Padrões de Projeto J2EE

4

Padrões da Camada de Integração

Helder da Rocha (helder@acm.org)

Introdução

- *A camada de integração encapsula a lógica relacionada com a integração do sistema com a camada de informação distribuída (EIS)*
- *É acoplada com a camada de negócios sempre que esta camada precisar de dados ou serviços que residem na camada de recursos (dados)*
- *Os componentes nesta camada podem usar tecnologias de acesso aos serviços específicos que isolam (JDBC, JMS, middleware proprietário, etc.)*

Padrões da Camada de Integração

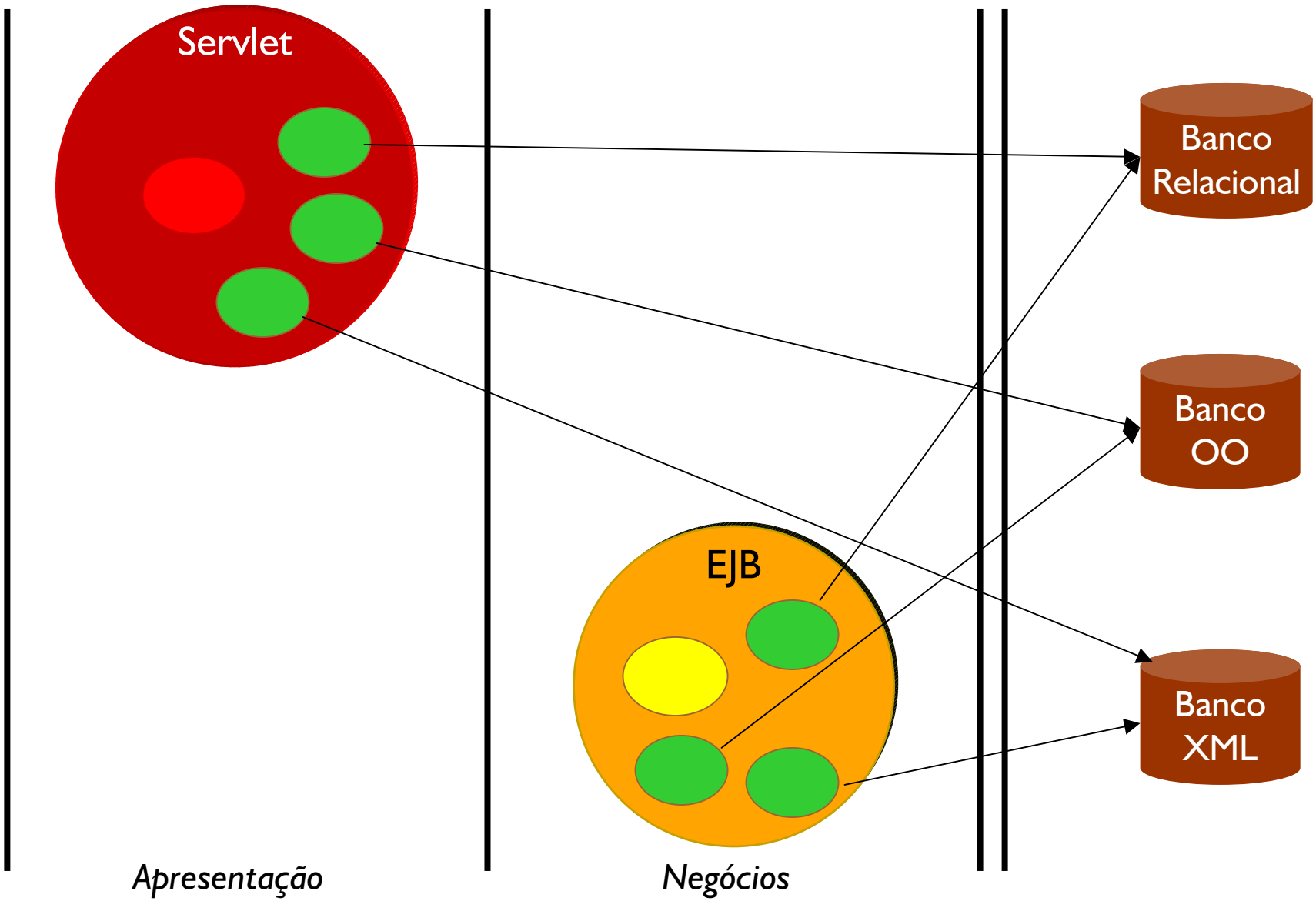
- *(14) Data Access Object*
  - *Abstrai fontes de dados e oferece acesso transparente aos dados*
- *(15) Service Activator*
  - *Facilita o processamento assíncrono para componentes EJB*

14

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.

