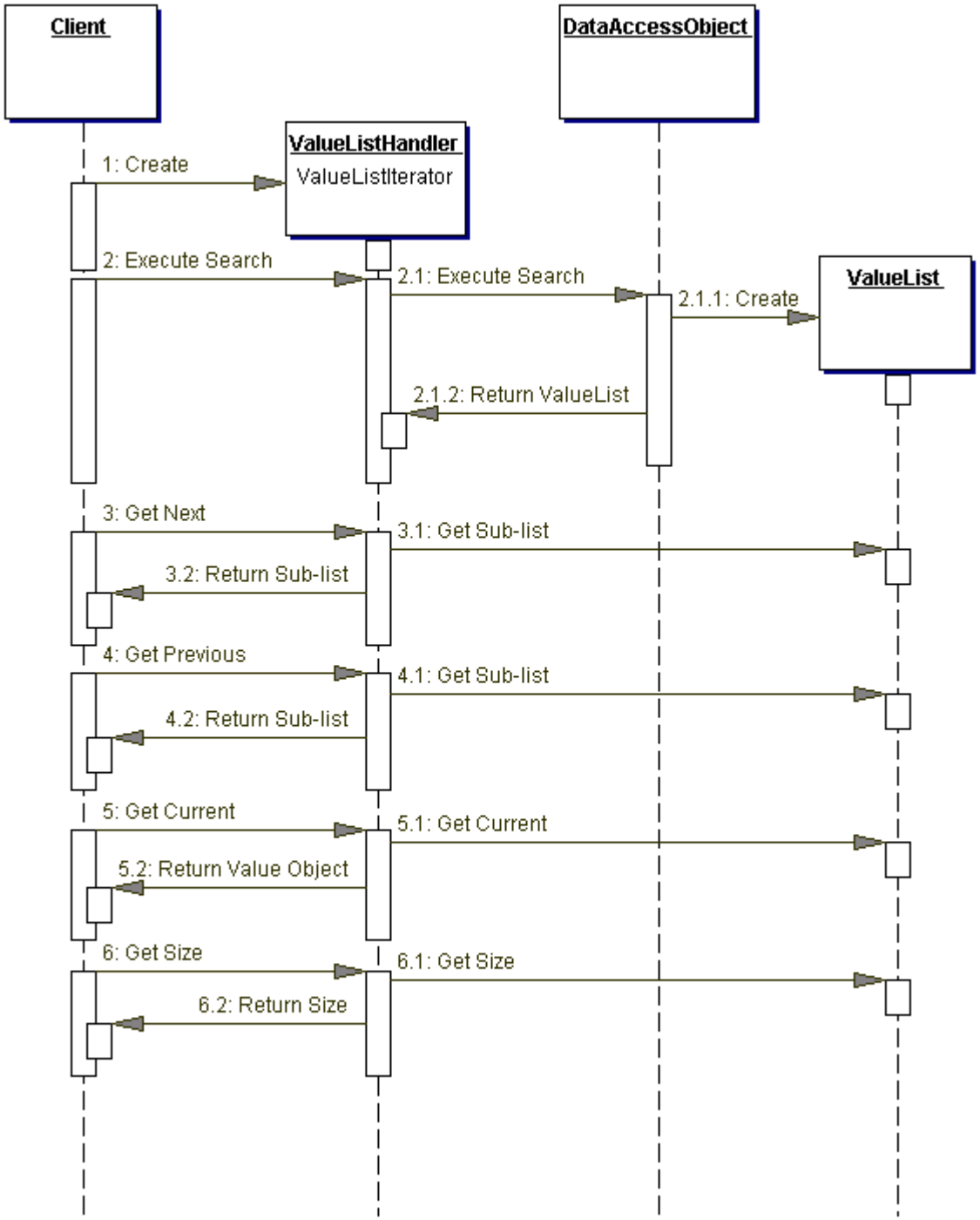
- *O Value List Handler acessa o DAO diretamente e faz a sua pesquisa*
- *Os dados são armazenados como uma Collection de Value Objects*
- *O cliente solicita o ValueListHandler que contém os objetos desejados*
- *O Value List Handler implementa o padrão Iterator e permite que o cliente extraia as informações seqüencialmente*

Estrutura UML



Fonte: [Core]

Diagrama de Seqüência



Fonte: [Core]

Melhores estratégias de implementação

- *Java Object Strategy*
  - *Value List Handler implementado como um objeto qualquer pode ser usado por qualquer cliente que precise da facilidade.*
  - *Útil para aplicações que não usam EJB*
  - *Business Delegates podem usar um Value List Handler deste tipo para obter uma lista de valores*
- *Stateful Session Bean Strategy*
  - *Ideal quando a aplicação usa EJBs na camada de negócio.*
  - *Bean pode conter estado que implementa o Value List Handler ou pode ser um proxy para um.*

