UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Edjandir Corrêa Costa

ROTEIRO PARA DESENVOLVIMENTO DE COMPONENTES ENTERPRISE JAVABEANS

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação

Orientador: João Bosco Mangueira Sobral, Dr.

Florianópolis, Outubro de 2000.

ROTEIRO PARA DESENVOLIVIMENTO DE COMPONENTES ENTERPRISE JAVABEANS

Edjandir Corrêa Costa

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação Área de Concentração Sistemas de Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

_____

João Bosco Mangueira Sobral, Dr.

Orientador

_____

Fernando Álvaro Ostuni Gauthier, Dr.

Coordenador do CPGCC

Banca Examinadora

_____

Mirela Sechi Moretti Annoni Notare, Dr.

_____

Bernardo Gonçalves Riso, Dr.

_____

Ricardo Pereira e Silva, Dr.

_____

Euclides de Moraes Barros Júnior, M.Sc.

Gostaria de agradecer sinceramente a todos aqueles que direta ou indiretamente contribuíram para a elaboração deste trabalho. De maneira particular expresso minha gratidão ao professor João Bosco Mangueira Sobral, pela dedicação e amizade que marcaram o seu papel de orientador durante todo o desenvolvimento deste trabalho.

SUMÁRIO

3 - OBJETOS DE NEGÓCIO DISTRIBUÍDOS

4 - ENTERPRISE JAVABEANS

5 – O ROTEIRO PROPOSTO

6 – APLICAÇÃO DO ROTEIRO

7 – CONCLUSÕES E SUGESTÕES PARA TRABALHOS FUTUROS

CONTENTS

3 – DISTRIBUTED BUSINESS OBJECTS

4 - ENTERPRISE JAVABEANS

5 – THE GUIDE OF COMPONENT DEVELOPMENT

6 – THE GUIDE APPLICATION

LISTA DE FIGURAS

LISTAGENS DE CODIFICAÇÃO

RESUMO

Este trabalho apresenta um roteiro que mostra as atividades existentes no processo de desenvolvimento de componentes de negócio. O roteiro mostra, também, a sequência de execução dessas atividades.

O roteiro sugere, como forma de organização, a divisão do processo de desenvolvimento em quatro etapas: modelagem de negócio, projeto, implementação e instalação de componentes. O roteiro assume que os componentes sejam implementados de acordo com o modelo *Enterprise Javabeans.*

Além do roteiro desenvolvido, este trabalho apresenta tecnologias envolvidas no processo de construção de componentes de negócio como objetos de negócio, objetos distribuídos e o modelo *Enterprise Javabeans.*

ABSTRACT

This work presents a guide that shows the activites of business components development process. The guide shows the sequence of execution of that activities too.

The guide suggests the division of process development in four global activities: business modelling, design, implementation and deployment of components. The guide assumes that business components are implemented using the Enterprise Javabeans model.

This wok shows too, tecnologies envolved in business components construction such as business objects, distributed objects and Enterprise Javabeans component model.

1 – INTRODUÇÃO

O presente trabalho situa-se no contexto da Computação de Objetos Distribuídos e propõe um roteiro para ser aplicado ao processo de desenvolvimento de objetos de negócio. Este roteiro contempla, apenas, o desenvolvimento de objetos de negócio, e não o desenvolvimento de aplicações. Os termos "objeto de negócio" e "componente de negócio" podem ser utilizados de forma intercambiável. Um objeto de negócio é um componente de software que representa uma entidade da vida real, em contraste com objetos a nível de sistemas que representam entidades que fazem sentido somente a programadores e sistemas de informação.

O roteiro apresentado neste trabalho tem como foco o desenvolvimento de objetos de negócio baseados no modelo de componentes de servidor *Enterprise Javabeans*. Os objetos de negócio, implementados de acordo com a especificação *Enterprise Javabeans*, podem ser instalados em qualquer plataforma e servidor que implemente o modelo. Além da independência de plataforma, objetos de negócio implementados de acordo com o modelo Enterprise Javabeans, podem utilizar os serviços oferecidos pelos sistemas de monitoramento de transações baseados em componentes, de forma automática. Essa característica, permite que o desenvolvedor do objeto se preocupe apenas com a lógica de negócio.

A adoção de objetos de negócio no desenvolvimento de aplicações, combinada com um método consistente e uma infra-estrutura que ofereça os serviços necessários, torna possível a obtenção de um número de benefícios como flexibilidade, facilidade de adaptação, facilidade de manutenção, reutilização e interoperabilidade.

1.1 – Justificativa

O surgimento das técnicas de reengenharia de negócios, no início da década de 90, impulsionou uma reformulação na forma como os sistemas eram então arquitetados. Nessa mesma época, as técnicas de modelagem com o enfoque orientado por objetos, estavam se firmando como ferramentas de análise e projeto de sistemas de informação. Taylor foi um dos pioneiros na aplicação de técnicas de modelagem, utilizadas por desenvolvedores de sistemas de informação, na reformulação de atividades de negócio (TAYLOR, 1995). Jacobson propôs, em seu livro "*The Object Advantage: Business Process Reengineering With Object Technology*", um método para reengenharia de negócios baseado em técnicas de modelagem de objetos (JACOBSON, 1995). A atividade de reformulação de negócios baseada em técnicas de modelagem foi chamada de Reengenharia de Processos de Negócios. Os sistemas de informação que gerenciavam as atividades do negócio foram, também, alvos de uma reformulação.

A presença de características como flexibilidade, reutilização, facilidade de adaptação e interoperabilidade, nos sistemas de informação de suporte aos negócios, passou a ser exigida. Estas características são os princípios da tecnologia de objetos. Até então, os sistemas desenvolvidos sob a abordagem orientada por objetos, não ofereciam, efetivamente, as vantagens que a tecnologia poderia oferecer.

Outra característica, que passou a ser exigida nos sistemas de informação, era a de que a sua evolução deveria acontecer em sincronia com a evolução do negócio. Essa característica implicava no uso das mesmas técnicas aplicadas na modelagem de negócio (ou reengenharia de negócio) na modelagem do sistema de informação. O método proposto por Jacobson (JACOBSON, 1995) pode ser utilizado na modelagem de negócio assim como na modelagem de sistemas de informação. Jacobson, no seu trabalho, foi além, classificando a atividade de construção do sistema de informação como parte do processo da reengenharia de negócios.

O trabalho de reengenharia de negócio envolve, também, a reformulação de sistemas de informação. A técnica de modelagem baseada em objetos pode ser utilizada em todo processo de reengenharia de negócio (incluindo a construção do sistema de informação), como foi comprovado por Jacobson (JACOBSON, 1995).

Os objetos de negócio representam os elementos fundamentais em um negócio. O cliente de uma empresa, assim como o pedido que ele faz, podem ser visto como objetos de negócio. O trabalho na modelagem de um negócio, utilizando uma abordagem orientada por objetos, consiste na identificação dos objetos de negócio e na definição das interações destes objetos com as atividades e recursos existentes no negócio. Se, em um projeto de reengenharia de negócios, o desenvolvimento de um sistema de informação estiver envolvido, os objetos identificados na modelagem do negócio podem ser implementados como componentes de negócio. Para isso ser possível, é necessária, inicialmente, a utilização de uma linguagem de programação orientada por objetos. Para que os objetos possam ser utilizados em um ambiente distribuído, também, tornam-se necessários serviços que forneçam recursos de sistemas distribuídos como serviço de segurança, transações, e outros. Para que o sistema de

informação tenha as características exigidas pelo negócio, é essencial que exista uma plataforma que ofereça facilidades, para que o desenvolvedor de objetos de negócio possa se preocupar apenas com o mapeamento dos objetos identificados na modelagem de negócio. O modelo de componentes *Enterprise Javabeans* (SUN MICROSYSTEMS, 1997), proposto pela Sun Microsystems, atende a estes requisitos. Os objetos de negócio são implementados como componentes de servidor. O servidor, no qual residem os componentes, oferece a eles os serviços necessários para atender várias aplicações-cliente, como controle de transação, serviço de segurança e serviço de persistência.

1.2 – Objetivos

Este trabalho tem como objetivo a elaboração de um roteiro para ser aplicado ao processo de desenvolvimento de objetos de negócio baseado no modelo de componentes de servidor *Enterprise Javabeans*.

1.2.1 - Objetivos específicos

- Propor técnicas que possam ser utilizadas no desenvolvimento de componentes (implementação) baseado em objetos de negócio (lógica de negócio).
- Dividir o processo de desenvolvimento em etapas, permitindo que trabalhos específicos possam ser realizados em cada etapa. O benefício da divisão do processo em etapas, é que, além da organização obtida, torna possível a participação de profissionais especialistas em cada fase do processo.

- Oferecer um roteiro (manual de desenvolvimento) aos profissionais envolvidos no processo de desenvolvimento de componentes.
- Facilitar o gerenciamento, pelos gerentes de processo, no desenvolvimento de objetos de negócio.
- Possibilitar a integração de tecnologias de objetos distribuídos no desenvolvimento de componentes.
- Experimentar o modelo *Enterprise Javabeans* e seus serviços para a implementação de objetos de negócio.

1.3 - Abrangência

O roteiro apresentado neste trabalho oferece suporte ao desenvolvimento de componentes *Enterprise Javabeans*. O processo de desenvolvimento de aplicações está fora do escopo deste trabalho. O roteiro sugere que o processo seja divido em etapas, cada uma representando um conjunto de atividades relacionadas no processo de desenvolvimento objetos de negócio.

Na disciplina de Engenharia de Software, o termo “ciclo de vida do projeto de sistema” é utilizado para representar todas as fases envolvidas no processo de desenvolvimento de sistemas (YOURDON, 1990). Algumas fases do ciclo, como análise e projeto de sistemas, existem na maioria das metodologias de desenvolvimento. O roteiro desenvolvido sugere a inclusão de fases correspondentes ao ciclo de vida de projeto de sistema tradicional, mas direcionadas ao desenvolvimento de componentes.

O ciclo de vida do projeto de sistema tradicional deve ser adaptado quando, na construção de aplicações, são utilizados componentes já fabricados.

1.4 – Forma de organização do trabalho

Como forma de organização, o trabalho foi dividido em sete capítulos. Os capítulos iniciais fazem um estudo dos conceitos e tecnologias envolvidas no processo de desenvolvimento de componentes *Enterprise Javabeans*, enquanto os capítulos finais mostram o roteiro, proposto neste trabalho, e a sua aplicação.

O Capítulo 2 apresenta os conceitos básicos da tecnologia de objetos de negócio. Nele é mostrada a vantagem da utilização de objetos de negócio na representação das entidades e processos existentes em um negócio. São também citados os serviços necessários para a implementação de objetos de negócio e justifica a necessidade de uma plataforma para implementação dos mesmos.

O Capítulo 3 faz um estudo sobre a tecnologia de objetos distribuídos. Os objetos de negócio atuam em um ambiente distribuído, necessitando assim, de serviços que ofereçam a integração dos processos envolvidos em um negócio. Algumas tecnologias de sistemas distribuídos, como os sistemas de monitoramento de transações, já existem há muitos anos. É mostrada a evolução desses sistemas em direção à tecnologia de objetos e apresentados, também, conceitos importantes, como modelo de componentes de servidor.

O Capítulo 4 mostra como o modelo *Enterprise Javabeans* oferece suporte a implementação de objetos de negócio. O modelo proposto pela *Sun Microsystems* em 1997 (SUN MICROSYSTEMS, 1997) apresentou uma plataforma que realmente poderia ser utilizada por desenvolvedores de componentes de negócio. Os componentes construídos de acordo com o modelo *Enterprise Javabeans* utilizam os serviços dos

sistemas de monitoramento de transações de forma automática, permitindo assim, que o desenvolvedor de componentes se preocupe apenas com a lógica de negócio que o componente implementa. Conceitos importantes como servidor, container, e tipos de componentes *Enterprise Javabeans* são apresentados neste capítulo.

Neste capítulo são mostrados, também, os mecanismos que permitem a flexibilidade existente nos componentes construídos de acordo com a arquitetura *Enterprise Javabeans.* Os componentes podem ser configurados no momento da instalação, permitindo a configuração adequada para cada negócio.

O Capítulo 5 apresenta o roteiro para implementação de objetos de negócio como componentes *Enterprise Javabeans*. É mostrado como o processo de desenvolvimento de objetos de negócio baseados em componentes *Enterprise Javabeans* deve acontecer, identificando as etapas bem como as atividades realizadas em cada etapa do processo.

O capítulo 6 mostra como o roteiro pode ser aplicado ao processo de desenvolvimento de componentes utilizando, para isso, um estudo de caso.

O Capítulo 7, finalmente, apresenta as conclusões sobre o trabalho e, também, comenta sobre possíveis trabalhos a serem realizados no futuro.

2 - OBJETOS DE NEGÓCIO

Os objetos de negócio podem atuar como recursos para modelagem e reengenharia dinâmica de negócios. Eles podem ser utilizados no empacotamento de políticas de negócio, de processos de negócio, de dados e de definições. Além disso, os objetos de negócio ajudam a gerenciar a complexidade da arquitetura de sistemas baseados em objetos distribuídos – uma tendência atual na computação cliente-servidor em três camadas.

Em sistemas convencionais, apenas os dados de um negócio são compartilhados, já em sistemas que utilizam uma arquitetura baseada em objetos de negócio, o compartilhamento se estende a dados, processos e políticas de negócio. A utilização de objetos, como um meio de entender e modelar processos de negócio, pode ser um grande passo em direção ao sucesso para organizações.

2.1 - Objeto – o elemento fundamental

Para entender o que são objetos de negócio, primeiro é essencial estabelecer o que realmente significa o termo objeto. Booch, informalmente, define objeto como sendo uma entidade que apresenta algum tipo de comportamento bem definido (BOOCH, 1991). Jacobson define objeto como uma entidade capaz de gravar o seu

estado (informação) e que oferece um número de operações (comportamento) para pesquisar ou modificar o seu estado (JACOBSON et al, 1992).

Os objetos não são apenas blocos de programação. Eles podem, também, descrever processos de negócio. Sendo assim, os objetos podem ser considerados uma alternativa útil para representar ambos, elementos de programação e elementos de negócio de forma intercambiável.

2.2 - As categorias de objetos

Para Shelton, os objetos podem ser classificados em categorias distintas: objetos de negócio, objetos de implementação e objetos de aplicação (SHELTON, 1995). Um objeto pode ser, apenas, de uma categoria. O processo de desenvolvimento e obtenção é diferente para cada categoria de objeto. Os sistemas de informação podem ser reestruturados e colocados em funcionamento no momento da aquisição dos objetos requeridos pelo negócio em questão. O paradigma tradicional de desenvolvimento de aplicações pode ser substituído por uma construção de software muito mais flexível.

2.2.1 - Objetos de negócio

Objetos de negócio são objetos que representam pessoas, lugares, coisas ou conceitos no domínio do negócio. Eles empacotam procedimentos de negócio e políticas em volta dos dados. Objetos de negócio servem como um lugar de armazenamento para políticas e dados afins. A normalização semântica é o processo

que identifica os dados certos para as políticas de negócio certas, endereçando a um lugar de armazenamento comum – o objeto de negócio.

A OMG (Object Management Group) estabeleceu a definição de objeto de negócio da seguinte forma (OMG, 1997 a):

> "Um objeto de negócio é uma representação de uma entidade ativa em um domínio de negócio, incluindo no mínimo seu nome e definição, atributos, comportamento, relacionamentos e regras. Um objeto de negócio pode representar, por exemplo, uma pessoa, lugar ou conceito. A representação pode ser em linguagem natural, em linguagem de modelagem, ou em linguagem de programação."

2.2.2 - Objetos de implementação

Objetos de implementação representam conceitos de programação. Eles são os componentes dos sistemas de informação e ambientes de aplicação. Exemplos de objetos de implementação incluem componentes de interfaces visuais como janelas e botões de comando, objetos de linguagens de programação como as classes *string* e *integer*, serviços CORBA, banco de dados e *frameworks* de aplicação. Estes são (ou deveriam ser) recursos prontos que os projetistas e programadores utilizam para a construção dos sistemas de informação. Os objetos de implementação devem ser selecionados de acordo com o seu grau de conformidade em relação a padrões estabelecidos (como CORBA, por exemplo). A padronização favorece a integração dos

objetos de implementação com uma variedade de outros objetos compatíveis. Os objetos de negócios, quando implementados, tornam-se também objetos de implementação.

2.2.3 - Objetos de aplicação

Objetos de aplicação são programas que apresentam informações e gerenciam a interação com usuários (pessoas), processam informação, e produzem relatórios. Eles são soluções para problemas específicos, manipulam uma série de objetos de implementação (incluindo objetos de negócio implementados) para realizar uma tarefa específica. Exemplos incluem programas utilizados nos registros de pedidos, programas que geram relatórios periódicos, programas para gerenciamento de contas, etc.

Uma classificação semelhante é sugerida pela Andersen Consulting no seu documento "*Understanding Components*" que relata as experiências com o desenvolvimento baseado em componentes (incluindo objetos de negócio) e que estabelece uma arquitetura para construção de sistemas de informação baseados em componentes – a arquitetura Eagle (ANDERSEN CONSULTING, 1998). Nessa arquitetura, os objetos são classificados em: objetos de negócio, objetos particionados e objetos de implementação.

O termo *componente* representa todos os tipos de objetos utilizados na construção de aplicações. Este termo tem um significado muito amplo. Segundo Orfali, *componente* é um objeto que não está ligado a uma única aplicação, a uma linguagem de programação específica ou a uma implementação (ORFALI & HARKEY, 1996).

Podem ser considerados como *componentes* aqueles objetos de interface visuais e objetos de negócios que já se encontram implementados (objetos particionados).

Objetos de interface não podem ser considerados objetos de negócio. Objeto de negócio é um termo mais específico que *componente* e que pode ser utilizado com maior precisão na modelagem de negócios e no desenvolvimento de *frameworks* de negócio. Um *framework* é uma coleção de classes que oferece uma série de serviços para um domínio de aplicação específico (BOOCH, 1994).

2.3 - Os tipos de objetos de negócio

Os objetos de negócio são categorizados por Eeles e Sims (EELES & SIMS, 1998) em duas variedades: objetos de negócio do tipo-entidade e objetos de negócio do tipo-processo. Esta categorização é útil na implementação de soluções práticas na modelagem e reengenharia de negócios e no desenvolvimento de sistemas de suporte ao negócio.

2.3.1 - Objetos de negócio do tipo-entidade

Objetos de negócio do tipo-entidade são os objetos mais comuns. Eles representam pessoas, lugares ou coisas, da mesma maneira que uma entidade na modelagem de dados. As diferenças fundamentais entre a entidade encontrada nos diagramas de Entidade–Relacionamento (ER) e o objeto de negócio do tipo entidade são:

- O objeto encapsula funções e regras que são específicas para o conceito representado, enquanto a entidade de dados encapsula apenas dados;

- O objeto de negócio do tipo-entidade pode possuir relações mais complexas que a entidade do modelo ER, como composições, por exemplo.

2.3.2 - Objetos de negócio do tipo-processo

Objetos de negócio do tipo processo são os verbos de um negócio. Eles representam os processos de negócio caracterizados pela interação de uma série de objetos de negócio. Os objetos de negócio interagem, de uma maneira esperada e repetida, para produzir um resultado também esperado.

Os objetos de negócio do tipo-entidade são atores que participam de um processo de negócio. Cada interação entre um par de objetos de negócio do tipo-entidade representa uma etapa do processo. Os objetos de negócio do tipo-entidade encapsulam a política que determina como o processo de negócio será executado. Os objetos de negócio do tipo-processo encapsulam o processo de negócio em um objeto.

Um processo de negócio é uma ação que sempre envolve mais de um ator. Suas instâncias (objetos de negócio do tipo-processo) representam unidades de trabalho em curso, e não fatos sobre uma pessoa envolvida no negócio nem mesmo lugares ou coisas. Seus atributos são referências a objetos de negócio do tipo-entidade que participam do processo. Seus serviços iniciam, finalizam, avançam e classificam unidades de trabalho. O objeto de negócio do tipo-processo é similar, em conceito, a um fluxo de trabalho *(workflow)* de um programa, comparando a forma como ele organiza e faz o controle dos passos que são necessários para completar uma unidade de trabalho. A Figura 2-1 mostra o fluxo de trabalho *(workflow)* gerenciado pelo objeto de negócio do tipo-processo de nome Venda. Este objeto de negócio representa um processo de

negócio que é a venda de algum bem de valor. Este objeto de negócio encapsula as interações entre os objetos de negócio do tipo-entidade. O fluxo por inteiro representa uma transação, onde valores (dados) dos objetos de negócio do tipo-entidade (Cliente, por exemplo) podem ser modificados.

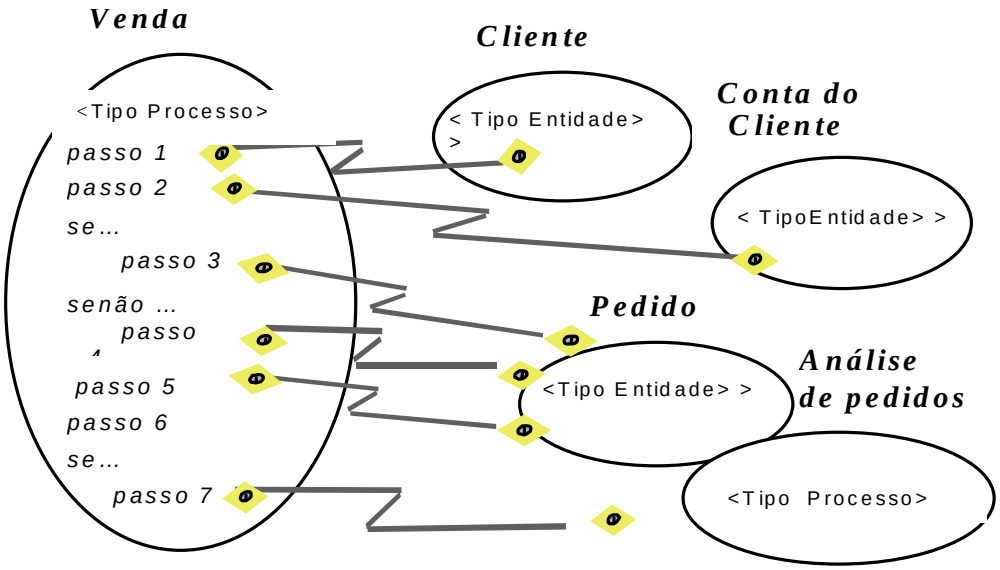


Figura 2.1- Objetos de negócio.