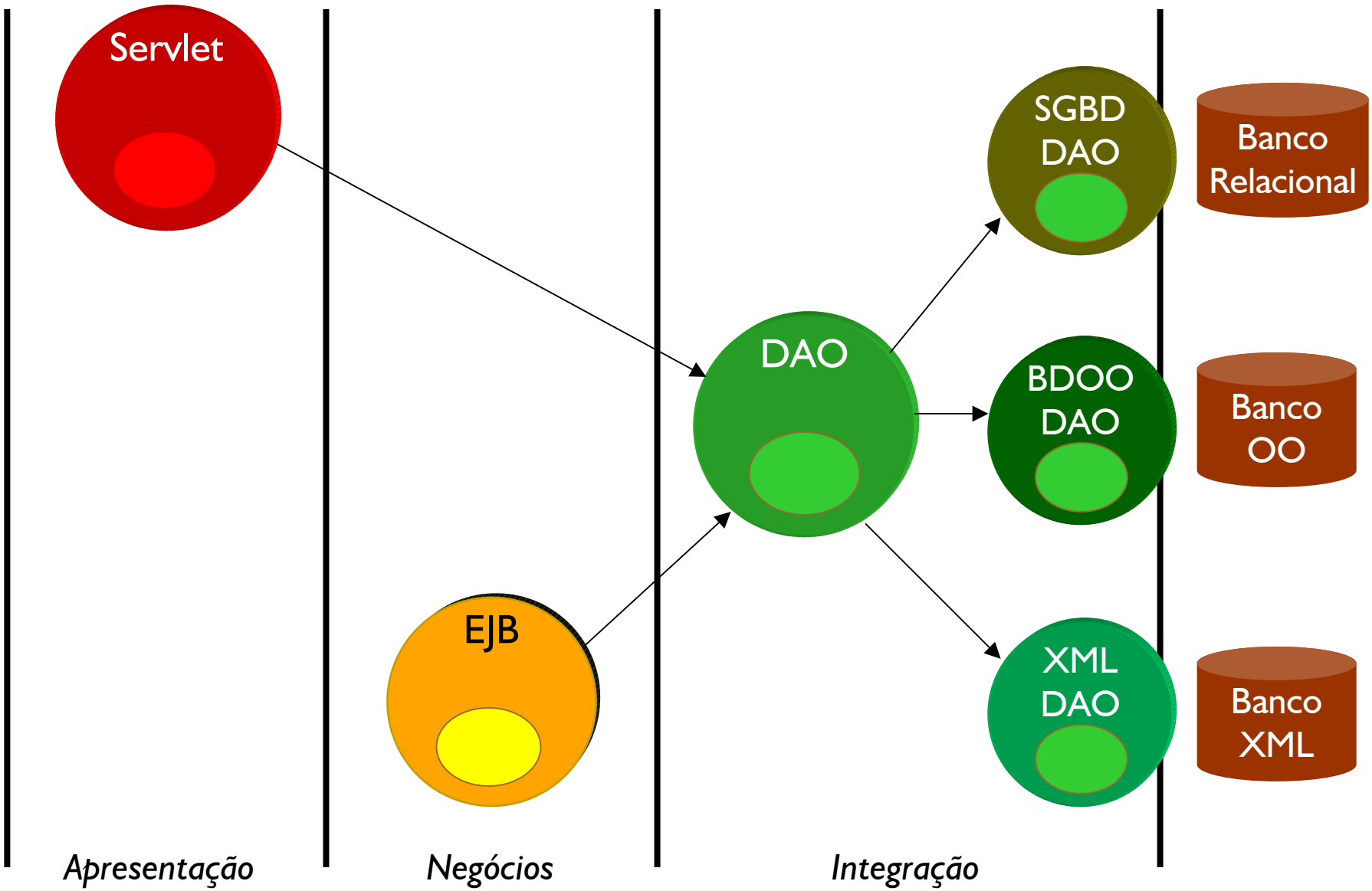
Problema


Servlet
Banco Relacional
Banco OO
EJB
Banco XML
Apresentação
Negócios

Descrição do problema

- *Forma de acesso aos dados varia consideravelmente dependendo da fonte de dados usado*
  - *Banco de dados relacional*
  - *Arquivos (XML, CSV, texto, formatos proprietários)*
  - *LDAP*
- *Persistência de objetos depende de integração com fonte de dados (ex: entity beans)*
  - *Colocar código de persistência (ex: JDBC) diretamente no código do objeto que o utiliza ou do cliente amarra o código desnecessariamente à forma de implementação*
  - *Ex: difícil passar a persistir objetos em XML, LDAP, etc.*

