WANDERSON FERREIRA LEÃO

DESENVOLVIMENTO DE UM COMPONENTE PARA O DOMÍNIO DE GESTÃO DE RECURSOS DE NEGÓCIOS UTILIZANDO O PADRÃO JAVABEANS:
COMPONENTE COMPRA _VENDA

Trabalho de conclusão de curso apresentado ao Curso de Ciência da Computação.

UNIVERSIDADE PRESIDENTE ANTÔNIO CARLOS

Orientador: Prof. Ms. Frederico de Miranda Coelho

BARBACENA
2004

WANDERSON FERREIRA LEÃO

DESENVOLVIMENTO DE UM COMPONENTE PARA O DOMÍNIO DE GESTÃO DE RECURSOS DE NEGÓCIOS UTILIZANDO O PADRÃO JAVABEANS:
COMPONENTE COMPRA _VENDA

Este trabalho de conclusão de curso foi julgado adequado à obtenção do grau em Ciência da Computação e aprovado em sua forma final pelo Curso de Ciência da Computação da Universidade Presidente Antônio Carlos.

Barbacena – MG, 22 de Junho de 2004

_____

Prof. Ms. Frederico de Miranda Coelho

_____

Prof. Ms. Elio Lovisi Filho

_____

Prof. Ms. Eduardo Macedo Bhering

AGRADECIMENTOS

Aos Professores, Mestres e Doutores que se puseram de prontidão para passar seus conhecimentos e experiências profissionais.

Os meus sinceros agradecimentos ao Mestre e orientador Frederico de Miranda Coelho pela disponibilidade, atenção e boa vontade em compartilhar os seus conhecimentos e experiências. Foi o principal responsável pela publicação desse trabalho de conclusão de curso, sem seu apoio não teria chegado até aqui.

Aos amigos e companheiros de sala pela convivência. Aprendi muito, principalmente que não somos nada se não conhecemos mingúem.

Aos meus pais, Raimundo Leão e Maria Irene Leão, pelo amor implícito em seus olhos, pela preocupação e orações.

A todos meus irmão e sobrinhos, pelo carinho e incentivo dado a todo o momento.

A Mirelle pelo amor, companheirismo e amizade. Mesmo distante se fez presente em todos os momentos, juntos conquistamos o nosso diploma.

A todos os amigos feitos nesta universidade, companheiros de viagem, motoristas, funcionários e diretores.

Ao Bom Jesus por ter me iluminado durante esses 4 anos de viagens pela BR 040 nunca deixou que nada de grave acontecesse. Obrigado meu Deus!! Obrigado por ter me dado a chance de chegar até aqui num novo começo.

À minha mãe, meu pai, minha noiva, as meus familiares, professores, amigos e todas as pessoas que contribuíram para o sucesso e a conclusão deste projeto. A Bom Jesus pela luz em todos os momentos difíceis.

"…cada um de nós carrega sua história cada um de nós carrega o dom ser capaz de ser faliz…
…todo mundo ama um dia, todo mundo chora
um dia a gente chega o outro vai embora…"
"Almir Sater"

RESUMO

Desenvolver Softwares com um auto nível de qualidade e num curto espaço de tempo é uma das principais preocupações da Engenhara de Softwares, com isso surgiram várias técnicas e novos paradigmas, com o objetivo de minimizar o tempo gasto no processo de desenvolvimento do Software. Técnicas como por exemplo a Programação Orientada a Objetos, Padrões, Linguagens de Padrões, Componentes de Software, *Frameworks* Baseados em Classes e *Frameworks* Baseados em Compontentes, Proramação Orientada por Componentes.

Sendo assim, este trabalho visa o desenvolvimento de um componente de software estruturada com Padrões de Projeto, de um *Framework* Orientado a Objetos, utilizando o Padrão *JavaBeans* da SUN para posteriormente ser analisado sua interoperabilidade e reusabilidade com o componente "Locar_e_Reservar_Recursos", desenvolvido no padrão COM da Microsoft. Para isso será utilizado a ferramenta *JavaBeans Bridge for ActiveX* da SUN que inclui características que permitem *JavaBeans* e componentes COM/OLE/ActiveX trocar informação em ambas direções.

O resultado desse trabalho é a afirmação positiva das vantagens de estruturar um Componente aplicando-lhe Padrões de Projeto, como a facilidade de manutenção e documentação do mesmo e as vantagens de se utilizar padrões distintos na comunicação entre componentes do mesmo domínio.

SUMÁRIO

FIGURAS

1 INTRODUÇÃO

Há várias técnicas como o paradigma Orientado a Objetos (OO), Frameworks Orientado a Objetos, Componentes de Software, Padrões e Linguagens de Padrões, Programação Orientada por Componentes que surgiram com o intuito de obter o máximo de reusabilidade dos softwares, assim também obtendo menor esforço e qualidade no seu processo de implementação, documentação e manutenção.

Embora nenhuma tecnologia tenha, até o momento, preenchido completamente estes requisitos, a nova tecnologia de componentes de softwares JavaBeans da *Sun Microsystems* é indiscutivelmente aquela que mais se aproximou desta visão (Morrison *et al.*, 1999).

1.1 A PROPOSTA

O trabalho consiste re-projetar um componente de uma Arquitetura Estruturada em Componentes, fazer a um processo de reengenharia, aplicar Padrões de Projeto em sua estrutura, para analisar e avaliar as possibilidade de tal componente se comunicar com outro componente desenvolvido em outro padrão.

Para realizar tal comunicação é proposto a utilização da ferramenta JavaBeans Bridge for ActiveX, que empacota componentes JavaBeans, tornando possível utilizá-los em ambientes ActiveX.

O Componente a ser desenvolvido será o Componente Compra_Venda da Arquitetura Estruturada em Componentes de um Framework voltado para a Gestão de Recursos de Negócios (Coelho, 2002).

1.2 OBJETIVOS

Os principais objetivos com este trabalho são:

- A comunicação entre componentes de padrões diferentes que poderá abrir caminho para difusão do Paradigma “Programação Orientada em Componentes” tendo como ponto principal desta a reusabilidade;

- Oferecer uma base para programadores que desejam desenvolver componentes no padrão *JavaBeans*, uma vez que este próprio traz documentado todas as etapas do desenvolvimento e os conceitos utilizados nessas etapas;

- Oferecer um estudo analítico sobre como os Padrões de Padrões de Projetos ajudam no desenvolvimento de um componente e quais as vantagens de se utilizar estes padrões;

1.3 ORGANIZAÇÃO DA MONOGRAFIA

O Capítulo 2 apresenta as definições das Tecnologias utilizadas neste trabalho.

O Capitulo 3 apresenta o metodologia utilizada para o desenvolvimento do Componente, proposta pro (Melo,2003) e os resultados obtidos em cada etapa desse processo, exceto.

No Capítulo 4 são descritos os detalhes da implementação de um componente modelo para detalhar a forma correta de implementar um componente *JavaBeans* e como empacotar tal componente utilizando a tecnologia *JavaBeans Bridge for ActiveX*.

No Capítulo 5, são descritas as conclusões desta monografia, suas contribuições e os planos de trabalhos futuros.

2 CONCEITOS INICIAIS

Antes de iniciar o processo de desenvolvimento do componente, que ainda será escolhido, é preciso definir alguns conceitos que serão utilizados ao longo deste trabalho e que são de fundamental importância para o seu entendimento.

2.1 ENGENHARIA DE APLICAÇÕES (EA)

Engenharia de Aplicações se dedica ao estudo das melhores técnicas, processos e métodos para a produção de aplicações no contexto do desenvolvimento baseado em reutilização (Simos,1998).

Possibilidade de utilização de um método de desenvolvimento conhecido pela organização, porém enfatizando atividades de reutilização

2.2 DESENVOLVIMENTO BASEADO EM COMPONENTES (DBC)

Um componente de software é qualquer subsistema que possa ser separado e que possui uma interface reusável e potencialmente padronizáveis (Morrison *et. al.*, 1999). As partes podem ser reutilizadas em outras aplicações. Funciona como as peças de um carro. Pode-se substituir o carburador ou colocar um turbo no motor do carro sem ser necessário levar o carro de novo para a linha de montagem.

A pesar de ser um conceito muito antigo na indústria, só agora esta sendo possível de implementar em software. A figura abaixo ilustra modelos de componentes utilizados na ciência da computação:

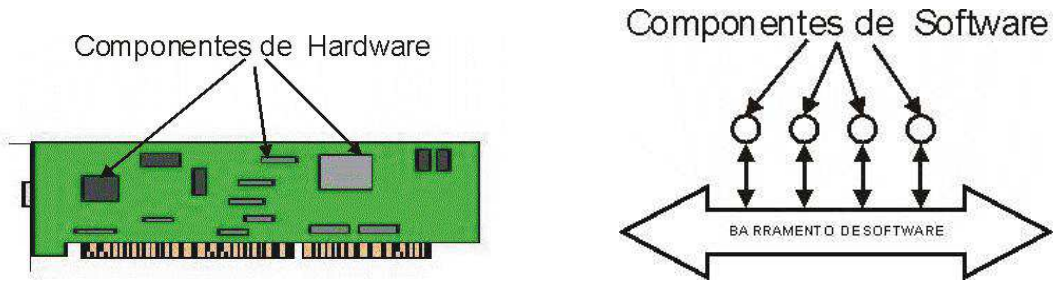


Figura 1.1 – Tipos de coponentes utilizados na ciênci da computação.

O funcionamento do hardware é todo basedo em componentes que se comunicam entre si e cada um com sua finalidade predefinida, o paradgina da programeção baseada em componentes segue o mesmo princípio.

Não é suficiente dividir uma aplicação em partes, encapsular cada parte em um componente *JavaBeans* e chamar o resultado de uma aplicação com base em componentes.

Tal tipo de aplicação terá partes identificáveis, da mesma forma como acontece em um quebra-cabeças que se encaixam apenas de uma maneira correta e útil.

Para serem reutilizáveis a mais atrativos do que escrever um novo código desde o zero os componentes devem levar ao refinamento e à reutilização em outras aplicações, eles devem atrair outros programadores que não aqueles que originalmente criaram os componentes (Morrison *et al.*, 1999).

Os componentes, segundo a literatura, são de dois tipos:

Componente físico: refere-se aos Componentes que possuem um código binário, ou seja, são módulos que realmente têm capacidade de executar determinadas tarefas e podem ser constituídos de um ou mais Componentes Lógico.

Componente Lógico: engloba os Componentes que possuem uma Interface bem definida e dependências de contexto explicitas através de suas Interfaces.

Ambos podendo ser "**Caixa Preta**" onde sua reutilização independe da implementação do Componente de Software, ou "**Caixa Branca**" onde sua reutilização depende da implementação do Componente, ou "**caixa cinza"** que é a união dos dois anteriores.

2.3 REUTILIZAÇÃO DE SOFTWARE

"Utilização de produtos de software, construídos ao longo do processo de desenvolvimento, em uma situação diferente daquela para a qual foram originalmente produzidos" (Freeman, 1980).

A reutilização é inerente ao processo de solução de problemas utilizado pelos seres humanos. O problema não é a falta de reutilização na Engenharia de Software, mas a falta de um padrão amplo e formal para realizá-la (Werner *et. al*., 2000).

Alguns benefícios são concebidos a partir da reutilização de software como, por exemplo, qualidade, produtividade, redução de custos/ tempo, flexibilidade. Existem também

dificuldades a se superar, como a identificação, recuperação, compreensão do código a ser reutilizado a modificação, composição e principalmente a qualidade e confiabilidade de componentes (Werner *et. al*., 2000).

2.4 ENGENHARIA DE DOMÍNIO (ED)

Para entender melhor a Engenharia de Domínios primeiro é necessário saber o sentido que é aplicado à palavra domínio. Domínio é uma coleção de problemas reais e de aplicações que compartilham características comuns (Werner *et. al*., 2000).

A Engenharia de domínio é o processo de identificar e organizar o conhecimento sobre uma classe de problemas, o domínio do problema, para suportar sua descrição e solução (Prieto, 1991).

Uma abordagem baseada em reuso deve conter um definição do escopo, especificação da estrutura, e construção de recursos para uma classe de sistemas, subsistemas ou aplicações, com o objetivo de originar meta-sistemas, ou seja, sistemas que são reutilizados na construção de aplicações específicas, deve-se descobrir e definir modelos de domínio e arquiteturas comuns às famílias de aplicações para suportar reuso pré- planejado (Prieto, 1993)

2.5 FRAMEWORK

Um *Framework* é um esqueleto de uma aplicação que pode ser particularizado por um desenvolvedor, isto é, um projeto reutilizável de um sistema que descreve o sistema decomposto em um conjunto de objetos e elementos que interagem entre si. Um *Framework* descreve a interface de cada objeto e o fluxo de controle entre eles (Fayad *et. al*., 1999).

Um *Framework* provê uma solução para uma família de problemas semelhantes, usando um conjunto de classes e interfaces que mostra como decompor a família de problemas, e como objetos dessas classes colaboram para cumprir suas

responsabilidades. O conjunto de classes deve ser flexível e extensível para permitir a construção de várias aplicações com pouco esforço, especificando apenas as particularidades de cada aplicação (Fayad *et. al*., 1999).