Conseqüências

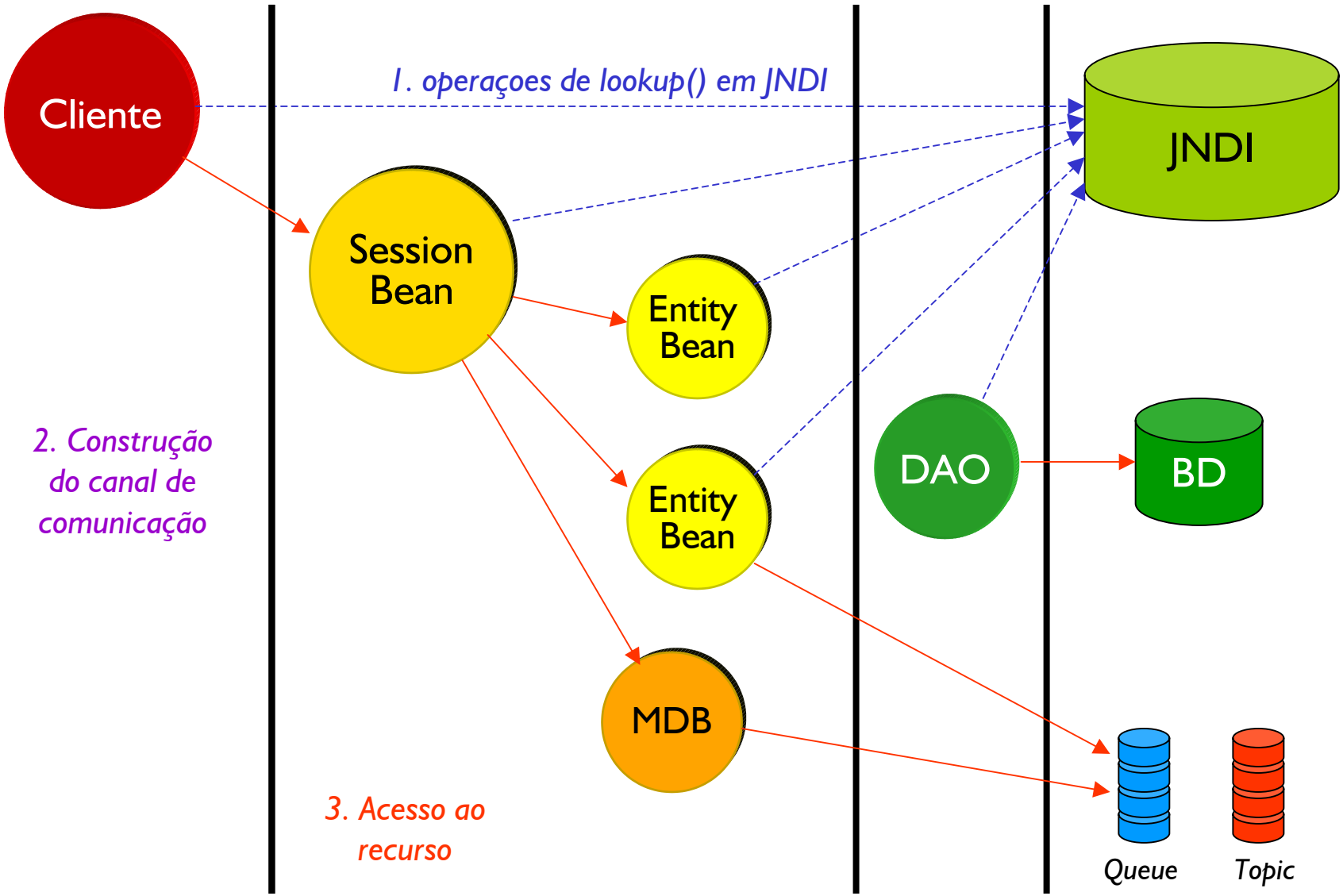
- *Alternativa a EJB finders em pesquisas grandes*
- *Faz cache de resultados do lado do servidor*
- *Maior flexibilidade de pesquisa*
- *Melhora performance da rede*

13

Service Locator

Objetivo: Isolar dos clientes de serviços de localização de recursos (JNDI) a lógica e informações relacionadas ao serviço oferecendo uma interface neutra. Service locator pode ser usado para abstrair todo uso de JNDI e ocultar as complexidades da criação de objetos.