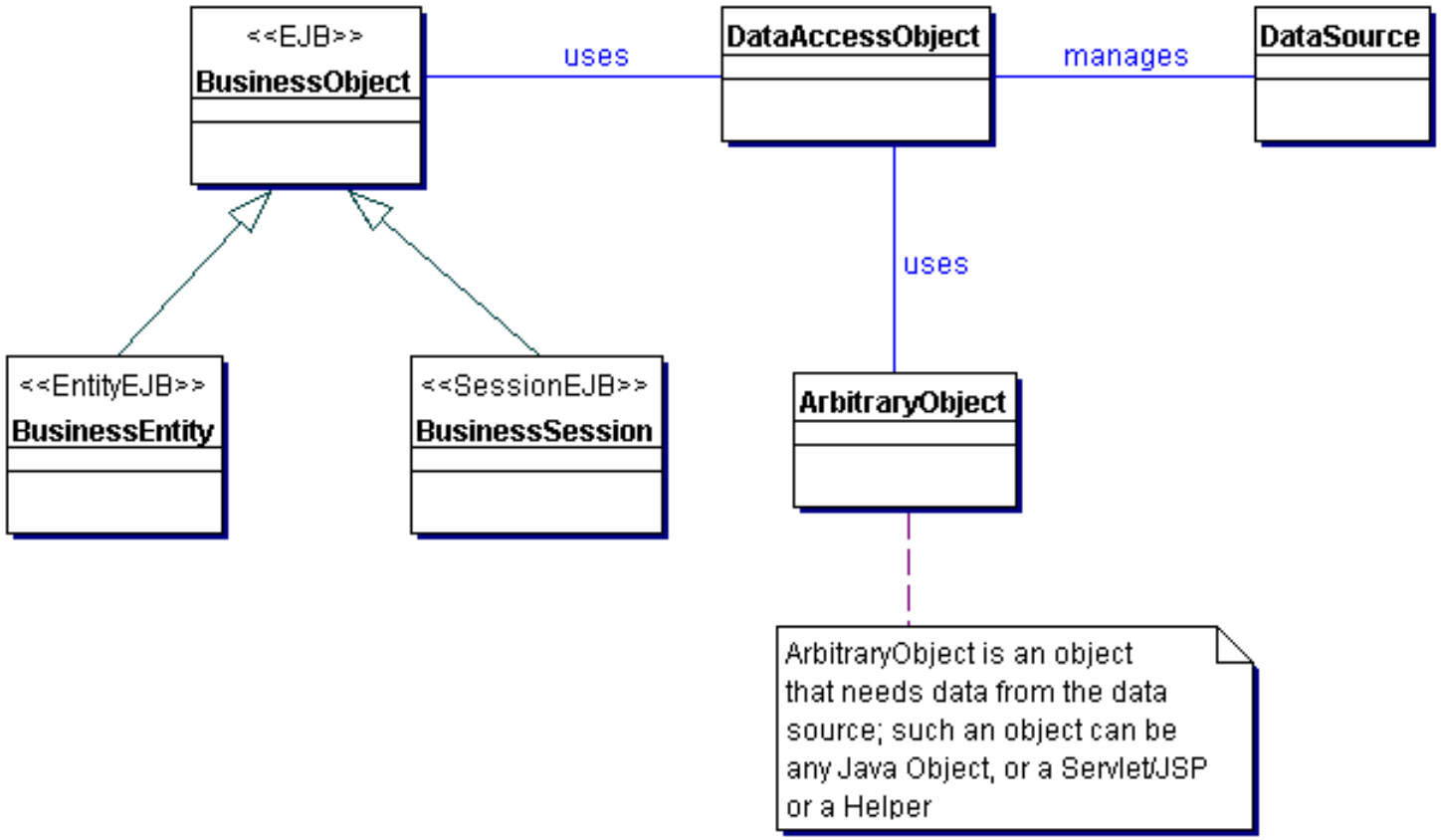
Solução: Data Access Object



Descrição da solução

- *Data Access Object oferece uma interface comum de acesso a dados e esconde as características de uma implementação específica*
  - *Métodos genéricos para ler e gravar informação*
  - *Métodos genéricos para concentrar operações mais comuns (simplificar a interface de acesso)*
- *DAO define uma interface que pode ser implementada para cada nova fonte de dados usada, viabilizando a substituição de uma implementação por outra*

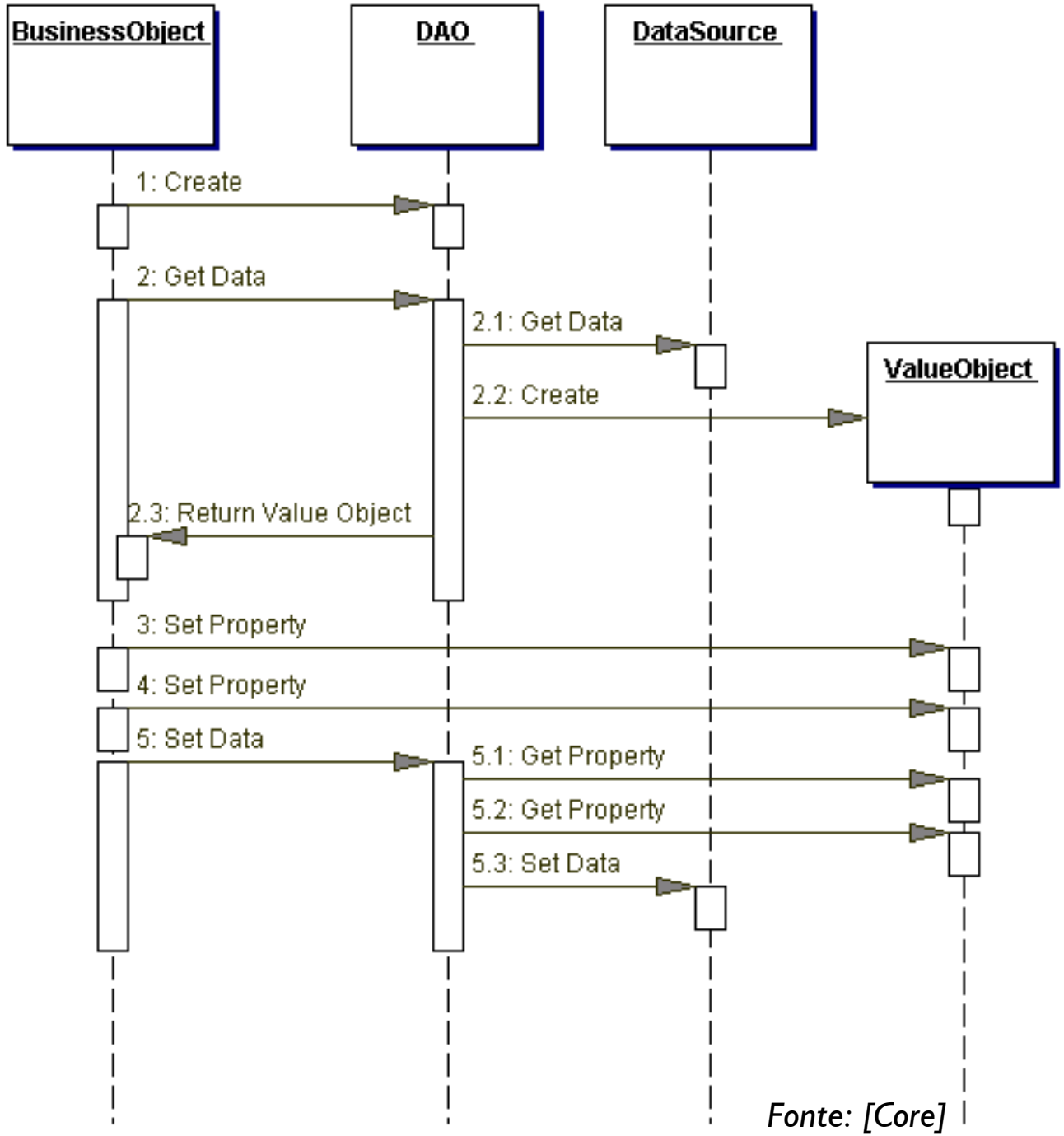
Estrutura UML



Fonte: [SJC]