Existe uma diferença entre uma biblioteca de classes e um *framework* orientado a objetos. Numa biblioteca de classes, cada classe é única e independente das outras, as aplicações criam as colaborações. Em um *framework*, as dependências/colaborações estão embutidas. Já que a comunicação entre objetos já está definida, o projetista de aplicações não precisa saber quando chamar cada método: é o framework que faz isso. Na figura 2.2 vê-se, portanto que um *framework* impõe um modelo de colaboração (o resultado da análise e design) ao qual será ser utilizado neste projeto.

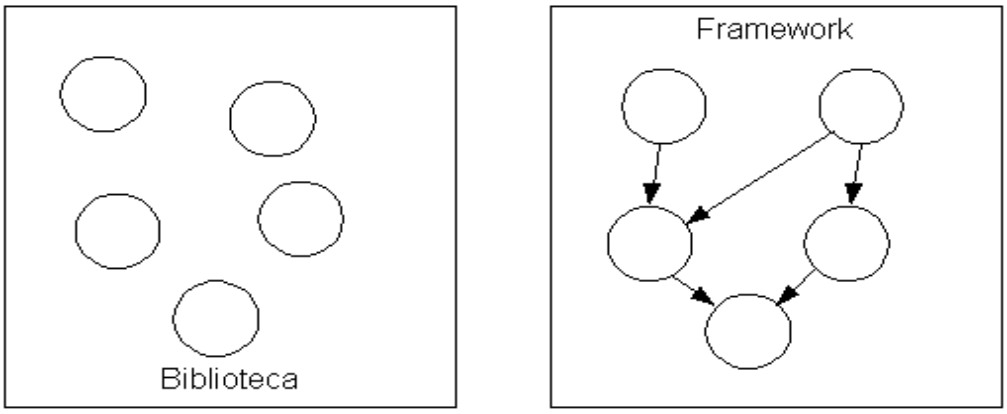


Figura 2.2 – Representação de uma Biblioteca e um Framework onde o Framework impõe um modelo de colaboração

Biblioteca: Classes instanciadas pelo cliente; Cliente chama funções; Não tem fluxo de controle predefinido; Não tem interação predefinida; Não tem comportamento *default* (Jacqueline,UFPB).

***Framework*:** Customização com subclasses ou composição; Chama dunções da "aplicação"; Controla o fluxo de execução; Define interação entre objetos; Provê comportamento *defalt* (Jacqueline,UFPB).

2.6 MODELAGEM UML

A Modelagem é de extrema importância para o desenvolvimento de um sistema de boa qualidade, pois permite visualizar toda a estrutura e comportamento do sistema, possibilitando também uma melhor compreensão do sistema construído e uma melhor análise dos riscos do sistema (Booch *et. al.*, 1999).

A UML (*Unified Modeling Language*) é uma Linguagem de Modelagem para Análise e Projeto Orientado a Objetos (Fowler *et. al.*, 2000) e será utilizada para a modelagem do componente em questão.

2.7 INTERFACES

As interfaces são um ponto chave para a construção de um componente; bem definidas elas permitem a composição entre componentes sem a necessidade de conhecer a parte interna do mesmo (Melo,2003).

As Interfaces são os serviços oferecidos pelo componente de forma que a implementação desses serviços fica oculto do usuário, possibilitando uma segurança maior, fazendo com que o usuário compreenda o seu funcionamento, e interaja com o mesmo sem dificuldades.

Elas determinam como este componente pode ser reutilizado e interconectado com outros componentes. Um componente exporta (provê, Interface Provida) serviços que outro componente importa (requer, Interface Requerida).

A figura 2.3 ilustra essa relação entre as Interfaces Providas e as Interfaces Requeridas num modelo feito em UML. O Componente 2 possui uma Interface Provida (Interface_3) e uma Requerida (Interface_2) que usa a Interface Provida (Interface_2) do Componente 1 .

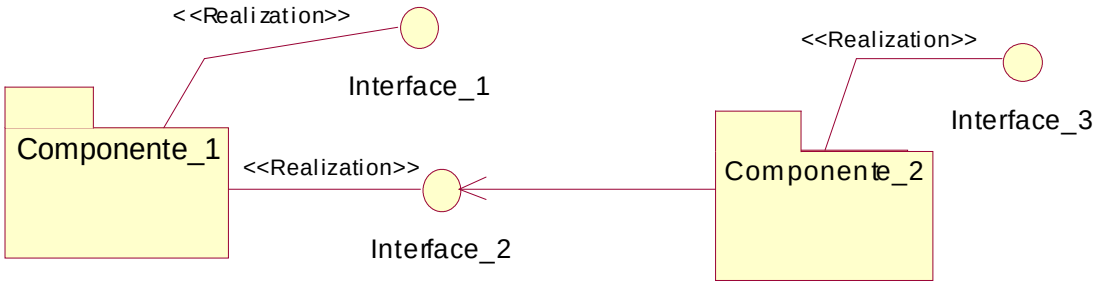


Figura 3.3 – Representação de um Componente Lógico e suas Interfaces Providas e Requeridas em UML.

2.8 PADRÕES DE PROJETO

Uma ferramenta muito importante na Engenharia de Software são os padrões de projeto que tem o objetivo principal obter-se um alto grau de reutilização de software (Gamma *et. al*., 2000).

Um Padrão fornece uma solução comum para um problema básico em um determinado contexto (Boch *et.al*., 1999).

Os Padrões são criados com base nos experimentos obtidos com diferentes problemas em um certo domínio, com o objetivo de não ter que perder tempo criando uma solução para esses problemas sempre que se defrontar com algum deles. Quando se deparar com um problema ou situação comum, basta escolher o Padrão apropriado e aplicá-lo em seu sistema e grande parte desse problema estará solucionado restando apenas algumas alterações para adaptá-lo às suas necessidades e limitações de forma que lhe for conveniente (Gamma *et. al*., 2000).

Um conjunto de Padrões que são utilizados em um domínio comum que possui um grau de abstração e de reutilização bem maior do que a soma das suas partes (Padrões), este conjunto é chamado Linguagem de Padrões (Buschmann *et. al.*, 1996).

2.9 RATIONAL ROSE

Ferramenta para Modelagem que será usada na criação de diagramas criados na etapa de Análise e Projeto do componente: como Diagramas de Classe, Diagramas de Estado e Diagramas de Seqüência, segundo a Linguagem de Modelagem UML, com o objetivo de facilitar o entendimento de sua estrutura.

2.10 JAVA

Java é uma linguagem de programação orientada a objetos desenvolvida pela *Sun Microsystems*. Modelada depois de C++, a linguagem Java foi projetada para ser pequena, simples e portável a todas as plataformas e sistemas operacionais, tanto o código fonte, como os binários, é uma linguagem que possibilita o desenvolvimento de programas seguindo uma abordagem orientada a objetos. (Deitel *et. al*., 2001)

2.10.1 AS VERSÕES DO JAVA

Em geral as versões dos diversos produtos Java seguem o padrão: versão **1.2.3a-000**, onde (JavaMan, 2004):

1 - indica a versão base. Até hoje nunca ocorreu mudança na versão base (estamos ainda no Java 1.x).

2 - indica mudanças de biblioteca. Você tem mais objetos e/ou métodos disponíveis, pode ocorrer de existirem métodos "*deprecated*" (métodos não mais recomendados).

3 - indica bug-fixes, ou seja, não ocorre mudança de bibliotecas. A partir do Java 2 a intenção é ter uma versão de *bug-fixes* a cada trimestre.

a - em alguns poucos casos surgiram versões com uma letra no final. Isso indica uma versão que foi lançada para corrigir um *bug* de segurança .

000 - a partir da versão do Java 2 (ver abaixo), a letra no final foi substituída por três números, mas no fundo, significam a mesma coisa que o "**a**" acima (JavaMan, 2004).

2.10.2 AS VERSÕES DO JAVA DEVELOPMENT KIT (JDK)

O JDK (*Java Development Ki*) é o conjunto básico de ferramenta para o desenvolvedor Java. Fazem parte do JDK ferramentas importantes como o javac (compilador), java (a máquina virtual), javadoc (gerador automático de documentação), jdb (Java debugger), javap (decompilador) e diversas outras ferramentas importantes (JavaMan, 2004).

Em geral, toda a funcionalidade do JDK já faz parte das ferramentas de desenvolvimento disponíveis no mercado (ferramentas como *NetBeans*, *JBuilder*, *Visual Age for Java*, *etc*), mas o conhecimento do JDK e de suas ferramentas é importante para qualquer desenvolvedor Java (JavaMan, 2004).

Desde o lançamento do primeiro JDK, em 1995, várias versões já surgiram, abaixo é listado as principais, e algumas das que precisam ser levadas em consideração e que foram utilizadas neste trabalho (JavaMan, 2004):

JDK 1.1 - essa versão incorporou uma grande quantidade de funcionalidades, entre elas algumas das mais importantes são: Java RMI (*Remote Method Invocation*), JavaBeans, novo modelo de eventos, JDBC (JavaMan, 2004).

JDK 1.2 - essa era para ser a próxima versão do JDK, que vinha sendo anunciada há algum tempo. Várias melhorias e a incorporação de uma grande quantidade de funcionalidades. Quando foi finalmente lançada, foi renomeada para *Java 2 Standard Editon Software Development Kit v*1.2.

Java 2 - nesse momento foi considerado que a plataforma Java tinha atingido uma maturidade e uma evolução suficiente para ser realizada uma troca de nome. O que aconteceu na verdade é que foi dado um nome para as versões da Tecnologia Java. Antes, as versões da tecnologia eram confundidas com as versões de um produto (o JDK), já que não havia outra forma de identificá-las. Portanto não só foi anunciado que aquele que seria o JDK

1.2 seria renomeado para SDK 1.2, mas também foi anunciada a divisão da plataforma em 3 plataformas básicas, de forma a permitir a inclusão de tecnologias pertinentes (JavaMan).:

Java 2 Micro Edition (J2ME) - a plataforma Java voltada para pequenos devices (telefones celulares, agendas eletrônicas, televisores, aparelhos eletrônicos em geral).

Java 2 Standard Edition (J2SE) - a plataforma voltada para aplicações cliente (equivalente ao antigo JDK). Essa é a plataforma que será encontrada nos browsers web e instalada nos sistemas operacionais que já incluírem a plataforma Java.

Java 2 Enterprise Edition (J2EE) - essa é a plataforma Java para desenvolvimento e execução de aplicações servidores. Possui todo o suporte para desenvolver aplicações robustas e escaláveis, para suportar grandes números de usuários. Os servidores de aplicação implementam essa plataforma.

Java 2 Standard Editon Software Development Kit v1.2 (J2SE SDK 1.2) - esse é o antigo JDK, que foi renomeado como SDK no Java 2. Inclui várias melhorias em relacao à versão 1.1, uma das principais é a inclusão das funcionalidades da JFC (Java Fundation Classes, a biblioteca que contém os componentes do Swing), além de inclusão do suporte a CORBA (*Common Object Request Broker Architecture*), melhorias no Java RMI, testes de compatibilidade muito superiores, estabilidade, performance, e muitas outras coisas. Esse é o ambiente básico para o desenvolvimento de aplicações Java atualmente e será utilizado para o desenvolvimento deste trabalho.

2.10.3 O JAVA RUNTIME ENVIRONMENT (JRE)

Enquanto o JDK é o ambiente voltado para os desenvolvedores, o JRE (*Java Runtime Environment*) é basicamente composto da Máquina Virtual Java (JVM) e o conjunto de bibliotecas, ou seja, tudo o que precisa para executar aplicações Java (JavaMan, 2004).

Entre os atrativos do JRE esta o Java Plugin. Desde a versão 1.2 (Java 2) que o JRE já traz embutido o *Java Plugin*, a JVM que substitui a JVM dos browsers web e permite com que mesmo versões mais antigas de browsers suportem aplicações Java 2. Ao instalar o JRE, junto será instalando o *Java Plugin*, automaticamente (JavaMan, 2004).

O Java *Plugin* fornece uma ferramenta chamada *JavaBeans Bridge for ActiveX* que será utilizada neste trabalho de conclusão de curso e será descrita a seguir.

2.11 COMPONENTES ENTERPRISE JAVABEANS

Enterprise JavaBeans é a tecnologia que define um modelo para desenvolvimento e implantação de componentes de aplicação que rodam em um servidor de aplicação, estes componentes são denominados Componentes Servidores. Os compenentes *Enterprise JavaBeans* estende o modelo *JavaBeans* para suportar componentes servidores.

A arquitetura EJB (*Enterprise JavaBeans*) é constituida por três elementos (Vinagreiro).:

EJB Server: É o ambiente que suporta a execução de aplicações EJB, gerencia a alocação de recursos (*threads*, processos, memória, pool de conexões a um Banco de Dados) e fornece um conjunto padrão de serviços de: Transações; Nomes/diretórios JNDI; Segurança (p.ex.: as classes AccessControl); Persistência JDBC.

EJB Container: Fornece uma thread/processo para execução de um componente, tem função de prover contexto, registrar no serviço de nomes, fornecer interfaces (OBJECT, HOME), criar/destruir instâncias, gerenciar transações e estados, podendo gerenciar persistência (CMP).