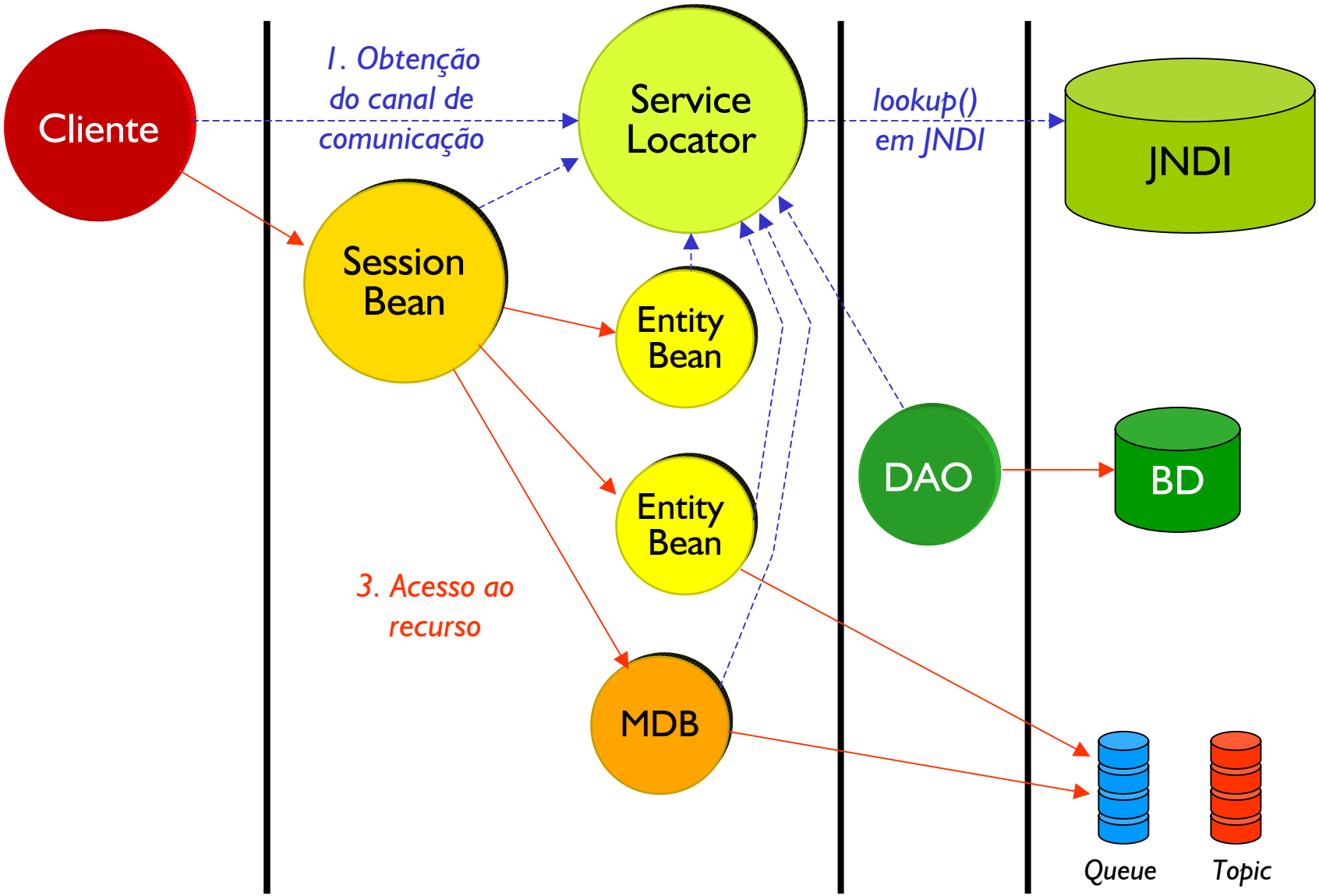
Problema


Cliente
1. operaçoes de lookup() em JNDI
JNDI
Session Bean
Entity Bean
Entity Bean
DAO
BD
MDB
2. Construção do canal de comunicação
3. Acesso ao recurso
Queue
Topic

Descrição do problema

- *Clientes de EJBs ou de recursos compartilhados precisam conhecer e usar a API JNDI para obter referências para os recursos desejados*
- *Em alguns casos, após a obtenção da referência, é preciso realizar conversões e outras tarefas antes de usar o objeto*
- *Criação de objetos, se requerida frequentemente, pode impactar na performance da aplicação, principalmente se clientes e serviços estão localizados em camadas diferentes*