Diagrama de Seqüência

Usando DAO e Value Object



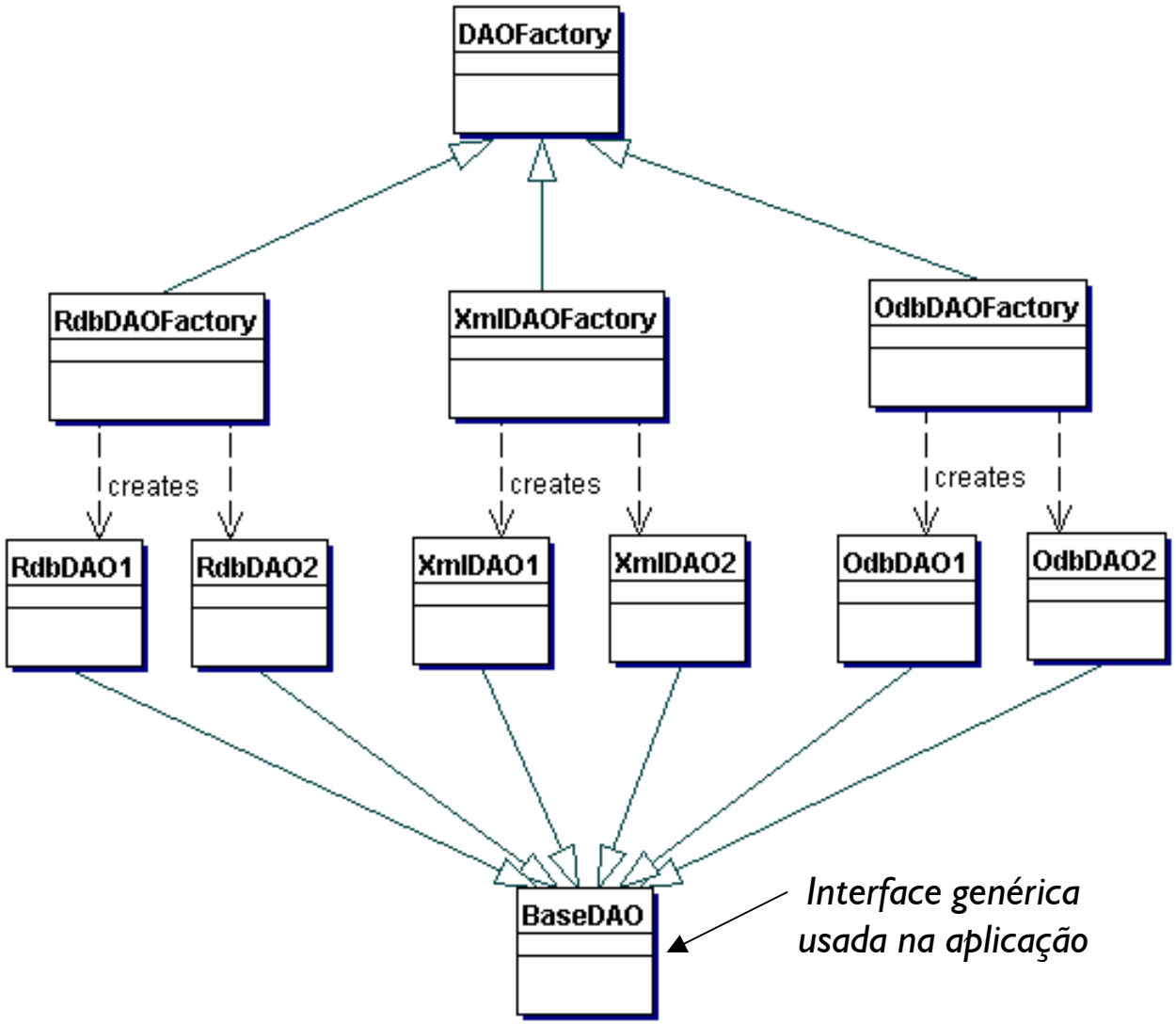
Fonte: [Core]

*Melhores estratégias de implementação**

- *Automatic DAO Code Generation Strategy*
  - *Sabendo-se a interface do DAO, pode-se gerar o código para um DAO específico (usando, por exemplo, ferramentas de introspecção, mapeamento O-R, etc.)*
- *Factory for Data Access Objects Stategy*
  - *Se a implementação de persistência não muda com freqüência, pode-se criar uma fábrica dos DAOs necessários para acessar os dados usando Factory Method*
  - *Para suporte a implementações plugáveis, a solução pode ser estendida para usar Abstract Factory, onde uma fábrica genérica pode ter diversas implementações*

** Veja exemplos de código em exemplos/eislayer/*

Factory for Data Access Objects Strategy (com Abstract Factory)



Fonte: [SJC]