EJB Component: Componente servidor Java e contém a lógica negocialda aplicação.

EJB Server (*Enterprise JavaBeans Server*) fornece um container para gerenciar o componente no servidor, o cliente invoca o componente e o container aloca uma *thread* para executar o componente. O container gerencia todas as interações com o componente como pode ser visto no modelo abaixo:

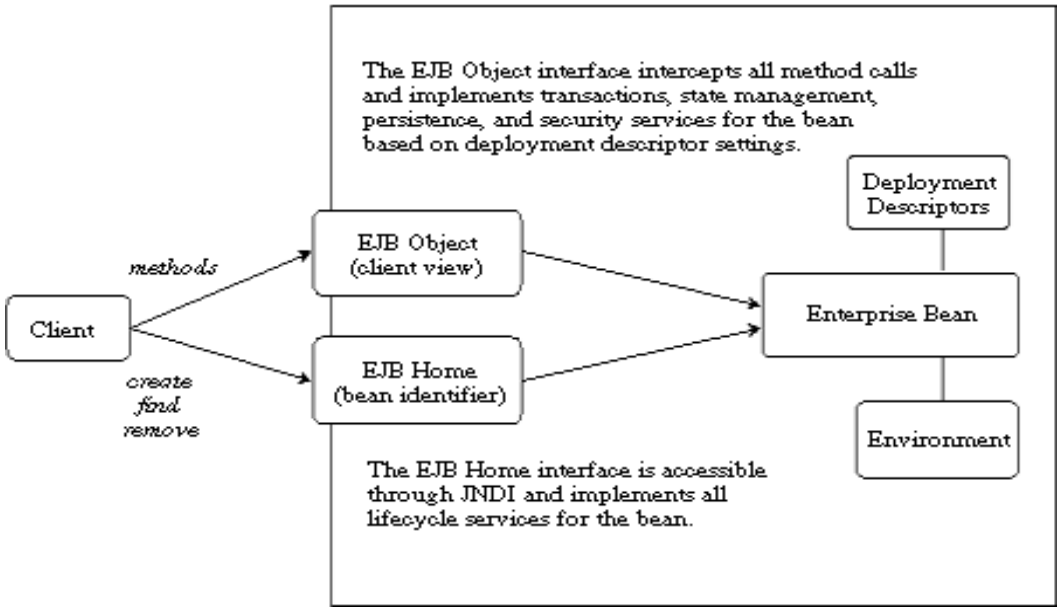


Figura 4.4 – Representação das Interfaces de um Bean e um conjunto de métodos públicos.

2.12 SISTEMAS DISTRIBUÍDOS CORBA

Sistema distribuído é aquele que roda em um conjunto de máquinas sem memória compartilhada, máquinas estas que, mesmo assim, aparecem como um único computador para seus usuários (Morrison *et. al*., 1999).

O padrão CORBA (*Common Object Request Broker)* do grupo OMG (*Object Management Group*) propõe uma arquitetura de software para suportar distribuição e garantir a interoperabilidade entre diferentes plataformas de hardware e sistemas operacionais. Resumidamente, o padrão CORBA possibilita a comunicação entre aplicações independentemente de sua localização ou ambiente (Júnior,2001).

ORB (*Object Request Broker*): É o middleware que estabelece os relacionamentos cliente-servidor entre os objetos. Utilizando um ORB, um cliente pode invocar transparentemente um método de um objeto no servidor, que pode estar na mesma máquina ou em qualquer ponto da rede.

OMA (*Object Management Architecture*): Arquitetura padrão para objetos distribuídos, define as especificações dos componentes da arquitetura de objetos distribuídos em um nível de abstração mais elevado.

CORBA (*Common Object Request Broker Architecture*): Define as especificações técnicas de uma implementação da OMA em um nível de abstração mais baixo.

Algumas vantagens e desvantagens da utilização de sistemas distribuídos CORBA (Júnior,2001):

Economia: Os microprocessadores oferecem uma melhor relação preço/performance do que a oferecida pelos mainframes.

Velocidade: Um sistema distribuído pode ter um poder de processamento maior que o de qualquer mainframe.

Distribuição inerente: Algumas aplicações envolvem máquinas separadas fisicamente.

Confiabilidade: Se uma máquina sair do ar, o sistema como um todo pode sobreviver.

Crescimento incremental: O poder computacional pode crescer em doses homeopáticas.

Software: Até o presente momento, não há muita disponibilidade de softwares para os sistemas distribuídos.

Ligação em rede: A rede pode saturar .

Segurança: Os dados secretos também são acessíveis facilmente.

2.13 COMPONENTES JAVABEANS

JavaBeans é um modelo de componente de software reutilizável programado na linguagem Java da Sun, não podendo ser classificado como um sistema, quem faz este papel é o RMI (*Remotd Methods Invocation*) e a JVM (*Java Virtual Machine*). Escreve-se um JavaBeans da mesma forma que escreveria qualquer outra classe (SUN, 2004).

O conceito de aplicações baseadas em componentes permaneceu nas mentes dos programadores durante 30 anos, desde que M. D. McIlroy (1968) fez um artigo, agora histórico para catálogos de componentes de software. As ferramentas práticas necessárias para a visão de McIlroy como bibliotecas de componentes de software agora existe, fundamentalmente com a linguagem de programação Java, mas também no JavaBeans API para implementação de componentes a nível de cliente, e o Enterprise JavaBeans para implementação de componente a nível de servidor (SUN, 2004).

O objetivo do JavaBeans APIs é definir um modelo de componente de software para Java, que podem compor juntos aplicativos para o usuário final (SUN, 2004).

A especificação de JavaBeans prescreve convenções e mecanismos de descoberta dinâmicos que minimize o desígnio e esforço de implementação de componentes de software pequenos apoiando no padrão de implementação, e montagem de componentes mais sofisticados (SUN, 2004).

Simplesmente refere-se a JavaBean como um Bean. Ferramentas construtoras, como Delphi ou Visual Basic, utilizam Beans que podem ser manipulados visualmente. Botões, formulários de entrada de dados, visualizadores de bancos de dados, receptor de mensagens em uma aplicação servidora são exemplos de *Beans* (SUN, 2004).

JavaBeans definem uma interface, esta permite ferramentas construtoras, examinar componentes para determinar os tipos de propriedades que os componentes definem e os tipos de eventos que eles geram ou para a qual eles respondem. A figura 2.3 ilustra as Interfaces de um *bean* e um grupo de métodos públicos.

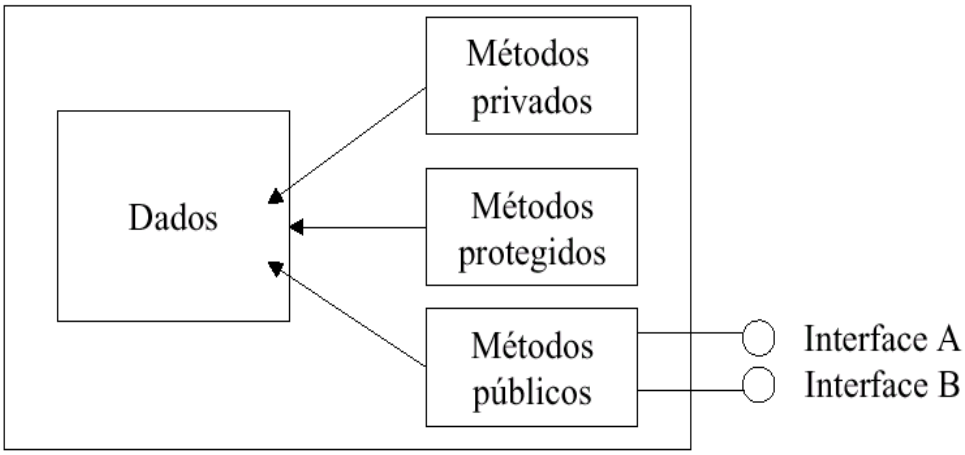


Figura 5.5 – Representação das Interfaces de um Bean e um conjunto de métodos públicos.

As suas funcionalidades são definidas não só através de atributos métodos e construtores, mas também pelas características de, Propriedades – Introspecção – Comunicação por Eventos – Customização – Persistência (Morrison *et. al*., 1999), descritas abaixo:

Introspecção: *Beans* suportam introspecção, ou permite uma ferramenta construtora analisar o trabalho de um *Beans*. Eles aderem a regras específicas chamadas Padrões de Projetos por nomear características do *Bean*. Cada *Bean* tem uma classe de informação de *Bean* relacionada, que provê propriedade, método, e informação de evento sobre o próprio *Bean*. Cada *Bean* informação implementa uma classe e uma interface de *BeanInfo* que explicitamente lista as características do *bean* que serão expostas a ferramentas de construtor de aplicação.

Propriedades: As propriedades controlam o aparecimento de um *bean* e seu comportamento. Acessando ferramentas de construtor em um *Bean* para descobrir suas propriedades e os expor para manipulação. Como resultado, pode-se mudar a propriedade de um *Bean* em um desejado momento.

Customization: As propriedades de um *Bean* podem estar customizadas em um determinado momento. A customização permite um usuário alterar o aparecimento e comportamento de um *Bean*. Os *Beans* apóiam customização usando os editores de propriedade ou usando custonizadores de *Beans* especiais.

Eventos: Os *Beans* usam eventos para se comunicar com outros *Beans*. Os *Beans* podem disparar eventos por meio de um *Bean* que envia um evento para um outro *Bean*. Quando um *Bean* dispara eventos é considerado um *Bean* de fonte. Um *Bean* pode receber um evento neste caso é considerado um *Bean* de ouvinte. Um *Bean* ouvinte registra seu interesse no evento com o *Bean* fonte. Ferramentas construtoras usam introspecção para determinar esses eventos que um *Bean* envia e recebe.

***Beans* de Persistência**: Implementam o *java.io*. É usado para economizar e restabelecer estado que pode ter mudado como resultado da customização. Por exemplo, estado é salvo quando se customiza um *Bean* em um construtor de aplicação, de forma que as propriedades mudadas podem ser restabelecidas mais tarde.

Métodos: Todos os métodos do *JavaBean* são idênticos a métodos de outras classes de Java. Métodos de *Bean* podem ser chamados através de outros *Beans* ou por linguagem de *scripting*. Um método público *JavaBean* que é exportado por padrão.

Os *Beans* são pretendidos para ser usados principalmente com ferramentas construtoras. Os *Beans* podem ser manipulados manualmente através de ferramentas de texto ou por interfaces programáveis. São projetados APIs, apoio para eventos, propriedades, e persistência, para ser lido facilmente e serem entendidos por programadores, como também através de ferramentas de construtor (Morrison *et. al*., 1999).

2.14 JAVABEANS API'S

O *JavaBeans* API inclui várias interfaces e classes no pacote de java.beans. Emprega interfaces e classes de outra tecnologia de Java API como por exemplo (SUN, 2004).

1. **Java event model**: java.util.EventObject, java.awt.event;
2. **Object serialization**: java.io.Serializable, java.io.Object;
3. **Reflection**: java.lang.reflect;

A API representa, as classes, interfaces, e objetos da atual Máquina Virtual Java. Pode-se usar a API se estiver sendo escrito ferramentas de desenvolvimento como *debuggers*, *browsers* de classe, e construtores de GUI (*Guide User Interfaces*). Com a API que pode-se:

1. Determinar a classe de um objeto,
2. Adquirir informação sobre os modificadores de uma classe, campos, métodos, constructors, e superclasses.
3. Descubrir que constantes e declarações de método pertença a uma interface,
4. Criar uma instância de uma classe cujo nome não é conhecido até runtime.
5. Usar "*Get*" e "*Set*" no valor do campo de um objeto, até mesmo se o nome de campo é desconhecido a seu programa em tempo real.
6. Invocar um método em um objeto, até mesmo se o método não é conhecido em tempo real.
7. Criar uma ordem nova cujos tamanho e tipo de componente não são conhecidos em tempo real, e então modifique os componentes da ordem.

2.15 ARQUIVOS JAR (JAVA "ARCHIVES")

Embora *JavaBeans* seja uma tecnologia muito poderosa por si só, ela depende muito do suporte de algumas outras áreas centrais do sistema Java. Uma delas diz respeito à forma como os componentes *JavaBeans* são distribuídos aos usuários finais. Os "*Archives*" Java, são arquivos compactados destinados a guardar *applets* Java, aplicações ou *Beans* juntamente com qualquer recurso associado. Os *archives Java* são comumente referidos como arquivos JAR (*Java ARchive*) (Morrison *et. al*., 1999).

Os arquivos JAR são bem parecidos com outros tipos de arquivos compactados como ZIP e TAR, exceto que eles, adicionalmente fornecem suporte a autenticação de segurança (Morrison *et. al*., 1999).

Motivo pelos quais os arquivos JAR são tão importantes para o Java (Morrison *et. al*., 1999).

São independentes da plataforma: São independentes porque o padrão é inteiramente escrito em Java.

Dão suporte a diferentes tipos de arquivos: São capazes de manipular muitos tipos de arquivos, inclusive aqueles contendo imagens, sons e classes.

