J931

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*
  - *Eventos e operações assíncronas com message-driven beans*
  - *Cache de referências para EJBs em Business Delegates*

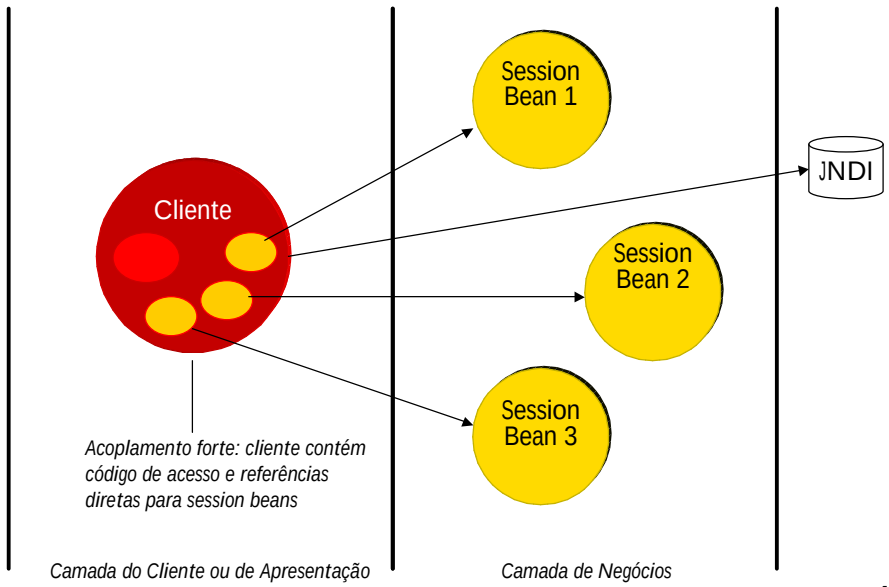
2

9

Business Delegate

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

Problema



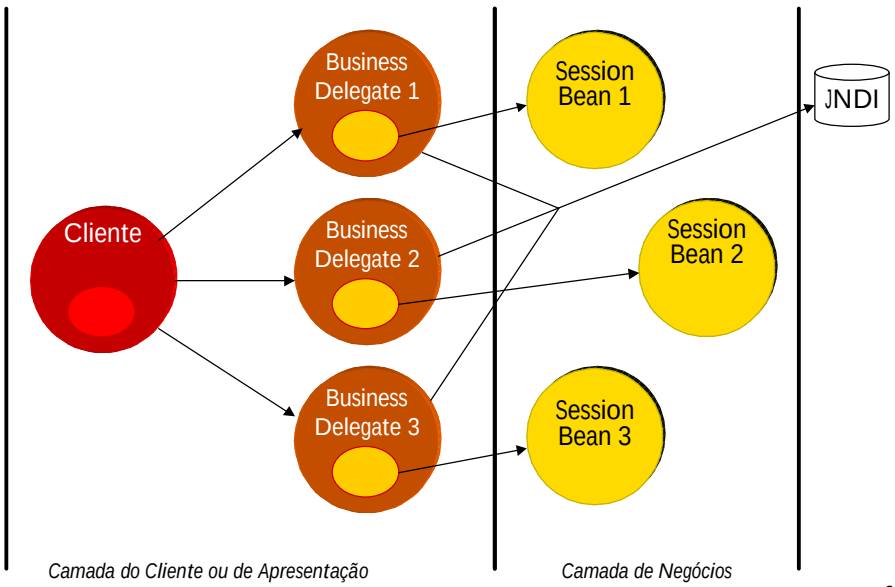
4

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*

5

Solução: Business Delegate



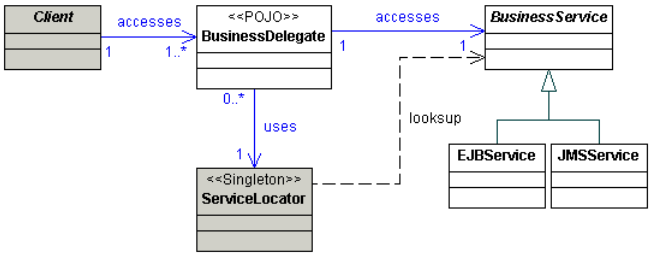
6

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)*

7

Estrutura UML (1)

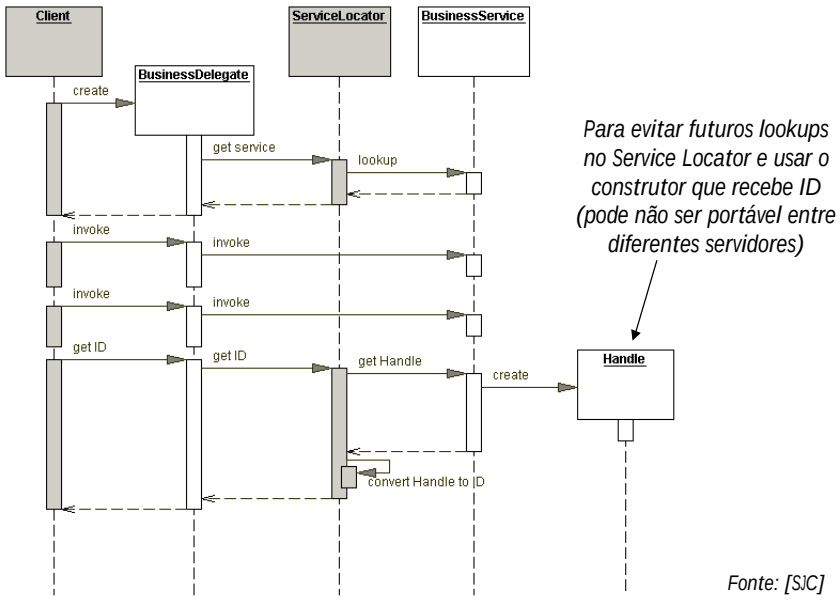


- *BusinessDelegate: tem o papel de prover controle e proteção para o BusinessService. Pode oferecer dois tipos de construtores*
  - *Construtor default que utiliza um ServiceLocator para obter uma fábrica para o BusinessService (como um EJBHome)*
  - *Construtor que recebe um string de ID e o utiliza para obter um Handle para o BusinessService*
- *ServiceLocator: encapsula detalhes de localização*
- *BusinessService: tipicamente um Session Façade*

Fonte: [Core]

8

Diagrama de Seqüência



Fonte: [SJC] 9

Melhores estratégias de implementação

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

10

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

11

Exercícios

- 1. Analise a implementação das estratégias de Business Delegate
- 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