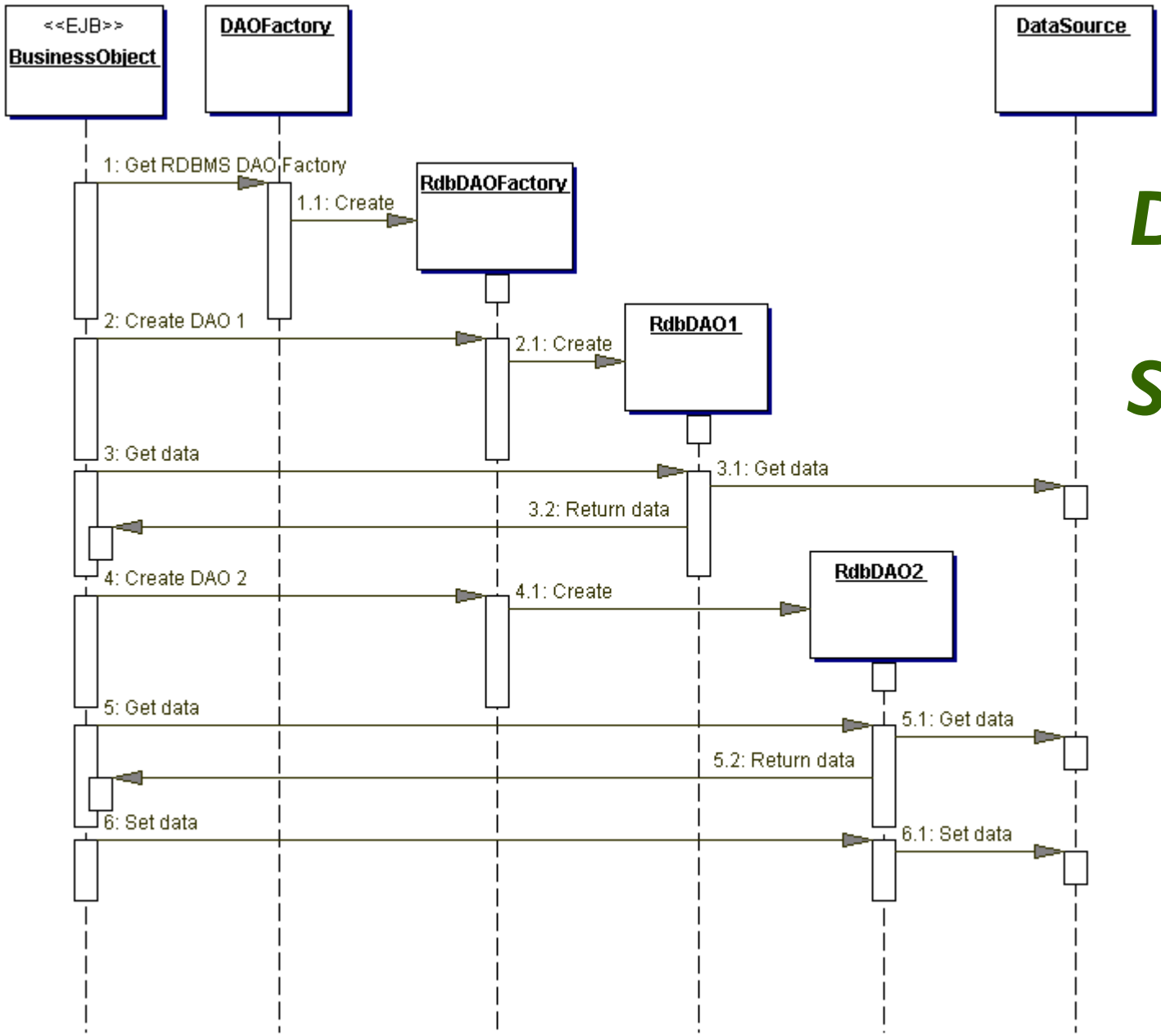


Diagrama de Seqüência

Fonte: [SJC]