Quando implementados, os objetos de negócio devem oferecer integridade em relação aos dados. Cada instância de um objeto de negócio do tipo-processo deve possuir objetos de negócio do tipo-entidade únicos. Isso significa que um processo não pode competir com outro (uma outra instância do mesmo objeto de negócio do tipo-processo) que modifica os dados das entidades envolvidas.

Um outro requisito é que a modificação dos valores dos objetos de negócio do tipo-entidade só pode ser efetivada se o fluxo de trabalho for executado por completo. Nos gerenciadores de bancos de dados, o recurso que garante a integridade dos dados

das entidades envolvidas num fluxo de trabalho, se chama *controle de transação* e existe há mais de uma década. Os objetos de negócio também têm a necessidade de utilizar um recurso semelhante.

2.4 - Serviços de suporte para objetos de negócio

A OMG criou uma especificação para padronização de serviços de controle de transação que suportam a arquitetura CORBA (OMG, 1997c). Para suportar transações, os objetos de negócio do tipo-entidade devem ser desenvolvidos de acordo com o padrão CORBA. Esta especificação, não é proprietária e é um dos principais padrões adotados pela indústria de desenvolvimento de sistemas de informação baseados em objetos distribuídos.

Um outro serviço importante é o serviço de nomes (OMG, 1997d). Este serviço permite que um objeto de negócio implementado possa ser identificado por um nome dentro de um ambiente distribuído. O serviço de nomes é, também, uma especificação baseada na arquitetura CORBA (OMG, 1993).

Vários outros serviços tiveram suas especificações desenvolvidas pela OMG e podem ser utilizados como suporte de uma arquitetura para a construção de objetos de negócio. O serviço de persistência (OMG, 1997e), que fornece suporte a persistência de objetos, e o serviço de segurança (OMG, 1997b) são exemplos desses serviços.

2.5 - Arquitetura para construção de objetos de negócio

A OMG vem fazendo esforços em busca de uma padronização para a construção e a utilização plena de uma arquitetura baseada em objetos de negócio (OMG, 1993). Inúmeras dificuldades são encontradas quando a tarefa é obter uma forma única de trabalho que todos possam utilizar. O grande obstáculo é fazer com que grandes organizações abram mão de suas arquiteturas proprietárias, as quais servem de base para produtos comercializados, em benefício de um padrão universal. Um exemplo desta dificuldade é a ausência da empresa Microsoft do consórcio de empresas que formam a OMG. A Microsoft tem uma arquitetura que serve de base para os seus produtos. A arquitetura ou modelo COM(*Component Object Model*) promove uma integração natural entre aplicações que são executadas no ambiente *Windows* (MICROSOFT, 1995) . Um objeto pode ser escrito em qualquer linguagem, desde que esteja de acordo com o modelo COM, ele pode ser utilizado em qualquer aplicação. A tecnologia *ActiveX* é construída sobre a arquitetura COM. Um componente *ActiveX* (uma planilha, um gráfico, ou um objeto de interface) pode ser compartilhado por várias aplicações, inclusive por aplicações executadas em rede (Internet, por exemplo). A tecnologia *ActiveX* é uma das mais sólidas tecnologias para o compartilhamento de componentes entre aplicações escritas em diferentes linguagens de programação do cenário atual. Pelo fato de ser baseada no modelo COM, o grande problema com esta tecnologia, é que ela só pode ser utilizada por aplicações que executam no ambiente *Windows*, não existindo interoperabilidade com outros ambientes operacionais. Como a Microsoft já possui um padrão adotado por grande parte dos usuários do seu ambiente operacional, é muito difícil que esta empresa abra mão de seu modelo de objetos, pois

teria que reformular todos os produtos baseados neste modelo. Produtos que proporcionam a integração entre componentes construídos sobre a arquitetura COM e aplicações executadas em outros ambientes, ampliam suavemente a possibilidade de utilização desses componentes.

Um outro problema enfrentado pela OMG é a dificuldade que os desenvolvedores de objetos de negócio encontram na utilização e integração dos serviços oferecidos. Apesar dos serviços simplificarem tarefas como controle de transação e controle de segurança, não existe uma integração de tal forma que os desenvolvedores se preocupem somente com a lógica do negócio. Também não existem frameworks que facilitem a integração dos objetos de negócio com os serviços CORBA. Na tentativa de amenizar estes problemas, a OMG designou a um grupo denominado *Business Object Task Force*, o trabalho de desenvolver uma arquitetura que sirva de base para a construção de objetos de negócio (OMG, 1997a).

Enquanto a OMG tentava encontrar um modelo que fosse adequado aos desenvolvedores de objetos de negócio e que suportasse os serviços CORBA mais importantes, a Sun Microsystems (uma das consorciadas da OMG), desenvolvia o seu próprio padrão para construção de objetos-servidores. Este padrão foi denominado *Enterprise Javabeans* (SUN MICROSYSTEMS, 1997). *Enterprise Javabeans* é um modelo para construção de objetos que atuam nos chamados servidores de aplicação. Estes objetos devem ser escritos em linguagem Java. *Enterprise Javabeans* é compatível com o padrão CORBA, deste modo, os objetos de negócio escritos em Java, podem ser utilizados por aplicações escritas em outras linguagens, desde que suportem também o padrão CORBA. A grande motivação, para a utilização do padrão *Enterprise Javabeans* pelos desenvolvedores de objetos de negócio, é a facilidade de acesso aos

serviços. Esses serviços são semelhantes aos serviços CORBA. Os objetos de negócio são acoplados a servidores de aplicação. Os servidores de aplicação fornecem os serviços de forma integrada aos objetos de negócio. Os desenvolvedores de objetos de negócio não precisam se preocupar com detalhes de configuração dos serviços. Estes detalhes são de responsabilidade do administrador do servidor de aplicação.

A OMG apresenta uma tendência muito forte para a adoção do padrão *Enterprise Javabeans* como arquitetura para o desenvolvimento de objetos de negócio. A adoção dessa arquitetura pode significar um direcionamento para a construção de aplicações baseadas em objetos de negócio voltadas, principalmente, para o ambiente WEB.

3 - OBJETOS DE NEGÓCIO DISTRIBUÍDOS

Coulouris define um sistema distribuído como uma coleção de computadores autônomos, ligados por uma rede, com software projetado para oferecer um recurso integrado de computação (COULOURIS, 1994).

A computação distribuída oferece uma infra-estrutura que permite a divisão de sistemas e alocação de suas partes em diferentes computadores. Sendo assim, a computação distribuída permite que a lógica de um negócio (dados e serviços) possa ser acessível a partir de localizações remotas. Clientes, fornecedores e outros parceiros podem utilizar um sistema de negócio em qualquer lugar a qualquer hora.

A mais recente tecnologia desenvolvida na área de computação distribuída é constituída pelos chamados objetos distribuídos. Tecnologias que oferecem suporte a objetos distribuídos como Java, RMI, CORBA e DCOM permitem que, objetos ativos em uma determinado computador, possam ser utilizados por aplicações-clientes em computadores diferentes. Os objetos distribuídos evoluíram a partir de uma arquitetura em três camadas, a qual é utilizada por sistemas de monitoramento de transações como IBM CICS e BEA TUXEDO. Esses sistemas, separam uma aplicação em três camadas distintas: apresentação, lógica de negócio e banco de dados. A Figura 3.1 mostra uma arquitetura em três camadas.

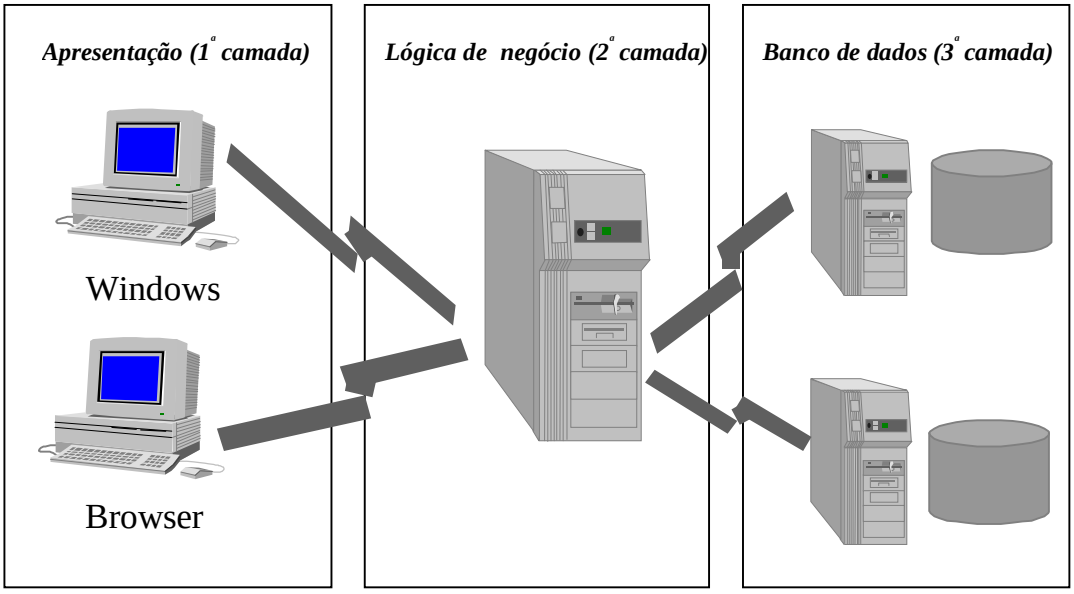


Figura 3.1 Arquitetura em três

No passado, esses sistemas eram geralmente compostos por "telas verdes" ou terminais "burros" representando a primeira camada (apresentação), aplicações escritas em COBOL ou em outra linguagem procedural, para a segunda camada (lógica de negócio) e um banco de dados como ADABAS, por exemplo, como a terceira camada (banco de dados). Com o surgimento dos objetos distribuídos, uma nova forma de arquitetura em três camadas foi adotada.

Tecnologias de objetos distribuídos tornam possível a substituição de aplicações implementadas em linguagens procedurais, como COBOL na segunda camada, por objetos de negócio. Uma arquitetura distribuída, baseada em objetos de negócio, deve ser composta por uma interface gráfica de usuário na primeira camada, objetos de negócio na camada intermediária e um banco de dados relacional ou orientado por objetos na terceira camada. Arquiteturas mais complexas são geralmente compostas por

mais camadas, com objetos residindo em diferentes servidores e interagindo de várias formas para realizar uma tarefa.

3.1 - Arquitetura de objetos de negócio distribuídos

Todos os protocolos utilizados por tecnologias de objetos de negócio distribuídos são construídos com base na mesma arquitetura. Essa arquitetura, foi projetada para facilitar a utilização de objetos remotos por aplicações. Uma aplicação acessa um objeto remoto como se estivesse acessando um objeto local (que reside no mesmo computador). Os três elementos que compõe a arquitetura de objetos de negócio distribuídos são: o objeto-servidor, o *skeleton* e o *stub*.

O objeto-servidor é o objeto de negócio que reside na camada intermediária. O termo "servidor" é utilizado para diferenciar o objeto de negócio dos demais elementos - o *stub* e o *skeleton*. O objeto-servidor é uma instância de um objeto com seu estado próprio. Cada classe de objeto-servidor tem classes *stubs* e classes *skeletons* correspondentes, construídas especificamente para aquele tipo de objeto-servidor. Deste modo, por exemplo, um objeto de negócio distribuído chamado `Cliente` deve possuir as classes `Cliente_Stub e Cliente_Skeleton` correspondendo aos elementos *stub* e *skeleton* gerados para o respectivo objeto-servidor.

O *stub* e o *skeleton,* têm a tarefa de fazer com que o objeto-servidor, que reside na camada intermediária, se pareça com um objeto local, no computador da aplicação-cliente. Isto é possível graças a utilização de um protocolo de invocação de métodos remotos. Um protocolo de invocação de métodos remotos é utilizado para estabelecer a comunicação na chamada de um método remoto na rede. CORBA, Java RMI, e o

DCOM da Microsoft são exemplos de tecnologias que possuem este protocolo. Cada instância de objeto-servidor na camada intermediária tem uma instância da sua classe *skeleton*. O *skeleton* é configurado para "escutar", em um endereço e porta IP, as chamadas feitas pelo *stub*. O *stub* reside no computador da aplicação-cliente e é conectado via rede ao *skeleton*. O *stub* age como um tradutor do objeto-servidor para a aplicação-cliente, e é responsável pela comunicação entre as chamadas da aplicação-cliente e o objeto-servidor, através do *skeleton*. O *stub* e o *skeleton* escondem, da aplicação-cliente e da implementação do objeto-servidor, respectivamente, os detalhes de comunicação específicos do protocolo de invocação de métodos remotos.

O *stub* implementa uma interface com os mesmos métodos do objeto-servidor, mas os métodos do *stub* não possuem nenhuma lógica de negócio. Os métodos do *stub* implementam as operações necessárias para enviar chamadas e receber resultados do objeto-servidor através da rede. Quando uma aplicação-cliente faz uma chamada a um método no *stub*, a chamada é comunicada pela rede. Nesta chamada, o nome do método do objeto-servidor e os valores passados como parâmetros são "empacotados". O *skeleton* recebe o "pacote" e executa o método requerido do objeto-servidor. Qualquer valor retornado pelo método chamado no objeto-servidor é "empacotado" pelo *skeleton* e enviado ao *stub*. O *stub*, então, retorna o valor para a aplicação-cliente. Para a aplicação-cliente fica a impressão de que a lógica do negócio tenha sido processada localmente. A Figura. 3.2 mostra a "conversação" entre *stub*, *skeleton* e objeto-servidor.

A maioria das implementações de tecnologias de objetos distribuídos como CORBA, DCOM, e Java RMI oferecem ferramentas para geração automática dos *stubs* e *skeletons*, a partir da definição da interface do objeto-servidor. Outros recursos

oferecidos pelas implementações, como tratamento de exceções e mecanismo de segurança, são bastante comuns.

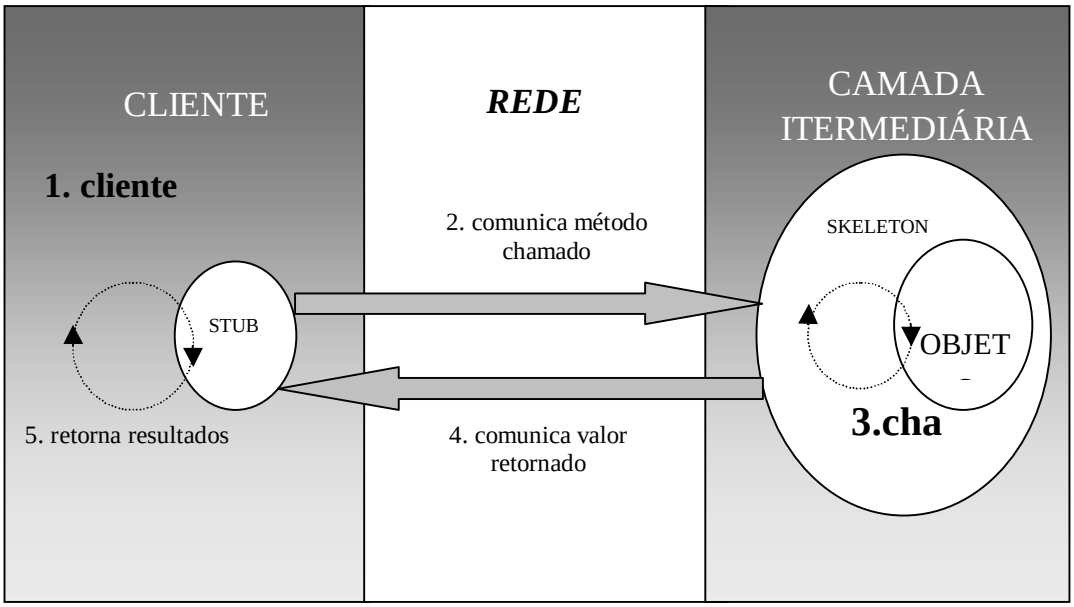


Figura 3.2 - Invocação de métodos remotos.

Apesar da maioria das implementações de tecnologias de objetos distribuídos oferecer alguns serviços como controle de transação e segurança, a utilização destes serviços não é automática. No caso dos ORBs (*Object Request Brokers*), a maior parte da responsabilidade de criação de infra-estrutura de sistema para transações e esquemas de segurança é de responsabilidade do desenvolvedor da aplicação.

3.2 - Componentes de servidor

Linguagens orientadas por objetos, como Java, C++ e Smalltalk, são geralmente utilizadas para escrever software que possuam características tais como flexibilidade, possibilidade de extensão e possibilidade de reutilização de código – algumas vantagens

oferecidas pela programação orientada por objetos. Em sistemas de negócio, linguagens orientadas por objetos são utilizadas para facilitar a criação de interfaces de usuário, simplificar o acesso a dados, e para encapsular a lógica do negócio. O encapsulamento da lógica de negócio em objetos de negócio, é um dos aspectos mais importantes do desenvolvimento sob o paradigma orientado por objetos. As regras de negócio de uma organização evoluem rapidamente. Se o software que representa o negócio é baseado em objetos de negócio, ele é flexível, pode ser estendido e reutilizado facilmente, e portanto, pode evoluir à medida que o negócio evolui.

Um modelo de componente de servidor define uma arquitetura para desenvolvimento de objetos de negócio distribuídos. Objetos de negócio distribuídos combinam a facilidade de acesso, oferecida pelos objetos distribuídos, com a facilidade de representação das regras de negócio, oferecida pelos objetos de negócio. Os modelos de componentes de servidores são utilizados na camada intermediária. Servidores de aplicação implementam estes modelos para gerenciar os componentes em tempo de execução e são responsáveis pela distribuição dos componentes para que clientes remotos possam acessá-los.

Os modelos de componentes de servidor, suportados por servidores de aplicação, oferecem uma estrutura para facilitar o desenvolvimento de objetos de negócio distribuídos, que interagindo através de um fluxo de trabalho, vão representar soluções para a implementação de sistemas de suporte ao negócio (MONSON-HAEFEL, 2000).

Componentes de servidores, como outros componentes, podem ser comercializados como peças independentes de software. Eles obedecem a um padrão definido pelo modelo de componente e podem ser ativados, sem qualquer modificação, em um servidor de aplicação que suporte aquele modelo de componente de servidores.

Os atributos de um componente podem ser modificados (desta forma, alterando o seu comportamento na execução) quando forem distribuídos, sem a necessidade de alteração de qualquer código de implementação. Dependendo do modelo de componente, o administrador do servidor de aplicação pode declarar que um componente suporta transações, níveis de segurança e até mesmo persistência, apenas definindo atributos no momento da distribuição do componente.

Ã medida que novos produtos são desenvolvidos e mudanças operacionais ocorrem, os componentes de servidores podem ser alterados, estendidos e redistribuídos para que o sistema de componentes de servidor reflita essas mudanças. Um sistema de componentes de servidor pode ser imaginado como uma coleção de componentes de servidores que representam conceitos como clientes, pedidos, produtos e estoque. Cada componente é uma peça de montagem que pode ser combinada com outro componente para oferecer uma solução de negócio. Em um sistema de negócio, os produtos podem ser armazenados em um estoque ou podem fazer parte de um pedido de cliente. Um cliente pode fazer pedidos que podem conter referência a um ou mais produtos. Os componentes de servidores representam a implementação destes objetos de negócio, encapsulando suas características como atributos de objeto. A interação lógica desses objetos também pode ser encapsulada em componentes de servidores. Os componentes podem ser agrupados, ou utilizados individualmente e terem suas definições alteradas. Um sistema baseado em componentes de servidor é flexível porque ele é baseado em objetos e é acessível porque componentes de servidor são objetos distribuídos.

3.3 - Modelos de componentes

Um modelo de componentes define um conjunto de classes, em forma de pacotes, que devem ser utilizadas para encapsular uma série de funcionalidades. Uma vez que o componente seja definido de acordo com o modelo, ele passa a ser uma peça de software que pode ser distribuída e utilizada em várias aplicações. Um componente é desenvolvido para um fim específico e não para uma aplicação específica.

Javabeans, proposto pela Sun Microsystems, pode ser considerado um exemplo de modelo de componentes (SUN, 1997). As classes e interfaces desse modelo estão no pacote `java.beans.` Esse pacote é a interface de programação para construção de componentes na linguagem Java. Botões de comando e Grades são exemplos de componentes que podem ser criados de acordo com o modelo *Javabeans*. Esses componentes podem ser utilizados na construção de interface visual para qualquer aplicação escrita em Java.

3.4 - Modelo de componentes de servidor

Um modelo de componentes de servidor é um tipo especializado de modelo de componentes. Este modelo define uma especificação para construção de componentes para servidores. Os objetos de negócio podem ser implementados como componentes de servidores. Na implementação dos objetos de negócio, é necessário seguir as convenções adotadas pelo modelo de componentes de servidor, para que o componente desenvolvido possa ser utilizado por qualquer servidor que suporte o modelo.