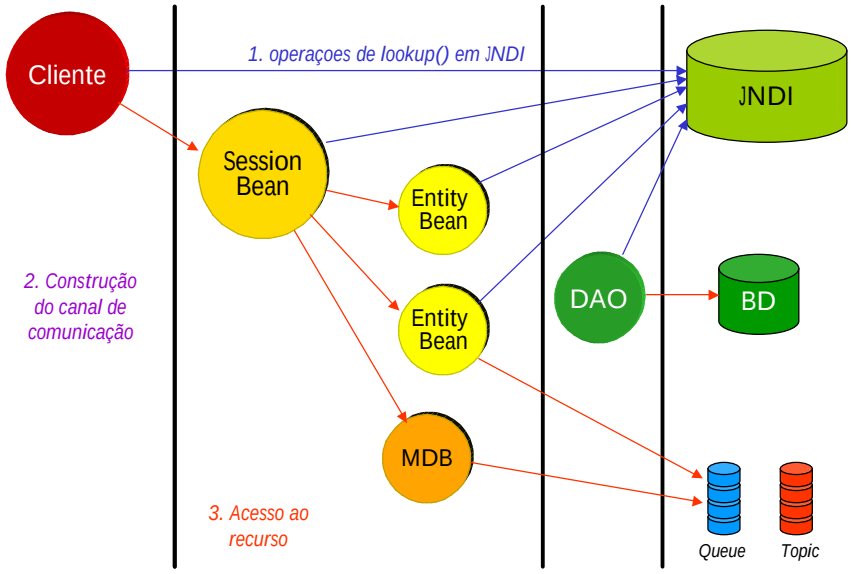
12

10

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



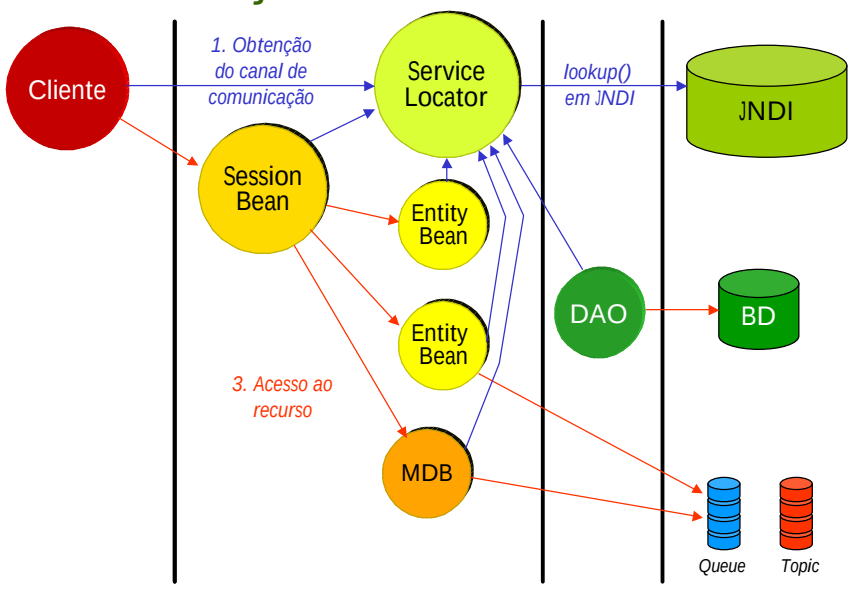
14

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*

15

Solução: Service Locator



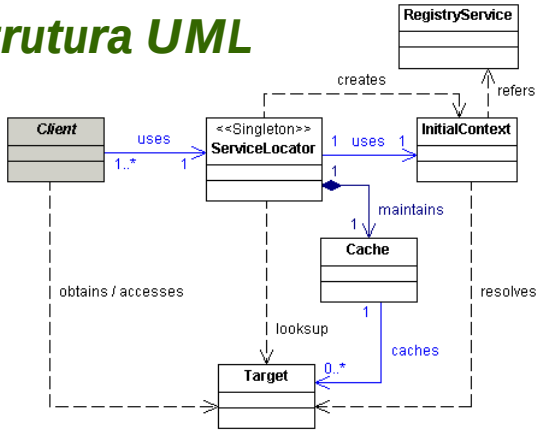
16

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.

17

Estrutura UML

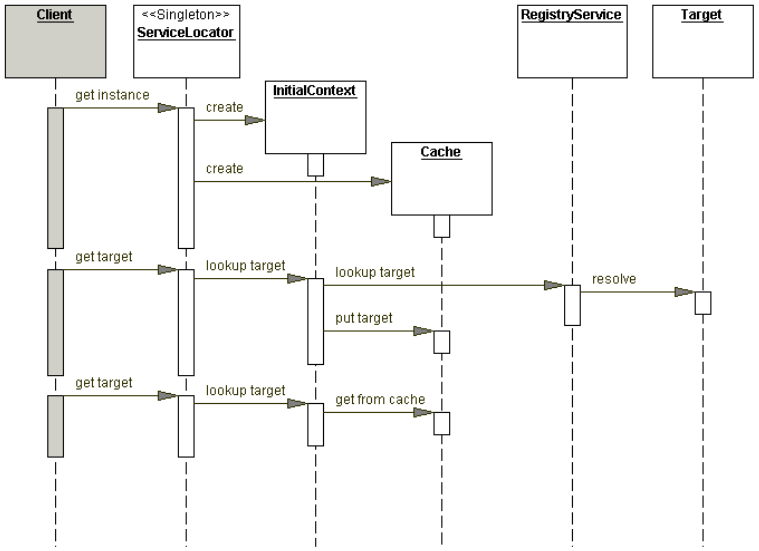


- Client: tipicamente é um Business Delegate para localizar beans ou um DAO para localizar bancos de dados
- ServiceLocator: isola do cliente os detalhes da API de pesquisa
- Cache: ServiceLocator opcional para guardar referências previamente localizadas
- Target: serviço ou componente que o cliente busca no ServiceLocator

[SJC]

18

Seqüência



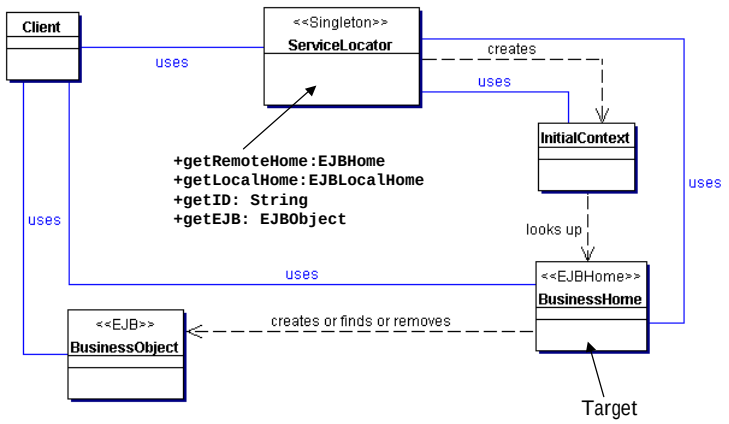
[SJC] 19

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())*
- *Web Service Locator Strategy*

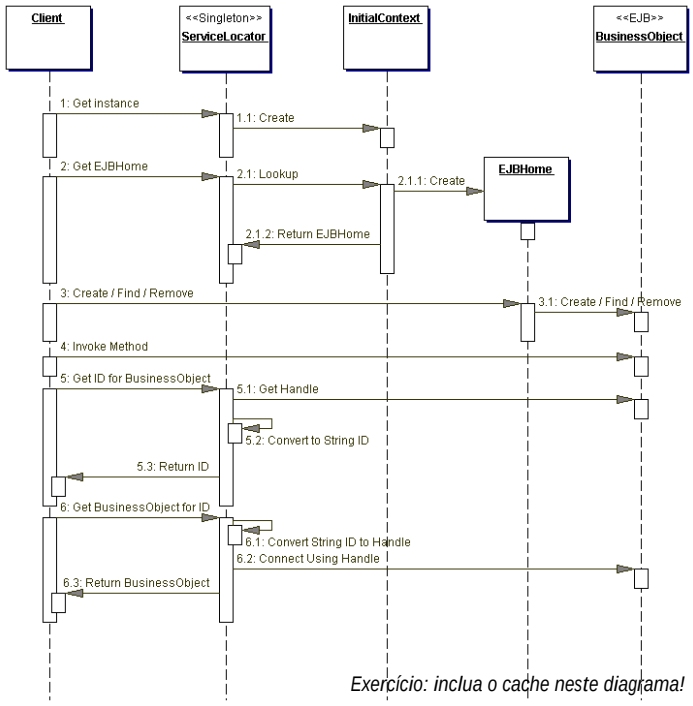
20

EJB Service Locator Strategy



21

EJB Service Locator Strategy



Exercício: inclua o cache neste diagrama!

22

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*

23

Exercícios

- *1. Analise a implementação das estratégias de Service Locator*
- *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*

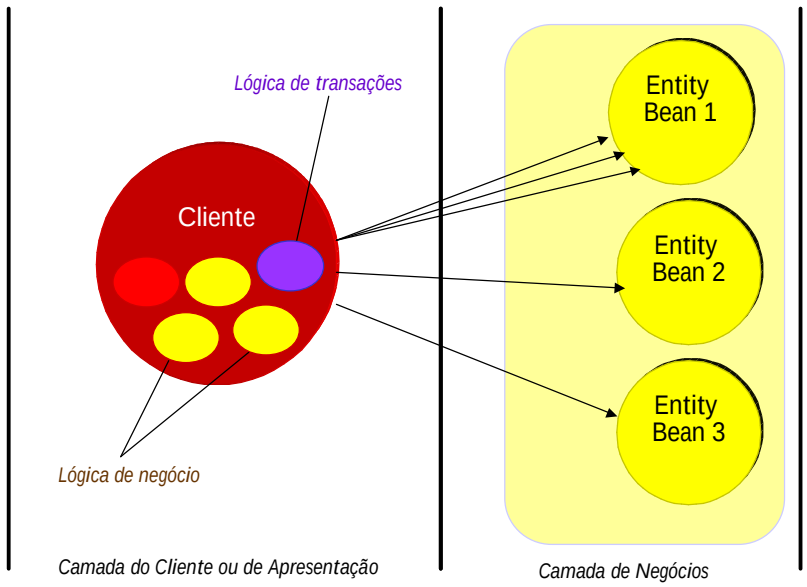
24

11

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



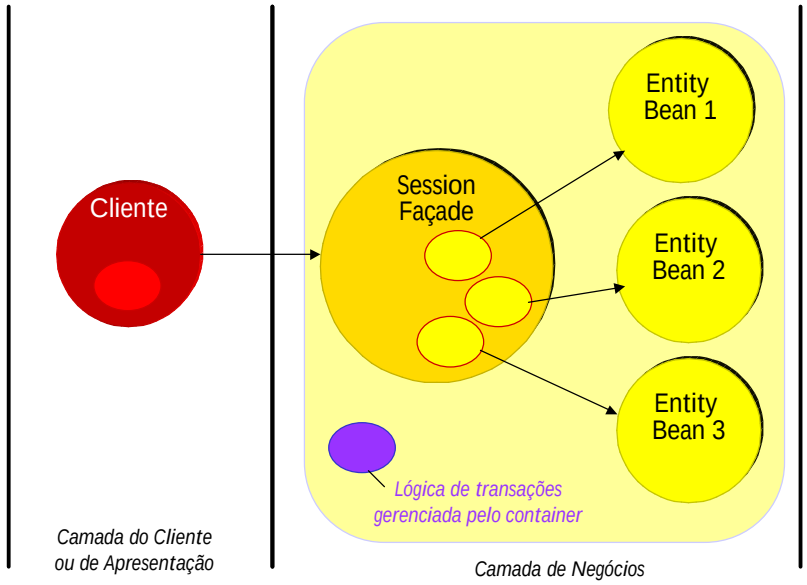
26

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 transações para cada requisição*
  - *Interface complexa e difícil de usar*
  - *Fortemente acoplada ao modelo de objetos (vulnerável)*