Conseqüências

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

Exercícios

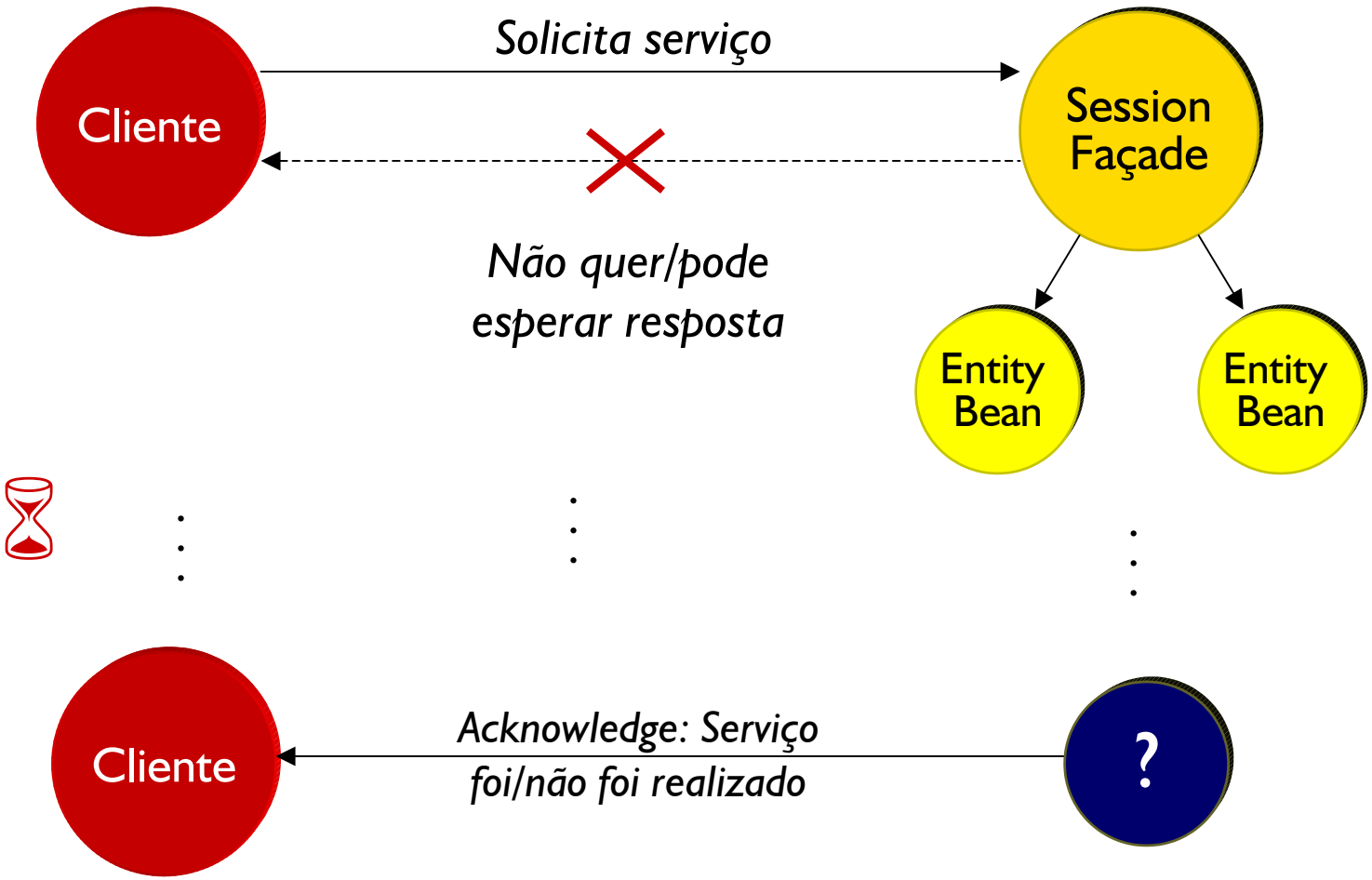
- *1. Analise a implementação das estratégias de Data Access Object (código 9.1 a 9.6)*
- *2. Analise o código do DAO existente (XML) e implemente um DAO e código para armazenar as mensagens no banco de dados Cloudscape:*
  - *a) Implemente a interface MessageBeanDAO*
  - *b) Implemente um mecanismo 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*

15

Service Activator

Objetivo: Receber requisições e mensagens assíncronas do cliente. Ao receber uma mensagem, o Service Activator localiza e chama os métodos de negócio necessários nos componentes para atender à requisição, de forma assíncrona.