Têm compatibilidade retroativa: Os arquivos JAR são compatíveis com versões anteriores do Java. Isso significa, por exemplo, empacotar *applets* criadas no Java 1.0.2 em arquivos JAR. O JAR é basicamente, um padrão para empacotamento de arquivos.

Fornecem um padrão aberto e extensível: significa que o padrão JAR pode crescer e evoluir para suportar futuras melhorias do Java. Já que a tecnologia Java encontra-se em constante estado de evolução. (Morrison *et. Al*,1999).

2.15.1 ARQUIVOS JAR E JAVABEANS

Sob uma perspectiva de projeto e desenvolvimento, os arquivos JAR não tem nada a ver com o *JavaBeans*, ou seja, não precisa fazer nada de especial em um *Bean* a nível de código. Os arquivos JAR entram em cena com o *JavaBeans* quando é chegada o momento de empacotar um *Bean* para a distribuição.

Usando os arquivos JAR, pode-se agrupar as classes e recursos para um Bean em uma unidade compactada para organizá-los, preservar espaços, e em alguns casos fornecer uma medida de segurança.

Todos os *Beans* devem ser empacotados em arquivos JAR, um dos motivos é que as ferramentas para a construção de aplicações que suportam a integração de *Beans* esperam encontrar os *Beans* empacotados na forma de arquivos JAR.

Exemplos de ferramentas que serão usadas neste projeto e requer que os *beans* sejam empacotados em arquivos JAR são o contêiner de teste *BeanBox,* que acompanha o *Benas Developement Kit* e o *JavaBeans Bridge for AcitiveX* que é um utilitário que estabelece uma funcionalidade a um *Bean* de forma que ele possa ser usado com um componente *ActiveX* . (Morrison *et. al*,1999).

2.16 BEANBOX

Um *bean* fora de um ambiente construtor ele não pode mostrar qualquer um de seus recursos especiais, assim a questão é: Como mostrar o poder desse *bean*? Essa pergunta lava à questão de qual ferramenta construtora de beans utilizar. Em vez de favorecer o produto de um fornecedor em particular, será utilizado o ambiente de menor denominador comum, chamado BeanBox, que faz parte do *Beans Developer Kit* ou BDK. A *BeanBox* não é muito completa, mas está disponível em muitas plataformas e permite testar beans simples. O BeanBox é considerado um referência de ambiente de ferramenta construtora, porém não é projetado para construir aplicações de GUI (*Graphical User Interfaces*), nem é necessário saber usar ferramentas de construtoras Visuais como Delphi, ou Visual Basic. Caso o *bean* funcione na *BeanBox* ele deverá funcionar em outros ambientes de desenvolvimento sem quaisquer alterações (SUN, 2004). A figura 2.6 abaixo mostra a *BeanBox* em todo seu esplendor.

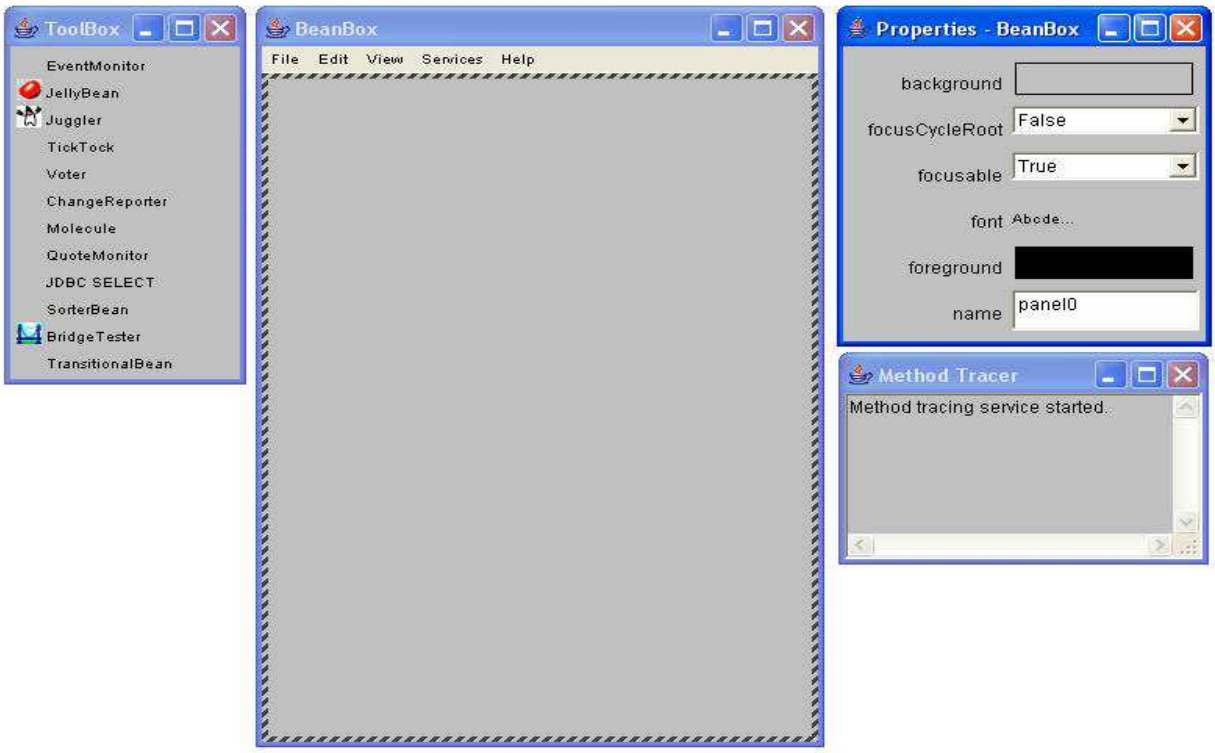


Figura 6.6 – Aspecto inicial do BeanBox, a janela contêiner principal do BeanBox e as janelas ToolBox e Propertysheet.

A **BeanBox** aparece inicialmente com a janela vazia que poderá ser chamada de formulário através do qual outras ferramentas construtoras crian suas aplicações.

A janela **ToolBox** exibe o *JavaBeans* que está instalado atualmente no *BeanBox*, quando o *BeanBox* é aberta, a *ToolBox* automaticamente carregada com os *Beans* em formato JAR que estão contidos no diretório de *bean/jars*, podendo-se adicionar outros *Beans*.

A janela **Properties** exibe as propriedades atuais do *Bean* selecionado sedo usada também para editar as propriedades de um *Bean*, caso nenhum *bean* seja selecionado, pode-se exibir as propriedades *BeanBox* dano um click no centro da janela.

2.17 COM

A tecnologia COM (*Componente Object Mode*) (Modelo de Objetos Componentes) foi criada pela Microsoft® para a sua plataforma Windows, com o intuito de promover a integração de aplicativos, definindo uma maneira padrão para um módulo (Aplicativo ou DLL) cliente e um módulo servidor (COM Server) se comunicarem, por meio de uma interface específica (Melo, 2003).

2.18 ACTIVEX

ActiveX é um conjunto de tecnologias desenvolvidas pela Microsoft em meados da década de 1990 no sentido de facilitar a criação de "Web sites" interativos e um rápido desenvolvimento de controles especiais para a Internet (Activex,2004).

A criação de um componente COM em uma DLL é indicada no Delphi como uma biblioteca Active X (*Library ActiveX*) (Melo, 2003).

2.19 JAVABEANS BRIDGE FOR ACTIVEX

O *JavaBeans Bridge* da *Sun* para *ActiveX* inclui características que permitem *JavaBeans* e componentes *ActiveX* trocar informação em ambas direções. Isto permite os clientes embutir componentes *JavaBeans* dentro de container de *ActiveX* existentes, como Microsoft Office, Internet Explorer, aplicações do *Borland Delphi* e *Visual Basic* , entre outros (Morrison et. al,1999).

O *JavaBeans Bridge para ActiveX* contém uma GUI packager (*Graphical User Interfaces*) que cria OLE (*Object Linking and Embedding*), cria informações da biblioteca e win32 de registro para um *Bean* selecionado. Isto permite o container de OLE/COM (*Object Linking and Embedding/Component Object Model*) analisar corretamente e apresentar um *Bean*. Os *Beans* podem acionar eventos que podem ser pegos através de recipientes de

OLE/COM/ActiveX. *Beans* podem agir como servidores para OLE/COM/ActiveX. Isto significa que ferramentas como Visual Basic pode invocar métodos em *Beans* (SUN, 2004).

A Bridge é formada pelos seguintes itens (SUN,2004):

Classes Bridge: consistem de classes Java e C++ que agem como um tradutor entre o JavaBeans e o ActiveX.

Utilitário ActiveX Packager: é um utilitário que estabelece uma funcionalidade maior em um *Bean* de modo que ele possa ser capaz de ser usado como um componente ActiveX.

Beans demonstrativos: são dois *Beans* do BDK (*Beans Devoloper Kit*) que já foram rodados no *Packager*.

Documentação: consiste basicamente de um tutorial de como utilizar um *Bean* no Visual Basic.

A *ActiveX Bridge* foi concebida para ser a mais automática possível, os *Beans* requerem pouco esforço adicional para serem utilizáveis como componentes ActiveX, para operar em tal ambiente requerem os três seguintes elementos que são gerados automaticamente pelo *ActiveX Packager* (SUN, 2004).

Biblioteca de tipos: é um arquivo binário que descreve as propriedades, métodos e eventos de um *Bean* em termos de ActiveX. Pode-se imaginar uma biblioteca de tipos como o equivalente grosseiro no ActiveX de uma classe informativa de um *Bean*.

Arquivo de registro: contem informações sobre o *Bean* tal como o caminho para a biblioteca de tipos.

Classes "stub" do Java: correspondem à ponte de ligação real entre um *Bean* e o ActiveX. Estas classes são geradas e adicionadas ao arquivo JAR de um *Bean* pelo *ActiveX Packager* (Morrison *et al.*, 1999).

A figura 2.7 abaixo mostra uma comparação entre ActiveX e JavaBeans, e as ferramentas que apóiam o caminho de migração entre eles. Usando JavaBeans, pode-se criar um única fonte para suas aplicações de desktop, aplicações de Rede, e componentes, podendo criá-los em Java e exportá-los para VB (*Visual Basic*).

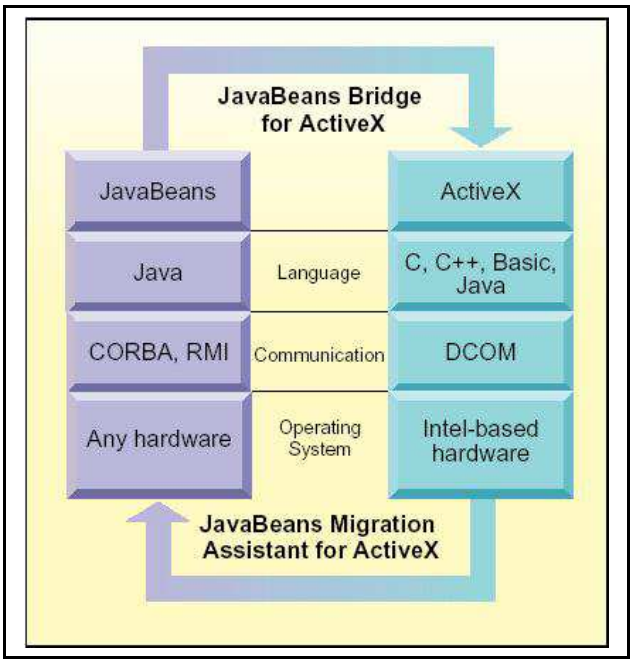


Figura 7.7 – ActiveX e JavaBeans, e as ferramentas que apóiam o caminho de migração entre eles.

2.20 JAVABEANS X ACTIVEX

A um nível alto, a diferença principal entre *JavaBeans* e *ActiveX* são que enquanto componentes *JavaBeans* são para plataforma independentes, ActiveX só pode ser usado em plataformas de Windows.

Assim enquanto é possível escrever controles ActiveX usando a linguagem de programação Java, por causa da dependência do ActiveX no COM de Microsoft (*Component Object Model*), esses controles ainda serão limitados a sistemas baseados Windows os JavaBeans são independentes da plataforma (Devx,2004).

2.21 JAVABEANS X ENTERPRISE JAVABEANS

O *JavaBeans* foi projetado originalmente para interface dos usuários e tarefas do lado do cliente. Para permitir, desenvolvimento de componentes baseado em Java do lado do servidor, a Sun lançou o *Enterprise JavaBeans* (EJB) especificação que soma características do lado do servidor como transações, persistência, e escalabidade (Boris Feldman, devx.com).

2.22 CONSIDERAÇÕES FINAIS

As ferramentas para o desenvolvimento de um *JavaBeans* são muitas, mais importante do que saber que elas existem é saber usá-las de forma correta. Com todas as ferramentas em mãos parte-se agora para o desenvolvimento do componente, etapa fundamental para se obter sucesso no funcionamento do componente.

3 DESENVOLVIMENTO DO COMPONENTE

Antes de iniciar a desenvolvimento de um componente é necessário ter bem definido qual é realmente a finalidade deste componente, é preciso analisar, projetar em diversos níveis de abstrações para se discutir toda a estrutura do componente. Para analisar todas as etapas do desenvolvimento do componente utiliza-se a Engenharia de Software, que guiará o processo de maneira consciente e segura.