27

Solução: Session Façade



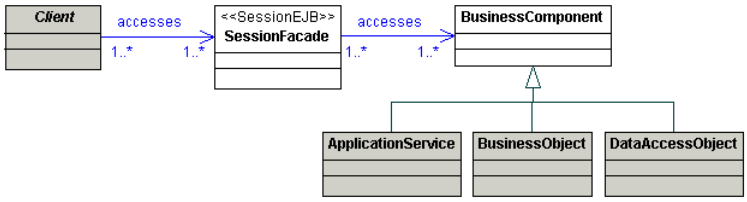
28

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*
- *Ideal é ter pouca ou nenhuma lógica de negócio*
  - *Se existir, deve ser colocada em um Application Service*

29

Estrutura UML

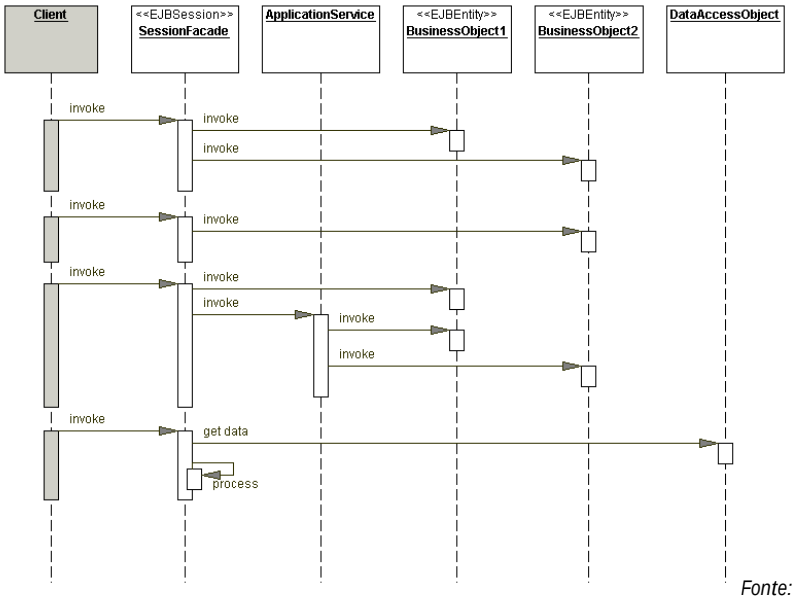


- *Cliente: tipicamente um Business Delegate*
- *SessionFacade: implementada como um session bean. Oculta a complexidade ao lidar com diferentes BusinessComponents*
- *BusinessComponent: pode ser um BusinessObject (EJB), DAO ou ApplicationService*
- *ApplicationService: encapsula os BusinessObjects e implementa a lógica de negócios para oferecer o serviço*
- *DataAccessObject: representa os dados*

Fonte: [Core]

30

Diagrama de Seqüência



Fonte: [SJC] 31

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*

32

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

33

Exercícios

- 1. Analise a implementação das estratégias de Session Façade
- 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)

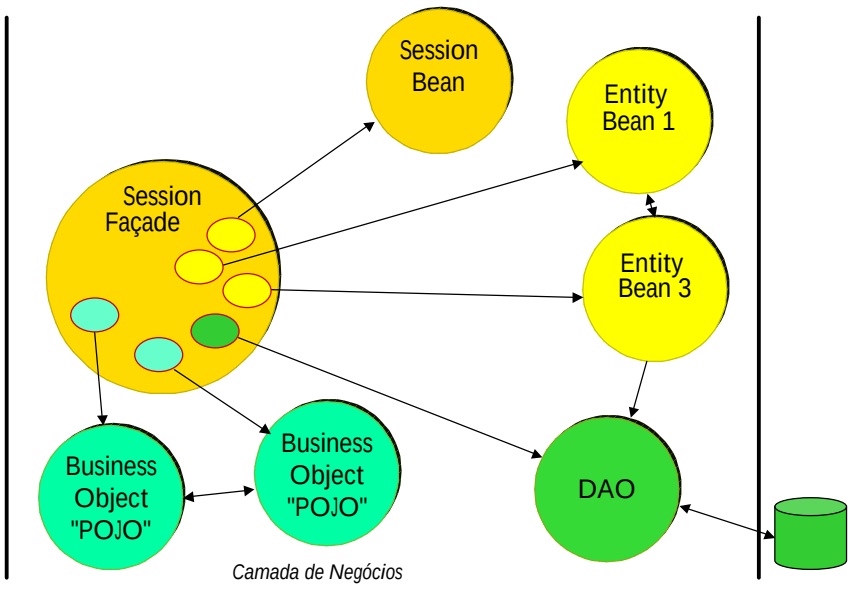
34

12

Application Service

Objetivo: Centralizar e agregar comportamento para prover uma camada de serviços uniforme. Application Service serve para oferecer uma camada de serviços que implementa casos de uso para um conjunto de Business Objects

Problema



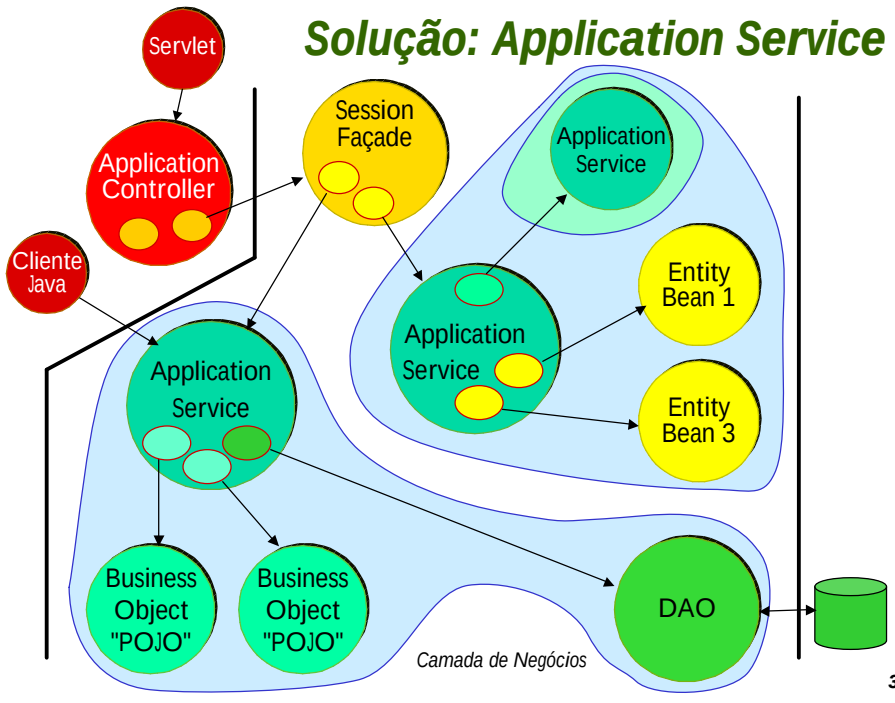
36

Problema

- *Deseja-se centralizar lógica de negócio atraves de diversos componentes de negócio e serviços*
  - *Session Façades poderiam fazer isto, mas não devem conter lógica de negócio e devem expor uma interface simples*
- *Deseja-se encapsular lógica específica para um use-case fora de Business Objects individuais e independente de Session Façades*
  - *A coordenação de diversos Business Objects deve ser feita em objetos que possam ser usados por Session Façades e outras fachadas de negócio (nem sempre Business Objects são EJBs)*

37

Solução: Application Service



38

Solução

- *Application Services oferecem a infraestrutura básica de lógica de negócio usada por Session Façades,*
  - *As fachadas ficam mais fáceis de implementar e contém menos código*
  - *Reduzem o acoplamento entre Business Object ao conter a lógica de integração entre eles*
  - *Permitem ter uma camada central de lógica de negócio mesmo em situações onde não se está usando Business Objects: pode usar um Data Access Object diretamente para acessar dados.*
- *Pode oferecer uma interface mais detalhada que um Session Façade e mais genérica que os serviços que isola*

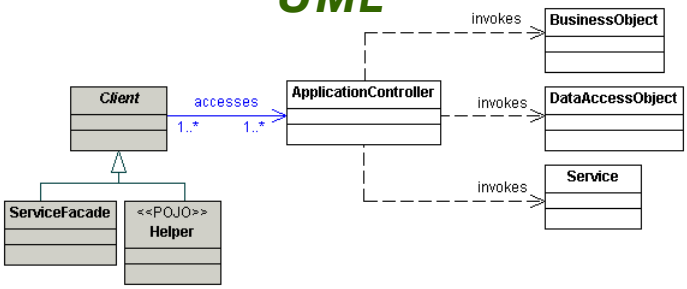
39

Por que Application Services?