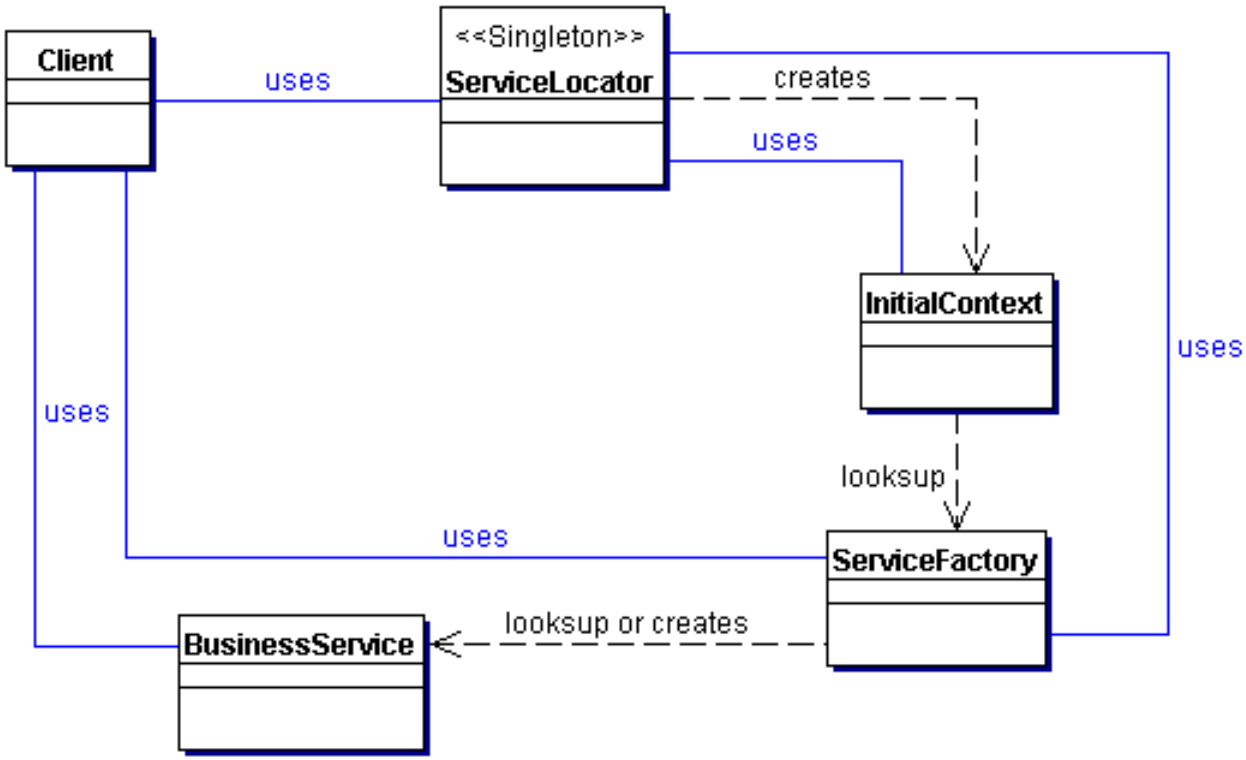
Solução: Service Locator



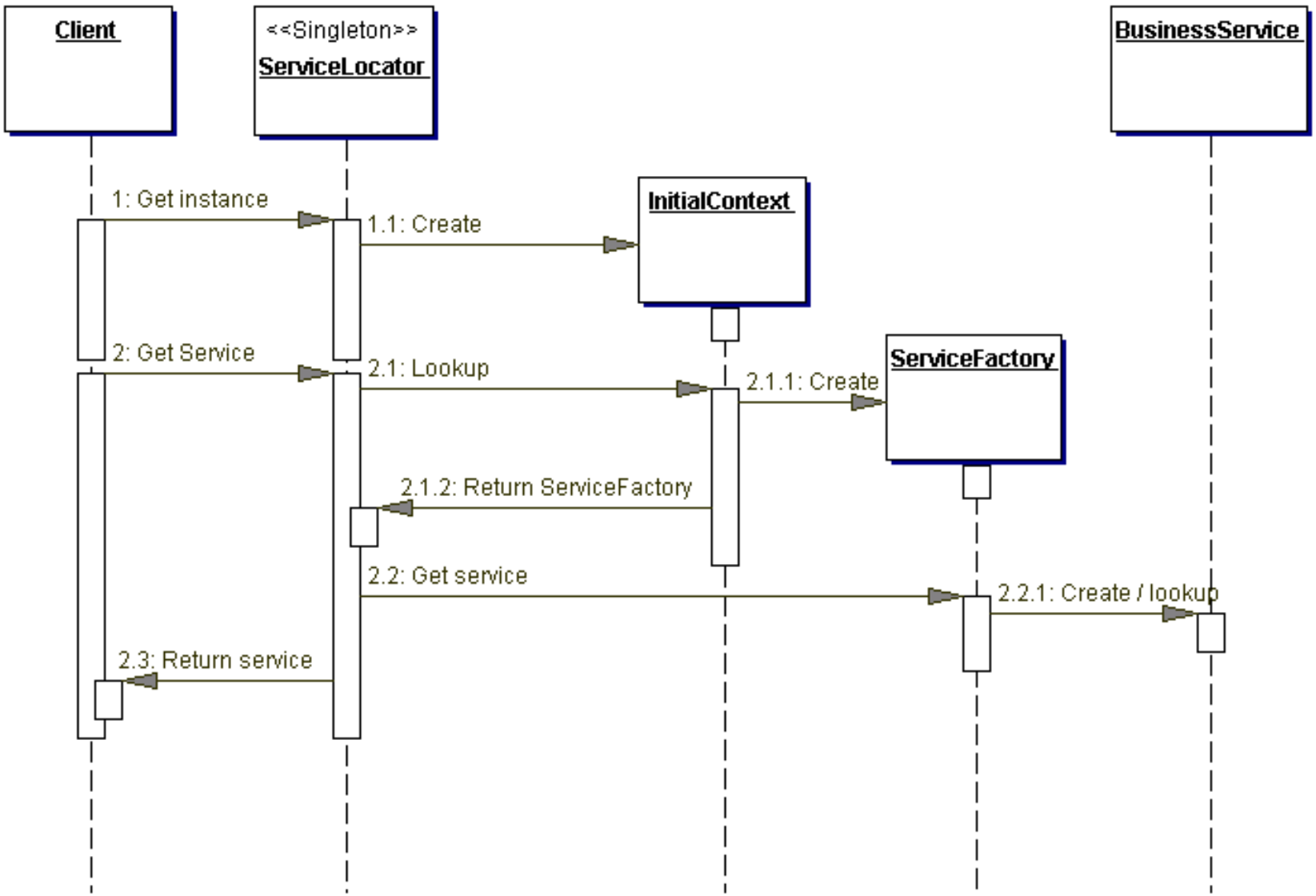
Descrição da solução

- *Service Locator centraliza todo o acesso ao servidor JNDI, facilitando*
  - *Localização de objetos EJBHome (obtenção da referência já convertida para o tipo correto)*
  - *Localização de serviços como conexões de bancos de dados ou conexões de servidores de messaging*
  - *Localização de canais e filas JMS*
- *Service Locator pode melhorar a performance da pesquisa oferecendo um cache para as pesquisas mais frequentes.*