Enterprise Javabeans é o modelo de componentes de servidor desenvolvido pela Sun Microsystems. Utilizando uma série de classes e interfaces do pacote `javax.ejb`, os desenvolvedores podem criar e distribuir componentes compatíveis com a especificação *Enterprise Javabeans*.

O modelo de componentes *Javabeans* não é um modelo de componentes de servidor. Apesar de existir uma semelhança entre os nomes dos modelos e de ambos serem modelos desenvolvidos para aplicações escritas em Java, *Enterprise Javabeans* e *Javabeans* são dois tipos de modelos de componentes completamente diferentes. O primeiro, é um modelo de componentes de servidor, utilizado na implementação de objetos de negócio distribuídos em uma arquitetura de três camadas. O segundo, é um modelo de componentes geral, cuja maior utilização se dá na construção de controles de interfaces visuais.

3.5 - Sistemas de monitoramento de transações

Os primeiros monitores de transações apareceram no final dos anos 60 (CICS da IBM foi introduzido em 1968). Desde o início, os fabricantes de monitores de transações têm acrescentado muitos recursos e principalmente melhorado o desempenho de seus produtos, tornando-os desta forma, soluções adequadas para o desenvolvimento de aplicações distribuídas.

Alguns monitores de transações, como CICS e TUXEDO, são muito utilizados. Monitores de transação oferecem um ambiente operacional para sistemas de negócio escritos em linguagens de programação como COBOL.

Os sistemas de monitoramento de transação gerenciam automaticamente os recursos do sistema, oferecendo características como controle de transações e tolerância a falhas. A lógica do negócio, que reside em um servidor (monitor de transações), é organizada em aplicações estruturadas, que geralmente, são acessadas pela rede através de mecanismos como sistemas de envio de mensagens ou por chamadas a procedimentos remotos.

Sistemas de mensagens permitem que a aplicação-cliente envie mensagens diretamente ao servidor, solicitando que, uma aplicação servidora, seja executada de acordo com determinados parâmetros. Esse mecanismo é muito parecido com o modelo de eventos em Java. O processo de envio de mensagem pode ser síncrono ou assíncrono, isto significa que a aplicação-cliente pode precisar, ou não, esperar por uma resposta para continuar a sua execução.

Chamadas a métodos remotos constituem um mecanismo distribuído que permite que, aplicações-clientes, invoquem procedimentos em aplicações residentes em um servidor. Esse mecanismo deixa a impressão, para a aplicação-cliente, que o procedimento remoto está sendo executado localmente.

A principal diferença entre chamada a procedimentos remotos e chamada a métodos remotos é que, a primeira, é utilizada por aplicações estruturadas e é baseada em procedimentos, e a segunda, é utilizada por aplicações distribuídas baseadas em objetos. Na concepção do mecanismo de chamada a procedimentos remotos, não se faz referência ao conceito de objeto, desta forma, um procedimento ou função, em vez de trabalhar com atributos privados de um objeto, trabalha com parâmetros, que podem ser informações de várias entidades.

Pelo fato de estarem sendo utilizados por vários anos, os sistemas monitoramento de transações podem ser considerados uma tecnologia sólida. Hoje em dia, eles são muito utilizados em sistemas críticos. O grande problema é que estes sistemas não são baseados em objetos. O acesso a esses sistemas, através de chamadas a procedimentos remotos, é feito sem nenhum conceito de identidade, da mesma forma como é feito em chamadas a métodos estáticos de classes, não existindo objetos. Deste modo, não é possível estruturar a lógica de negócio de sistema em objetos, não existindo encapsulamento, diminuindo a flexibilidade e possibilidade de reutilização desta lógica.

3.5.1 - *Object request brokers*

Sistemas distribuídos, baseados em objetos, permitem a utilização dos recursos oferecidos pela programação orientada por objetos como encapsulamento e identidade de objetos. Nesses sistemas, os objetos são unidades lógicas, onde cada unidade possui o seu próprio estado. Como cada objeto tem sua identidade esta utilizada quando um objeto, que faz parte de um sistema distribuído, deve ser acessado.

Tecnologias de objetos distribuídos, como CORBA e Java RMI, são evoluções do mecanismo de chamadas a procedimentos remotos. Os objetos de negócio, depois de implementados, são distribuídos (instalados) em um barramento chamado ORB, o qual facilita o acesso a esses objetos por aplicações-clientes.

ORBs, entretanto, não definem um ambiente operacional para os objetos distribuídos. Eles, simplesmente, oferecem recursos de comunicação utilizados para acessar e interagir com os objetos remotos. No desenvolvimento de aplicações baseadas em objetos distribuídos, toda responsabilidade pelo controle de transações,

concorrência, gerenciamento de recursos do sistema e tolerância a falhas é do desenvolvedor. Muitas vezes, esses serviços estão disponíveis e implementados, mas, o desenvolvedor, tem que integrá-los aos objetos de negócio.

3.5.2 – Sistemas de monitoramento de transações baseados em componentes

Os sistemas de monitoramento de transações baseados em componentes, ou monitores de transações baseados em componentes, é uma espécie de tecnologia híbrida, que oferece recursos necessários para a implementação de sistemas baseados em objetos distribuídos, com suporte a transações. A indústria de monitores de transações baseados em componentes surgiu quando os fabricantes de monitores de transações e de ORBs se preocuparam em oferecer uma solução integrada. Sendo assim, os servidores de monitoramento de transações baseados em componentes oferecem um infra-estrutura de ORB e um controle de transações que torna possível manter a integridade do estado de um objeto de negócio em um ambiente distribuído.

As principais atividades de um servidor de monitoramento de transações baseado em componentes são identificadas por Orfali e Harkey (ORFALI & HAKEY, 1998) :

- **Ativar e desativar os componentes**. Os servidores ativam e desativam os componentes instalados quando são iniciados e finalizados, respectivamente.
- **Coordenar transações distribuídas**. Os servidores permitem que transações sejam iniciadas declarativamente por instruções existentes nos objetos de negócio. O controle de transações pode ser feito de forma automática pelos servidores quando estes são configurados. Se um fluxo de trabalho envolver mais de uma transação, o servidor coordena a ordem de execução, permitindo a

confirmação ou cancelamento de conjuntos de instruções de cada transação individualmente.

- **Notificar um componente quando acontecerem eventos durante o seu ciclo de vida**. Os servidores notificam os componentes quando eles são criados, ativados, desativados, e destruídos. Um componente, quando notificado, pode realizar atividades como gerenciamento do seu estado e alocação de recursos de sistema.
- **Gerenciar automaticamente o estado de componentes persistentes**. Os servidores mais sofisticados, podem carregar o estado de um componente, automaticamente, a partir de um banco de dados e salvar este estado dentro de limites transacionais.

Como as vantagens da utilização de objetos distribuídos se tornaram muito visíveis, o número de sistemas desenvolvidos utilizando ORBs cresceu rapidamente. Utilizando um modelo de componentes bastante primitivo, os ORBs permitem que os objetos de negócio sejam acoplados ao sistema de comunicação, mas não oferecem acesso direto aos serviços necessários para que um sistema de negócio possa operar satisfatoriamente. O acesso aos serviços, pelos objetos, é feito através de uma interface de programação, fazendo com que, o desenvolvedor, tenha que se preocupar não somente com a lógica de negócio mas, também, com a integração com os serviços.

Além da falta de integração entre os objetos de negócio e os serviços, os ORBs não possuem estratégias de melhoramento de performance e de otimização dos sistemas. Se o desenvolvedor deseja que sua aplicação tenha performance satisfatória e utilize recursos de forma racionada, ele deve implementar melhoramentos de desempenho e

otimização em sua aplicação, o que requer um conhecimento aprimorado sobre computação distribuída. A tarefa de gerenciar recursos e de otimização é muito complicada em aplicações cuja infra-estrutura se resume somente ao ORB.

Os monitores de transações sempre realizaram muito bem tarefas ligadas à administração de sistemas distribuídos. Com três décadas de experiência, não demorou muito para que empresas como IBM e BEA iniciassem o desenvolvimento de sistemas híbridos que combinassem as melhores características dos ORBs e dos sistemas de monitoramento de transações. Esses novos sistemas foram denominados sistemas de monitoramento de transações baseados em componentes. Esse tipo de sistema combina os recursos da programação orientada por objetos, oferecidos pelos objetos distribuídos, com o ambiente operacional, disponibilizado pelos monitores de transação. Eles representam ambientes ideais para componentes de servidor, oferecendo gerenciamento de concorrência, transações, distribuição de objetos, segurança e gerenciamento dos recursos do sistema, de forma automática. Os desenvolvedores de objetos de negócio ainda precisam conhecer esses recursos para utilizá-los, mas não precisam implementá-los, se preocupando, apenas, com a lógica do negócio.

Muitos fabricantes desenvolveram produtos para o monitoramento de transações orientados por objetos. Esses produtos são oferecidos por fabricantes de banco de dados, servidores de aplicação, servidores web, CORBA ORBs e monitores de transações. Cada produto reflete a área de atuação do fabricante. O mais importante, no momento da escolha de um produto, é identificar o modelo de componentes adotado, pois esta decisão, vai afetar o desenvolvimento dos objetos de negócio, uma vez que os objetos devem ser desenvolvidos de acordo com o modelo de componente suportado pelo servidor.

3.6 - Sistemas de monitoramento de transações baseados em componentes e modelo de componentes de servidor

Para os objetos de negócio utilizarem a infra-estrutura oferecida pelos sistemas de monitoramento de transações baseados em componentes, eles devem ser desenvolvidos de acordo com as especificações do modelo de componentes implementado pelo servidor.

Um bom modelo de componentes de servidor é um item fundamental para o sucesso de um projeto. O modelo de componentes de servidor é um contrato que define as responsabilidades do sistema de monitoramento de transações e dos objetos de negócio.

Ao empregar um bom modelo de componentes, o desenvolvedor tem como saber a maneira como os recursos de sistema serão gerenciados e oferecidos, e o servidor pode, então, gerenciar os objetos de negócios da melhor forma possível. Sendo assim, os modelos de componentes de servidor, estabelecem as responsabilidades do desenvolvedor da aplicação e do fabricante do servidor.

3.6.1 - Sistemas de monitoramento de transações baseados em componentes existentes no mercado

Os produtos que fazem o monitoramento de transações baseados em componentes existentes no mercado se dividem em duas categorias: os que são baseados no modelo DCOM (proposto pela Microsoft), que tem como único representante o *Microsoft*

Transaction Server (MTS), e os que são baseados no modelos CORBA e *Enterprise Javabeans*.

3.6.1.1 - *Microsoft Transaction Server* (MTS)

A Microsoft foi uma das primeiras empresas a lançar um sistema de monitoramento de transações baseado em componentes, o *Microsoft Transaction Server* (MTS). O MTS utiliza um modelo de componentes de servidor e um serviço de componentes distribuído baseado no modelo DCOM (*Distributed Common Object Model*). Como foi lançado em 1996, foi um dos primeiros servidores a oferecer um ambiente para desenvolvimento de objetos de negócio. Com o MTS, os desenvolvedores de aplicação podem escrever componentes compatíveis com o modelo COM (*Common Object Model*), sem se preocupar com detalhes de infra-estrutura de sistema distribuído.

Se o objeto de negócio for projetado de acordo com o modelo COM, o MTS faz, automaticamente, o gerenciamento de transações, concorrência e o gerenciamento de recursos. Com esta proposta, utilizar o MTS como servidor para os objetos de negócio, facilita muito o trabalho realizado pelo desenvolvedor de aplicação. Se existe uma necessidade de se configurar a forma como o objeto de negócio interage com os serviços de sistema, os desenvolvedores devem utilizar editores de configuração, um recurso que já vem embutido no servidor. A definição dos atributos que influenciam o comportamento do objeto de negócio em tempo de execução, é feita, de maneira similar, à configuração de propriedades de um componente visual em um ambiente de programação como o *Microsoft Visual Basic*. Basta selecionar a propriedade a ser

configurada, e escolher o tipo de controle de transação, ou atributo de segurança, ou nome de identificação, ou outras propriedades. Os valores para estas propriedades são normalmente exibidos em componentes visuais como *list box*, por exemplo, evitando assim, que o desenvolvedor tenha que fornecer os valores por meio de digitação (uma fonte natural de erros).

Depois que os objetos de negócio são instalados no MTS, as suas interfaces remotas podem ser utilizadas na montagem de aplicações, um processo muito parecido com a construção de um prédio, onde cada componente, previamente construído, tem uma função específica no projeto.

Apesar de possuir recursos consideráveis, e de ter uma interface de programação fácil de ser utilizada pelo desenvolvedor de aplicação, o MTS não pode ser encarado como um padrão aberto. O MTS é um sistema de monitoramento de transações baseado em componentes cuja tecnologia é de propriedade da Microsoft, o que significa que ele está preso à plataforma oferecida pela Microsoft. Caso um projeto utilize apenas produtos Microsoft ou produtos fabricados por outras empresas, mas compatíveis com o ambiente *Microsoft Windows*, a utilização do MTS como servidor de aplicação deve ser considerada. Um outro fator importante são as aplicações-clientes. Caso o MTS seja a plataforma escolhida, as aplicações-clientes devem ser executadas no ambiente *Windows* ou outro ambiente que ofereça integração com DCOM.

Como, em um ambiente distribuído, várias plataformas são utilizadas, existe a necessidade de se utilizar um modelo de componentes aberto. O MTS não se encaixa nesta classificação. Além disso, o MTS oferece suporte somente a componentes “sem estado”, ou seja, ele não suporta objetos persistentes. Apesar dos objetos sem estado oferecerem um bom desempenho, para implementar objetos de negócio é necessário

uma flexibilidade que é encontrada, somente, em sistemas de monitoramento de transações que suportam componentes com e “sem estado”.

3.6.1.2 - Sistemas de monitoramento de transações que utilizam os padrões CORBA e *Enterprise Javabeans*

Os primeiros sistemas de monitoramento de transações baseados em componentes que utilizaram uma arquitetura baseada no padrão CORBA começaram a aparecer em meados do ano de 1997. Naquela época, quando o MTS já era um produto consolidado no mercado, produtos promissores como *Weblogic* da BEA e o *Gemstone/J* da Gemstone eram apenas projetos. Apesar de serem projetos, todos eles tinham uma característica comum: eram baseados em um padrão aberto, o CORBA.

Pelo fato de utilizarem, como serviço de objetos distribuídos, o padrão CORBA, os objetos de negócio implementados nos servidores “não Microsoft” podiam ser instalados em qualquer plataforma e podiam suportar aplicações-clientes que não necessitassem do ambiente *Microsoft Windows*. O padrão CORBA é independente de linguagem e plataforma, deste modo, os fabricantes de servidores passaram a oferecer mais opções a seus clientes. O problema com os servidores de aplicação baseados no padrão CORBA, era que, todos eles, apresentavam diferentes modelos de componentes de servidor. Isto significa que, se um componente é desenvolvido para um servidor específico, ele não pode ser utilizado em outro servidor de aplicação. Isto porque os modelos de componentes de servidor são diferentes.

Apesar de oferecerem independência de plataforma e acesso para qualquer aplicação-cliente, os sistemas de monitoramento de transação baseados no padrão

CORBA não ofereciam independência a nível de componente, ou seja, o desenvolvedor de componentes tinha que criar um componente para um servidor específico. Desta forma, assim como o MTS não oferecia uma solução aberta, os sistemas de monitoramento de transações baseados puramente no padrão CORBA também não ofereciam uma plataforma totalmente flexível.

O grande problema era a fragmentação do mercado, onde cada fabricante implementava a forma de interação entre os objetos de negócio e o servidor, utilizando sua tecnologia proprietária. Como o mercado era fragmentado, nenhum fabricante podia obter vantagens sobre outro. A solução seria a adoção de um padrão que fosse utilizado para que, objetos de negócio, fossem implementados como componentes universais, capazes de serem instalados em qualquer servidor, e que pudessem ser utilizados por aplicações-clientes que estivessem sendo executadas em qualquer plataforma.

A idéia inicial seria, obviamente, desenvolver um modelo de componentes de servidor baseado no padrão CORBA, que fosse utilizado por todos os fabricantes de sistemas de monitoramento de transações baseados em componentes. A OMG iniciou, então, um projeto que definiria um modelo de componentes de servidor. Como o grupo envolvido com esta atividade estava demorando muito para apresentar um modelo concreto, a Sun Microsystems, uma das empresas consorciadas a OMG, anunciou no ano de 1997, um modelo de componentes de servidor chamado *Enterprise Javabeans*.

O modelo proposto pela Sun Microsystems, oferecia algumas vantagens importantes, que até então, não tinham sido oferecidas por nenhum outro modelo de componente de servidor. Em primeiro lugar, a Sun era uma empresa que tinha uma certa reputação, e era conhecida pelos trabalhos realizados em conjunto com outros fabricantes, no sentido de obter as melhores soluções possíveis. Além disso, a Sun tinha

o costume de adotar as melhores idéias do mercado no desenvolvimento de tecnologias de suporte à linguagem Java. Um exemplo bastante conhecido é o JDBC (*Java Database Connectivity*) baseado no padrão ODBC, proposto pela Microsoft. O JDBC ofereceu aos fabricantes de banco de dados um modelo flexível, para que estes pudessem desenvolver seus drivers, e às aplicações-clientes, uma forma padronizada de acesso a dados, independente do banco de dados utilizado.

Apesar do CORBA ter sido oferecido como um padrão aberto, a tentativa de padronizar serviços internos do servidor, como serviço de segurança e controle de transações, teria um custo enorme para empresas que já possuíam produtos consolidados. Para produtos consolidados como TUXEDO e CICS, por exemplo, não seria vantajoso rescrever os serviços para se adequarem ao padrão CORBA. O suporte, ao modelo *Enterprise Javabeans*, não implicava em modificações em serviços de baixo nível do servidor. A idéia proposta era que, os serviços, pudessem ser utilizados pelos componentes da forma estabelecida pela especificação, não importando a forma de implementação desses serviços. Isto representou uma solução muito mais viável para produtos já estabelecidos e para novos produtos. Além disso, o modelo era baseado na linguagem Java, uma linguagem independente de plataforma e que fornecia acesso direto a tecnologias emergentes como Internet.

Depois de anunciar o modelo *Enterprise Javabeans*, a Sun Microsystems iniciou um trabalho, em conjunto com vários fabricantes, para definir um padrão aberto e flexível, que pudesse ser adaptado a produtos existentes no mercado. Este trabalho envolveu fabricantes de servidores web, servidores de banco de dados, servidores de aplicação e servidores de aplicação baseado em componentes. Então, em dezembro de

1997, a Sun Microsystems publicou a primeira versão da especificação *Enterprise Javabeans*.

3.6.2 - Vantagens oferecidas por um modelo de componentes de servidor padronizado

A principal vantagem de se utilizar um modelo de componentes de servidor padronizado é que, o objeto, ao ser implementado de acordo com o modelo, pode ser utilizado por qualquer servidor baseado em componentes compatível com aquele modelo. *Enterprise Javabeans* é o modelo de componentes de servidor que tem mais chances de se tornar um padrão de mercado. Sendo assim, implementar objetos de negócio de acordo com o padrão E*nterprise Javabeans* pode ser uma garantia de que o objeto possa ser utilizado por qualquer servidor por muitos anos.

Outra vantagem é a possibilidade de utilização de componentes de negócio terceirizados. Os componentes de negócio podem ser utilizados da mesma forma que componentes visuais (componentes *ActiveX*, por exemplo) no desenvolvimento de aplicações. Isto permite a criação de bibliotecas de componentes de negócio especializadas (*frameworks)*, o que simplificaria em muito o desenvolvimento de aplicações.

Oferecendo flexibilidade aos fabricantes de servidores, e portabilidade aos desenvolvedores de componentes, o modelo de componentes de servidor *Enterprise Javabeans* é uma alternativa interessante para implementação de objetos de negócio.

4 - ENTERPRISE JAVABEANS

Enterprise Javabeans é um modelo de componentes desenvolvido para sistemas de monitoramento de transações (servidores de aplicação baseados em componentes). Este modelo foi proposto pela Sun Microsystems, no final do ano de 1997. Em 1999, a Sun Microsystems publicou a segunda versão do modelo *Enterprise Javabeans* (SUN MICROSYSTEMS, 1999).

As similaridades que existem entre os diferentes servidores de aplicação baseados em componentes, permitem que, a abstração definida pela arquitetura *Enterprise Javabeans*, seja um modelo de componentes padrão entre eles. Um fabricante de servidor de aplicação, implementa seu produto de forma diferente de outro fabricante, mas, todos os produtos, oferecem os mesmos serviços e técnicas de gerenciamento de recursos semelhantes.

Todos os servidores de aplicação, compatíveis com o modelo *Enterprise Javabeans*, devem oferecer a mesma estrutura, para que, desenvolvedores de objetos de negócio, criem seus objetos obedecendo apenas a especificação proposta por este modelo. Desta forma, a especificação do modelo *Enterprise Javabeans*, serve de

contrato, para que, as responsabilidades de fabricantes de servidores de aplicação e desenvolvedores de objetos de negócio, sejam bem definidas.

4.1 - Infra-estrutura do servidor *Enterprise Javabeans*

A infra-estrutura necessária, para implementação de componentes *Enterprise Javabeans*, é formada por dois elementos: o *container* e o servidor *Enterprise Javabeans*.

4.1.1 - *Container Enterprise Javabeans*

O ambiente que envolve os componentes (*beans*) no servidor *Enterprise Javabeans* é chamado de *container*. O termo *container* é mais um conceito do que um recurso de programação. Ele atua como um elemento intermediário, situado entre a classe *bean* e o servidor *Enterprise Javabean*s. O *container* gerencia os componentes *Enterprise Javabean*s e oferece, para as instâncias destes componentes, os serviços primários como controle de transações, segurança, concorrência e serviço de nomes, assim como o gerenciamento do ciclo de vida do componente.