- *Onde implementar lógica de negócio?*
  - *Objetos de negócio devem representar os dados, mas devem ter pouca (ou nenhuma) lógica de negócio*
  - *Application Services são uma camada adicional que pode centralizar a lógica de negócios*
- *E Session Façades?*
  - *Em aplicações EJB, Session Façades tipicamente implementavam a lógica de negócio entre Business Objects*
  - *Mas a intenção dos Session Façades é oferecer uma interface de baixa granularidade, o que às vezes requer uma sub-fachada com mais detalhes (que pode ser um Application Service)*
  - *Aplicações que não usam EJB frequentemente precisam de um serviço similar às Session Façades*

40

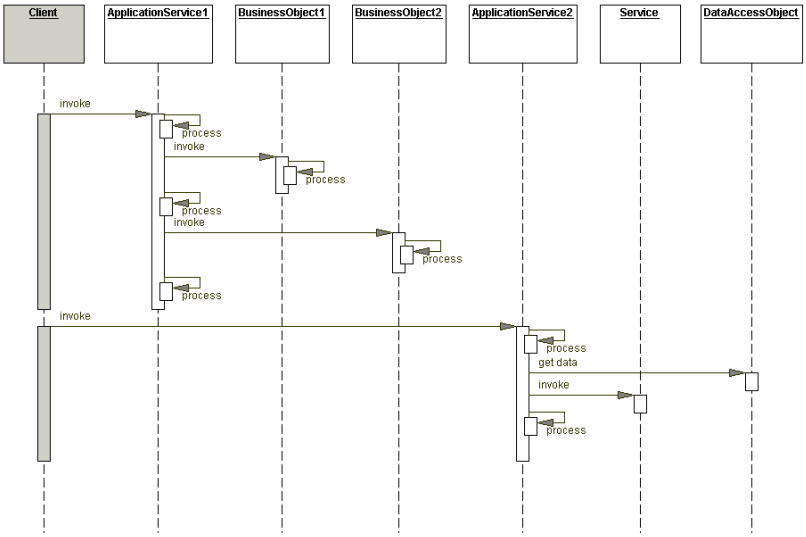
UML



- *Client: cliente POJO, um Session Façade, outro Application Service*
- *ApplicationService: encapsula lógica de negócios para prover um determinado serviço.*
- *BusinessObject: objetos necessários para executar o serviço*
- *Service: componente arbitrário que oferece um serviço qualquer*
- *DataAccessObject: oferece acesso direto aos dados*

41

Seqüência



42

Principais Estratégias

- *Application Service Command Strategy*
  - *Solução utilizando o Command Pattern (GoF)*
  - *Cada chamada do cliente, por um Application Controller utiliza uma instância de um objeto de ação, que é executado e que pode chamar métodos de um Application Service*
- *Application Service Layer Strategy*
  - *Os Application Services são distribuídos em cascata*
  - *Serviços mais genéricos chamam serviços mais específicos*
  - *Um ApplicationService genérico poderia ser usado por toda a aplicação*

43

Conseqüências

- *Centraliza lógica de workflow reutilizável*
  - *Tira a lógica dos Session Façades*
- *Melhora o reuso de lógica de negócio*
- *Evita duplicação de código*
  - *Session Façades e POJO Façades podem compartilhar os mesmos serviços*
- *Simplifica implementação de fachadas*
- *Introduz camada adicional de negócios*

44

Exercícios

- *1. Analise os exemplos de código com as estratégias de ApplicationService*
- *2. Refatore a aplicação fornecida para que use Application Service*
  - *Remova lógica de negócios do Session Façade*
  - *Remova lógica de comunicação dos Entity Beans*

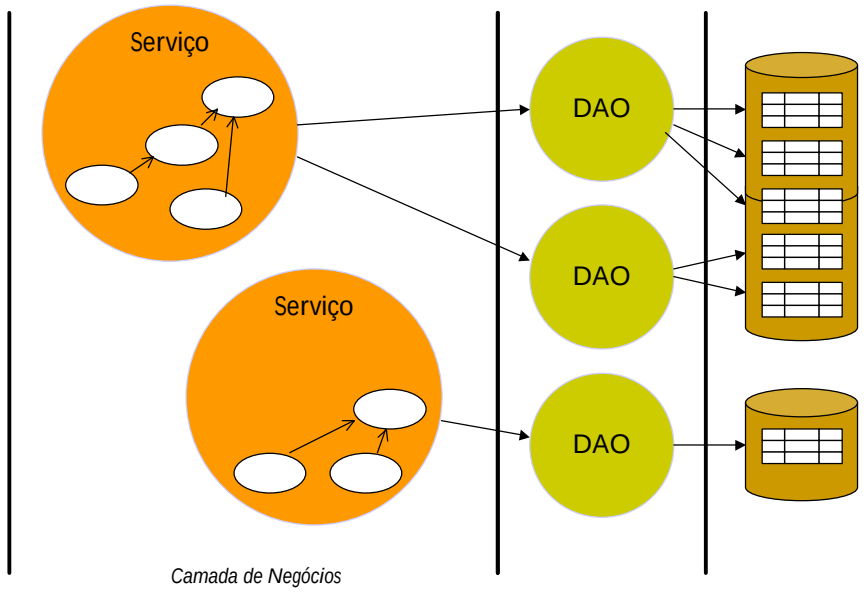
45

13

Business Object

Objetivo: Separar dados de negócio e lógica usando um modelo de objetos. O Business Object é uma abstração que representa uma entidade. Pode ser implementado como um Entity Bean.

Problema



47

Problema e Motivação

- *Temos um modelo de domínio conceitual que contém lógica de negócio e relacionamento.*
  - *Objetos compostos que possuem relacionamentos entre si, lógica complexa, regras de validação e negócios*
- *Queremos separar o estado e lógica de negócios do comportamento associado ao resto da aplicação*
  - *Melhorar a coesão*
  - *Aumentar a chance de reuso*
  - *Evitar duplicação de código*
- *Separar serviços (ações) de estado (entidades)*

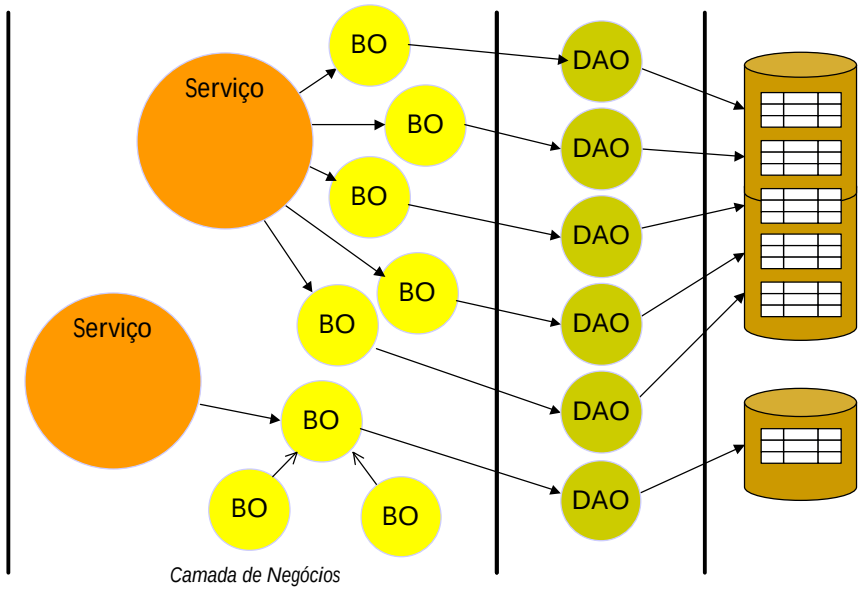
48

Definições (USDP)

- *Modelo de negócios*
  - *Modelo de casos de uso: descreve atores e processos*
  - *Modelo de objetos: descreve as entidades usadas*
- *Modelo de domínio ou modelo conceitual*
  - *Modelo abstrato que captura os mais importantes tipos de objetos no contexto do sistema*
  - *Representam as entidades que existem ou eventos que ocorrem no sistema*
- *Modelo de objetos*
  - *Implementação concreta do modelo conceitual*
- *Modelo de dados*
  - *Implementação para banco de dados (ER-model)*

49

Solução:Business Object



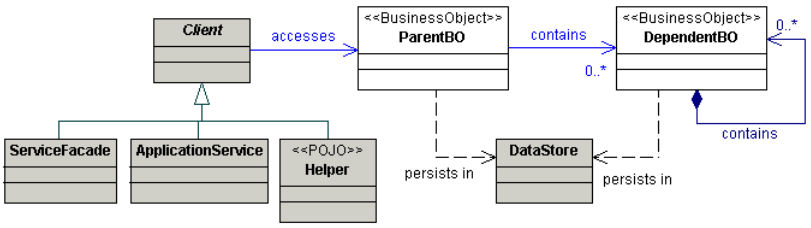
50

Solução

- *Usar objetos de negócio (Business Objects) para separar dados e lógica de um modelo de objetos*
  - *Business Objects encapsulam e gerenciam dados, comportamento e persistência de negócios.*
  - *Ajudam a separar lógica de negócio de lógica de persistência*
- *Duas principais estratégias de implementação*
  - *Usar objetos Java comuns (POJO) e escolher um mecanismo de persistência (DAOs, Domain Store, JDO)*
  - *Usar entity beans (Composite Entity) e escolher BMP ou CMP como forma de persistência*