Estrutura UML



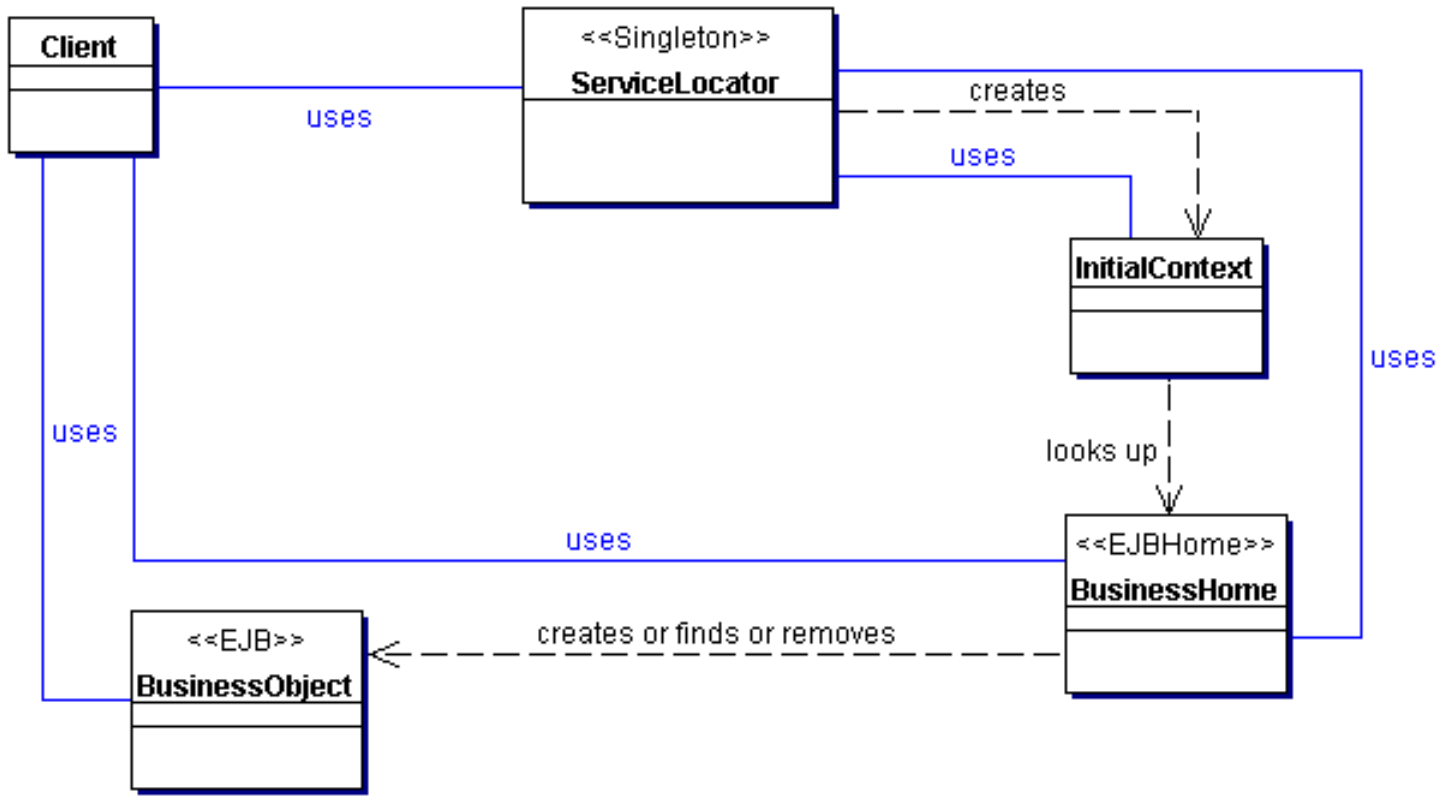
Seqüência


Client
<<Singleton>>
ServiceLocator
BusinessService
InitialContext
ServiceFactory
1: Get instance
1.1: Create
2: Get Service
2.1: Lookup
2.1.1: Create
2.1.2: Return ServiceFactory
2.2: Get service
2.2.1: Create / lookup
2.3: Return service

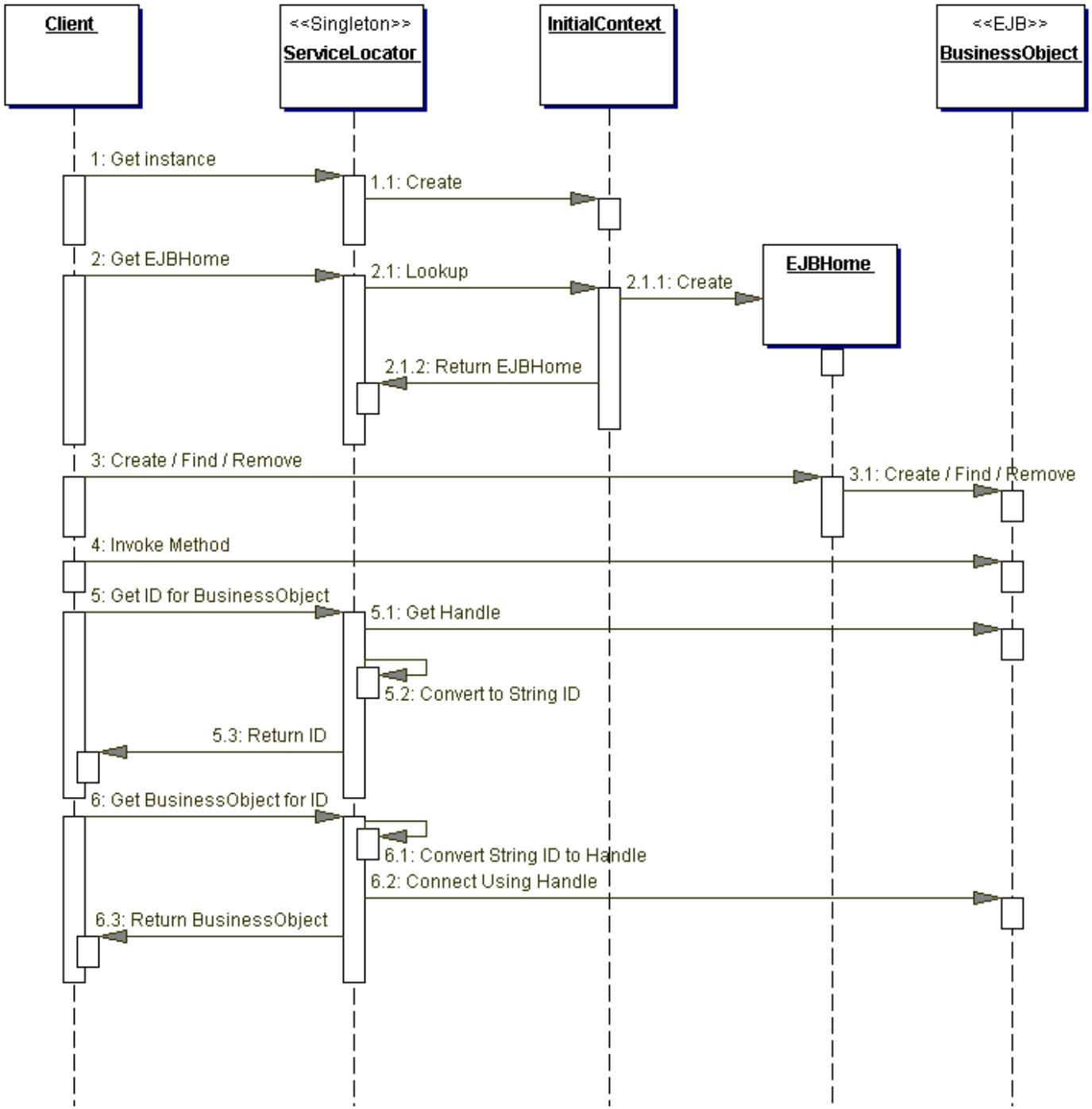
Melhores estratégias de implementação

- *EJB Service Locator Strategy*
  - *Usa objeto EJBHome que pode ser armazenado em um cache para evitar uma pesquisa futura*
- *JMS Queue Service Locator Strategy e JMS Topic Service Locator Strategy*
  - *Retornam QueueConnectionFactory e TopicConnectionFactory usados em conexões JMS*
  - *Retornam filas (queues) e canais (topics)*
- *Type Checked Service Locator Strategy*
  - *Realiza conversões de tipos (ex: narrow())*