Um servidor *Enterprise Javabean*s pode ter vários *containers*, cada um deles responsável pelo gerenciamento de vários componentes. Na verdade, os termos *container* e servidor são, muitas vezes, utilizados para representar o mesmo elemento da arquitetura *Enterprise Javabeans*. A especificação *Enterprise Javabeans*, define o modelo de componentes em termos de responsabilidades do *container*.

4.1.2 - Servidor *Enterprise Javabeans*

A forma como é feita a integração do *container* com o servidor não é definida pela especificação *Enterprise Javabeans*. Isto foi feito para permitir que, os fabricantes, utilizassem suas tecnologias na implementação dos servidores. A grande responsabilidade do *container* é isolar os componentes do servidor. A especificação *Enterprise Javabeans* define, apenas, as responsabilidades do *container* e a forma como deve ser feita a integração dos componentes com *container*. É muito difícil determinar, por exemplo, exatamente onde o *container* termina e onde o servidor começa em um produto.

Nesta primeira geração de servidores *Enterprise Javabeans*, esta ambigüidade não é um problema porque, a maioria dos fabricantes de servidores *Enterprise Javabeans*, fornece, também, *containers Enterprise Javabeans*. Os produtos são oferecidos de forma integrada, isto é, um produto é composto por servidor e *container*, desta forma, a interface entre os dois é proprietária. É possível que, no futuro, algum trabalho seja realizado no sentido de definir uma interface comum entre todos os *containers* e servidores *Enterprise Javabeans*. Isto definiria as responsabilidades de cada peça da infra-estrutura de servidor, e não permitiria soluções proprietárias por parte dos fabricantes.

4.2 - Os papéis no desenvolvimento de aplicações baseadas em componentes *Enterprise Javabeans*

A abordagem adotada, para simplificar o desenvolvimento de aplicações construídas a partir de componentes *Enterprise Javabeans*, é baseada na divisão do processo de desenvolvimento em tarefas diferentes e, na atribuição de diferentes papéis aos envolvidos no processo (VOGEL & RANGARAO, 1999). A especificação *Enterprise Javabeans* identifica seis papéis, os quais se referem às tarefas de disponibilização da infra-estrutura para utilização de componentes, desenvolvimento da aplicação, sua distribuição, e procedimentos operacionais. Como algumas dessas tarefas são realizadas pelos fabricantes de servidores *Enterprise Javabeans*, na construção de seus produtos, a tarefa do desenvolvedor da aplicação fica mais simples. A Figura 4.1 mostra a arquitetura *Enterprise Javabeans* e os papéis no desenvolvimento de aplicações baseadas nesta arquitetura.

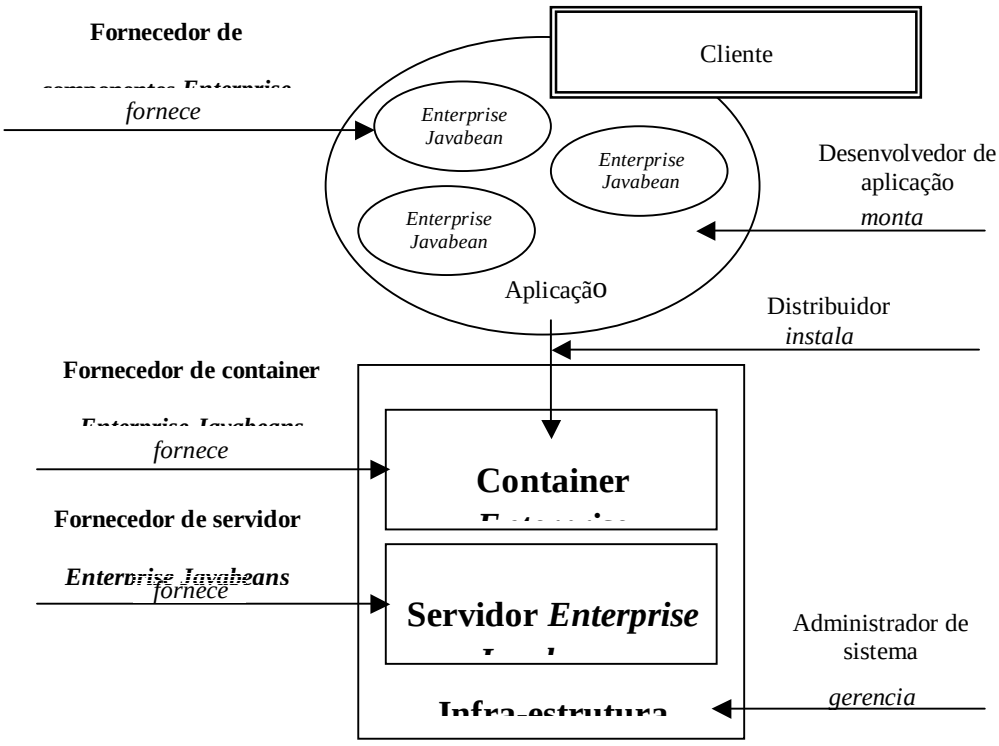


Figura 4.1 - Arquitetura *Enterprise Javabeans*.

4.2.1 - Infra-estrutura

A infra-estrutura para implementação dos componentes *Enterprise Javabeans* é oferecida pelos fabricantes de servidores e containers *Enterprise Javabeans.*

4.2.1.1 - Fornecedor de servidores *Enterprise Javabeans*

Um fornecedor de servidores *Enterprise Javabeans* é, tipicamente, um fabricante de servidores de aplicação ou de produtos de outras áreas, como banco de dados ou

ORBs, por exemplo, com experiência em infra-estrutura para sistemas distribuídos e serviços. Ele implementa uma plataforma, a qual facilita o desenvolvimento de aplicações distribuídas e fornece um ambiente para execução das mesmas.

4.2.1.2 - Fornecedor de *containers Enterprise Javabeans*

Um fornecedor de containers *Enterprise Javabeans* tem experiência em sistemas distribuídos, controle de transações e segurança. Um *container* é o ambiente onde ficam os componentes *Enterprise Javabeans*. Ele faz a ligação desses componentes com o servidor *Enterprise Javabeans*.

4.2.2 - Aplicação

Os fornecedores de componentes e os desenvolvedores de aplicações são os envolvidos no processo de construção de aplicações baseadas no modelo *Enterprise Javabeans*.

4.2.2.1 - Fornecedor de componentes *Enterprise Javabeans*

Um fornecedor de componentes *Enterprise Javabeans* é um especialista em um determinado negócio, como sistema financeiro ou indústria de telecomunicações, por exemplo. O fornecedor de componentes, implementa a lógica de negócio em componentes sem se preocupar com distribuição, transação, segurança, e outras tarefas não ligadas ao negócio.

4.2.2.2 - Desenvolvedor de aplicações

Desenvolve a aplicação a partir de componentes prontos (componentes *Enterprise Javabeans*) adicionando outros componentes, como componentes de interface visual, por exemplo. Quando os desenvolvedores estão montando a aplicação, eles não se preocupam com a implementação dos componentes.

4.2.3 - Distribuição e operação

Os distribuidores de aplicação e os administradores de sistemas são responsáveis pela instalação da aplicação e operação do servidor *Enterprise Javabeans*, respectivamente.

4.2.3.1 - Distribuidor

Um distribuidor é especializado na operação das aplicações. O distribuidor adapta uma aplicação, composta por um número de componentes, em um ambiente operacional, modificando as propriedades dos componentes *Enterprise Javabeans*. A tarefa do distribuidor inclui, por exemplo, configuração de controle de transação, políticas de segurança, e a integração com o container.

4.2.3.2 - Administrador de sistema

Um administrador de sistema se preocupa com as aplicações distribuídas (instaladas). O administrador faz o monitoramento das aplicações, que estão sendo executadas, e toma medidas apropriadas quando acontece um evento anormal na execução da aplicação. Geralmente, um administrador utiliza ferramentas oferecidas pelo servidor para modificar propriedades dos componentes que compõe a aplicação, quando necessário.

4.3 - O componente *Enterprise Javabean*

Existem dois tipos de componentes de servidores na arquitetura *Enterprise Javabeans*: *entity beans* (componente do tipo entidade) e *session beans* (componente do tipo sessão).

De um modo geral, componentes do tipo-entidade representam conceitos de negócio que podem ser expressos por nomes. Por exemplo, um componente do tipo-entidade pode representar um cliente, um equipamento, ou um item de estoque. Componentes do tipo-entidade representam objetos do mundo real. Esses objetos, são geralmente persistentes (seu estado deve ser gravado em um banco de dados).

Componentes do tipo-sessão são uma espécie de extensão da aplicação-cliente e são responsáveis pelo gerenciamento dos processos ou tarefas. Um componente Produto oferece métodos para alterar o seu estado, mas, utilizando somente esses métodos, não é possível realizar uma tarefa ou processo. Compra de Produtos é um processo de negócio que pode ser modelado como um componente do tipo sessão. Ele envolve não somente

o componente Produto, mas outros como Nota Fiscal de Compra, Fornecedor, Transportadora, Forma de Pagamento, Fatura e Estoque. Um componente do tipo-sessão é responsável pela coordenação que deve existir entre os todos componentes do tipo-entidade envolvidos em um determinado processo.

A atividade que um componente do tipo-sessão representa é, fundamentalmente, transiente (um componente do tipo-sessão não tem nenhuma representação no banco de dados). Apesar de não ser mapeado diretamente para o banco de dados, a sua utilização, modifica as informações armazenadas no banco. Por exemplo, no processo de compra de produtos, as faturas são registradas e o estoque é modificado. Para representar este processo, um componente do tipo-sessão é criado – `CompraProdutoBean`. Este componente é responsável pela criação dos componentes do tipo-entidade envolvidos no processo, como `ProdutoBean`, `FaturaBean` e `EstoqueBean`, por exemplo. Estes componentes têm representação direta no banco de dados. O componente bean do tipo-sessão cria quantas instâncias forem necessárias de cada componente do tipo-entidade. A cada instância criada de um componente do tipo-entidade, um registro na tabela correspondente é inserido.

Em resumo, o que diferencia os dois tipos de componente é a presença, ou não, do serviço de persistência. Componentes do tipo-entidade são persistentes e componentes do tipo- sessão não são persistentes.

4.4 - Componentes *Enterprise Javabeans* e objetos de negócio

O modelo *Enterprise Javabeans* é adequado para o desenvolvimento de objetos de negócio, pois além de estabelecer um padrão que é suportado pela maioria dos

servidores de aplicação baseados em componentes, proporciona o mapeamento direto dos objetos de negócio do tipo-entidade e objetos de negócio do tipo-processo para componentes do tipo-entidade e componentes do tipo-sessão, respectivamente. Como o mapeamento é direto, o trabalho de engenharia de negócio pode ser inteiramente baseado em objetos de negócio. Os modelos de análise, como o modelo conceitual de classes, e os diagramas de interação, podem ser desenvolvidos sem nenhuma preocupação com futuras implementações. Quando os objetos de negócio forem implementados, os objetos identificados no modelo de classes, serão representados ou por componentes do tipo entidade ou componentes do tipo-sessão. Grande parte das classes do diagrama de classes é implementada como componentes do tipo-entidade, porque seus objetos devem ser persistentes. Os objetos de negócio que representam um processo, devem ser implementados como componentes do tipo-sessão.

4.5 - Classes e *interfaces*

Para implementar um componente *Enterprise Javabeans*, é necessário definir duas *interfaces* – *interface remote* e *interface home* e uma classe – classe do componente *bean*. Se o componente for do tipo-entidade, então será necessário definir uma segunda classe – classe *primary key*.

4.5.1 - *Interface remote*

A *interface remote* define os métodos de negócio do componente *bean*. A *interface remote* é derivada de `javax.ejb.EJBObject`, que por sua vez é derivada

de `java.rmi.Remote.` A Listagem 4.1 mostra um exemplo de definição de uma *interface remote* para o objeto de negócio Produto.

```
import java.rmi.RemoteException;

public interface Produto extends javax.ejb.EJBObject
{
      public String getNome() throws RemoteException;

      public void setNome(String n)
            throws RemoteException;

      public double getPreco() throws RemoteException;

      public void setPreco(double p)
           throws RemoteException;
 }
```

Listagem 4.1 : *Interface remote.*

Todos os métodos definidos nesta interface devem possuir, na sua assinatura, a cláusula `throws` indicando a possibilidade de geração da exceção `RemoteException`. Esta cláusula indica que, a chamada ao método, pode gerar uma exceção remota. Este procedimento, é necessário, quando se utiliza chamada a métodos remotos. *Enterprise Javabeans* utiliza as convenções estabelecidas pelo protocolo *Java*

RMI-IIOP, apesar de ser possível, uma aplicação utilizar outro protocolo de comunicação como *CORBA-IIOP* ou *Java Remote Method Protocol (JRMP)*.

4.5.2 - *Interface home*

A *interface home* define os métodos do ciclo de vida do componente *bean*. Isto significa que esta interface define métodos para criação, remoção e busca do componente *bean*. A *interface home* é derivada da *interface* `javax.ejb.EJBHome`, que por sua vez é derivada da *interface* `java.rmi.Remote.` A Listagem 4.2 mostra a definição da interface home para o objeto de negócio Produto.

```
import java.rmi.RemoteException;
import javax.ejb.CreateException;
import javax.ejb.FinderException;

public interface ProdutoHome extends javax.ejb.EJBHome
{
        public Produto create(int cod)
           throws CreateException, RemoteException;

        public Produto findByPrimaryKey(ProdutoPK pk) throws
           FinderException, RemoteException;
}
```

Listagem 4.2: *Interface home*.

O métodos `create()` são responsáveis pela inicialização das instâncias de componentes *bean*. Como *Enterprise Javabeans* permite a sobrecarga de métodos, uma *interface home* pode possuir mais de um método `create()`.

O método `findByPrimaryKey()` é um método de busca, que retorna a instância de componente bean do tipo-entidade que corresponde à classe *primary key*, passada como argumento. Além deste método, podem existir outros métodos de busca definidos na *interface home*. Métodos de busca são definidos apenas para componentes bean do tipo-entidade e não são utilizados em componentes bean do tipo-sessão.

4.5.3 - A Classe do componente *bean*

A classe do componente *bean* implementa os métodos do objeto de negócio. Essa classe, não implementa as interfaces *remote* e *home*, entretanto, deve possuir todos os métodos definidos na *interface remote*, além de alguns dos métodos definidos na *interface home*. Um componente *bean* do tipo-entidade, deve implementar a *interface* `javax.ejb.EntityBean.` Um componente *bean* do tipo-sessão, deve implementar a *interface* `javax.ejb.SessionBean.` Tanto a *interface* `EntityBean` como a *interface* `SessionBean`, são derivadas da *interface* `javax.ejb.EnterpriseBean.` A Listagem 4.3 mostra a definição de uma classe de componente bean para o objeto de negócio Produto.

```
import javax.ejb.EntityContext;

public class ProdutoBean implements javax.ejb.EntityBean
```

```
{
        public int codigo;
        public String nome;
        public double preco;

        public ProdutoPK ejbCreate(int cod)
        {
            codigo = cod;
            return null;
        }
        public void ejbPostCreate(int cod){}

        public String getNome()
        {
            return nome;
        }
        public void setNome(String n)
        {
            nome = n;
        }

        public double getPreco()
        {
            return preco;
        }

        public void setPreco(double p)
```

```
        {
            preco = p;
        }

        public void setEntityContext(EntityContext ctx) {}

        public void unsetEntityContext() {}

        public void ejbActivate() {}

        public void ejbPassivate() {}

        public void ejbLoad() {}

        public void ejbStore() {}

        public void ejbRemove() {}
    }
```

Listagem 4.3: Classe do componente *bean*.

Os métodos de negócio são os únicos métodos visíveis para uma aplicação-cliente que utiliza componentes bean. Outros métodos, exigidos pelo modelo de componentes *Enterprise Javabeans*, são visíveis somente para o *container* e não fazem parte da definição de negócio do componente *bean*.

Os métodos `ejbCreate()` e `ejbPostCreate()`, no caso do componente *bean* ser do tipo-entidade, inicializam a instância de componente *bean*, quando um registro é inserido na tabela de banco de dados que representa o componente. Caso o componente seja do tipo-sessão, não há necessidade de se definir o método `ejbPostCreate()` na classe do componente *bean*, pois este método não existe na *interface* `SessionBean.` O método `ejbRemove()` notifica uma instância de componente *bean* do tipo-entidade quando um registro está para ser excluído da tabela de banco de dados. O mesmo método, quando utilizado em um componente *bean* do tipo-sessão, notifica o componente *bean* que a aplicação-cliente não precisa mais dele. Entretanto, o método `ejbRemove()` não informa ao componente *bean* do tipo-sessão, que um registro está para ser excluído do banco de dados, porque um componente do tipo-sessão não é mapeado diretamente para o banco de dados. Os métodos `ebjLoad()` e `ejbStore()` notificam a instância de componente *bean* do tipo-entidade que seu estado está sendo lido ou escrito no banco de dados. Estes métodos não são utilizados em componentes bean do tipo-sessão. Os métodos `ejbActivate()` e `ejbPassivate()` notificam uma instância de componente *bean* que ela está para ser ativada ou desativada - um processo que gerencia memória e outros recursos. Os métodos `setEntityContext()`, para componentes *bean* do tipo- entidade, e `setSessionContext()`, para componentes *bean* do tipo-sessão, fornecem uma interface para o servidor *Enterprise Javabeans* que permite que os componentes *bean* obtenham informações a respeito de si próprios e do ambiente onde estão atuando. O método `unsetEntityContext()` é chamado pelo servidor *Enterprise Javabeans*, para notificar a instância de componente *bean* do tipo-entidade,

que ela está para ser destruída pelo mecanismo de *garbage collection*. Não existe um método correspondente para componentes *bean* do tipo-sessão.

Os métodos `ejbPostCreate()`, `ejbRemove()`, `ejbLoad()`, `ejbStore()`, `ejbActivate()`, `ejbPassivate()`, `setEntityContext()` e `setSesssionContext()`, e `unsetEntityContext()` são métodos *callback.* Os métodos *callback* notificam a instância de componente *bean*, quando uma ação está para ser realizada, ou quando uma ação acabou de ser realizada, no ambiente do servidor *Enterprise Javabeans*. Estas notificações, simplesmente, informam o componente *bean* de um evento. O componente *bean* não precisa fazer nada quando o evento acontece. Um evento pode acontecer quando a instância de componente *bean* está para ser carregada, removida ou desativada, por exemplo. Se um componente *bean* deve fazer alguma coisa quando um evento acontecer, a sua classe deve implementar o método correspondente ao evento. Por exemplo, se o componente *bean* deve fazer alguma coisa quando uma de suas instâncias está para ser ativada, então ele deve implementar o método `ejbActivate().`

4.5.4 - A classe *primary-key*

A classe *primary-key* é uma classe bastante simples que fornece um ponteiro para o banco de dados. Somente componentes *bean* do tipo-entidade precisam de uma classe *primary-key*. A classe deve implementar a interface `java.io.Serializable.` A Listagem 4.4 mostra a definição de uma classe *primary-key* para o objeto de negócio Produto.

```
public class ProdutoPK implements java.io.Serializable
{
        public int codigo;

        public int hashCode()
        {
            return codigo;
        }

        public boolean equals(Object obj)
        {
            if (obj instanceof ProdutoPK)
            if (((ProdutoPK)obj).codigo = = codigo)
                 return true;
            return false;
        }
}
```

Listagem 4.4: Classe primary-key.

Os atributos públicos do componente *bean* do tipo-entidade são armazenados no banco de dados. Esses atributos são chamados *atributos persistentes.* Quando o componente *bean* é instalado no servidor, seus atributos persistentes devem ser mapeados para o banco de dados. Os atributos persistentes são obtidos com a utilização da classe *primary-key* (chave-primária). A chave-primária identifica cada registro do banco de dados.

Alguns fabricantes de *containers Enterprise Javabeans*, fornecem ferramentas, utilizadas na instalação dos componentes beans, que ajudam a mapear a classe *primary-key* e os atributos persistentes para o banco de dados.

4.6 - Utilização de componentes *bean* por aplicações-clientes

Para poder utilizar um componente *bean*, uma aplicação-cliente deve, inicialmente, obter uma referência remota à *interface home*. Isto pode ser feito utilizando o serviço de nomes e diretórios da linguagem Java (JNDI). Esta *interface* é uma poderosa API para localizar recursos como objetos remotos, em redes (SUN MICROSYSTEMS, 1998). O passo seguinte é obter, a partir da *interface home*, uma referência remota ao objeto que implementa os métodos de negócio – a classe do componente *bean*. A partir do momento que a aplicação-cliente obtiver a referência à classe do componente *bean*, ela pode fazer chamadas a qualquer método definido pela *interface* remota.

A Listagem 4.5 mostra um trecho de código de uma aplicação-cliente que utiliza o componente *bean* do tipo-entidade Produto.

```
public void cadastraProduto(int cod, String n, double p)
{
        Context jndiContext = getInitialContext();

        //obtém a referência para a interface home
        ProdutoHome home = (Produto)jndiContext.lookup
        (“ProdutoHome”);
```

```
            //obtém a referência para uma instância do
            //componente bean Produto
            Produto produto = home.create(cod);

            //utiliza os métodos set definidos na interface
           //remota
            produto.setNome(n);
            produto.setPreco(d);
    }
```

Listagem 4.5: Aplicação-cliente.