51

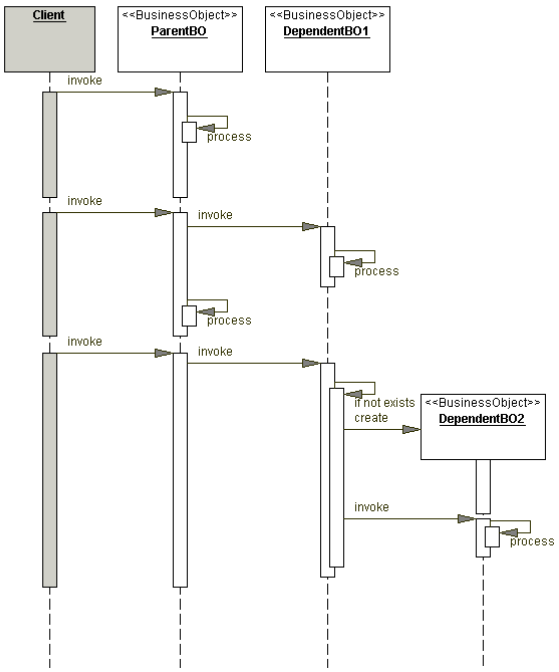
UML



- *Client: tipicamente um Session Façade, objeto Helper (View Helper ou similar) ou um Application Service*
- *ParentBO: No modelo de Business Object composto, este é o BO principal que contém BOs dependentes.*
- *DependentBO: Objetos independentes gerenciados pelo ParentBO. São fortemente acoplados aos seus pais e deles dependem para gerência do ciclo de vida.*

52

Seqüência



53

Estratégias

- *POJO Business Object Strategy*
  - *Implementação usando objetos Java comuns*
  - *Requer que programador lide com cache de objetos, pooling, concorrência, sincronização de dados, persistência, segurança, transações*
- *Composite Entity Business Object Strategy*
  - *Implementação usando Entity Beans locais*
  - *Permite tratamento automático de concorrência, sincronização, pooling, cache e configuração declarativa de transações, segurança, persistência*

54

Conseqüências

- Promove um enfoque orientado a objetos à implementação do modelo de negócios
  - Distingue-se serviços dos objetos
- Centraliza comportamento e estado de negócios
  - Cada objeto cuida de sua lógica local.
  - Lógica que age em múltiplos objetos deve ser implementada com Application Service.
- Separa lógica de persistencia de lógica de negócios
- Promove arquitetura orientada a serviços

55

Exercícios

- 1. Analise o código contendo diferentes estratégias de implementação.
- 2. Refatore a aplicação fornecida
  - Identifique potenciais objetos de negócio
  - Crie os objetos e centralize seu código de lógica de negócios

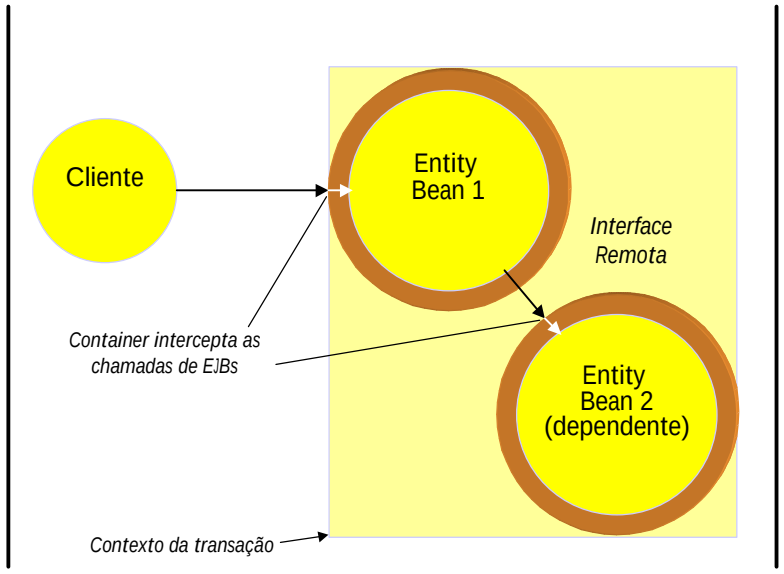
56

14

Composite Entity

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

Problema



58

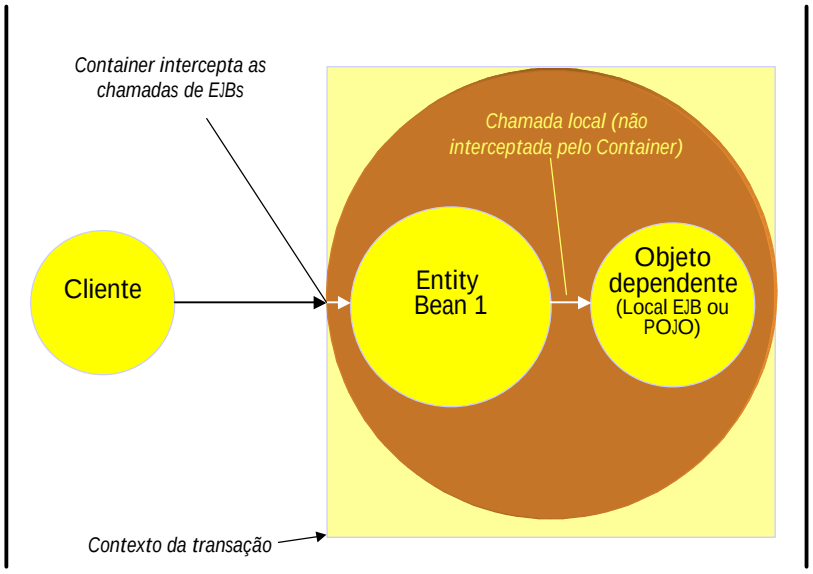
Descrição do problema

- *Relacionamentos entity-to-entity através de interfaces remotas devem ser evitados*
  - *Difíceis de manter: acoplamento forte*
  - *Quaisquer chamadas são interceptadas pelo container, trazendo overhead desnecessário (muito overhead se bean acessar o outro via interface remota)*
  - *Container 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*

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

59

Solução: Composite Entity



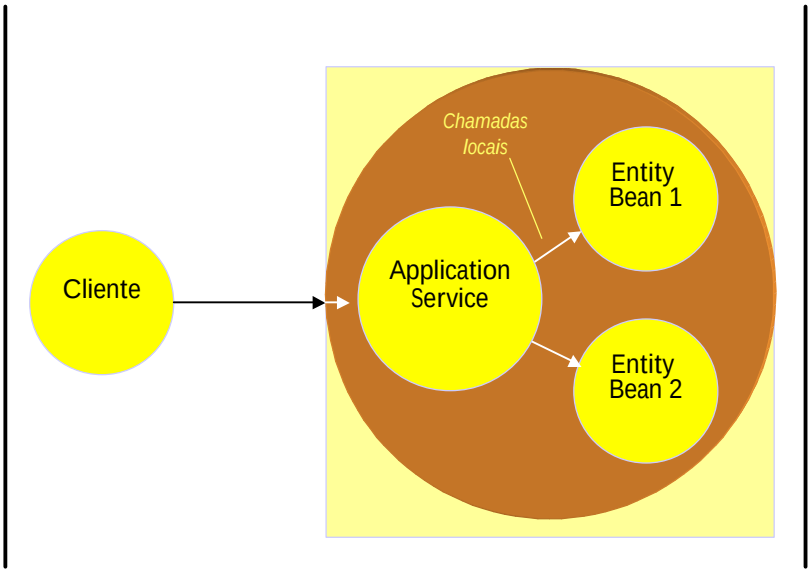
60

Descrição da solução

- *Transformar o Entity Bean dependente em um objeto Java comum ou Entity Local*
- *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)*

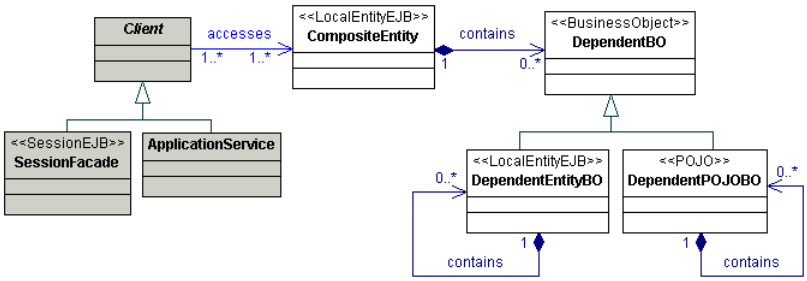
61

Solução 2: Application Service



62

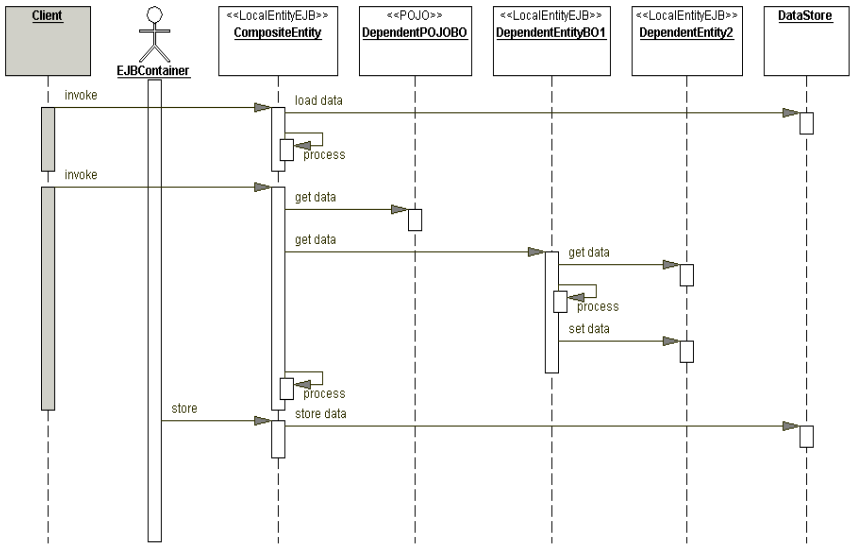
Estrutura UML



- *CompositeEntity: entity bean de baixa granularidade que contém objetos dependentes*
- *DependentBO: objeto dependente que depende do objeto pai e gerencia seu próprio ciclo de vida. Pode conter outros objetos dependentes*