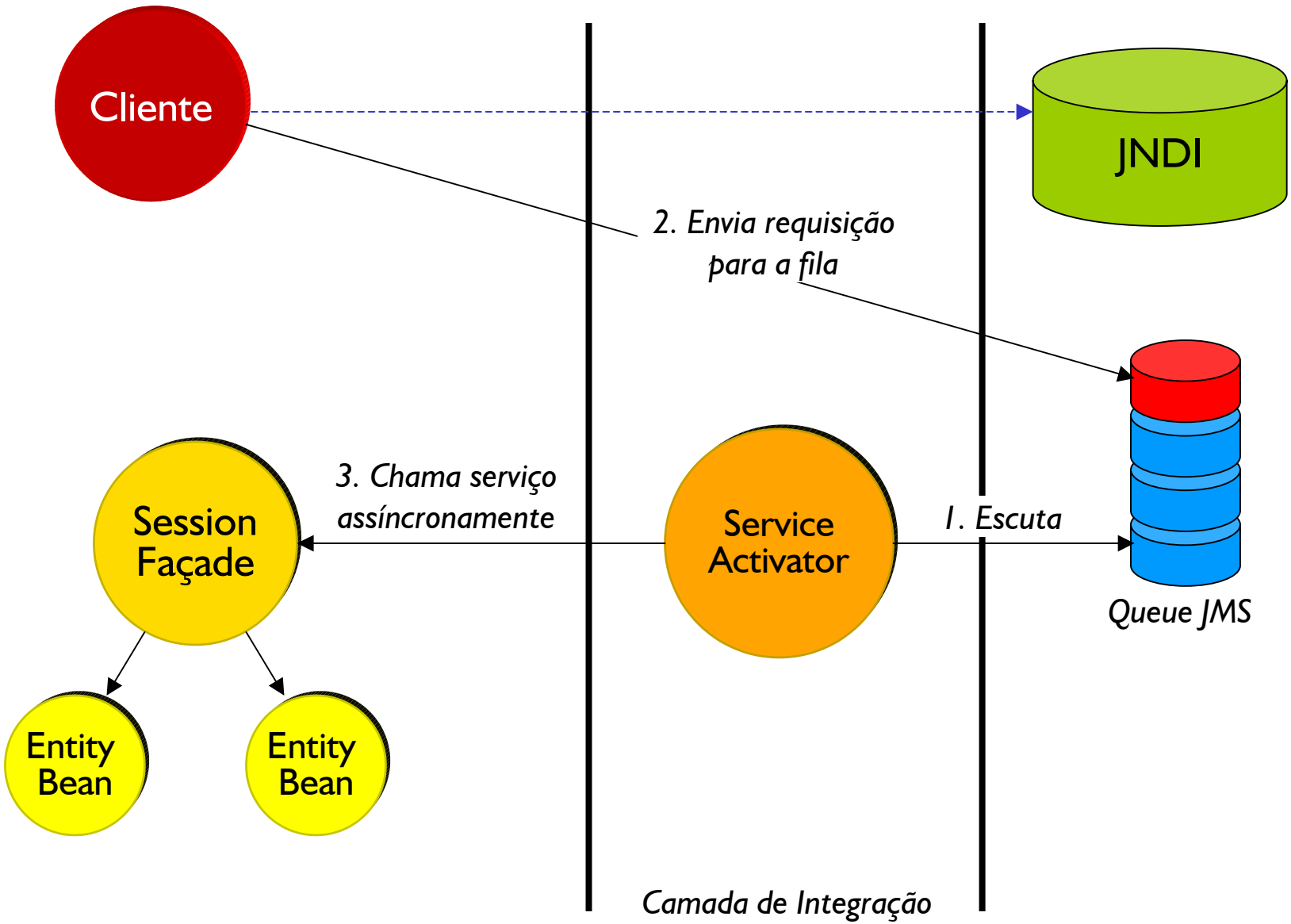
Problema


Cliente
Solicita serviço
Session Façade
Não quer/pode esperar resposta
Entity Bean
Entity Bean
Cliente
Acknowledge: Serviço foi/não foi realizado
?

Descrição do problema

- *Algumas aplicações precisam operar de forma assíncrona com um componente*
  - *Não pode esperar que uma requisição seja respondida ou não tem interesse na resposta (pelo menos não imediatamente)*
- *Acesso de Session beans e Entity beans da forma convencional (usando sua interface remota) é sempre realizado de forma síncrona*
  - *É preciso implementar um ponto de acesso no qual clientes possam deixar requisições para serem encaminhadas a EJBs logo que possível*
  - *EJBs devem poder "acordar" quando receberem a notificação e executar o serviço solicitado*