Na verdade, as referências obtidas pela aplicação-cliente não são exatamente referências à *interface home* e à classe que implementa o componente *bean*. A aplicação-cliente utiliza seus stubs para obter referência ao objeto *home* e ao objeto *Enterprise Javabeans*. O objeto *home* é um objeto criado pelo servidor *Enterprise Javabeans* que implementa a *interface home*. Utilizando este objeto, é possível obter uma instância de objeto *Enterprise Javabeans*. Este objeto tem uma referência para a classe que implementa o componente *bean*. Desta forma, a aplicação-cliente não acessa diretamente a instância do componente *bean*.

4.7 - As elementos do servidor *Enterprise Javabean*s utilizadas por aplicações-clientes

Em uma aplicação-cliente, os dois elementos utilizados são a classe do objeto *Enterprise Javabeans* e a classe que implementa a *interface home*. A definição dessas duas classes são proprietárias, cada fabricante de *container Enterprise Javabeans* tem a sua implementação - que geralmente não é pública.

O desenvolvedor de aplicações está interessado no seu ambiente de negócio e se preocupa em modelar este ambiente. Desta forma, sua atividade está relacionada com o desenvolvimento de componentes e aplicações que representam o seu negócio. Os desenvolvedores de *containers* e servidores *Enterprise Javabeans*, não se preocupam com lógica de negócio, mas sim em desenvolver sistemas de monitoramento de transações baseados em componentes.

Esta divisão de responsabilidades, permite que, cada área, evolua da sua maneira, e que os desenvolvedores, concentrem seus esforços em problemas específicos.

4.7.1 - A classe do objeto *Enterprise Javabeans*

Um objeto *Enterprise Javabeans* é um objeto distribuído que implementa a *interface* remota do componente *bean*. Ele envolve a instância de componente *bean* (um objeto da classe *Enterprise Javabeans* que implementa os métodos de negócio) no servidor e expande sua funcionalidade, incluindo o comportamento do objeto *Enterprise Javabeans* – `javax.ejb.EJBObject`. O objeto *Enterprise Javabeans* é gerado por utilitários fornecidos pelo fabricante do *container* e é baseado nas classes do componente *bean* e nas informações fornecidas no momento da instalação. O objeto

Enterprise Javabeans envolve a instância do componente *bean* e trabalha com o *container* para oferecer controle de transações, segurança e outros serviços para o componente no momento de execução.

Existem algumas estratégias que os fabricantes utilizam para implementar o objeto *Enterprise Javabeans*. A Figura 4.2 mostra a forma mais comum de implementação do objeto *Enterprise Javabeans*. Neste caso, o objeto *Enterprise Javabeans* é um típico *container* que possui uma referência para a classe do componente *bean* e delega as chamadas para a instância do componente.

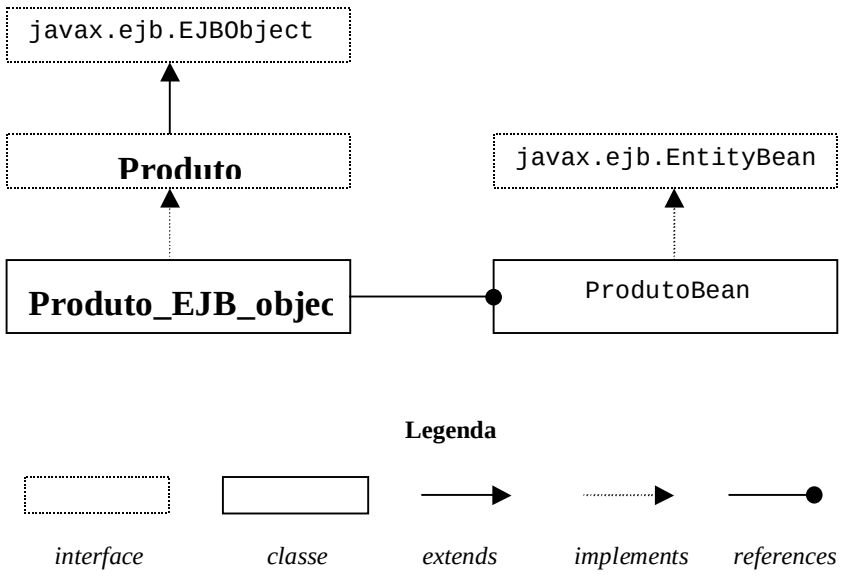


Figura 4.2 - Implementação do Objeto Enterprise Javabeans.

O desenvolvedor de componentes e de aplicações não precisa conhecer muito, a respeito do objeto *Enterprise Javabeans*. Os detalhes de implementação deste objeto são de responsabilidade do fabricante do *container*. Para o desenvolvedor, envolvido com o

negócio, basta, apenas, saber da existência do objeto. Os detalhes que realmente são importantes dizem respeito à *interface* remota e à *interface home* dos componentes *Enterprise Javabeans.*

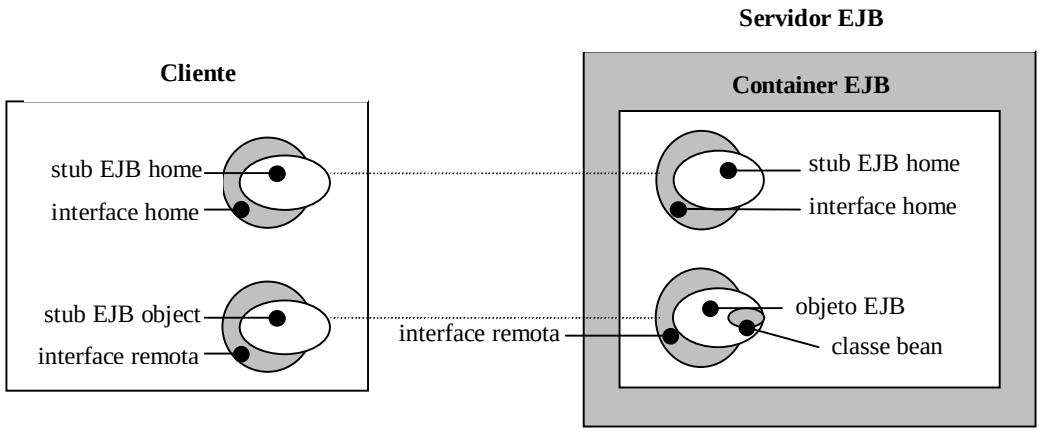
4.7.2 - O objeto *home*

O objeto *home* é muito parecido com o objeto *Enterprise Javabeans*. Sua classe também é gerada, automaticamente, quando o componente é instalado no *container*. A classe implementa todos os métodos definidos na *interface home* do componente *bean* e é responsável pela ajuda ao *container* em gerenciar o ciclo de vida do componente *bean*. Atuando em conjunto com o *container Enterprise Javabeans*, o objeto *home* é responsável pela localização, criação, e remoção dos componentes *beans*. Essas atividades, acionam mecanismos embutidos no servidor *Enterprise Javabeans*, como gerenciadores de recursos, gerenciadores de instanciação de componentes e mecanismos de persistência, cujos detalhes, são escondidos do desenvolvedor dos componentes.

Por exemplo, quando um método `create()` é chamado, o objeto *home* cria uma instância do objeto *Enterprise Javabeans*, o qual possui uma referência para uma instância de componente *bean* apropriado. Quando a instância de componente *bean* é associada ao objeto *Enterprise Javabeans*, o método `ebjCreate()` é chamado. Se o componente *bean* for do tipo-entidade, um novo registro é inserido no banco de dados. Se o componente *bean* for do tipo-sessão, a instância simplesmente é inicializada. Depois que o método `ejbCreate()` for concluído, o objeto *home* retorna uma referência remota ao objeto *Enterprise Javabeans* para a aplicação-cliente. A aplicação-cliente, então, pode fazer chamadas aos métodos implementados pela classe do

componente *bean* através do *stub*. O *stub* repassa a chamada para o objeto *Enterprise Javabeans* que, por sua vez, delega a mesma à instância do componente *bean*.

A Figura 4.3 mostra a arquitetura *Enterprise Javabeans*, com o objeto *home* e o objeto *Enterprise Javabeans*, implementando a interface *home* e a *interface* remota, respectivamente.



EJB – *Enterprise Javabeans*

Figura 4.3 - Objeto *home* e Objeto *Enterprise Javabeans.*

4.8 - Instalação dos componentes *bean*

Grande parte das informações sobre o gerenciamento de componentes *beans* em tempo de execução, não são definidas no momento da implementação das classes e *interfaces*. O código fonte, das classes e *interfaces* dos componentes *beans*, não mostram como esses componentes vão interagir com os mecanismos de segurança, com o controle de transação, e outros serviços utilizados por objetos distribuídos. Esses

serviços são, automaticamente, administrados pelo servidor *Enterprise Javabeans*. Apesar do servidor gerenciar, automaticamente, os serviço utilizados pelos componentes, ele precisa saber como aplicar e oferecer os serviços para cada componente instalado.

4.8.1 - *Deployment descriptors*

Deployment descriptors são arquivos utilizados para configuração de componentes *beans*. Eles permitem que o comportamento do software (conjunto de componentes *beans*) seja configurado em tempo de execução, sem a necessidade de mudança do próprio *software*. Arquivos de configuração são muito utilizados por aplicações convencionais. *Deployment descriptors* são arquivos de configuração utilizados, especificamente, por componentes *Enterprise Javabeans*. Eles permitem que comportamentos de tempo de execução, como por exemplo, a forma de controle de transação utilizada por um determinado método do componente *bean*, possam ser alterados sem a necessidade de modificação de nenhuma das classes ou interfaces de definição do componente.

Quando uma classe de componente *bean* e suas interfaces são definidas, um *deployment descriptor* para o componente deve ser criado. A definição do *deployment descriptor* e a sua configuração adequada, é o primeiro passo para que o componente possa ser instalado em um *container Enterprise Javabeans*. Existem ferramentas, disponíveis no mercado, que oferecem recursos como editores com *interface* visual, por exemplo, que o desenvolvedor pode utilizar quando for definir o *deployment descriptor* para o componente bean. Depois que a definição do *deployment descriptor* for

concluída e salva em um arquivo, o componente bean pode ser “empacotado” em um arquivo JAR, para instalação em um *container Enterprise Javabeans*.

4.8.2 - Arquivo JAR

Os arquivos JAR (arquivos Java) são arquivos de formato ZIP (arquivos de compactação) que são utilizados, especificamente, para empacotamento de classes Java (e outros recursos como imagens, por exemplo) que devem ser utilizadas por algum tipo de aplicação. Os arquivos JAR são utilizados para empacotamento de *applets*, aplicações Java, componentes *Javabeans*, e componentes *Enterprise Javabeans*. Um arquivo JAR, pode conter, um ou mais componentes *Enterprise Javabeans* com suas respectivas classes e interfaces. No arquivo JAR, utilizado para empacotamento de componentes *Enterprise Javabeans*, deve existir, também, o *deployment descriptor*, cuja configuração, é utilizada por todos os componentes incluídos no arquivo.

Quando um componente *Enterprise Javabeans* é instalado, a localização do arquivo JAR, que contém o componente, deve ser informada ao *container*, para que as suas ferramentas de instalação possam ler o arquivo JAR. O *container* utiliza o *deployment descriptor*, para descobrir quais componentes existem no arquivo JAR, e a forma como cada componente deve ser gerenciado em tempo de execução. O *deployment descriptor* informa, às ferramentas de instalação, que tipo de componente bean existe no arquivo JAR (componente *bean* do tipo sessão ou componente *bean* do tipo entidade), como o componente deve trabalhar com transações, quais usuários podem acessar o componente, e outros atributos de tempo de execução.

Na instalação do componente, algumas desses configurações, como os atributos de transação e segurança, por exemplo, podem ser alteradas para adaptação do componente a uma determinada aplicação. Existem ferramentas que lêem, diretamente, o d*eployment descriptor* contido em um arquivo JAR. Essas ferramentas possuem editores que facilitam a configuração de cada atributo do componente *bean*.

Os *deployment descriptors* auxiliam as ferramentas na instalação de um componente *bean* no *container*. Uma vez que o componente *bean* tenha sido instalado, os atributos descritos no *deployment descriptor* são utilizados, para informar ao container *Enterprise Javabeans*, como cada componente deve ser gerenciado em tempo de execução.

4.8.3 - Instalação de componentes baseados na especificação Enterprise Javabeans 1.1

Os componentes *Enterprise Javabeans* baseados na especificação 1.0 tinham, como *deployment descriptors*, conjuntos de objetos serializados. Um programa deveria ser escrito para configurar, apropriadamente, um objeto da classe `DeploymentDescriptor` e, então, o objeto deveria ser serializado. Depois de serializado, o *deployment descriptor* era empacotado com as classes e *interfaces* do componente em um arquivo JAR.

Os componentes *Enterprise Javabeans* baseados na especificação 1.1 têm como *deployment descriptors* arquivos *XML*. Arquivos XML são fáceis de editar, mesmo sem ferramentas visuais, e são muito flexíveis. Além disso, grande parte dos fabricantes de ambientes de desenvolvimento integrado e servidores *Enterprise Javabeans* fornecem ferramentas para criação de *deployment descriptors* baseados em XML. A abordagem

adotada pela especificação *Enterprise Javabeans* 1.1 é superior em relação a proposta pela especificação anterior.

Os *deployment descriptors,* baseados em XML, são arquivos texto estruturados de acordo com o padrão *Enterprise Javabeans* DTD (*Document Type Definition*), que podem ser estendidos. Isto permite que, o tipo de informação de instalação armazenada no *deployment descriptor*, possa evoluir a medida que a especificação evolui. A Listagem 4.6 mostra um *deployment descriptor* que pode ser utilizado na instalação do componente *bean* Produto.

```
<?xml version="1.0"?>

<!DOCTYPE ejb-jar PUBLIC
"-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans
1.1//EN" "http://java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd">

<ejb-jar>
 <enterprise-beans>
   <entity>
      <ejb-name>ProdutoBean</ejb-name>
      <home>ProdutoHome</home>
      <remote>Produto</remote>
      <ejb-class>ProdutoBean</ejb-class>
      <persistence-type>Container</persistence-type>
      <prim-key-class>java.lang.Integer</prim-key-class>
      <primkey-field>codigo</primkey-field>
      <reentrant>False</reentrant>
```

```
    </entity>
 </enterprise-beans>
</ejb-jar>
```

Listagem 4.6: *Deployment descriptor*.

O exemplo da Listagem 4.6 mostra um *deployment descriptor* muito simples, com poucas tags. Normalmente, um *deployment descriptor* para um componente *bean* possui muito mais informações.

O primeiro elemento do documento é a versão da linguagem XML utilizada. O documento exibido na Listagem 4.6 utiliza a versão 1.0 da especificação XML.

O segundo elemento no documento XML é `!DOCTYPE`. Este elemento descreve a organização que definiu o DTD para o documento, a versão do DTD, e a localização URL para o DTD. O DTD descreve como um documento XML está estruturado.

Os elementos `xml version` e `!DOCTYPE` existem em qualquer documento XML, seja ele um deployment descriptor ou não. As outras tags existentes no exemplo da list. 4.6 são utilizadas apenas em *deployment descriptors* de componentes *Enterprise Javabeans*. As tags referentes a *deployment descriptors* que aparecem na Listagem 4-6 são descritas abaixo:

`ejb-jar`

A raiz de um *deployment descriptor* baseado em XML. Todos os outros elementos devem estar aninhados abaixo deste elemento. Ele deve conter um elemento `enterprise-beans` assim como outros elementos opcionais.

`enterprise-beans`

Contém declarações de todos os componentes *beans* descritos pelo documento XML. Este elemento pode conter elementos `entity` (para componentes do tipo-entidade) e elementos `session` (para componentes do tipo-sessão).

`entity`

Descreve um componente do tipo-entidade e sua informação de instalação. Deve existir um elemento `entity` para cada componente do tipo-entidade descrito pelo *deployment descriptor*. (O elemento `session` é utilizado da mesma forma para descrever componentes do tipo-sessão.)

`ejb-name`

O nome do componente *bean*. Este nome representa o componente como um todo – o conjunto de todas as classes e interfaces utilizadas pelo componente bean.

`home`

Nome da interface *home* do componente *bean*. Esta é a *interface* que define os métodos referentes ao ciclo de vida do componente (*create, find, remove*).

`remote`

Nome da interface remota do componente *bean*. Esta *interface* define os métodos de negócio do componente.

`ejb-class`

Nome da classe do componente *bean*. Esta classe implementa os métodos de negócio do componente *bean*.

`prim-key-class`

Nome da classe *primary-key* do componente *bean*. A classe *primary-key* é utilizada para localizar o componente no banco de dados.

`persistence-type`

Informação sobre o tipo de estratégia utilizada na persistência de componentes do tipo-entidade. Este elemento pode receber dois valores: `Container` ou `Bean` (persistência gerenciada automaticamente pelo container ou persistência gerenciada pelo componente bean, respectivamente).

`Reentrant`

Descreve se o componente *bean* do tipo-entidade permite acesso recursivo. Este elemento pode receber dois valores: `True` ou `False.` Se o valor for `True` o componente *bean* do tipo-entidade permite acesso recursivo*;* Se valor for `False` o componente *bean* gera uma exceção se um acesso recursivo ocorrer.

5 – O ROTEIRO PROPOSTO

A utilização de objetos, para entender e modelar processos de negócio, pode ser uma boa opção para o sucesso de organizações que atuam neste panorama complexo e dinâmico. O processo de reengenharia de negócios envolve, também, a reformulação dos sistemas de informação. Os sistemas de informação devem oferecer flexibilidade, possibilitar reutilização, atuar em ambientes distribuídos e, principalmente, evoluir de acordo com a evolução dos processos de negócio.

O processo de construção dos sistemas de informação deve ser um processo de montagem, no qual, se utilizam componentes que implementam as políticas de negócio. Para se desenvolver componentes, é recomendável que se tenha uma plataforma que ofereça os serviços de suporte e, também, uma metodologia que oriente o desenvolvedor de componentes, indicando as atividades que devem ser realizadas durante o processo de desenvolvimento.

O roteiro desenvolvido pode ser aplicado ao processo de desenvolvimento de objetos de negócio como componentes *Enterprise Javabeans*. O modelo de componentes de servidor *Enterprise Javabeans* oferece uma estrutura adequada para a implementação de objetos de negócio que, quando implementados, são chamados de "componentes de negócio".

O roteiro de desenvolvimento de componentes combina atividades, realizadas no processo de reengenharia de negócios (JACOBSON et al., 1995), com atividades oriundas do ciclo de vida do projeto de sistema tradicional.

A atividade de modelagem de negócio, principal atividade do processo de reengenharia de negócios, identifica os objetos de interesse, assim como os processos existentes em um negócio. Essa atividade deve representar o início do processo de desenvolvimento de componentes.

Após a obtenção do modelos que representam o negócio, os objetos e processos identificados na atividade de modelagem de negócio devem ser projetados para futura implementação. A etapa de projeto é proveniente do ciclo de vida de projeto de sistema tradicional. Essa etapa compreende uma série de atividades que, em conjunto, traduzem o resultado do trabalho de modelagem para um modelo mais próximo de uma implementação. Detalhes de implementação, como modelo de componentes utilizado, controle de transação e esquema de persistência, devem influenciar na construção dos modelos pertencentes a etapa de projeto de componentes. As tecnologias utilizadas na etapa de implementação devem ser definidas, ainda, na etapa de projeto de componentes.

A atividade de implementação de componentes deve ser realizada de forma direta, ou seja, nenhuma mudança que afete a estrutura dos componentes, representada pelos modelos criados na etapa de modelagem de negócio e projeto de componentes, deve ser realizada.

Depois de implementados, os componentes devem ser instalados em um servidor que ofereça compatibilidade com o modelo de componentes *Enterprise Javabeans*. O processo de instalação pode ser configurado de acordo com os requisitos de cada

negócio. O recurso que permite esse tipo de flexibilidade é o *deployment descriptor*, arquivo de configuração escrito em linguagem XML. A instalação de componentes Enterprise Javabeans pode ser feita com a ajuda de ferramentas oferecidas por grande parte dos servidores que implementam o modelo de componentes.

5.1 – Ferramentas utilizadas

A técnica de modelagem de negócios baseada em objetos, proposta por Jacobson (JACOBSON et al. 1995), é utilizada na etapa de modelagem de negócio sugerida pelo roteiro. O roteiro propõe a utilização desta técnica na obtenção do modelo de negócio. A principal ferramenta oferecida pela técnica, proposta por Jacobson, é o caso de uso (*Use Cases*). Esta ferramenta pode ser utilizada na representação dos processos de negócio.

O roteiro utiliza um subconjunto dos diagramas oferecidos pela linguagem UML (OMG, 1997b) – diagrama de casos de uso, diagrama de sequência e diagrama de classes. Esses diagramas constituem, em conjunto, o modelo que representa o negócio. Os diagramas podem representar estruturas estáticas, no caso do modelo de objetos, assim como o fluxo de trabalho de um processo de negócio, representado pelo diagrama de seqüência de mensagens .

Ferramentas utilizadas no ciclo de vida do projeto de sistema tradicional, também, podem ser utilizadas. O diagrama de fluxo de dados (YOURDON, 1990) pode ser utilizado para representar os processos de negócio, substituindo os casos de uso, enquanto o diagrama Entidade-Relacionamento (YOURDON, 1990) pode ser utilizado

na construção do modelo que representa a estrutura do banco de dados que armazena as informações de estado do componente.

O modelo de componentes *Enterprise Javabeans* (apresentado no Capítulo 4) é o modelo de componentes de servidor que serve de base para a implementação dos objetos de negócio.

Os componentes são implementados em linguagem de programação Java (HORSTMANN & CORNELL, 1999). O modelo de componentes *Enterprise Javabeans*, estabelece como requisito obrigatório, a utilização da linguagem Java na implementação de componentes. Apesar de impor a utilização da linguagem Java, na implementação dos componentes, o modelo de componentes *Enterprise Javabeans* permite que aplicações-clientes sejam implementadas em qualquer linguagem de programação (desde que haja o suporte ao padrão CORBA).