Fonte: [Core] **63**

Diagrama de Seqüência



Fonte: [Core] **64**

Melhores estratégias de implementação

- *Composite Entity é uma das estratégias do padrão Business Object*
- *Composite Entity Remote Façade Strategy*
  - *Nesta estratégia, um Entity Bean remoto é implementado como fachada para os outros beans ou business objects*
- *Estratégias para Composite Entity usando BMP*
  - *Lazy Loading Strategy: a carga de objetos dependentes é realizada apenas no momento do uso*
  - *Store Optimization Strategy: usa um token para sinalizar se os dados foram alterados para evitar uma atualização desnecessária*

65

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 TOs compostos*

66

Exercícios

- *1. Analise a implementação das estratégias de Composite Entity*
- *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 TO que possa ser enviado pelo cliente, lido, alterado e retornado.*

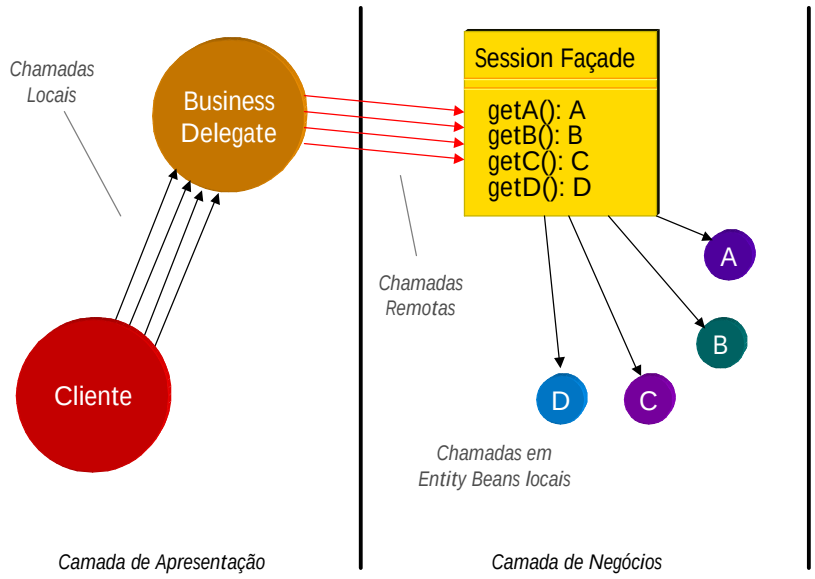
67

15

Transfer Object

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

Problema



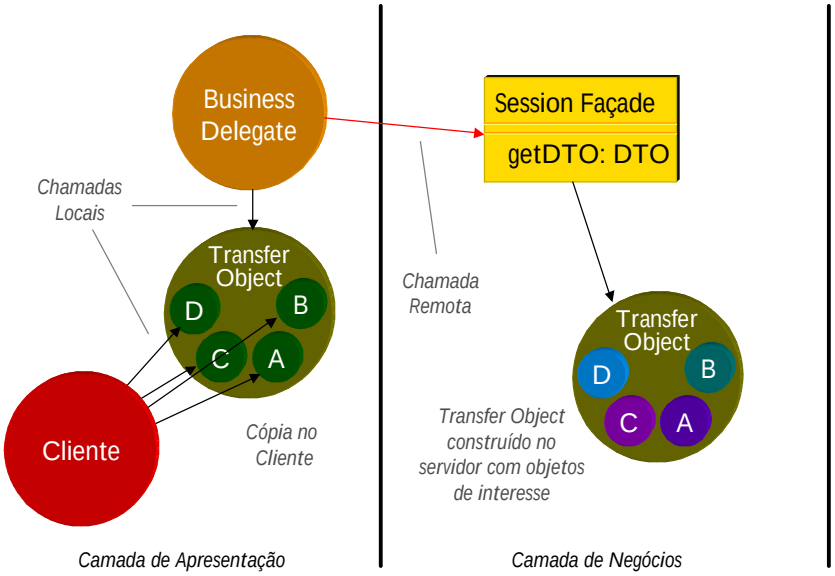
69

Descrição do problema

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

70

Solução: Transfer Object



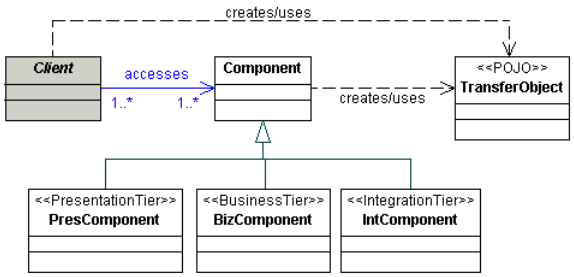
71

Descrição da solução

- *Uma única chamada remota é necessária para transferir o Transfer Object*
  - *O cliente pode extrair as informações de interesse através de chamadas locais*
- *Cópia do cliente pode ficar desatualizada*
  - *Transfer 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*

72

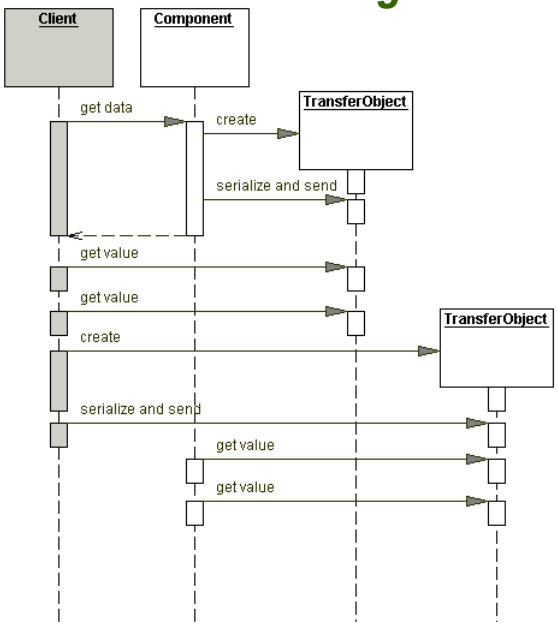
Estrutura UML



- *Client: geralmente um componente de outra camada.*
- *Component: qualquer componente de outra camada que o cliente usa para enviar e receber dados.*
- *TransferObject: é um POJO (Plain Old Java Object) serializável que contém atributos suficientes para agregar e carregar todos os dados*