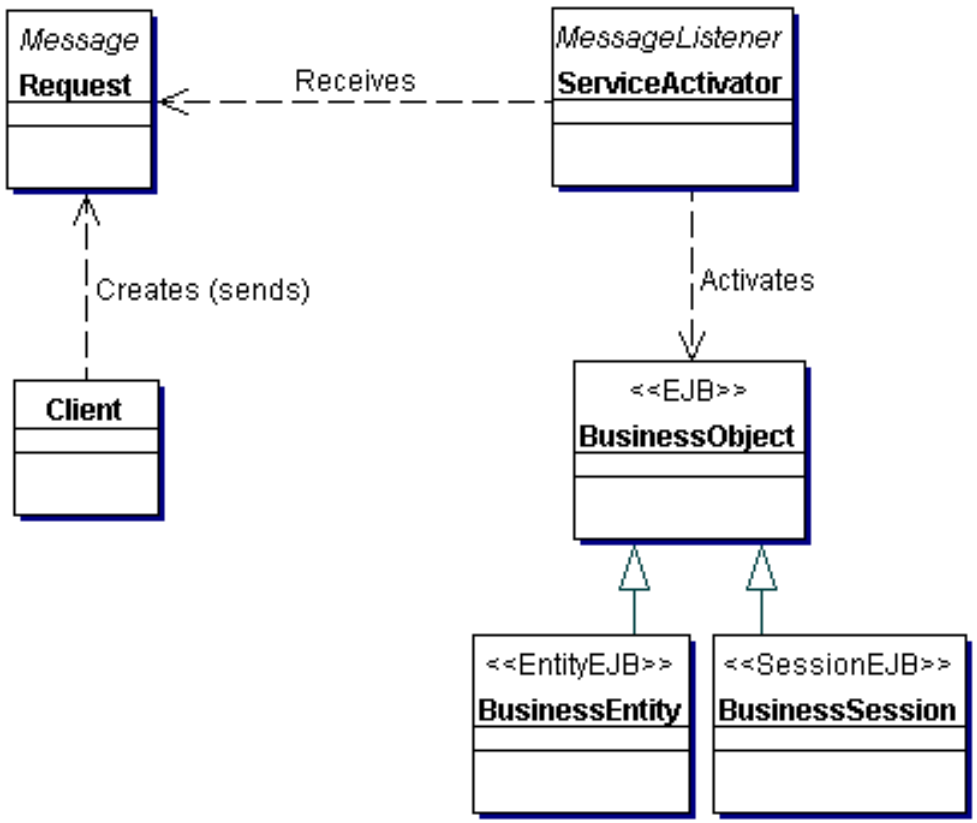
Solução: Service Activator



Descrição da solução

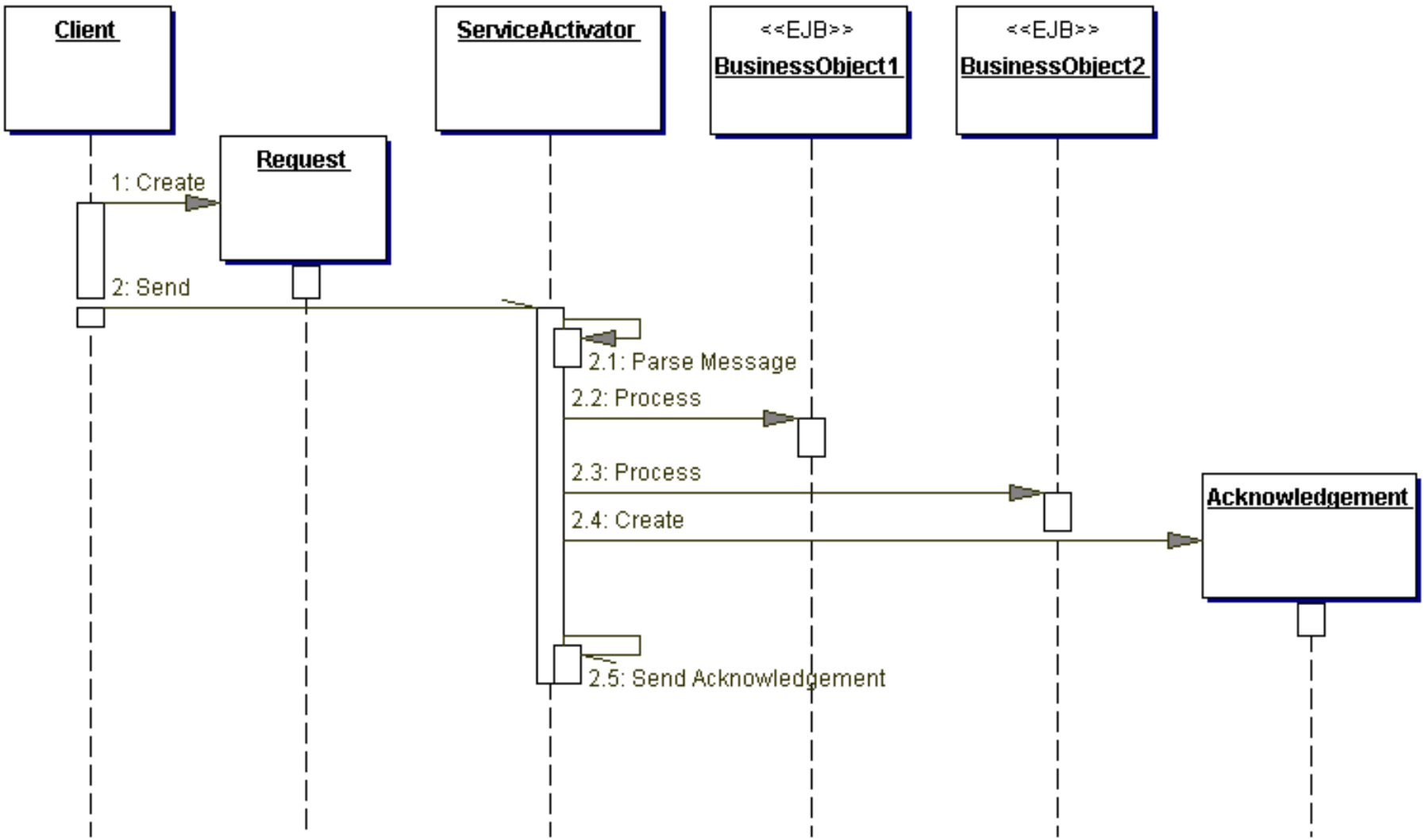
- *O Service Activator é um listener JMS*
  - *Pode ser implementado como um Message-driven bean ou como aplicação standalone*
- *Clientes que precisarem de chamar um serviço assíncronamente podem criar e enviar uma mensagem ao Service Activator*
  - *A mensagem é enviada a uma fila JMS usada pelo Activator*
  - *O Service Activator extrai as informações do serviço e localiza o componente que poderá executá-lo. Em seguida, chama os métodos necessários para atender à requisição do cliente.*
  - *Se o cliente desejar, o Service Activator pode enviar uma mensagem confirmando a realização do serviço com sucesso.*

Estrutura UML



Fonte: [Core]

Diagramas de Seqüência



Fonte: [Core]

Melhores estratégias de implementação

- *Entity Bean Strategy*
  - *Entity bean é o objeto de negócio interceptado*
- *Session Bean Strategy*
  - *Session bean é o objeto interceptado*
- *Service Activator Server Strategy*
  - *Implementa o Service Activator como um servidor standalone que escuta e processa mensagens JMS*
  - *Pode também ser implementado como um serviço do servidor de aplicações ou componente MDB*
- *Enterprise Bean as Client Strategy*
  - *Um EJB pode chamar serviços em outro EJB usando o Service Activator*

Conseqüências

- Permite processamento assíncrono de EJBs
  - Session e Entity Beans não podem ser chamados de forma assíncrona. Service Activator se coloca entre eles e o cliente para viabilizar isto
- Não precisa ser implementado com MDB
  - Todas as estratégias mostram como implementar Service Activator sem ou independente de MDB
  - Funciona em implementações antigas de EJB
- Pode ser executado como um processo standalone
- Pode ser implementado como um MDB
  - Porém está sujeito às mesmas limitações da plataforma (passivação, gerenciamento pelo container, etc.)

Exercícios

- *1. Analise a implementação das estratégias de Service Activator (código 9.7 a 9.9)*
- *2. Implemente um Service Activator para chamar o serviço de criar nova mensagem*
  - *a) Implemente-o como um MDB e configure-o*
  - *b) Crie um novo comando Mensagem Retardada que espere 30 segundos antes de criar uma mensagem (com texto fixo). Ele deve enviar uma mensagem contendo os dados necessários para a mesma fila no qual o Service Locator foi configurado*
  - *c) No método onMessage(), extraia os dados, crie um objeto hello.jsp.MessageBean e chame o serviço.*

Fontes

[SJC] SJC Sun Java Center J2EE Patterns Catalog. http://developer.java.sun.com/developer/restricted/patterns/J2EEPatternsAtAGlance.html. Versão desatualizada dos padrões J2EE do SJC, porém tem exemplos adicionais.

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

[Core] Deepak Alur, John Crupi, Dan Malks. Core J2EE Patterns: Best Practices and Design Strategies. Prentice-Hall, 2001. http://java.sun.com/blueprints/corej2eepatterns/index.html. Descrição detalhada dos padrões SJC. O site é mais atualizado que o livro.

Curso J931: J2EE Design Patterns

Versão 1.0

www.argonavis.com.br

(helder@acm.org)