A Figura 3.1 ilustra as etapas seguidas para a compreensão do domínio a que pertence o componente, para capturar a sua Arquitetura e comportamento, para especificar os Padrões de Projeto que se aplicam à sua estrutura e para o seu desenvolvimento proposto, por (Melo,2003), será acrescentado a uma nova etapa no desenvolvimento do componente que é a aplicação da tecnologia *JavaBeans Bridge for ActiveX*.

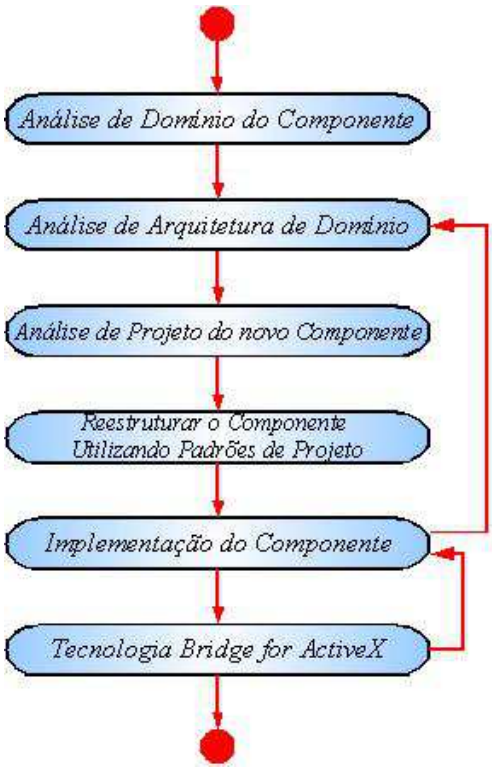


Figura 8.1 – Etapas do Desenvolvimento do Componente

3.1 ETAPAS PARA O DESENVOLVIMENTO DO COMPONENTE

Quando se pretende desenvolver um sistema para uma empresa qualquer, a maior parte das informações e requisitos necessários a sua construção são obtidas de entrevistas com o proprietário e funcionários dessa empresa. Porém, quando se pretende desenvolver um Componente que pertence a uma determinada Arquitetura já existente, a forma de se obter informações, requisitos e funcionalidades do Componente (Sistema) a ser desenvolvido não é algo trivial (Melo, 2003).

Será usado as mesmas etapas e procedimentos utilizadas por (Melo, 2003), sendo acrescentando apenas o *JavaBeans Bridge for ActiveX*. Será analisado posteriormente sua interoperabilidade dentro do sistema já desenvolvido por (Melo, 2003), daí a importância de se fazer uma análise e uma implementação perfeita. As etapas para se desenvolver um componente a partir de uma Arquitetura Estruturada em Componentes é a seguinte:

Domínio do Componente – Nesta etapa o Analista de Sistemas deve se preocupar em compreender bem o Domínio a qual pertence o Componente a ser desenvolvido, se abstraindo de pequenos detalhes, e, preocupando-se com a área compreendida pelo Sistema que contém o Componente, com quais situações ele se aplica e com as possíveis soluções que ele oferece para determinadas situações (Melo,2003).

Análise da Arquitetura do Domínio – Esta etapa é uma continuidade da descoberta do domínio do Componente, porém, em um nível de abstração inferior, preocupando-se em obter as características dos Componentes que pertencem ao Domínio em questão. A Análise da Arquitetura do Domínio ajuda a compreender melhor as soluções propostas, pois, pode-se observar o que cada um dos Componentes faz e como eles se relacionam uns com os outros para realizar tarefas específicas. Deve-se realizar esta etapa com muita atenção para evitar problemas durante o Projeto do novo Componente, como por exemplo, uma interface que depende de uma Interface Provida de outro Componente e que não foi identificada, provocará uma inconsistência no novo Componente (Melo, 2003).

Análise e Projeto do Novo Componente – Após a compreensão do Domínio e da sua Arquitetura pode-se dar inicio à Análise e Projeto do novo Componente a ser

desenvolvido. É preciso analisar a sua estrutura (classes, atributos, interfaces e operações) e o seu relacionamento com os outros Componentes da Arquitetura. Essa Análise visa captar as funcionalidades do Componente em geral e, se julgar necessário, excluir ou incluir novas funcionalidades ao novo Componente, que antes eram responsabilidade de outros Componentes ligados a este. Definidas as funcionalidades do novo Componente dar-se-á inicio ao Projeto do Componente, partindo da modelagem de um alto nível de abstração até chegar aos mínimos detalhes da sua estrutura (Melo, 2003).

Reestruturar o Componente Utilizando Padrões de Projeto – Com o Componente totalmente Projetado pode-se identificar alguns Padrões de Projeto que se aplicam à sua estrutura e, se for do interesse do desenvolvedor, aplicar estes Padrões de Projeto e refazer o Projeto do Componente. A identificação dos Padrões não é algo muito complexo, uma vez que, neste ponto já se conhece muito bem o Componente, bastando procurar por Padrões de Projeto que tratem de problemas que por ventura existam na estrutura do Componente (Melo, 2003).

Implementação do Componente - Esta é a etapa tão desejada, onde se inicia a construção física do Componente. Portanto, antes de sair programando é aconselhável uma revisão em todo o processo de Análise e Projeto do Componente, pois, alguma falha nessas etapas pode acarretar em sérios problemas durante a Implementação. Se possível escolha um padrão de modelagem de Componente para seguir, isso pode beneficiar o público alvo que vai realmente utilizar o Componente e também facilitar futuras manutenções ou expansão do Componente (criação de outros componentes que se relacionarão com este). Agora já é possível escolher uma linguagem de programação e dar início à programação do Componente. Caso o desenvolvedor se depare com situações inoportunas e sejam necessárias algumas alterações, estas devem ser feitas e a documentação das etapas de Análise e Projeto devem ser atualizadas concomitantemente ao processo de implementação, para chegar ao produto final com uma documentação bem consistente (Melo, 2003).

Tecnologia Bridge for ActiveX – Nesta esta etapa onde o componente já encontra implementado e testado, será aplicado uma tecnologia Java para que o componente implementado no padrão JavaBeans possa se comunicar e funcionar normalmente dentro de um sistema desenvolvido com a tecnologia COM.

3.2 DOMÍNIO DO COMPONENTE

O componente a ser desenvolvido neste trabalho, Compra_Venda_Recurso, foi escolhido de uma Arquitetura Estruturada em Componentes definida em (Coelho, 2002) construída a partir de uma reengenharia da Arquitetura do Framework GREN (Gestão de Recursos de Negócios) descrito em (Braga *et. al.*, 2001). Portanto, é importante entender o domínio do Framework GREN para que seja possível compreender as características do componente escolhido e onde ele poderá ser aplicado, é importante também entender a Arquitetura em componentes escrita por (Coelho,2002) bem como a implementação do componente Locar_Reservar_Recurso desenvolvido por (Melo, 2003)

O Framework GREN foi desenvolvido com base em uma linguagem de padrões para gestão de recursos de negócios – Linguagem de Padrões GRN – visando apoiar a construção de aplicações para tal domínio, que inclui sistemas para locação, comercialização e manutenção de bens, tais como, automóveis, imóveis, fitas de vídeo, dentre outros. Sua estrutura Arquitetural é toda constituída de classes (Braga *et. al.*, 2001).

Em um nível de abstração mais baixo, pode-se observar a Arquitetura Parcial do Framework GREN com suas classes e relacionamentos, apresentada na Figura 3.2 (Coelho, 2002). Observe-se um número muito grande de classes e relacionamentos entre elas o que torna o Framework GREN difícil de ser compreendido rapidamente, sendo assim a principal desvantagens, motivação pela qual foi apresentado por (Coelho, 2002) um estudo comparativo entre a Arquitetura do Framework GREN baseado exclusivamente em classes e outras duas arquiteturas, uma Estruturada em Componentes e uma Baseada em Componentes tendo como um dos objetivos verificar quais são as vantagens e desvantagens de cada uma destas arquiteturas e como elas influenciam na complexidade da arquitetura e na facilidade de utilização do Framework.

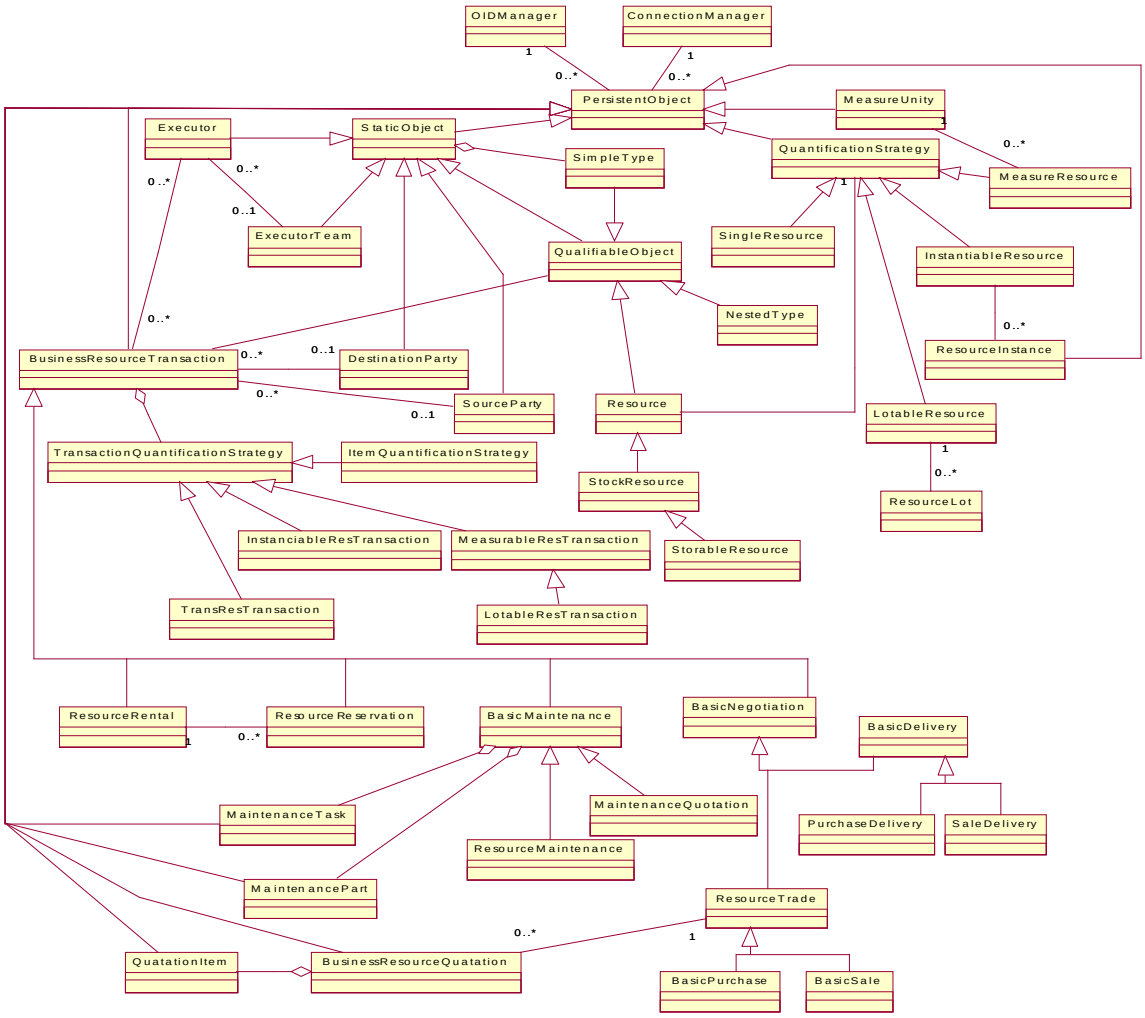


Figura 9.2 – Arquitetura Parcial do Framework GREN (Coelho, 2002).

3.3 ANÁLISE DE ARQUITETURA DE DOMÍNIO

Nessa seção será apresentada a Arquitetura Estruturada em Componentes do *Framework GREN*, que é a fonte do componente a ser estudado, e também o componente Locar_Reservar_Recurso sua estrutura aonde o novo componente será inserido.

A arquitetura de Componentes do domínio de recursos de negócios foi definida seguindo o processo de reengenharia e alguns passos definido por (Coelho, 2002). O resultado disto e a Arquitetura Estruturada em Componentes no domínio de gestão de recursos de negócio ilustrado na figura 3.3. A figura 3.4 ilustra a Arquitetura Estruturada em Componentes no domínio de gestão de recursos de negócio proposta por (Coelho, 2002) uma distribuição dos componentes nas camadas de Aplicação, de Regras de Negócio e de Elementos do Domínio – camadas 1, 2 e 3 respectivamente. A figura 3.5 mostra uma relação entre os componentes.