Fonte: [Core] 73

Diagrama de Seqüência



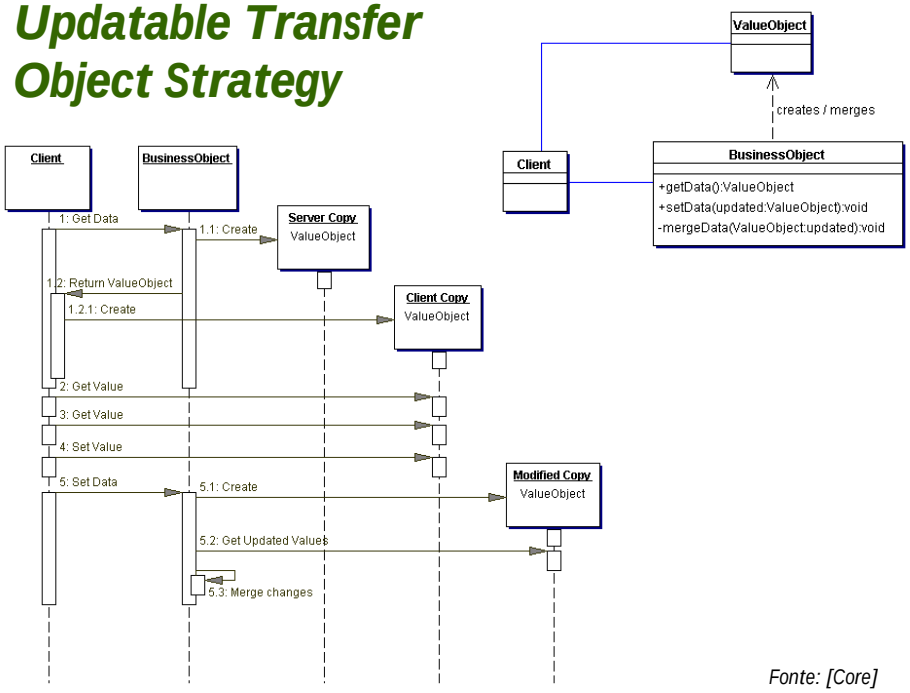
Fonte: [Core] 74

Melhores estratégias de implementação

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

75

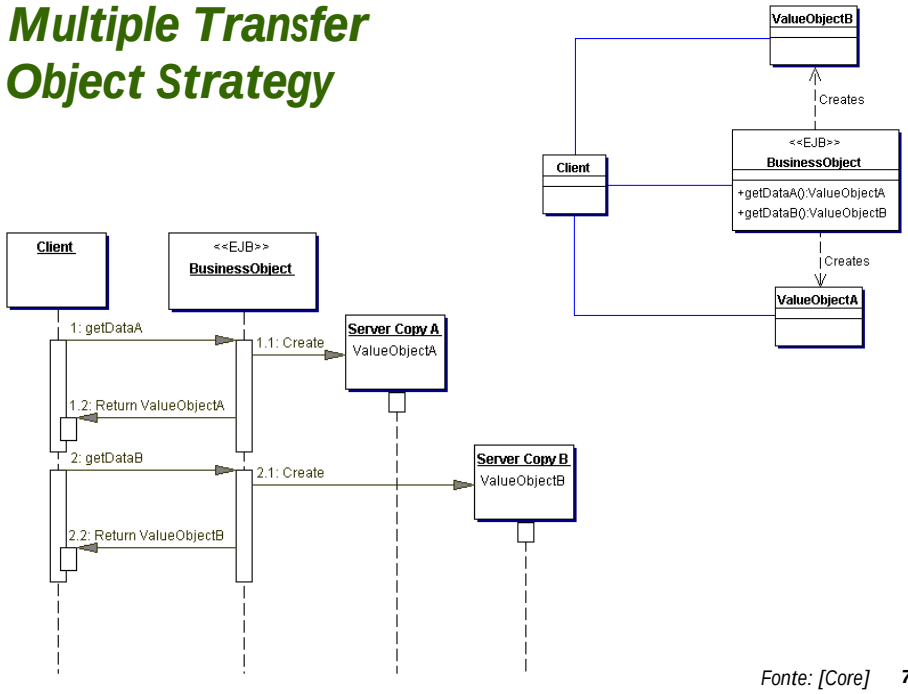
Updatable Transfer Object Strategy



Fonte: [Core]

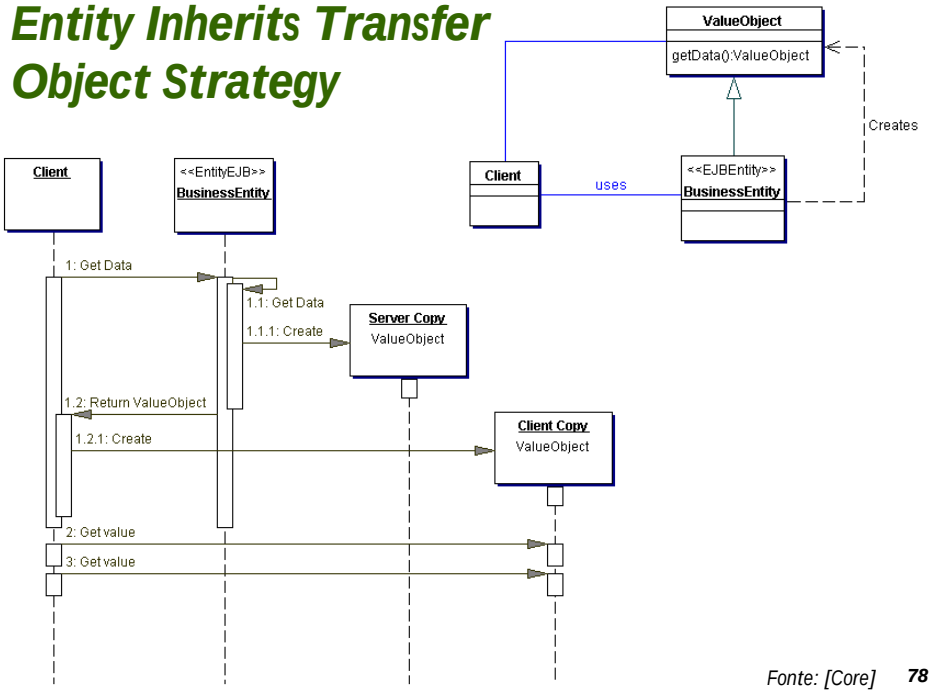
76

Multiple Transfer Object Strategy



Fonte: [Core] **77**

Entity Inherits Transfer Object Strategy



Fonte: [Core] **78**

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

79

Exercícios

- 1. Analise a implementação das estratégias de Transfer Object
- 2. Refatore a aplicação em ejblayer/vo/ para que utilize Transfer Object
  - a) Crie um Transfer Object que representa uma cópia do objeto ProdutoBean
  - b) Implemente no Façade um método que retorne o Transfer 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.

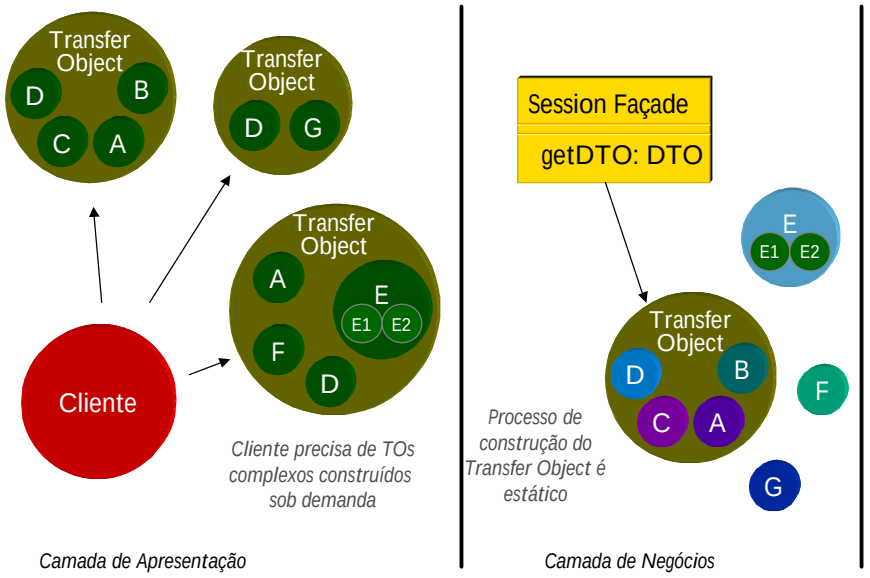
80

16

Transfer Object Assembler

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

Problema



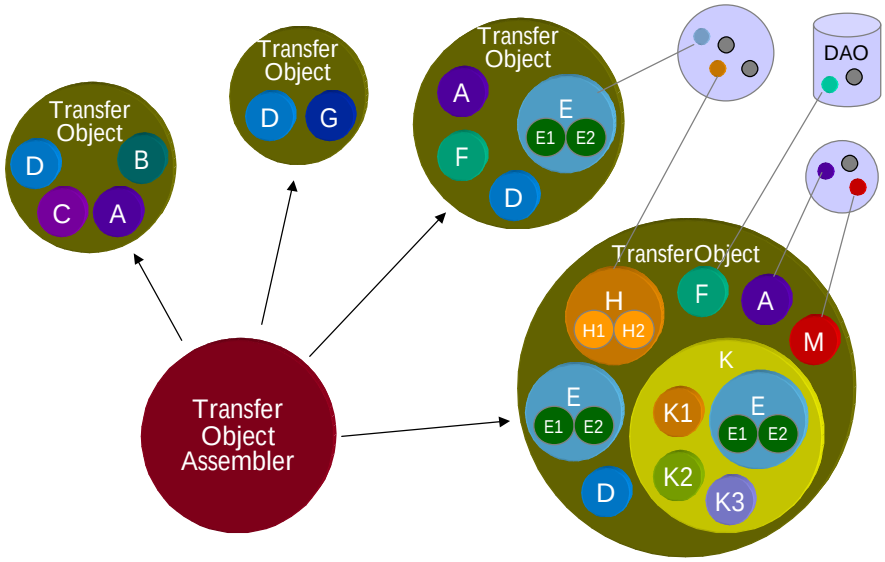
82

Descrição do problema

- *O cliente precisa interagir com múltiplos componentes para obter alguns dados de cada um deles*
  - *Chamar vários métodos ou vários Transfer 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.*