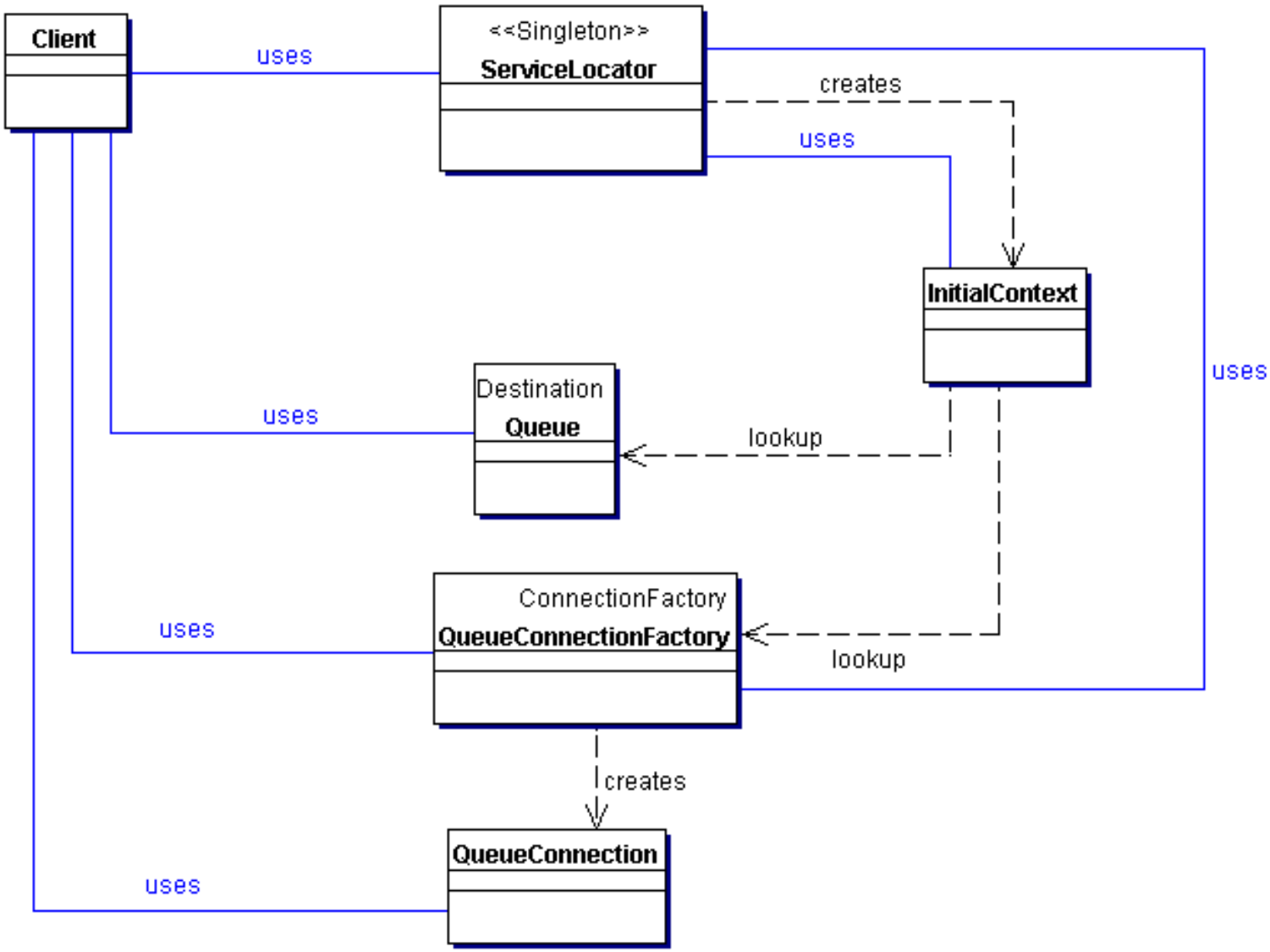
EJB Service Locator Strategy



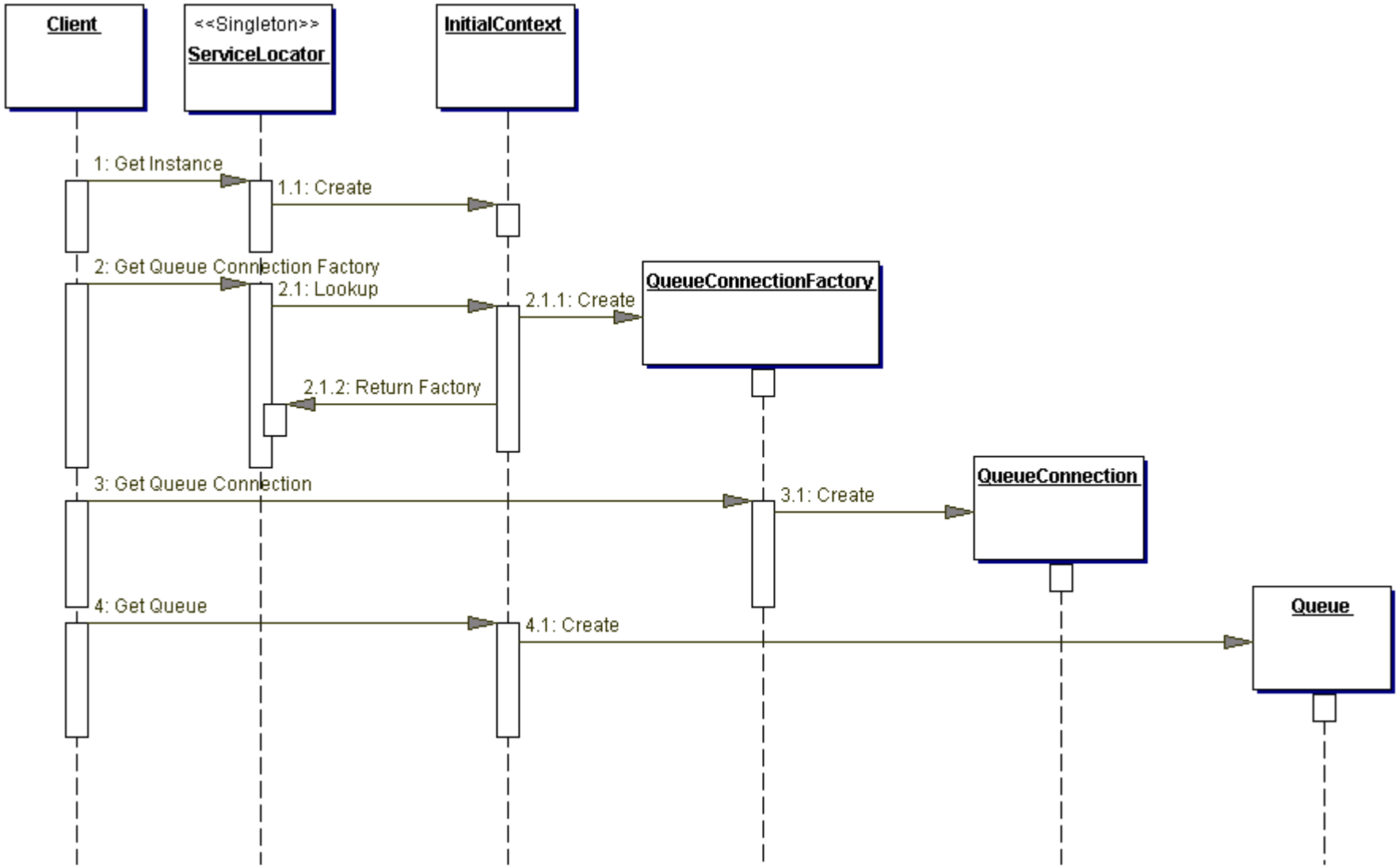
EJB Service Locator Strategy



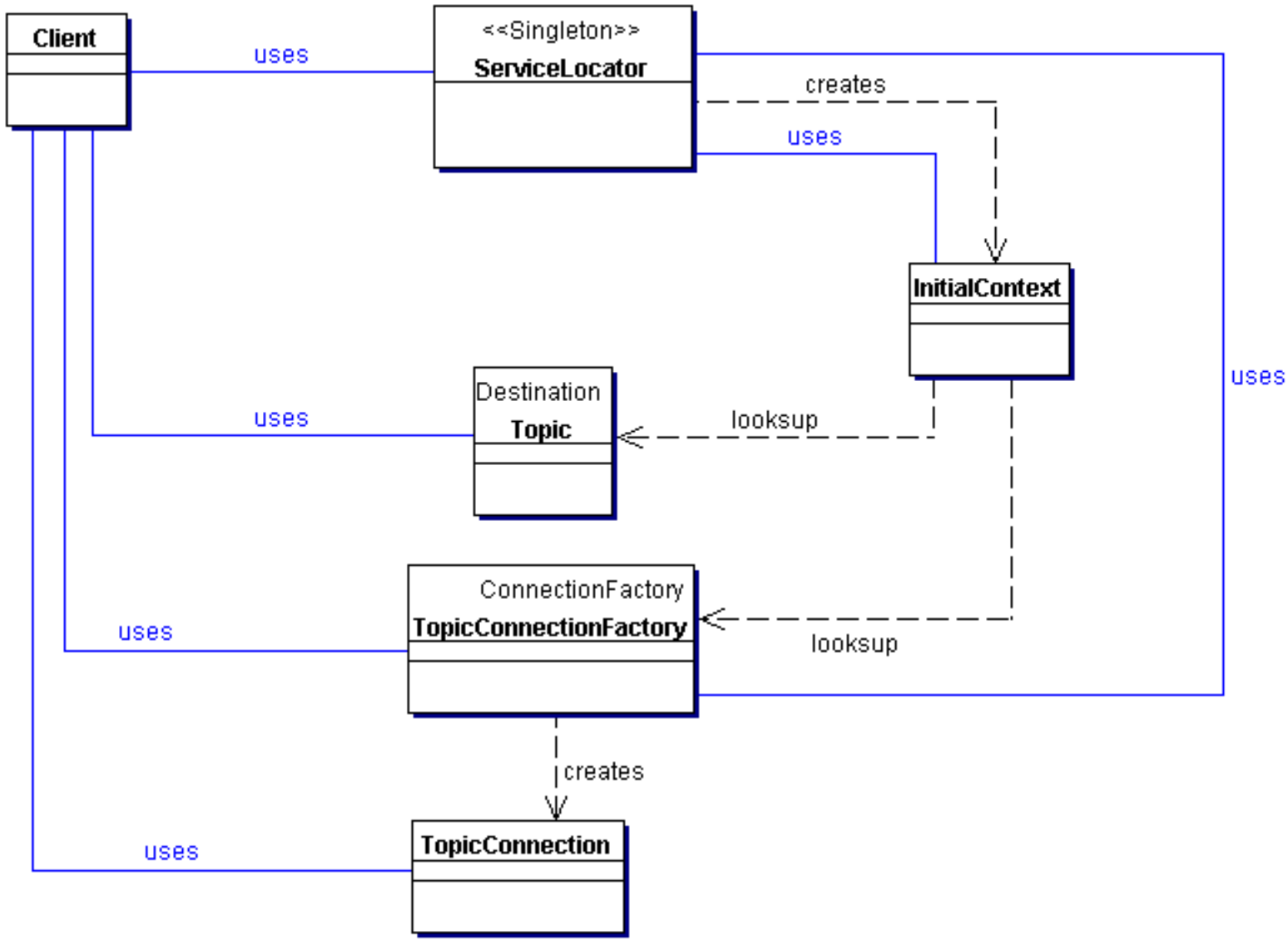
JMS Queue Service Locator Strategy



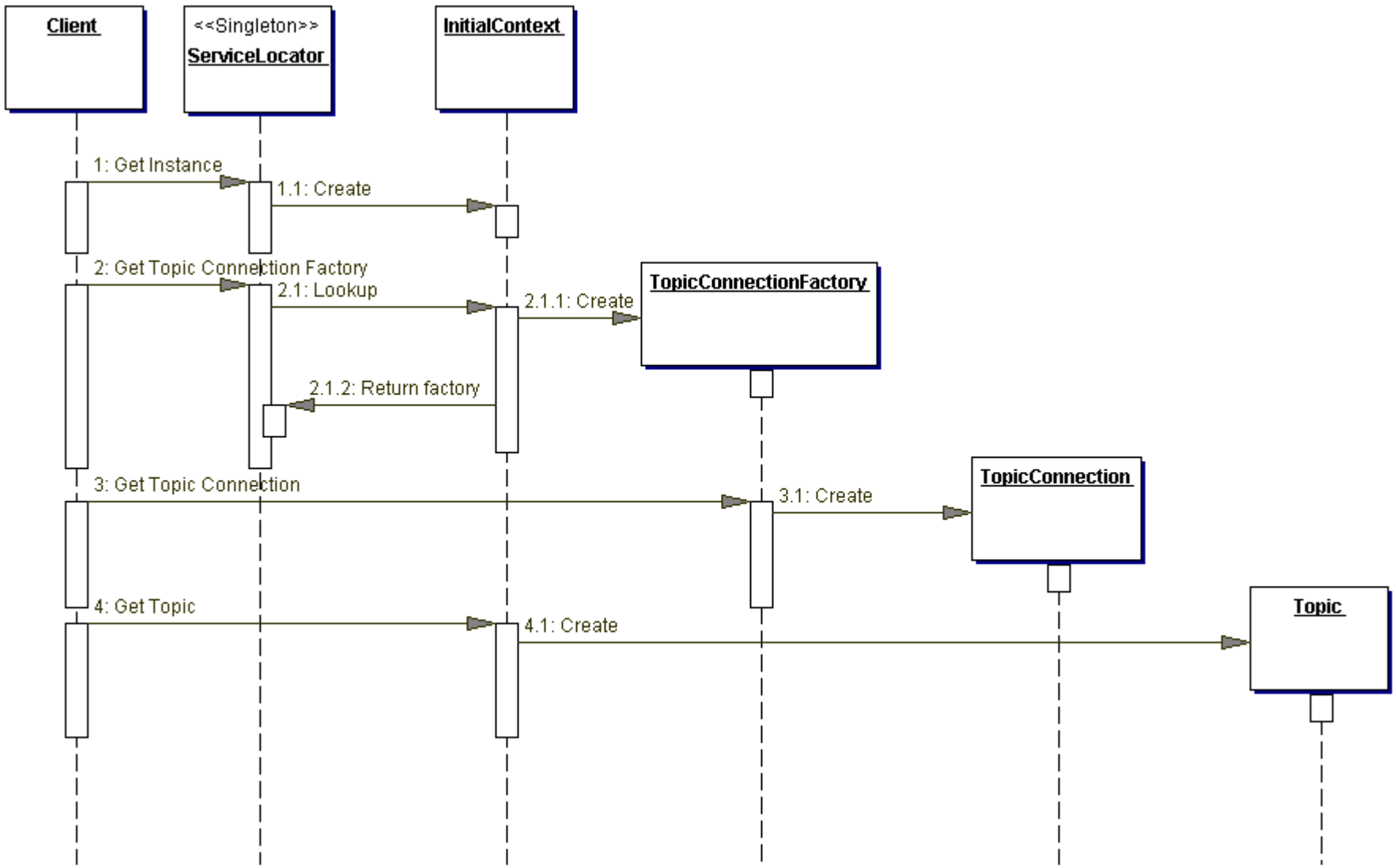
JMS Queue Service Locator Strategy



JMS Topic Service Locator Strategy



JMS Topic Service Locator Strategy



Conseqüências

- *Abstrai complexidade*
- *Oferece acesso uniforme a clientes*
- *Facilita adição de novos componentes de negócio*
- *Melhora performance de rede*
- *Melhora performance de cliente com cache*

Exercícios

- *1. Analise a implementação das estratégias de Service Locator (código 8.33 a 8.35)*
- *2. Refatore a aplicação em ejblayer/sl/ para que utilize Service Locator:*
  - *a) Isole todas as chamadas ao JNDI nesta classe*
  - *b) Altere os componentes (beans e clientes) para que utilizem o Locator para descobrir e obter os recursos desejados*

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.

[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)