O servidor que faz parte do pacote de instalação da plataforma *Java 1.2 Enterprise Edition* (SUN MICROSYSTEMS, 1999) foi utilizado para instalação, execução e teste dos objetos de negócio implementados. Esse produto é distribuído gratuitamente. A Sun Microsystems desenvolveu esse produto com o propósito de oferecer uma referência de implementação que pudesse ser utilizada pelos fabricantes de servidores e *containers Enterprise Javabeans*. O benefício se estende aos desenvolvedores de componentes que, podem criar os componentes *Enterprise Javabeans*, e testar o seu funcionamento. Esse produto oferece suporte a todos os elementos da plataforma *Java 1.2 Enterprise Edition* (*Enterprise Javabeans*, *Servlets* e *Java Server Pages*). O servidor oferecido pela Sun Microsystems, possui alguns recursos que facilitam o processo de instalação de componentes como ambiente visual, editores de configuração de propriedades de componentes e assistentes. O *deployment*

descriptor e o arquivo JAR (utilizados na instalação dos componentes) são construídos, automaticamente, com o auxílio desses recursos.

5.2 – As etapas do processo de desenvolvimento

O roteiro propõe a divisão do processo de desenvolvimento de componentes em quatro etapas principais: modelagem de negócio, projeto de componentes, implementação e instalação de componentes.

O modelo deve ser orientado por processos de negócio, isto significa que, o desenvolvimento dos componentes deve ser orientado por processos. A primeira atividade do processo de desenvolvimento de componentes é a identificação dos processos de negócio. Após a identificação dos processos de negócio, o processo de desenvolvimento deve abordar os processos de negócio de forma individual, respeitando as dependências existentes. Desta forma, o processo de desenvolvimento deve abordar processos de negócio como Venda de Produtos e Emissão de Pedidos, por exemplo, individualmente. Para cada processo de negócio deve-se realizar o conjunto de etapas proposto pelo roteiro. A Figura 5.1 mostra as etapas do processo de desenvolvimento de componentes propostas pelo roteiro.

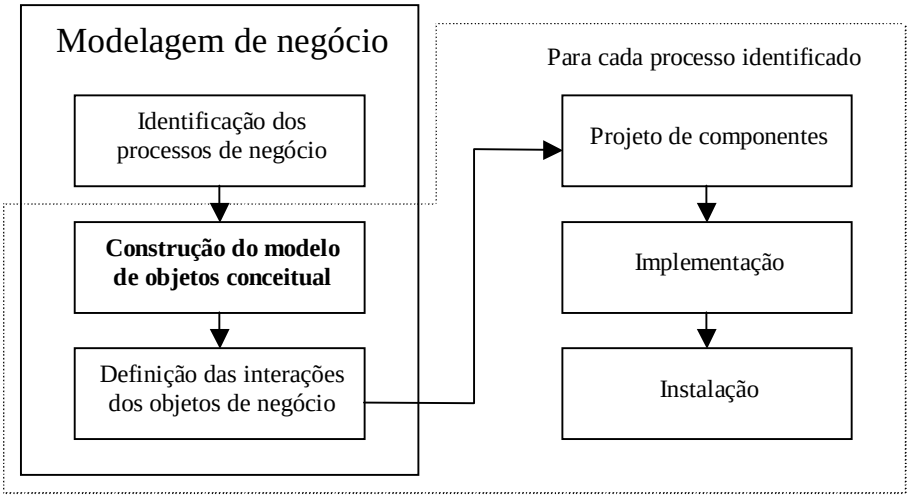


Figura 5.1 - Etapas do processo de desenvolvimento de componentes.

5.2.1 – Modelagem de negócio

A primeira etapa do processo de desenvolvimento de componentes é a modelagem de negócio. A obtenção do modelo conceitual de um negócio (objetos e processos) é o resultado desta primeira etapa. Nesta etapa, os processos de negócio são identificados. Outras duas atividades presentes nesta etapa são a construção do modelo conceitual de objetos e a definição das interações dos objetos de negócio.

5.2.1.1 – Identificação dos processos de negócio

A identificação dos processos existentes em um negócio é a primeira atividade realizada no processo de desenvolvimento de objetos de negócio. Esta atividade é fundamental na modelagem de negócios e implementação de objetos de negócio. Todos

os processos significativos de um negócio devem ser identificados. Venda de Produtos, Emissão de Pedidos, Pagamento de Faturas são exemplos de processos que podem existir em um negócio.

Na identificação dos processos de negócio, a utilização de ferramentas como diagrama de fluxo de dados ou casos de uso é recomendada. No desenvolvimento de componentes, as tecnologias utilizadas são baseadas em objetos, por isso, nesse caso, recomenda-se a utilização de casos de uso na representação dos processos. A utilização de casos de uso para representação de processos de negócio é, também, utilizada na reengenharia de processos de negócio (JACOBSON et al., 1995). Na técnica de Jacobson, cada caso de uso representa um processo de negócio. Um caso de uso é a descrição de uma seqüência de eventos que produz um resultado. Após a identificação e descrição dos processos de negócio, é interessante, quando casos de uso são utilizados, elaborar o diagrama de casos de uso. Este diagrama apresenta o conjunto de processos de um negócio e a relação desses processos com entidades externas . Na técnica de Jacobson, as entidades externas são chamadas de atores. A Figura 5.2 mostra um exemplo de diagrama de casos de uso.

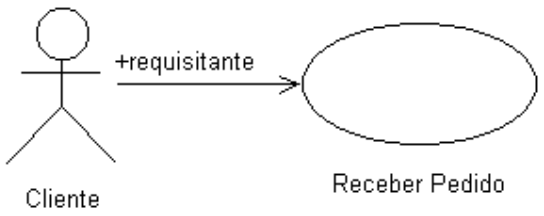


Figura 5.2 - Diagrama de casos de uso.

5.2.1.2 – Construção do modelo conceitual de objetos

O modelo conceitual é formado pelos objetos que participam de um processo de negócio. Este modelo representa a estrutura estática do processo de negócio. Ele pode ser representado pelo diagrama de classes contendo atributos e associações. A linguagem UML (OMG, 1997b) oferece uma notação que pode ser utilizada na construção do modelo conceitual de objetos.

Os objetos são identificados a partir da descrição do processo de negócio. Os objetos de interesse em um negócio são, geralmente, substantivos significativos encontrados na descrição do processo. A Figura 5.3 mostra um exemplo de modelo conceitual de objeto.

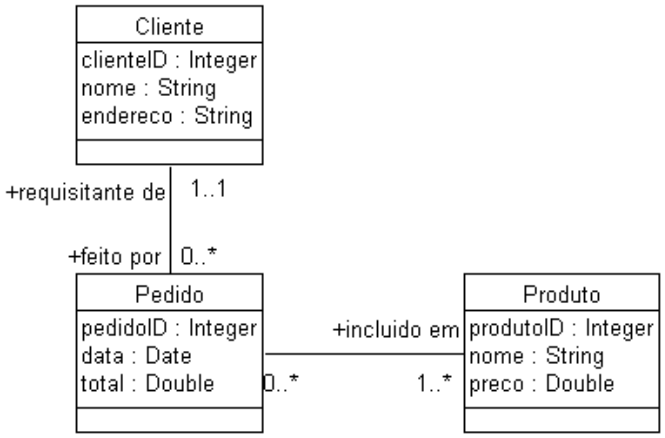


Figura 5.3 - Modelo conceitual de objetos (diagrama de classes).

É importante, no momento de identificação dos objetos que participam de um processo de negócio, verificar se, esses objetos, já foram identificados em um outro processo de negócio. Como o roteiro de desenvolvimento de componentes sugere uma abordagem orientada por processos de negócio, a ocorrência de um objeto pode ser observada mais de uma vez na modelagem de negócio. Desta forma, um objeto pode ser compartilhado por vários processos existentes em um negócio. A utilização de uma ferramenta CASE, na construção dos modelos que representam um negócio, pode auxiliar na tarefa de integração dos modelos. A maioria das ferramentas CASE possui um banco de dados, denominado repositório, no qual ficam armazenados todos os elementos identificados na modelagem de negócio. Esse recurso permite que um mesmo elemento de negócio possa ser utilizado em vários modelos.

5.2.1.3 - Definição das interações dos objetos de negócio

Nesta atividade, os processos de negócio são investigados. Essa investigação consiste no levantamento das interações que acontecem entre os objetos participantes do processo. O diagrama de seqüência de mensagens, recurso oferecido pela linguagem UML, pode ser utilizado para representar as interações dos objetos. Esse diagrama mostra os eventos que entidades ou processos externos podem gerar e o fluxo de trabalho como resposta a estes eventos. A Figura 5.4 mostra um exemplo de diagrama de seqüência de mensagens.

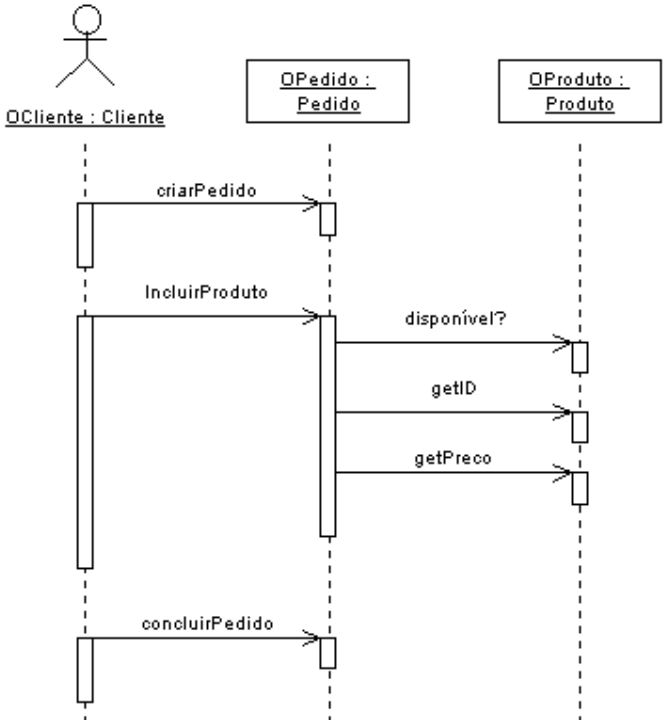


Figura 5.4 - Diagrama de seqüência de mensagens.

5.2.2 – Projeto de componentes

Os modelos, voltados para implementação dos objetos de negócio, podem ser construídos a partir dos modelos produzidos na etapa de modelagem de negócio. Todas as decisões, que afetam diretamente o trabalho de implementação, devem ser realizadas na etapa de projeto. Nesta etapa, os objetos de negócio são planejados para a implementação de acordo com o modelo de componentes *Enterprise Javabeans*. A construção do modelo de objetos, da lista de objetos persistentes e do plano de implementação do processo de negócio são as atividades realizadas nesta etapa.

5.2.2.1 – Construção do modelo de objetos de negócio

Nesta atividade, o modelo de objetos é construído a partir do modelo conceitual desenvolvido na etapa de modelagem de negócio. O diagrama de classes é estendido, mostrando, também, as operações dos objetos.

As operações dos objetos podem ser obtidas a partir da análise do diagrama de seqüência, desenvolvido na atividade de definição de interações dos objetos de negócio. O diagrama de seqüência de mensagens mostra as requisições feitas a cada objeto. Essas requisições podem ser mapeadas, nesta atividade, como operações de objeto.

Outras técnicas utilizadas na definição de operações de objetos, como utilização de padrões baseada na análise do diagrama de seqüência ou do diagrama de colaboração, são propostas por Larman (LARMAN, 1997). A Figura 5.5 mostra um exemplo de digrama de classes que representa o modelo de objetos de um negócio.

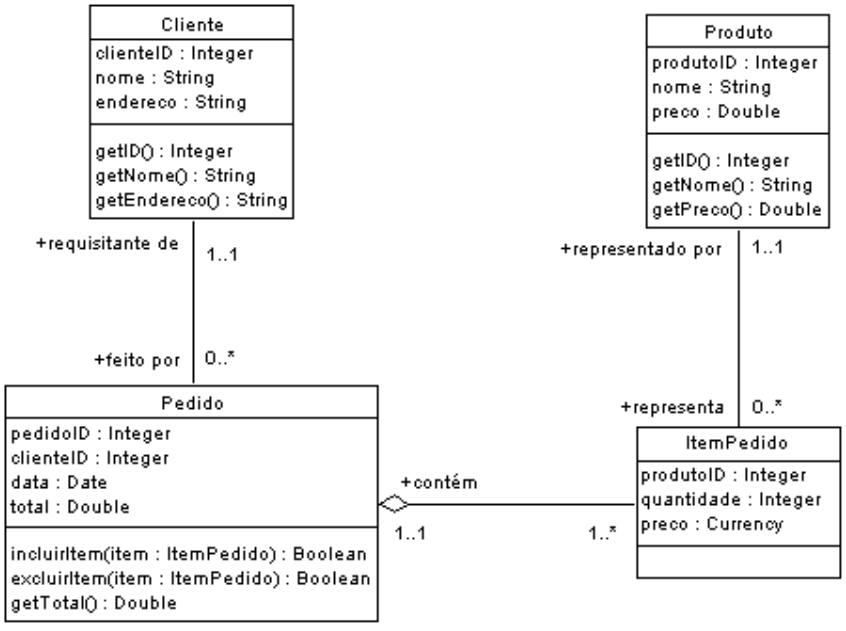


Figura 5.5- Modelo de objetos de negócio (diagrama de classes).

5.2.2.2 - Projeto de componentes *Enterprise Javabeans* do tipo-entidade

O trabalho realizado nesta etapa consiste na inclusão dos detalhes de implementação de componentes *Enterprise Javabeans* no diagrama de classes que representa o modelo de objetos de negócio. O resultado é a obtenção do modelo de objetos *Enterprise Javabeans.*

As atividades realizadas nesta etapa são: identificação dos objetos dependentes, definição do gerenciamento de persistência dos objetos, inclusão das operações de criação e busca no modelo e definição dos atributos ou classes `primary-key`. Essas atividades estão diretamente relacionadas com a implementação dos objetos de negócio de acordo com o modelo de componentes *Enterprise Javabeans*. A Figura 5.6 mostra o modelo de objetos de negócio com as modificações introduzidas em cada atividade.

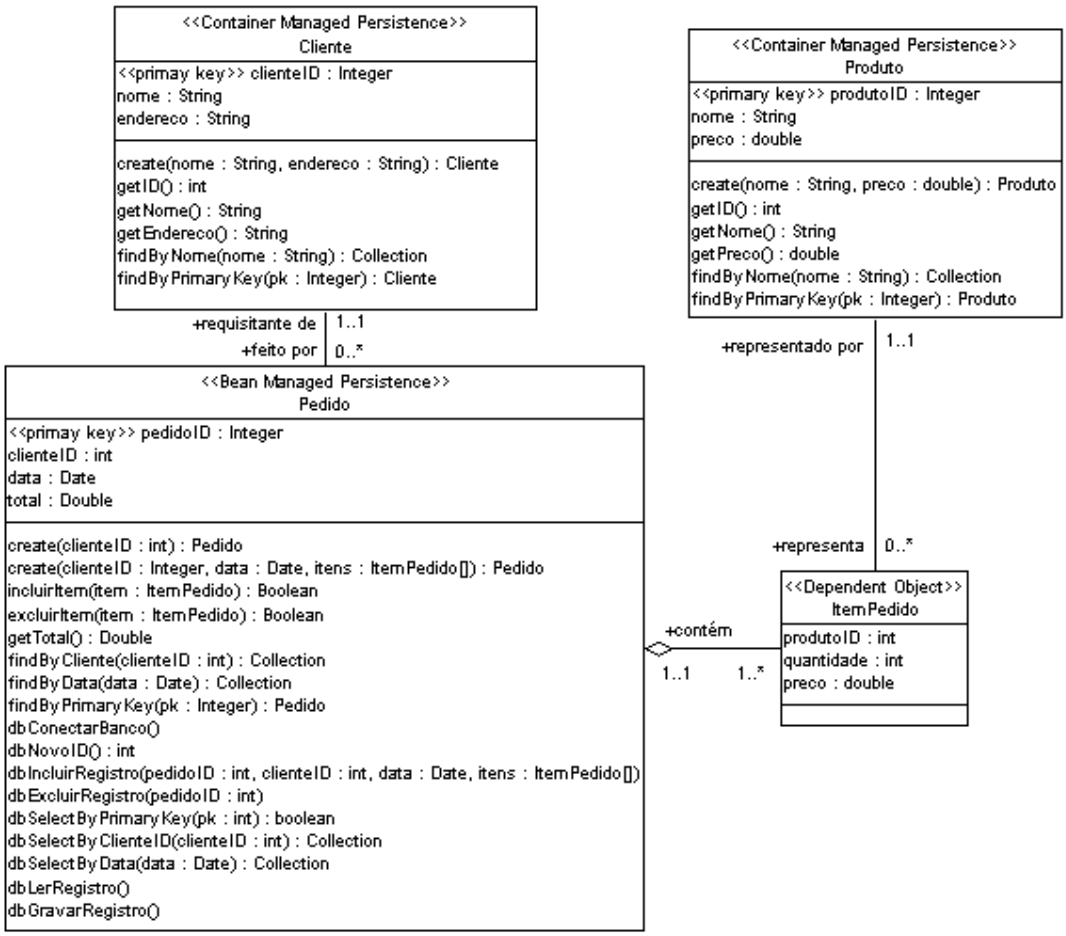


Figura 5.6 - Modelo de objetos Enterprise Javabeans.

- Identificação dos objetos dependentes

O primeiro detalhe que deve ser incluído, no modelo de objetos, é a indicação dos objetos persistentes e dos objetos dependentes. Nem todos os objetos identificados no modelo de objetos de negócio são mapeados para componentes *Enterprise Javabeans.* Um componente *Enterprise Javabeans* é um objeto de negócio persistente (neste caso ele deve ser mapeado para componente *Enterprise Javabeans* do tipo entidade) ou um

processo de negócio (quando ele é mapeado para componente *Enterprise Javabeans* do tipo-sessão.

Outra característica de um componente *Enterprise Javabeans* é que ele é um objeto de negócio independente, ou seja, o conceito que ele representa em um negócio é, além de único, independente. Por exemplo, o objeto de negócio Cliente representa um conceito de negócio que é independente de outros conceitos de negócio.

O objeto ItemPedido é um objeto que pode ser identificado em um modelo de objetos de negócio. O conjunto de informações que ele representa deve ser armazenado em um banco de dados para futuras necessidades, portanto, ele é um objeto persistente. Porém, este objeto não deve ser mapeado para componente *Enterprise Javabeans* do tipo-entidade, pois ele é um objeto dependente do objeto Pedido.

Os objetos dependentes não são mapeados para componentes *Enterprise Javabeans*. Eles devem, no modelo de objetos *Enterprise Javabeans*, ser identificados. O *stereotype* é um elemento da linguagem UML que pode ser utilizado para identificar um objeto dependente no diagrama de classes que representa o modelo de objetos *Enterprise Javabeans*. O roteiro sugere a utilização do *stereotype* <<`Dependent Object`>> para identificação dos objetos dependentes.

- Definição do gerenciamento de persistência dos objetos

Os objetos persistentes são aqueles que não foram identificados no modelo de objetos *Enterprise Javabeans* como objetos dependentes. Esses objetos são implementados como componentes *Enterprise Javabeans* do tipo entidade.

A persistência dos objetos, no modelo *Enterprise Javabeans*, pode ser de dois tipos: persistência gerenciada pelo *container* e persistência gerenciada pelo componente. Desta forma, o desenvolvedor de componentes *Enterprise Javabeans* do tipo-entidade pode implementar as instruções que realizam a persistência do objeto no próprio componente ou atribuir essa responsabilidade ao *container Enterprise Javabeans*. As informações dos objetos ficam armazenadas no banco de dados, que geralmente, é relacional. Cada objeto pode ter uma ou mais tabelas de banco de dados correspondentes. Se as informações de um objeto de negócio podem ser armazenada em uma única tabela de banco de dados, então, o gerenciamento de persistência desse objeto deve ser feito pelo *container*. Para identificar os objetos que devam ter a sua persistência gerenciada pelo *container,* no diagrama de classes que representa o modelo de objetos *Enterprise Javabeans*, recomenda-se a utilização do *stereotype* `<<Container Managed Persistence>>`.

Se as informações de um objeto de negócio são armazenadas em mais de uma tabela de banco de dados, o gerenciamento de persistência deve ser feito pelo componente *Enterprise Javabeans.* Para identificar os objetos de negócio que têm a persistência gerenciada pelo componente, no diagrama de classes que representa o modelo de objetos *Enterprise Javabeans*, recomenda-se a utilização do *stereotype* `<<Bean Managed Persistence>>`. Nesse caso, as instruções de acesso a dados devem ser implementadas pelo desenvolvedor do componente. As operações que contém essas instruções devem ser inseridas nas classes que representam os objetos de negócio, no modelo de objetos *Enterprise Javabeans*. É recomendável que o nome dessas operações comece com o prefixo `db`. O desenvolvedor do componente, também, pode construir classes de acesso a dados. Essas classes possuem apenas funções de

acesso a dados e são utilizadas como elemento intermediário entre os objetos de negócio e o banco de dados. Quando o desenvolvedor de componente utiliza classes de acesso a dados, é recomendável que o nome dessas classes seja o nome da classe do objeto de negócio acrescido do sufixo DAO (*Data Access Object*).

- Inclusão das operações de criação e busca

As funções construtoras de objetos de negócio são definidas, nos componentes *Enterprise Javabeans*, como operações `create()`. Um componente *Enterprise Javabeans* deve possuir uma ou mais operações `create()`. Essas operações devem ser chamadas, em aplicações-clientes, quando é necessário se obter uma referencia à instância do componente *Enterprise Javabeans*. As operações `create()`, no projeto de componentes do tipo entidade, devem ser incluídas no diagrama de classes que representa o modelo de objetos *Enterprise Javabeans*.