83

Solução: Transfer Object Assembler



Esses TOs não são um modelo de um componente específico mas da aplicação

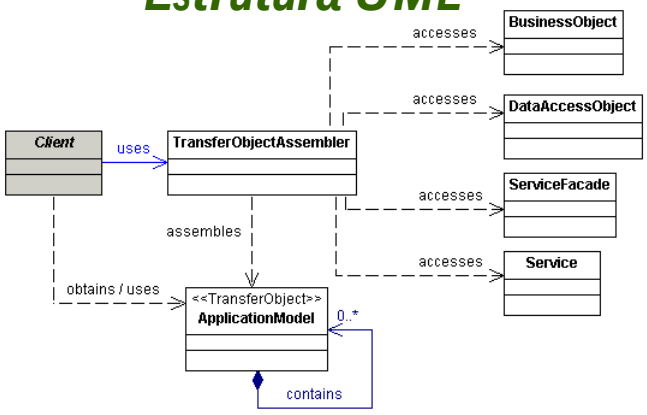
84

Descrição da solução

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

85

Estrutura UML

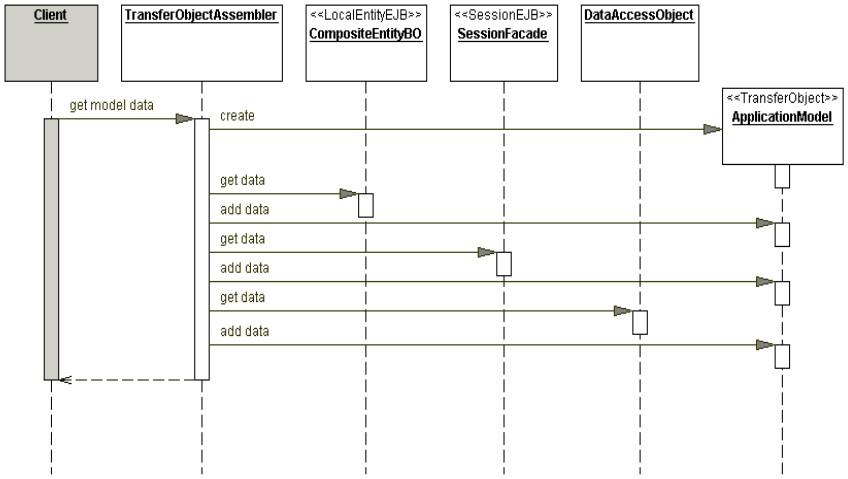


- *Client: objeto interessado em receber dados da aplicação*
- *TransferObjectAssembler: constrói um objeto TO composto baseado nos requerimentos da aplicação*
- *ApplicationModel: é o TO composto construído pelo TransferObjectAssembler*

Fonte: [Core]

86

Diagramas de Seqüência



Fonte: [Core]

87

Melhores estratégias de implementação

- *Plain Old Java Object Strategy (POJO)*
  - *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 Transfer Object Assembler*

88

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 TOs obsoletos*

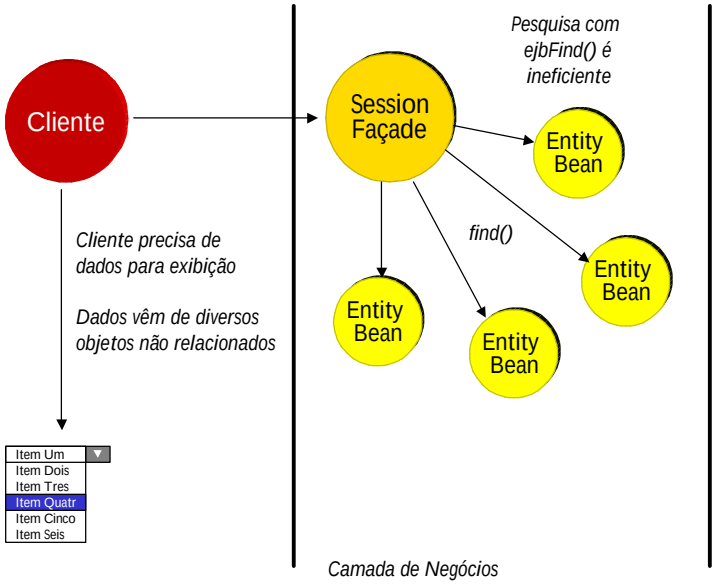
89

17

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



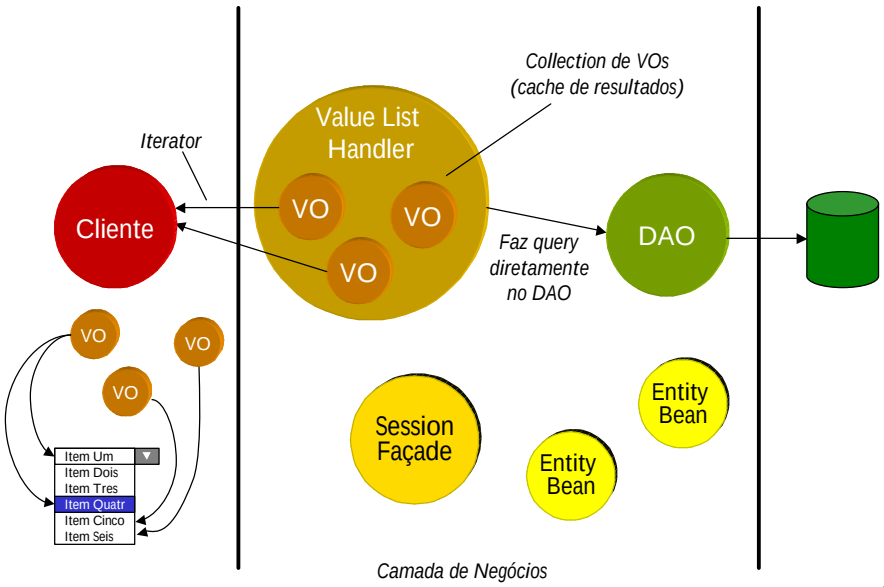
91

Descrição do problema

- *Cliente precisa de dados de diversas fontes (diversos objetos diferentes)*
- *Dados são somente para leitura*
- *Solução: Transfer Object ou TO 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)*

92

Solução: Value List Handler



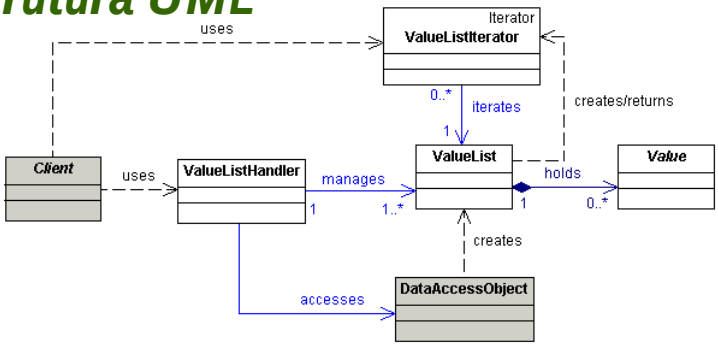
93

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*

94

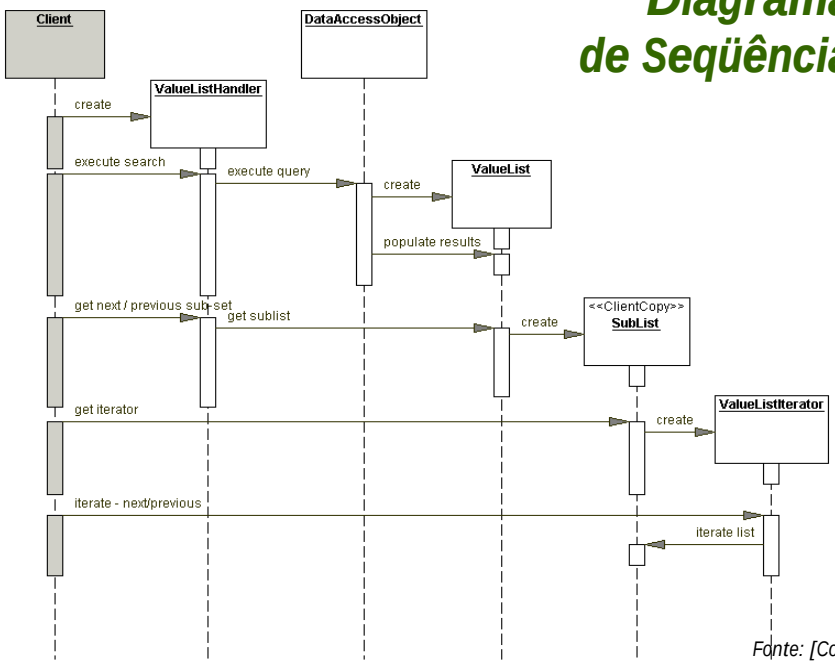
Estrutura UML



- *Client: qualquer cliente que precisa executar uma pesquisa que retorne grande quantidade de resultados.*
- *ValueListIterator: oferece métodos next(), hasNext(), etc.*
- *ValueListHandler: executa a pesquisa e obtém os resultados e os mantém em um objeto ValueList*
- *ValueList: Collection que guarda os resultados da pesquisa*
- *Value: um objeto de dados resultante da pesquisa*

Fonte: [Core] **95**

Diagrama de Seqüência



Fonte: [Core] **96**

Estratégias de implementação

- *Plain Old Java Object (POJO) Handler 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*
- *Value List Handler Session Façade 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.*
- *ValueList from DAO Strategy*

97

Conseqüências

- *Alternativa eficiente a EJB finders em pesquisas grandes*
- *Faz cache de resultados do lado do servidor*
- *Maior flexibilidade de pesquisa*
- *Melhora performance da rede*
- *Permite separação de interesses em camadas*
- *Criação de grandes quantidades de TOs pode ter alto custo*

98

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.

99

Curso J931: J2EE Design Patterns

Versão 1.1

www.argonavis.com.br

(helder@acm.org)