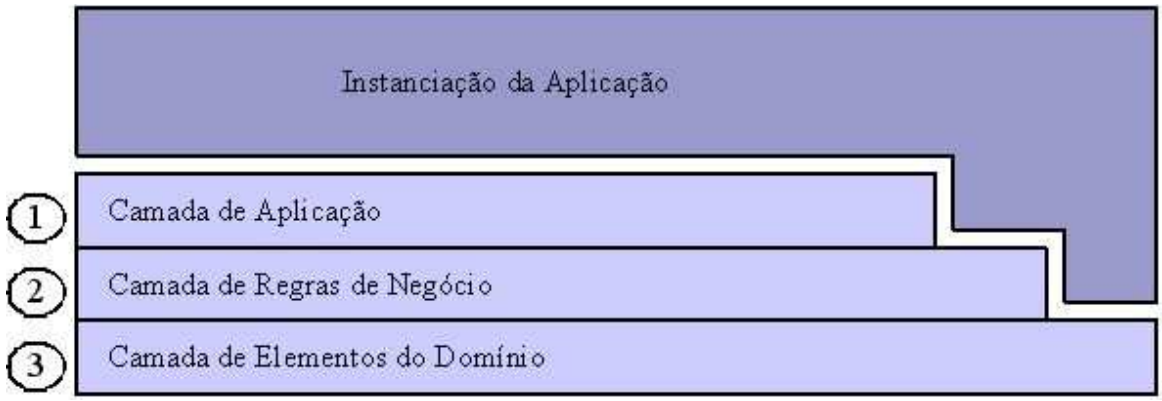


Figura 10.3 – Camadas da Arquitetura Estruturada em Componentes (Coelho, 2002)

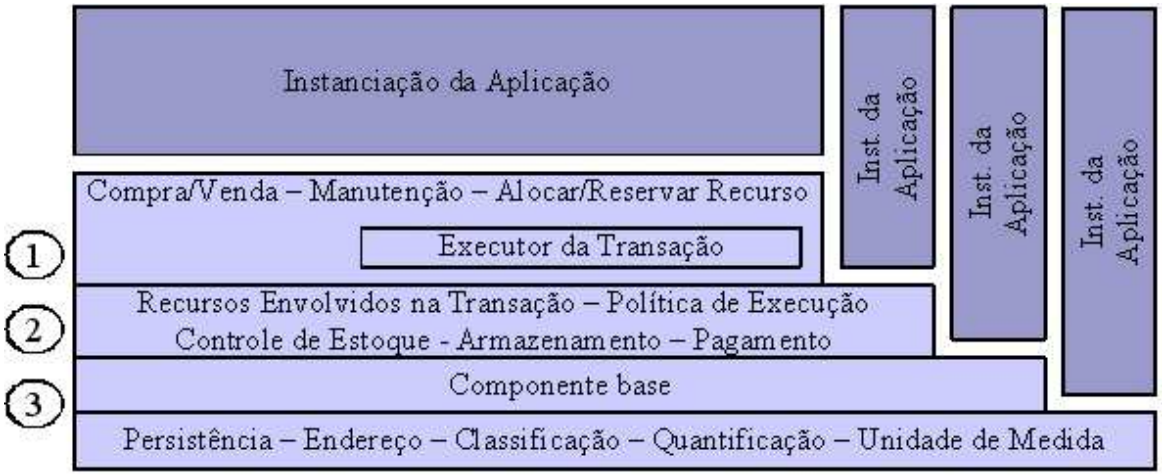


Figura 11.4 – Distribuição dos Componentes nas Camadas da Arquitetura Estruturada em Componentes (Coelho, 2002)

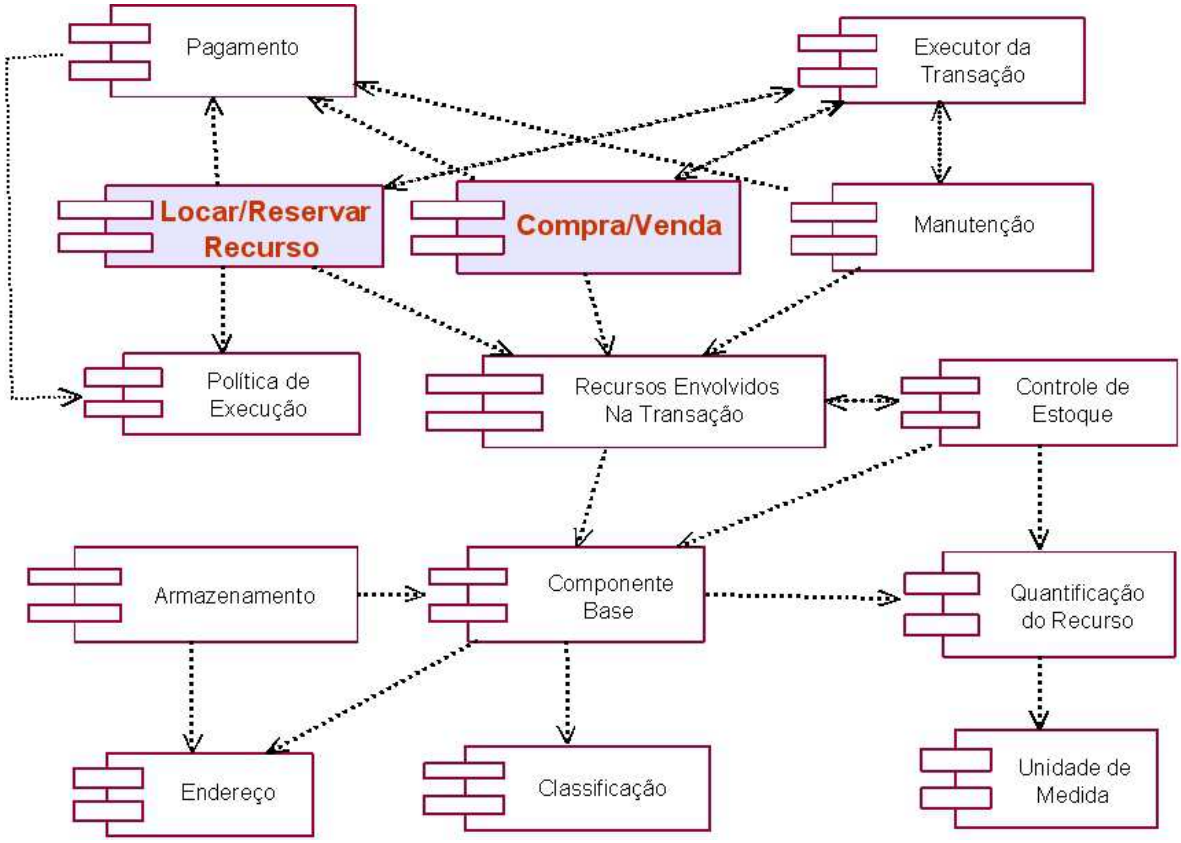


Figura 12.5 – Arquitetura Estruturada em Componentes (Coelho, 2002)

Uma das vantagens dessa Arquitetura é que para instanciar um novo sistema não é necessário estudar toda a sua documentação, uma vez que alguns dos componentes podem ser usados individualmente ou combinados com alguns outros, sendo preciso apenas estudar parte da documentação referente a estes componentes, pois estes, possuem um nível de abstração bem mais alto do que um emaranhado de classes e métodos, encapsulam características comuns a um determinado domínio (Coelho, 2002).

Outra característica relevante dos componentes é que eles podem possuir além das suas interfaces padrões, interfaces relacionadas a um sistema específico, o que adiantaria, em muito, a modelagem (Coelho, 2002).

Uma desvantagem dos componentes é que uma falha na etapa de análise, como uma interface não identificada ou novas operações a serem executadas por uma interface, pode acarretar em sérios problemas, pois fazer a correção de algumas interfaces é tão

complexo quanto às correções feitas em uma arquitetura constituída de classes (Coelho, 2002).

A especificação do Componente original Locar_Reservar_Recursos é fornecer a capacidade de alugar ou reservar um determinado recurso, a um cliente específico, de comparar a data da reserva com a data atual, de cancelar a reserva caso não tenha sido efetivado o aluguel até o fim do expediente, de comparar a data de entrega do recurso com a data atual e, se o recurso estiver sendo entregue em uma data posterior da combinada, de calcular a multa referente. Fornece também a habilidade de iniciar e terminar uma locação quer seja ou não executada até o fim (Melo, 2003).

O Componente Locar_Reservar_Recursos é responsável exclusivamente por efetuar uma Locação ou Reserva, outras funcionalidades não são provenientes deste componente e sim da combinação de outros Componentes (Melo, 2003).

A figura 3.6 mostra as interfaces Providas do Componente original Locar_Reservar_Recursos obtida em (Coelho, 2002).

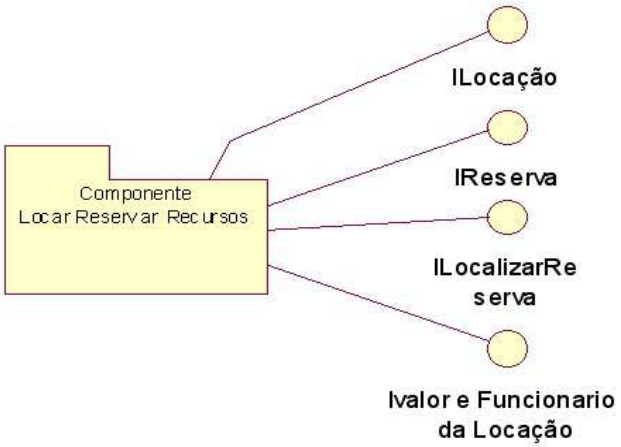


Figura 13.6 Interfaces Providas pelo Componente Original Locar_Reservar_Recursos.

A figura 3.7 mostra as interfaces providas da reestruturação feita por (Melo.2003) no componente Locar_Reservar_Recurso.

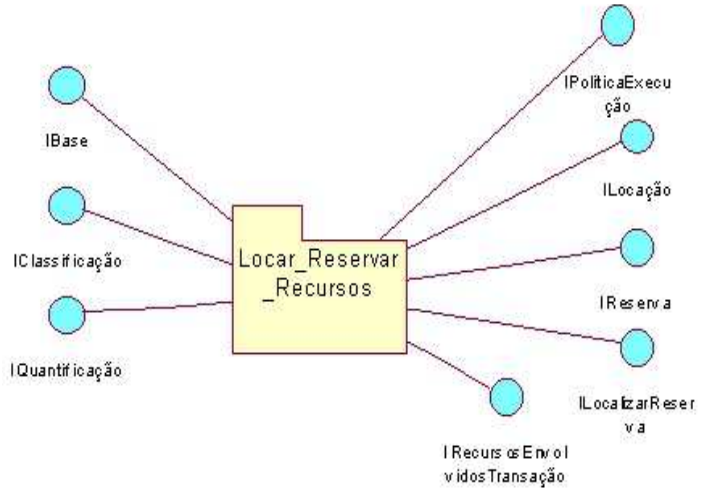


Figura 14.7 – Interfaces Providas pelo Novo Componente Locar_Reservar_Recursos.

Devido a este componente depender de alguns outros componentes, algumas interfaces serão incluídas na sua estrutura com o intuito de facilitar o seu entendimento, seu desenvolvimento e para que seja possível mostrar um exemplo prático de sua utilização, eliminando-se a necessidade da criação de outros Componentes (Melo, 2003).

3.4 ANÁLISE E PROJETO DO NOVO COMPONENTE

Dá início agora a análise do componente original Compra_Venda_Recurso e o projeto do novo componente Compra_Venda_Recurso, seguindo o mesmo desenvolvimento do componente Locar_Reservar_Recurso (Melo, 2003) do qual ele irá se comunicar. Também será constituído de alguns padrões de projetos tendo a mesma funcionalidade do componente original.

3.4.1 ANÁLISE DA ESTRUTURA DO COMPONENTE COMPRA_VENDA

Toda a estrutura do Componente Compra_Venda_Recurso será obtida a partir de um estudo da documentação da Arquitetura Estruturada em Componentes e dos Casos de Uso definidos em (Coelho, 2002) e da Linguagem de Padrões GRN (Braga *et. al*., 2001).

O componente Compra_Venda esta na camada de aplicação, fornece a capacidade de realizar comercialização ou cotação de um recurso, seja para compra ou para

venda, e se for a política da organização, de possibilitar a verificação da entrega do produto antes da confirmação final da transação e de verificar quais os orçamentos feitos que não se concretizaram em comercialização. Fornece também a habilidade de iniciar e terminar uma comercialização quer seja ou não executada até o fim (Coelho, 2002).

A figura 3.8 mostra as interfaces Providas do Componente original Compra_Venda obtida em (Coelho, 2002).

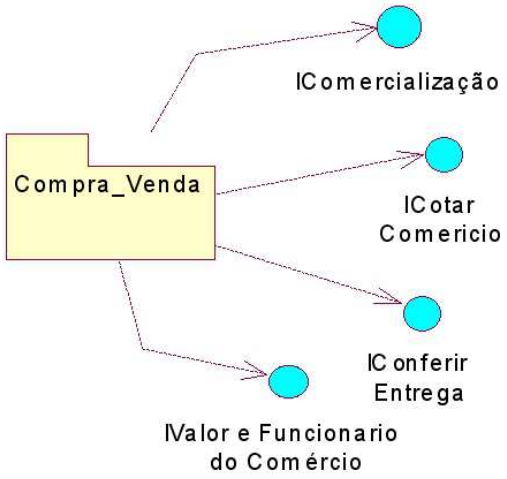


Figura 15.8 Interfaces Providas pelo Componente Original Compra_Venda.