As operações de busca são utilizadas quando aplicações-clientes ou outro componente *Enterprise Javabeans* precisam encontrar um determinado objeto. Essas operações possuem argumentos que são utilizados como referência na consulta ao banco de dados. É recomendável que, na definição das operações de busca, o nome dessas operações tenha como prefixo o termo `findBy`. Todo componente *Enterprise Javabeans* deve possuir a função de busca `findByPrimaryKey()`. Essa função (única função de busca definida como obrigatória pelo modelo de componentes *Enterprise Javabeans*) procura por um objeto que possua o argumento como valor para o seu atributo `primary-key`.

O componente *Enterprise Javabeans* do tipo-entidade deve possuir o maior número possível de operações de busca, pois, desta forma, ele pode oferecer flexibilidade na utilização, em aplicações-clientes, do objeto de negócio.

- Definição dos atributos ou classes *primary-key*

Os componentes *Enterprise Javabeans* do tipo-entidade devem possuir um atributo ou classe `primary-key`. Os objetos de negócio do tipo-entidade são identificados por esse atributo ou classe. O roteiro recomenda que, por motivos de simplicidade, sempre que possível, seja utilizado um atributo como `primary-key`. Esse atributo deve ser uma classe definida na linguagem Java (`Integer` ou `String`, por exemplo). Nesse caso, o atributo deve ser identificado no diagrama de classes que representa o modelo de objetos *Enterprise Javabeans.* Para identificação do atributo `primary-key`, no diagrama de classes, recomenda-se a utilização do stereotype `<<primary-key>>`. Se for necessário, para identificar um objeto de negócio, a utilização de mais de um atributo, então, uma classe `primary-key` deve ser utilizada. Nesse caso, a classe deve ser incluída no diagrama de classes. É necessário, também, que seja estabelecido o relacionamento de associação entre a classe do objeto de negócio e a sua classe `primary-key.` Os papéis nesse relacionamento devem ser "`identificado por`" para a classe do objeto de negócio e "`primary-key de`" para a classe `primary-key.`

5.2.2.3 - Projeto de componentes Enterprise Javabeans do tipo-sessão

Cada processo de negócio, identificado na primeira etapa do roteiro, deve ser implementado como componente *Enterprise Javabeans* do tipo-sessão. Esses componentes podem ser de dois tipos: com estado e sem estado.

Um componente *Enterprise Javabeans* do tipo-sessão com estado é um componente que funciona como extensão das aplicações-clientes que o utilizam. Esse componente possui atributos que podem ser acessados por todas as operações definidas na classe do componente. É recomendável que, esse tipo de componente, seja utilizado na representação de um processo de negócio, pois, desta forma, a montagem de aplicações-clientes pode ser simplificada.

Um componente *Enterprise Javabeans* do tipo-sessão sem estado tem, na sua classe, operações independentes entre si. As operações desse componente não podem dividir os mesmos atributos de classe e, as informações necessárias para a execução de cada operação, devem ser passadas como argumentos de função.

Se uma instância de processo de negócio deve ser única, ou seja, essa instância não pode ser utilizada por mais de uma aplicação-cliente ao mesmo tempo, então, o processo de negócio deve ser implementado como componente *Enterprise Javabeans* do tipo-sessão com estado. Por exemplo, o processo de negócio Receber Pedido, é único, ou seja, cada instância desse processo, refere-se a um pedido que está sendo realizado por um determinado cliente e que contém determinados produtos. A aplicação–cliente responsável por esse processo não pode dividir, com outra aplicação-cliente, a mesma instância desse processo.

Um diagrama de classes pode ser utilizado para representar o objeto de negócio do tipo processo como componente *Enterprise Javabeans* do tipo-sessão. Esse diagrama mostra a classe que representa o componente e a relação com os processos de negócio externos, quando utilizados. Se o componente *Enterprise Javabeans* for do tipo-sessão com estado, recomenda-se a utilização do *stereotype* `<<Stateful Session Bean>>`. Caso o componente *Enterprise Javabeans* seja do tipo-sessão sem estado então, no modelo de componentes *Enterprise Javabeans*, o *stereotype* `<<Stateless Session Bean>>` deve ser utilizado. A Figura 5.7 mostra um exemplo do diagrama que representa o componente *Enterprise Javabeans* do tipo-sessão.

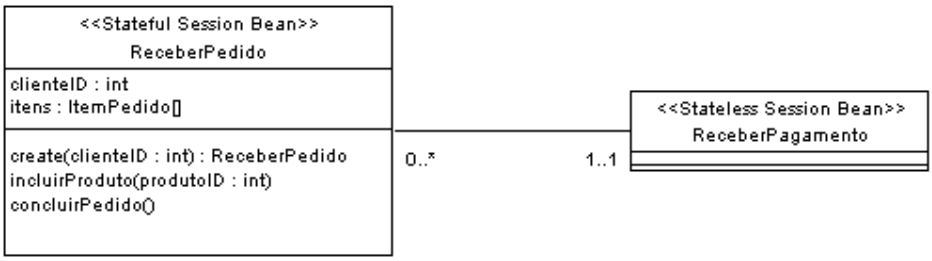


Figura 5.7 - Modelo de componente Enterprise Javabeans do tipo-sessão.

O roteiro sugere que, no projeto de componentes *Enterprise Javabeans* do tipo-sessão, seja criada uma matriz CRUD (Create, Retrive, Update, Delete) listando todos os objetos do tipo-entidade que participam do processo de negócio. A Figura 5.8 mostra um exemplo de matriz CRUD para o processo de negócio Receber Pedido.

	CREATE	RETRIVE	UPDATE	DELETE
Cliente		X		
Produto		X		
ItemPedido	X			
Pedido	X			

Figura 5.8 - Matriz CRUD.

O roteiro define, como atividade de caráter opcional, a construção de uma lista contendo as operações de objetos do tipo-entidade chamadas durante o processo de negócio.

5.2.3 – Implementação

As atividades realizadas nesta etapa são: implementação da interface `remote`, da interface `home`, dos objetos dependentes e da classe do componente *Enterprise Javabeans* Todos esses elementos, que constituem o modelo de componentes *Enterprise Javabeans*, foram apresentados no Capítulo 4. Essas atividades consistem na implementação, utilizando a linguagem Java, dos modelos definidos na etapa de projeto de componente.

5.2.3.1 – Implementação da *interface remote*

A implementação da *interface remote* é realizada com base no diagrama de classes que representa o modelo de objetos de negócio. Cada operação definida na classe que representa o objeto de negócio, deve ser incluída na *interface remote*.

O nome da *interface remote* deve ser o mesmo nome utilizado para representar o objeto de negócio no digrama de classes (Cliente, por exemplo).

5.2.3.2 – Implementação da *interface home*

A interface `home` deve ser implementada com base no modelo de objetos *Enterprise Javabeans*. As operações de criação (`create`) e as operações de busca (`findBy`) devem ser definidas nesta interface. Os componentes *Enterprise Javabeans* do tipo-sessão, que representam os processos de negócio, não possuem operações de busca. Nesse caso, a *interface home* para esses componentes, possui, apenas, as operações de criação. O nome da interface `home` deve ser o nome do objeto de negócio mais o sufixo `Home(ClienteHome`, por exemplo).

5.2.3.3 – Implementação dos objetos dependentes

Os objetos definidos, no modelo de objetos *Enterprise Javabeans* como dependentes, devem ser implementados. Estes objetos devem implementar a interface `Serialiazable.`

5.2.3.4 - Implementação da classe do componente *Enterprise Javabeans*

As classes de objetos *Enterprise Javabeans* são implementadas de acordo com o tipo do componente (entidade ou sessão) e conforme o tipo de gerenciamento de persistência (apenas para componentes do tipo entidade) adotado.

Os componentes do tipo-entidade, que fazem parte do modelo de objetos *Enterprise Javabeans,* devem ser implementados como classes que implementam a interface `EntityBean`. Se o gerenciamento da persistência for de responsabilidade do componente, então, as operações de busca (`findBy`) e as operações de acesso a dados (`db`)devem ser implementadas na classe do componente. Caso o componente tenha uma classe como *primary-key,* então, a implementação dessa classe deve ser feita. Se a responsabilidade do gerenciamento de persistência do objeto de negócio for atribuída ao *container Enterprise Javabeans,* as funções de busca não são implementadas na classe do componente. Nesse caso, ferramentas oferecidas pelos *containers Enterprise Javabeans*, podem ser utilizadas para definir os critérios de busca.

Os componentes *Enterprise Javabeans* do tipo- sessão implementam a *interface* `SessionBean`. As classes desses componentes devem definir as operações identificadas no modelo de objetos *Enterprise Javabeans* do tipo-sessão.

O nome da classe do componente *Enterprise Javabeans* deve ser o nome do objeto de negócio mais o sufixo `EJB` (`ClienteEJB`, por exemplo).

5.2.4 – Instalação de componentes

Nesta etapa, os componentes são instalados no *container Enterprise Javabeans*. O *deployment descriptor* é o arquivo de configuração, em formato XML, dos componentes *Enterprise Javabeans.* Neste arquivo, fica definido como o objeto de negócio, representado pelo componente *Enterprise Javabeans*, atua em um ambiente distribuído. Os mecanismos de segurança e as formas de controle de transação que o componente deve utilizar, são definidos nesta etapa. Alguns fornecedores de servidores *Enterprise*

Javabeans, oferecem ferramentas visuais que podem ser utilizadas para facilitar a configuração das características de tempo de execução dos componentes. A Figura 5.9 mostra um exemplo de ferramenta visual que pode ser utilizado na configuração das características dos objetos de negócio.

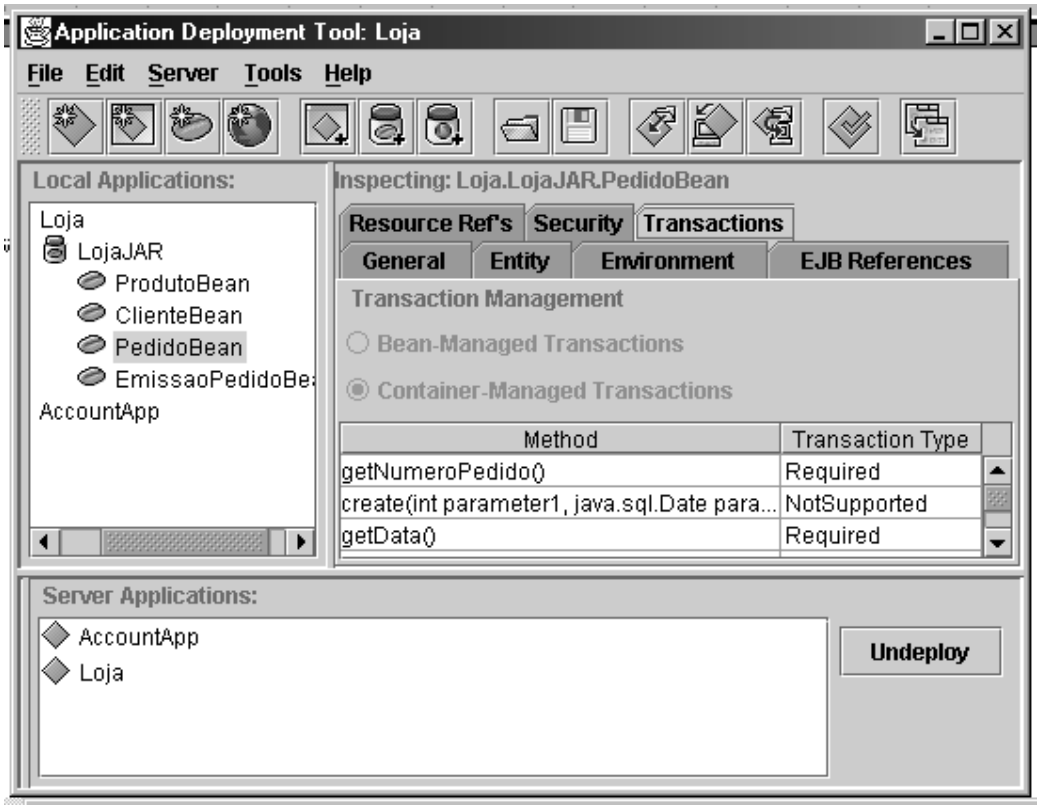


Figura 5.9 - Utilização de ferramenta visual na instalação de componente.

O gerenciamento da persistência do objeto de negócio pode ser feito, de forma automática, pelo container. Para isso ser possível, na instalação, o componente do tipo entidade, deve possuir, no seu item de configuração `Persistence Management`, a indicação `Container Managed Persistence.`

As operações de busca podem ser criadas, automaticamente, pelo *container*. Os critérios de pesquisa devem ser definidos, pelo desenvolvedor, nesta etapa de desenvolvimento. A Figura 5.9 mostra a definição do critério de pesquisa de uma operação de busca, utilizando uma ferramenta visual.

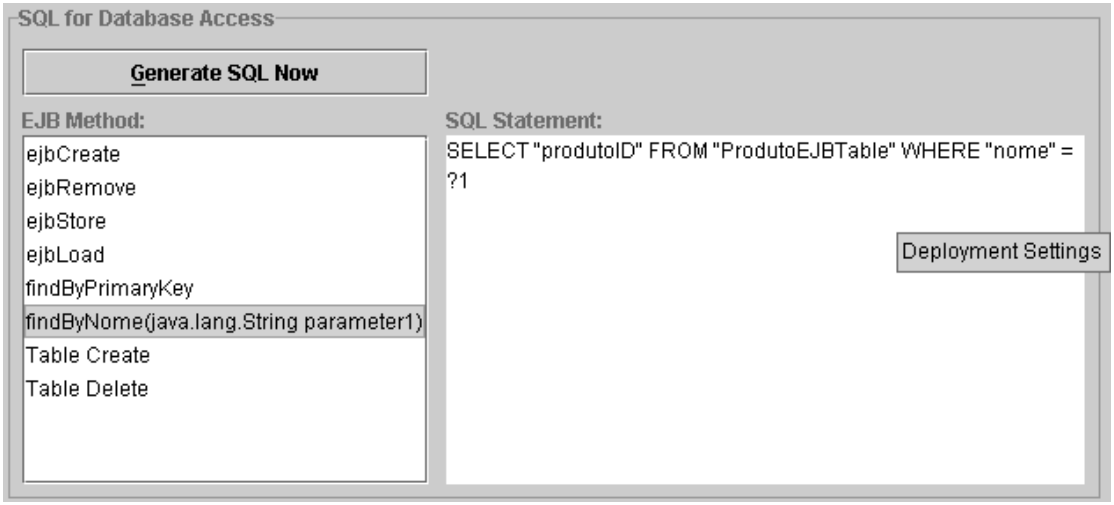


Figura 5.9- Definição dos critérios de pesquisa em uma operação de busca.

Cada componente, ou conjunto de componentes, fica, juntamente com o seu *deployment descriptor*, em um arquivo JAR. Esse arquivo é instalado no *container Enterprise Javabeans*. Depois de instalado, o funcionamento do componente pode ser testado, utilizando, para isso, aplicações-clientes simples (próprias para testes), para que, posteriormente, o componente possa ser efetivamente utilizado.

6 - APLICAÇÃO DO ROTEIRO

Um estudo de caso foi realizado para avaliação do roteiro desenvolvido neste trabalho. Neste estudo, foram aplicadas as técnicas propostas no roteiro de desenvolvimento de componentes.

A empresa a partir da qual o estudo de caso foi realizado atua no mercado de sistemas de informação voltados para a área de telecomunicações. Trata-se da *Security Technology Solutions Co*., situada em Florianópolis, Santa Catarina, Brasil.

6.1 – Aplicação

O estudo de caso foi feito de acordo com as informações fornecidas pela diretoria da empresa. Foram realizadas entrevistas com o propósito de identificar os principais processos de negócio no nível gerencial. O estudo de caso se ateve, apenas, às atividades relacionadas à gerência da empresa. Processos de produção não foram levados em consideração neste estudo de caso. Todas as etapas propostas no roteiro e suas respectivas atividades foram realizadas.

6.1.1 – Modelagem de negócio

Os principais processos de negócio (nível gerencial) existentes na empresa foram identificados nessa etapa. A sequência do trabalho se deu a partir do estudo de um processo de negócio único. O resultado desse estudo foi a obtenção do modelo conceitual, que mostra os objetos envolvidos no processo, e a obtenção do modelo dinâmico.

6.1.1.1 – Identificação dos processos de negócio

Os seguintes processos de negócio foram identificados neste estudo de caso:

- Ler e responder correspondências (escritas e eletrônicas).
- Realizar reuniões com equipes comercial, jurídica, contábil, de programação, de pesquisa científica e de incubação.
- Elaborar materiais impressos (notas fiscais, recibos e outros formulários).
- Acompanhar o trabalho realizado pelos programadores (testes).
- Elaborar e revisar contratos (contrato de venda, contrato com equipe comercial, contrato jurídico e contrato com o contador).
- Definir estratégia da empresa.

Neste estudo de caso, o processo “realizar reunião” foi utilizado para aplicação do roteiro desenvolvido. A Figura 6.1 mostra o diagrama de caso de uso para esse processo.

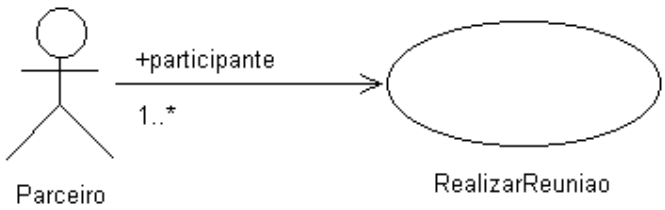


Figura 6.1 – caso de uso para o processo “realizar reunião”.

A entidade denominada “Parceiro”, ator identificado no diagrama de caso de uso, representa uma entidade que participa do processo “realizar reunião” (membro da equipe de programação ou membro da equipe comercial, por exemplo).

6.1.1.2 – Construção do modelo conceitual

As entidades obtidas a partir da análise do processo de negócio “realizar reunião” fazem parte do modelo conceitual desse processo. As entidades representam objetos de interesse dentro do processo. A Figura 6.2 mostra o modelo conceitual para o processo “realizar reunião”.

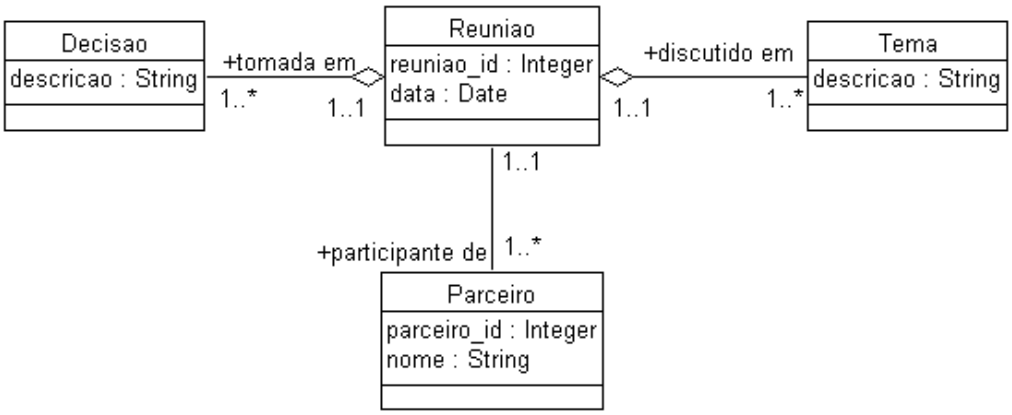


Figura 6.2 – Modelo conceitual para o processo “realizar reunião”.

Neste estudo de caso, foram identificadas as seguintes entidades de interesse para o processo “realizar reunião”: Reunião, Parceiro, Tema e Decisão. Reunião é a entidade central do processo e tem relação com as outras entidades identificadas no modelo. O relacionamento existente entre Reunião e Parceiro é o de associação com cardinalidade um para muitos. Nesse caso, esse tipo de relacionamento é adequado, pois Parceiro é uma entidade que também participa de outros processos. Já os relacionamentos existentes entre Reunião e Tema e Reunião e Decisão são do tipo agregação, pois Tema e Decisão fazem sentido apenas quando utilizados em conjunto com o objeto de negócio Reunião.

6.1.1.3 – Definição das interações dos objetos de negócio

A Figura 6.3 mostra o diagrama de sequência utilizado como ferramenta para identificação das interações dos objetos de negócio do processo “realizar reunião”.

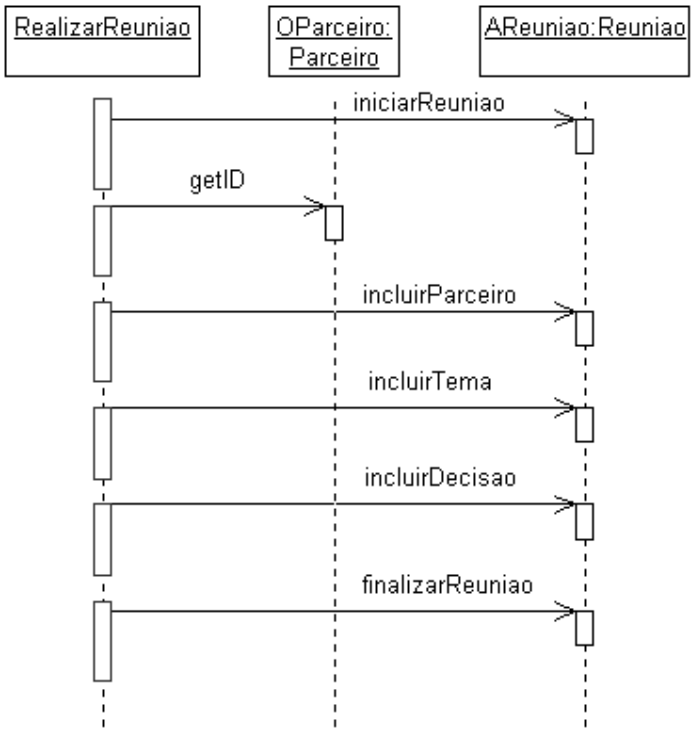


Figura 6.3 – Diagrama de seqüência para o processo “realizar reunião”.