3.4.2 PROJETO DO NOVO COMPONENTE COMPRA_VENDA

Observando a figura 3.8 na Seção 3.4.1, nota-se que o Componente não só faz compra e venda como também possibilita a verificação da entrega do produto, dos orçamentos feitos os que não se concretizavam. Porém o componente Compra_Venda a ser implementado simplesmente realizara a cotação, compra e venda de recursos, sendo assim só será necessário a implementação das interfaces que realizam tais funções. Assim tem-se abaixo, figura 3.9 o novo Componente com suas interfaces.

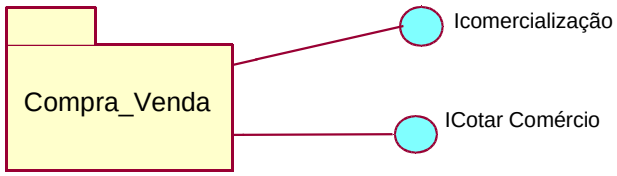


Figura 16.9 Interfaces Providas do novo Componente Compra_Venda.

As interfaces e os motivos de suas presenças no componente serão descritos a seguir:

Interface IComercialização

Trata dos pré-requisitos necessários à Compra e a Venda. Fornece métodos para inicializar uma nova transação, contém métodos para buscar o recurso a ser negociado a empresa o cliente o fornecedor e o atendente e assim comercializar o recurso sendo Compra ou Venda. É necessária esta interface no referido Componente, uma vez que, essas informações são essenciais para a Compra e Venda de Recursos.

Interface ICotar Comércio

Trata-se de uma interface simples com apenas um método para cotar recurso, podendo ser concretizado em venda ou não.

É apresentado a seguir na figura 3.10 o diagrama de classes do componente modelado para implementar as Interfaces descritas anteriormente, permitindo também se ter uma visão estática da estrutura do Componente permitindo visualizar quais as classes que realizam determinadas Interfaces.

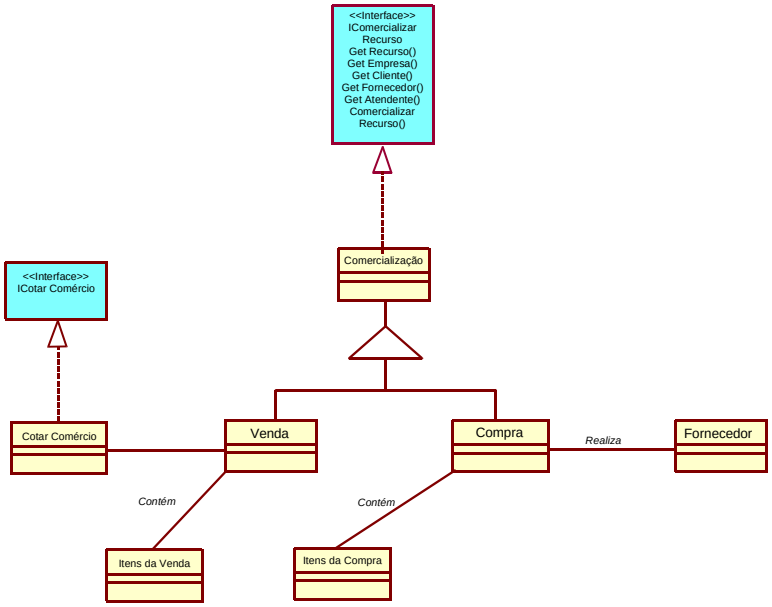
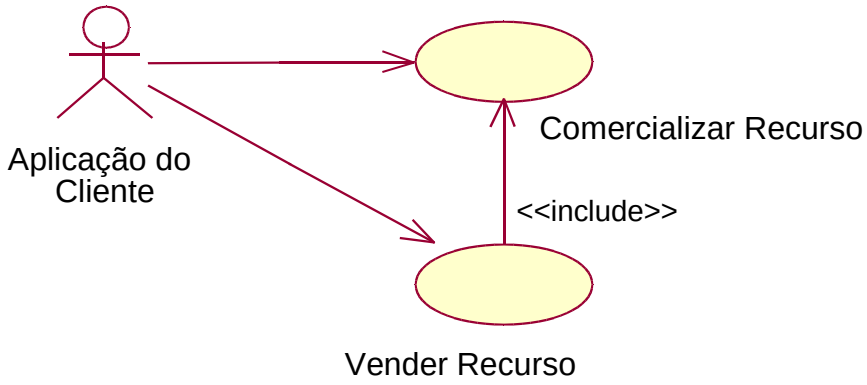


Figura 17.10 – Diagrama de Classes do Novo Componente Compra_Venda

3.4.3 DIAGRAMAS DE CASO DE USO DO COMPONENTE COMPRA_VENDA

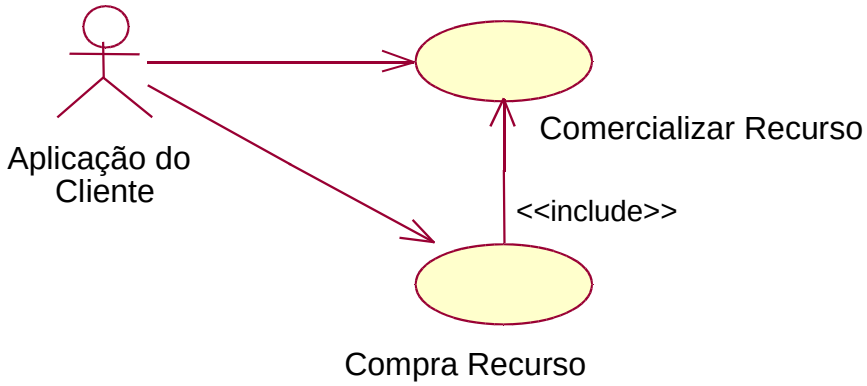
Os Diagramas de Caso de Uso mostrados a seguir fornecem uma visão do comportamento do sistema, suas funcionalidades essenciais e a interação dos elementos envolvidos com o sistema. A descrição detalhada de cada caso de uso encontra-se no anexo A.

Pacote: Comercialização_Venda



A aplicação do Cliente utiliza o Componente Compra_Venda. Solicita a execução de tarefas específicas através de chamadas às operações das Interfaces do Componente passando os dados necessários à execução dessas tarefas. Requisita uma cotação que resultara em uma compra ou venda.

Pacote: Comercialização_Compra



Pacote: Cotar Recurso

Para maior entendimento dos casos de uso Cotar Recurso, Comercialização_Venda, Comercialização_Compra criou-se um diagrama de seqüência para cada um deles. Os diagramas de seqüência abaixo fornecem uma visualização das interações entre a Aplicação do Cliente e o Componente e também uma visão das trocas de mensagens entre as Classes do Componente.

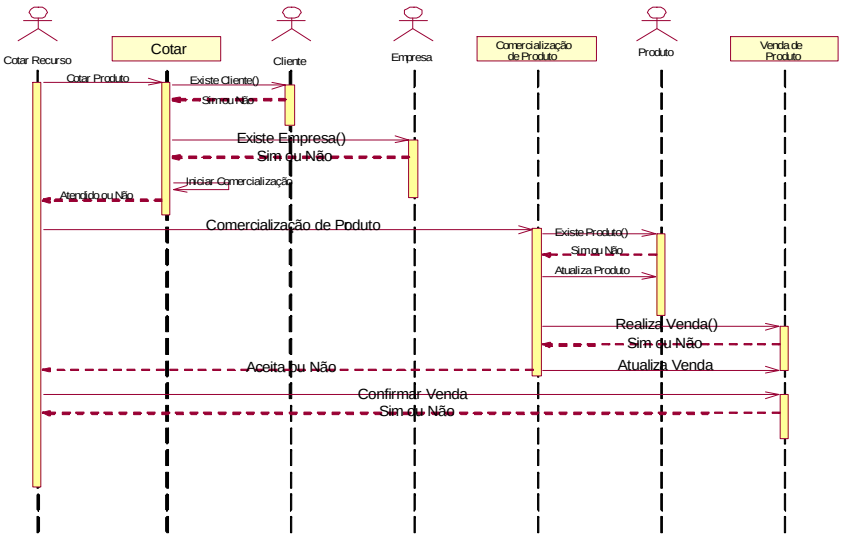


Figura 18.11 – Diagrama de Seqüência Cotação e Venda de Produtos

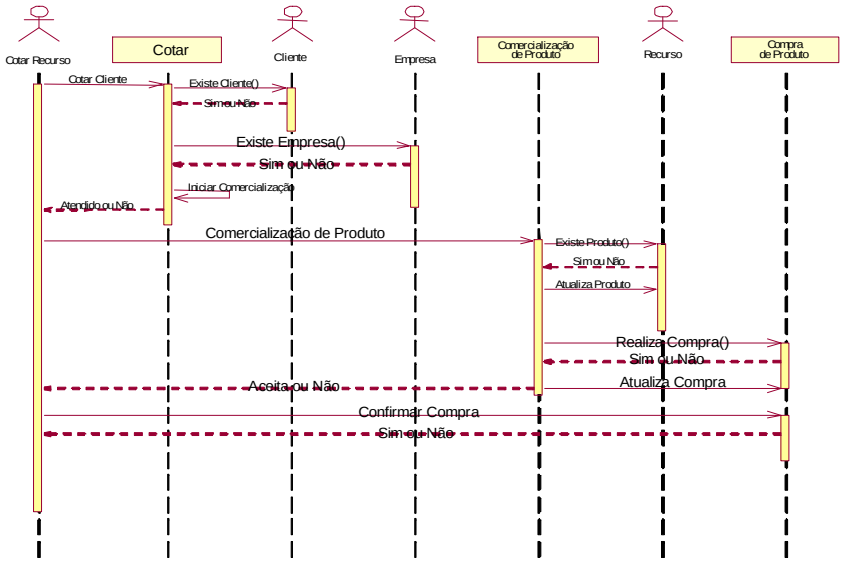


Figura 19.12 – Diagrama de Seqüência Cotação e Compra de Recurso

3.5 REESTRUTURAR O COMPONENTE UTILIZANDO PADRÕES DE PROJETO

Após concluir a Análise e o Projeto é proposto por (Melo, 2003) uma outra análise especial. Trata-se de uma verificação na estrutura do Componente em busca de partes da estrutura que tratam de problemas específicos, onde haja a necessidade de se aplicar uma estrutura mais forte (Padrão de Projeto), que traga benefícios para o Componente como um todo, como torná-lo mais fácil de usar, manter e deixá-lo o mais flexível possível (Melo, 2003).

Depois que esses problemas estruturais particulares foram identificados foi realizada uma análise minuciosa no catálogo do GAMMA (Gamma *et. al*., 2000), com o objetivo de identificar os Padrões de Projeto que melhor se encaixam na solução desses problemas. O primeiro Padrão identificado foi o padrão FAÇADE segundo GAMMA, tem a seguinte intenção:

“Fornecer uma interface unificada para um conjunto de interfaces em um subsistema. FAÇADE define uma interface de nível mais alto que torna o subsistema mais fácil de ser usado”.

Veja na seguir na Figura 3.13 a estrutura do Padrão de Projeto FAÇADE.

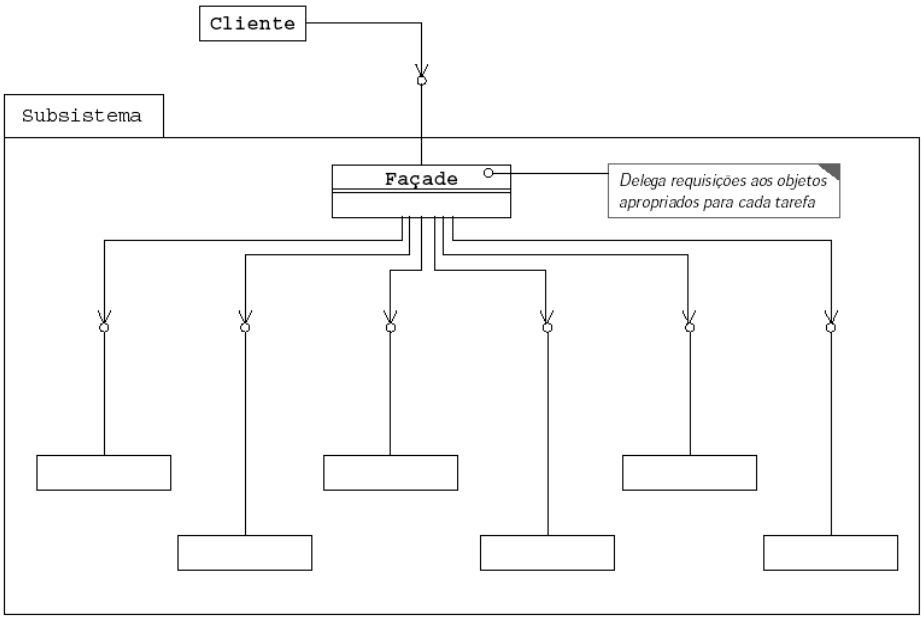


Figura 20.13 – Estrutura do padrão FAÇADE

Usa-se o padrão FAÇADE sempre que se for desejável criar uma interface para um conjunto de objetos e facilitar o uso da aplicação, permite que objetos individuais cuidem de uma única tarefa, deixando que a fachada se encarregue de divulgar suas operações (ArgoNavis, 2004).