Os objetos OParceiro e AReunião são instâncias das entidades Parceiro e Reunião identificadas no modelo conceitual. O diagrama da Figura 6.3 mostra as mensagens enviadas aos objetos participantes do processo. Esse diagrama mostra como o processo acontece. Uma mensagem é enviada à entidade Reunião para iniciar uma

reunião. Após a obtenção do identificador do Parceiro, este é incluído na reunião. Em seguida os temas e decisões são incluídos e, quando apropriado, o registro da reunião pode ser finalizado.

6.1.2 – Projeto de componentes

Os modelo conceitual, produzido na etapa de modelagem de negócio, foi ampliado nesta etapa, comportando operações e detalhes pertinentes à implementação dos componentes. As atividades realizadas nesta etapa foram construção do modelo de objetos, projeto de objetos *Enterprise Javabeans* do tipo-entidade e projeto de componentes *Enterprise Javabeans* do tipo-sessão.

6.1.2.1 - Construção do modelo de objetos de negócio

Nesta atividade as operações foram incuídas nas entidades do modelo conceitual. As operações incluídas nas entidades Reunião e Parceiro são as operações de negócio desses objetos. Essas operações, na implementação do objeto como componente *Enterprise Javabeans*, formaram a *interface remote* do objeto. Essa *interface* define as operações de negócio de um objeto distribuído. Neste estudo de caso, definiu-se, basicamente, uma estrutura simples para os objetos, sendo que esses possuem, na sua definição, apenas operações de negócio que retornam informações (as funções *get).*

A Figura 6.4 mostra o novo modelo desenvolvido: o modelo de objetos de negócio.

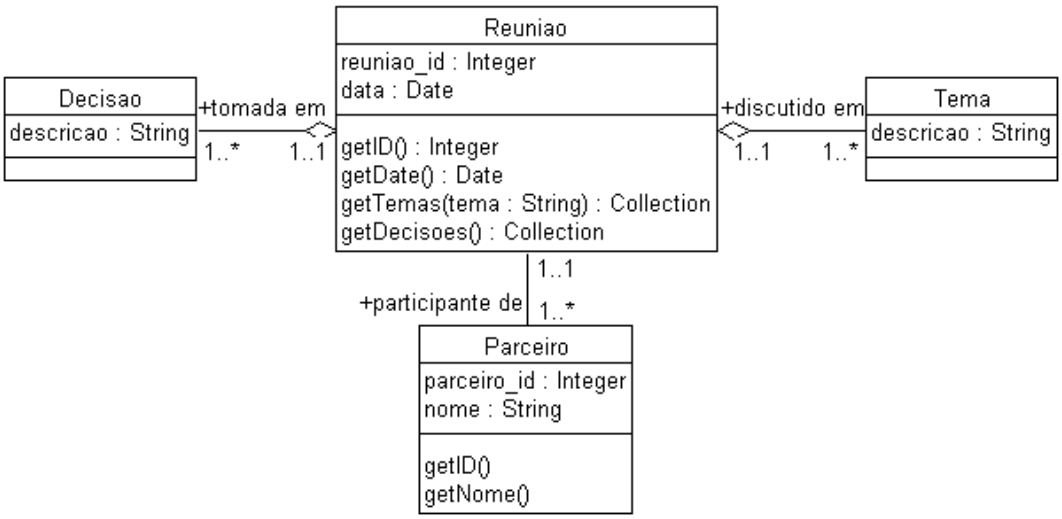


Figura 6.4 – Modelo de objetos de negócio.

Os objetos Decisão e Tema não possuem operações. Desse modo, tornou-se necessário no momento da implementação, que os atributos dessas entidades fossem definidos como públicos.

6.1.2.2 – Projeto de componentes *Enterprise Javabeans* do tipo entidade

Foram definidas, nesta atividade, as operações de criação e busca e, também, detalhes em relação à persistência dos objetos. Os *stereotypes* foram utilizados para classificar as instâncias das entidades Tema e Decisão como dependentes. Os *stereotypes* também definiram o tipo de gerenciamento da persistência dos objetos. A entidade Parceiro tem a persistência gerenciada pelo *container* enquanto a entidade Reunião possui funções para gerenciamento da sua persistência. No primeiro caso, é possível se fazer o mapeamento da entidade, diretamente, para uma tabela de banco de dados relacional, enquanto no segundo caso isso não é possível, pois a entidade possui

relacionamentos de agregação, sendo assim, necessária a utlilização de mais de uma tabela de banco de dados. Os *stereotypes* foram utilizados, também, para indicar o atributo *primary-key* das entidades que foram mapeadas para componentes *Enterprise Javabeans.*

As operações para criação e busca também foram incluídas no novo modelo. Neste estudo de caso, as operações de criação (operações *Create)* atribuem os valores passados como argumentos aos atributos privados. As operações de busca (operações cujo nome possui o prefixo *findBy)* foram definidas nas entidades Parceiro e Reunião. As operações de acesso a dados foram incluídas na definição da entidade Reunião (operações cujo nome possui o prefixo *db)*, pois essa entidade gerencia a sua persistência. A Figura 6.5 mostra o modelo de objetos *Enterprise Javabeans.*

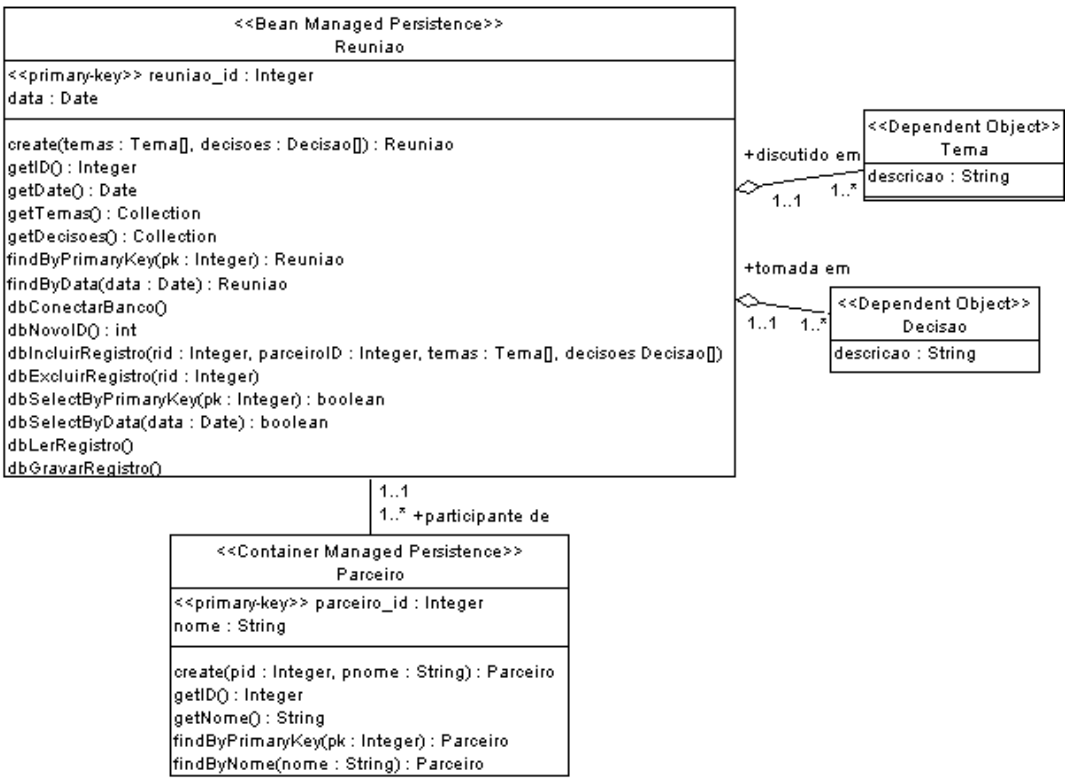


Figura 6.5 – Modelo de objetos *Enterprise Javabeans* do processo “realizar reunião”.

O modelo desenvolvido nesta atividade serve de base para implementação das classes dos componentes *Enterprise Javabeans*, para a definição das *interfaces home* e *remote*, e para a implementação dos objetos dependentes.

6.1.2.3 – Projeto de objetos *Enterprise Javabeans* do tipo-sessão

Nesta etapa, diagrama de classes que mostra a definição do objeto *Enterprise Javabeans* do tipo-sessão foi construído. A Figura 6.6 mostra esse diagrama.

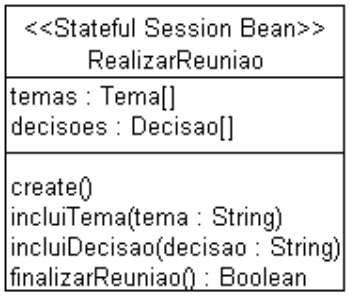


Figura 6.6 – Modelo de objeto *Enterprise Javabeans* do tipo sessão do processo "realizar reunião".

O processo "realizar reunião" foi modelado como componente *Enterprise Javabeans* do tipo-sessão com estado. Isto significa que, para cada reunião, haverá uma instância do componente *Enterprise Javabeans*, não podendo essa instância representar mais de uma reunião. Neste estudo de caso, foram identificadas as operações para inclusão de temas e decisões para a reunião. A função que finaliza a reunião confirma o registro de todas as informações no banco de dados.

6.1.3 – Implementação

Nesta etapa, inicialmente, as *interfaces remote* e *home* foram definidas para os objetos do modelo *Enterprise Javabeans* do tipo-entidade e do tipo-sessão. Os objetos definidos como dependentes (pelo *stereotype* <<Dependent Object>>) não possuem *interfaces remote* e *home*, pois eles não são mapeados para componentes *Enterprise Javabeans.* A Listagem 6.1 mostra a definição da *interface remote* para o objeto Reunião.

```
import java.rmi.RemoteException;
import java.sql.Date;
import javax.ejb.EJBObject;
import java.util.Collection;

public interface Reuniao extends EJBObject
{
   public int getID() throws RemoteException;
   public Date getData() throws RemoteException;
   public Collection getTemas() throws RemoteException;
   public Collection getDecisoes() throws RemoteException;
}
```

Listagem 6.1 – *interface remote* do objeto Reunião.

A definição dessa *interface* pode ser obtida, diretamente, a partir do modelo de objetos, desenvolvido na etapa de projeto de componentes.

A Listagem 6.2 mostra a definição da *interface home* para o objeto Reunião.

```
import java.rmi.RemoteException;
import javax.ejb.CreateException;
import javax.ejb.FinderException;
import javax.ejb.EJBHome;
import java.util.Collection;

public interface ReuniaoHome extends EJBHome
{
   public Reuniao create() throws
      CreateException, RemoteException;
   public Reuniao findByPrimaryKey
   (Integer pk) throws
     FinderException, RemoteException;
   public Collection findByData(Date d) throws
      FinderException, RemoteException;
}
```

Listagem 6.2 – *interface home* para o objeto Reunião.

A *interface home* foi escrita de acordo com a definição do objeto no modelo de objetos *Enterprise Javabeans* do tipo-entidade ou do tipo-sessão. O objeto Reunião possui uma função `Create` que inicializa os atributos com os valores passados como argumentos, e duas funções de busca, uma que utiliza a *primary-key* do objeto para busca e outra que utiliza a data como critério de pesquisa.

Após a definição das *interfaces home* e *remote*, as classes dos objetos foram implementadas. A definição da classe do objeto de negócio Reunião poder ser vista na Listagem 6.3.

```
import java.sql.*;
import javax.sql.*;
import java.util.*;
import javax.ejb.*;
import javax.naming.*;

public class ReuniaoEJB implements EntityBean
{
   private int reuniaoID;
   private java.sql.Date data;
   private Tema[] temas;
   private Decisao[] decisoes;

   private EntityContext context;
   private Connection con;
   private String db = "java:comp/env/jdbc/STSDB";

   //métodos de negócio
   public int getIDo() {}

   public java.sql.Date getData() {}
   public Collection getTemas() {}
   public Collection getDecisoes() {}
```

```
public Integer ejbCreate() throws CreateException {}
public Integer ejbFindByPrimaryKey(Integer pk)
     throws FinderException {}
public Collection ejbFindByData(java.sql.Date d)
    throws FinderException {}
public void ejbRemove() {}
public void setEntityContext(EntityContext ctx) {}
public void unsetEntityContext() {}
public void ejbActivate() {}
public void ejbPassivate() {}
public void ejbLoad() {}
public void ejbStore() {}
public void ejbPostCreate() {}

// rotinas de banco de dados
private void dbConectarBanco()
   throws NamingException,SQLException {}
private int dbBuscarProximoUid() throws
   SQLException {}
private void dbIncluirRegistro(
   int id, java.sql.Date d,Tema[] temas,
   Decisao[] decisoes) throws SQLException {}
private void dbExcluirRegistro(int id)
 throws SQLException {}
private boolean dbSelectByPrimaryKey
  (int pk) throws SQLException {}
private Collection dbSelectByData
```

```
        (java.sql.Date d) throws SQLException {}
    private void dbLerRegistro() throws SQLException {}
    private void dbGravarRegistro() throws
      SQLException {}
}
```

Listagem 6.3 – Classe do componente Reunião.

A Listagem 6.3 mostra, apenas, a definição das operações da classe do componente *Enterprise Javabeans* Reunião. Esta classe possui as operações definidas na *interface remote*, as operações definidas na *interface home* (é acrescido o prefixo *ejb* ao nome dessas operações), as operações de acesso a dados e as operações definidas pelo modelo de componentes *Enterprise Javabeans.*

6.1.4 – Instalação dos componentes

Os componentes desenvolvidos para o processo de negócio “realizar reunião” foram instalados em um *container Enterprise Javabeans.* Algumas configurações foram realizadas no momento da instalação como a definição do gerenciamento de transação e o gerenciamento da persistência . A estrutura de banco de dados para o objeto Parceiro foi definida, também, no momento da instalação dos componentes. O produto utilizado como *container* e servidor *Enterprise Javabeans* foi o servidor de referência que faz parte do pacote *Java Enterprise Edition 1.2* oferecido pela Sun Microsystems Co.

6.2 – Avaliação do resultado

A utilização do roteiro proposto neste trabalho, contribui para que o processo de desenvolvimento de componentes seja realizado de forma organizada. Isso se deve à divisão do processo em etapas e à identificação das atividades de cada uma. O estudo de caso mostrou como o roteiro dever ser aplicado – as etapas e atividades, propostas no roteiro, devem ser realizadas para cada processo de negócio.

O processo de negócio “realizar reunião” foi o processo implementado neste estudo de caso. A empresa *Security Technology Solutions Co*. possui outros processos (identificados na etapa de modelagem de negócio) que podem ser implementados da mesma forma como foi o processo “realizar reunião”.

Os componentes *Enterprise Javabeans* desenvolvidos podem ser compartilhados por várias aplicações em diversos ambientes como *internet, intranet* e outros. Como o padrão *Enterprise Javabeans* é compatível com o padrão CORBA as aplicações-cliente podem ser escritas em qualquer linguagem. Além de oferecer essas vantagens, a adoção de componentes *Enterprise Javabeans* permite a representação dos objetos e processos de negócio como componentes de aplicação.

7 – CONCLUSÕES E SUGESTÕES PARA TRABALHOS FUTUROS

Após o estudo realizado sobre as tecnologias envolvidas no desenvolvimento de aplicações distribuídas e, mais precisamente, sobre o suporte que o modelo de componentes *Enterprise Javabeans* oferece a implementação de objetos de negócio, algumas conclusões, sobre a viabilidade de utilização dessas tecnologias e sobre o método desenvolvido neste trabalho, podem ser observadas, assim como, podem ser feitas sugestões para trabalhos que venham a ser desenvolvidos no futuro.

7.1 – Conclusões

A representação dos conceitos de negócio, por meio de objetos, é uma técnica que pode ser utilizada na modelagem de negócio e na implementação em forma de componentes de negócio.

O modelo de componentes *Enterprise Javabeans* oferece uma infra-estrutura que permite a implementação dos objetos de negócio. Essa infra-estrutura é composta por dois elementos fundamentais: o servidor e o *container Enterprise Javabeans.* O servidor oferece os serviços necessários para que, o objeto de negócio implementado, possa atuar em um ambiente distribuído. O *container* atua como um agente facilitador na integração

dos objetos de negócio implementados (componentes) e o servidor. Essa flexibilidade, permite que, o desenvolvedor do objeto de negócio, se preocupe, apenas, com a sua tarefa principal – implementar o conceito de negócio como um componente reutilizável.

A divisão do processo em etapas, proposta pelo roteiro, é fundamental na organização de atividades relacionadas no processo. A execução de cada atividade sugerida pelo roteiro, representa um passo em direção à obtenção de componentes que representam adequadamente o conceito de negócio.

Os objetivos, apontados no início do trabalho, foram atingidos:

- Algumas técnicas como utilização de *stereotypes* e nomenclaturas especiais no projeto de componentes, desenvolvidas neste trabalho, podem ser utilizadas por desenvolvedores.
- O roteiro desenvolvido mostra um forma de dividir processo de construção de componentes em etapas.
- O roteiro pode ser utilizado, pelos desenvolvedores, como um manual de desenvolvimento de componentes *Enterprise Javabeans*.
- O roteiro desenvolvido permite, quando utilizado, uma maior organização no processo de desenvolvimento de componentes e facilita a execução de atividades relacionadas à gerência do processo.
- O trabalho possibilitou uma integração de tecnologias de objetos distribuídos no processo de desenvolvimento de componentes.
- O modelo de componentes *Enterprise Javabeans* foi utilizado e avaliado no desenvolvimento desse trabalho, assim como os serviços necessários para a construção de componentes.

7.2 – Sugestões para trabalhos futuros

O propósito deste trabalho foi a elaboração de um roteiro para o processo de desenvolvimento de objetos de negócio. O roteiro não foi desenvolvido para o processo de construção de aplicações distribuídas baseadas em componentes. A elaboração de tal método é uma das sugestões para futuros trabalhos. Outros trabalhos, complementares a este, podem ser realizados, como:

- Elaboração de um método para o desenvolvimento de aplicações distribuídas, baseadas em componentes *Enterprise Javabeans.*
- Utilização de padrões de análise, projeto e implementação no desenvolvimento de componentes *Enterprise Javabeans*, de forma integrada ao método de desenvolvimento de objetos de negócio.
- Criação de estratégias para o gerenciamento do processo de desenvolvimento de objetos de negócio.
- Desenvolvimento de um método para aplicação em testes e depuração de objetos em ambientes distribuídos.
- Implementação de um *framework* de objetos de negócio baseados em componentes *Enterprise Javabeans*.
- Criação de uma ferramenta CASE que suporte o método de desenvolvimento de objetos de negócio.

8. REFERÊNCIAS BIBLIOGRÁFICAS

ANDERSEN COMPUTING. Understanding Components. Andersen Computing, 1998.

BOOCH, Grady. Object Analysis And Design. Addison-Wesley, 1991.

COUROULIS, George; DOLLIMORE, Jean; KINDBERG, Tim. Distributed Systems: Concepts And Design. Addison-Wesley, 1994.

EELES, Peter; SIMS Oliver. Building Business Objects. John Wiley, 1998.

JACOBSON, Ivar; CHRISTERSON, Magnus; JONSSON, Patrick et al. Object Oriented Software Engeenering. Addison-Wesley, 1992.

JACOBSON, Ivar; ERICSSON, Maria; JACOBSON, Agneta. The Object Advantage: Business Process Reegineering With Object Technology. Addison-Wesley, 1995.

LARMAN, Craig. Applying UML And Patterns: An Introduction to Object-Oriented Analysis and Design. Prentice-Hall, 1997.

MICROSOFT CORPORATION. Component Object Model. Microsoft Corporation, 1995.

MONSON-HAEFEL, Richard. Enterprise Javabeans, Second Edition. O'Reilly, 2000.

OMG. Business Objects Task Force. OMG, 1995.

OMG. Business Objects DTF: Common Business Objects. OMG, 1997.

OMG. Object Management Architecture. OMG, 1993.

OMG. Security Service, Version 1.0 . OMG, 1997.

OMG. Transaction Service, Version 1.1. OMG, 1997.

OMG. Naming Service, Version 1.0. OMG, 1997.

OMG. Persistence Service, Version 1.0 . OMG, 1997.

OMG. The UML Specification, Version 1.1. OMG, 1997.

ORFALI, Robert; HARKEY, Dan. Client-Server Programming With Java And CORBA. John Wiley, 1998.

ORFALI, Robert; HARKEY, Dan. The Essencial Distributed Objects Survival Guide. John Wiley, 1996.

SHELTON, Robert. Business Objects. Nikkey Computer, Novembro de 1995.

SUN MICROSYSTEMS. Enterprise Javabeans Specification, Version 1.0. Sun Microsystems, 1997.

SUN MICROSYSTEMS. Enterprise Javabeans Specification, Version 1.1. Sun Microsystems, 1999.

SUN MICROSYSTEMS. Java Naming And Directory Interface**.** Sun Microsystems, 1998.

TAYLOR, David. Business Engineering With Object Technology**.** John Wiley, 1995.

VOGEL, Andreas; RANGARAO, Madhavan. Programming With Enterprise Javabeans, OTS, And JTS. John Wiley, 1998.

YOURDON, Edward. Análise Estruturada Moderna. Editora Campus, 1990.