A figura 3.14 mostra o diagrama de classes do componente Compra_Venda após aplicar o padrão FAÇADE.

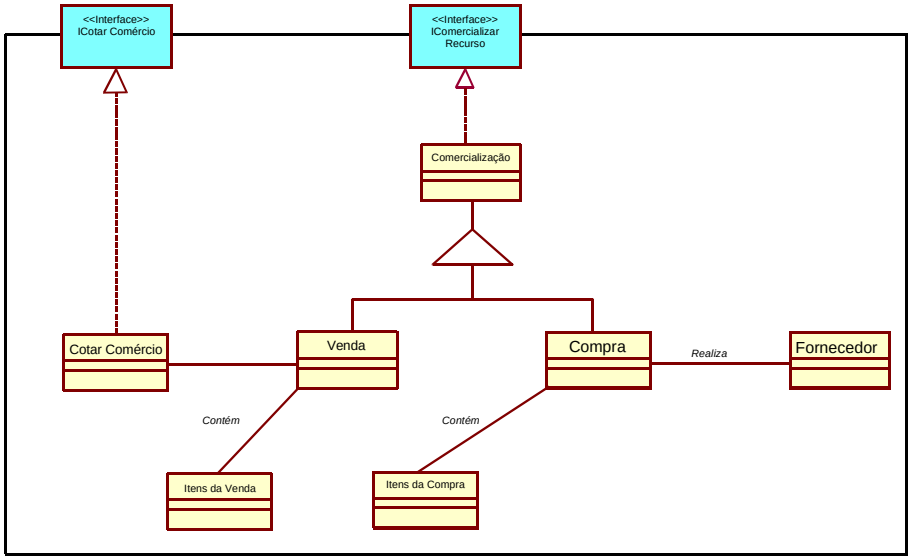


Figura 21.14 – Padrão FAÇADE presente no componte Compra_Venda

As interfaces do componente Compra_Venda torna-se FAÇADE (*fachada*), com o objetivo de minimizar a comunicação e as dependências entre subsistemas ou entre classes de um subsistema fornecendo assim uma interface única e simplificada para os recursos e funcionalidades mais gerais de um subsistema, resolve o problema onde o cliente necessita saber muitos detalhes do subsistema para utilizá-lo.

3.6 CONSIDERAÇÕES FINAIS

Foram obtidas as conclusões desejadas com desenvolvimento do Componente Compra_Venda, verificou-se vantagens no desenvolvimentos Estruturado em Componentes e, principalmente de Componentes estruturados com Padrões de Projetos. Como o componente estruturado pode-se dar inicio a fase de desenvolvimento e empacotamento e comunicação com componentes de outras tecnologias.

4 IMPLEMENTANDO UM COMPONENTE JAVABEANS

Neste Capítulo será descrito a implementação de um Componente modelo, utilizando o padrão *JavaBeans*, padrão que deverá ser utilizado na implementação do componente Compra_Venda projetado no Capítulo 3, para posteriormente inserir tal componente no sistema de Locadora de Vídeo e se comunicar com o componente Locar_Reservar_Recurs ambos desenvolvida por (Melo,2003). Será detalhado a Tecnologia JavaBeans que será empregada na construção do componente Compra_Venda, e também o Padrão de Modelagem de Componentes JavaBeans, e a forma correta de empacotar o componente JavaBeans para se comunicar com componentes COM.

4.1 TECNOLOGIA JAVABEANS

Na Seção 1.8 do capítulo 2 foi descrita uma visão geral da Tecnologia JavaBeans e agora será aprofundado um pouco mais o funcionamento de um componente JavaBeans sua implementação e interação com uma aplicação e componentes COM.

4.2 A IMPLEMENTAÇÃO DE UM COMPONENTE

O processo de implementação de um componente JavaBeans é muito simples, como já foi citado, o componente JavaBeans é uma classe Java. Abaixo está descrito uma implementação de um componentes JavaBeans simples que serve como modelo de implementação, foi descrito em (Deitel *et. al*., 2001).

4.2.1 A IMPLEMENTANDO UM COMPONENTE JAVABEANS

O modelo a seguir apresenta uma animação de um logotipo Deitel & Associates, Inc. A classe *LogoAnimator* foi inicialmente implementada como um aplicativo independente que executava em um *JFreme,* será demonstrado aqui como um *Bean* para ser utilizado na *BeanBox*. A classe *LogoAnimator* é uma subclasse de *JPanel* portanto ele tem propriedades e eventos herdados da classe *JPanel*, pode-se tirar proveito de algumas desses propriedades e eventos predefinidos, como a cor de fundo e eventos de mouse (Deitel *et. al*., 2001). A figura abaixo mostra o código da classe JavaBeans *LogoAnimator*.

```
2 // Bean de animação
3 package jhtp3beans;
4
5 import java.awt.*;
6 import java.awt.event.*;
7 import java.io.*;
8 import java.net.*;
9 import javax.swing.*;
10
11 public class LogoAnimator extends JPanel
12                 implements ActionListener, Serializable {
13       protected ImageIcon images[];
14       protected int totalImages = 30,
15                         currentImage = 0,
16                         animationDelay = 50; // retardo de 50 milissegundos
17       protected Timer animationTimer;
18
19       public LogoAnimator()
20       {
21                 setSize( getPreferredSize() );
22
23                 images = new ImageIcon[ totalImages ];
24
25                 URL url;
26
27                 for ( int i = 0; i < images.length; ++i ) {
28                         url = getClass().getResource(
29                                   "deitel" + i + ".gif" );
30                         images[ i ] = new ImageIcon( url );
31                 }
32
33       startAnimation();
34       }
35
36       public void paintComponent( Graphics g )
37       {
38                 super.paintComponent( g );
39
40                 if ( images[ currentImage ].getImageLoadStatus() ==
```

```
41                      MediaTracker.COMPLETE ) {
42                      g.setColor( getBackground() );
43                      g.drawRect(
44                              0, 0, getSize().width, getSize().height );
45                      images[ currentImage ].paintIcon( this, g, 0, 0 );
46                      currentImage = ( currentImage + 1 ) % totalImages;
47              }
48      }
49
50      public void actionPerformed( ActionEvent e )
51      {
52              repaint();
53      }
54
55      public void startAnimation()
56      {
57              if ( animationTimer == null ) {
58                      currentImage = 0;
59                      animationTimer = new Timer( animationDelay, this );
60                      animationTimer.start();
61              }
62              else // continua a partir da última imagem exibida
63                      if ( ! animationTimer.isRunning() )
64                              animationTimer.restart();
65      }
66
67      public void stopAnimation()
68      {
69              animationTimer.stop();
70      }
71
72      public Dimension getMinimumSize()
73      {
74              return getPreferredSize();
75      }
76
77      public Dimension getPreferredSize()
78      {
79              return new Dimension( 160, 80 );
80      }
81
82      public static void main( String args[] )
83      {
84              LogoAnimator anim = new LogoAnimator();
85
86              JFrame app = new JFrame( "Animator test" );
87              app.getContentPane().add( anim, BorderLayout.CENTER );
88
89              app.addWindowListener(
90                      new WindowAdapter() {
91                              public void windowClosing( WindowEvent e )
92                              {
93                                      System.exit( 0 );
94                              }
95                      }
96              );
```

```
97
98                app.setSize( anim.getPreferredSize().width + 10,
99                                   anim.getPreferredSize().height + 30 );
100               app.show();
101     }
102 }
```

Figura 22.1 : Linha de código da classe JavaBeans LogoAnimator.java

Para que a classe *LogoAnimator* pudesse ser utilizada como um JavaBean foram feitas pequenas modificações. Primeiro foi inserido uma instrução *packager* na linha 3, normalmente as classes que representam um bean são as primeiras colocadas em um pacote (Deitel et. al., 2001).

A comando para compilar classes empacotadas utiliza a opção " **– d** " com o compilador Java, onde "*jhtp3beans*" representa o diretório em que o pacote seve ser colocado como no exemplo abaixo:

```
Javac     – d     jhtp3beans     LogoAnimator.java
```

A segunda modificação está na linha 12, onde a classe especifica que ela implementa a interface *Serializable* para suportar persistência, todos os beans devem implementar a interface Serializable, implementar tal interface permite que programadores utilizem uma ferramenta de desenvolvimento para salvar seu *bean* personalizado serializando o *bean* para um arquivo (Deitel *et. al.*, 2001).

4.2.2 CRIANDO UM JAVABEANS

Para utilizar uma classe como um *JavaBean*, ela deve primeiro ser colocada em um *Java Archive File* (arquivo JAR) visto na seção capítulo 2. Porém antes de criar o arquivo JAR, primeiro cria-se um arquivo de texto chamado *manifest.tmp*. O arquivo manifesto (como é chamado) é utilizado pelo utilitário *jar* para descrever o conteúdo do arquivo *JAR*. Isso é importante para ambientes integrados de desenvolvimento que suportam *JavaBeans*. Quando um arquivo *JAR* que contem um *JavaBean* (ou um conjunto de JavaBeans) é carregado em um IDE, o IDE examina o arquivo manifesto para determinar as classes no *JAR* que representam *JavaBeans*. O utilitário de repositórios de arquivo Java *jar* utiliza *manifest.tmp* para criar um arquivo chamado *MANIFEST.MF* que é incluído no diretório *META-INF* do

arquivo *JAR*. Todos os ambientes de desenvolvimento sabem procurar o arquivo *MANIFEST.MF* no diretório *META-INF* do arquivo *JAR*. Além disso, o interpretador Java pode executar um aplicativo diretamente a partir de um arquivo JAR se o arquivo manifesto indicar qual classe no *JAR* contém *main* (Deitel et. al., 2001). O arquivo manifest.tmp para o bean LogoAnimator segue abaixo:

```
1    Main-Class: jhtp3beans.LogoAnimator2
2
3    Name: jhtp3beans/LogoAnimator.class
4    Java-Bean: True
```

Figura 23.2 : O arquivo manifest.tmp para o bean LogoAnimator

Nesse arquivo manifesto foi incluso a linha 1, que especifica que a classe *jhtp3beans.Logo-Animator* é a classe que contém o método *main* para executar o *bean* como um aplicativo. Quando a classe *main* de um aplicativo é armazenada em um JavaBean, o aplicativo pode ser executado diretamente do *bean* utilizando o interpretador Java com a opção de linha de comando *-jar* como segue:

```
java -jar LogoAnimator.jar
```

O interpretador automaticamente verá o arquivo manifesto para a classe especificada com a propriedade de arquivo manifesto ***Main-Class***: e iniciará a execução com o método *main* dessa classe.

Com *Java 2*, muitas plataformas constroem automaticamente o comando precedente quando o usuário executa o aplicativo *Java,* como fariam com qualquer outro aplicativo nessa plataforma. Por exemplo, no Microsoft Windows, o usuário pode executar um aplicativo Java a partir de um arquivo *JAR* dando clique duplo no nome de arquivo *JAR* no Windows Explorer.

Na linha 3 do arquivo manifesto especifica o ***Name:*** do arquivo contendo a classe do *bean* incluindo a extensão de nome de arquivo *.class* utilizando seu nome de pacote e de classe. Observe que os pontos “ . ” utilizados em um nome de pacote são substituídos por barras normais “ / “ para o ***Name***: no arquivo manifesto.

A linha 4 especifica que a classe nomeada na linha 3 é, de fato, um JavaBean "**Java-Bean: True**". É possível ter em um arquivo *JAR* classes que não são JavaBeans. Essas classes são geralmente utilizadas para suportar os JavaBeans no repositório de arquivos.

Por exemplo, um *bean* de lista encadeada talvez tenha uma classe de nodo de lista vinculada de suporte cujos objetos são utilizados para representar cada nodo na lista. Cada classe listada no arquivo manifesto deve ser separada de todas as outras classes por uma linha em branco. Se a classe é um *bean*, sua linha ***Name:*** deve ser imediatamente seguida pela sua linha **Java-Bean:**.

Um arquivo *JAR* para o *bean LogoAnimator* é realizado com o utilitário *jar* na linha de comando do *MS-DOS* com o seguinte o comando que cria o arquivo *JAR*:

jar cfm LogoAnimator.jar manifest.tmp jhtp3beans.**

No comando precedente, jar é o utilitário de repositório de arquivos Java utilizado para criar arquivos JAR. Em seguida, estão as opções para o utilitário jar, cfm. A letra "**c"** indica que esta se criando um arquivo JAR. A letra "**f"** indica que o próximo argumento na linha de comando "*LogoAnimator.jar"* é o nome do arquivo JAR a ser criado. A letra "**m"** indica que o próximo argumento na linha de comando é o arquivo *manifest.tmp*, que é utilizado pelo *jar* para criar o arquivo *MANIFEST.MF* no diretório *META-INF* do *JAR*. Especificamos *jhtp3beans*.**, para indicar que todos os arquivos no diretório *jhtp3beans* devem ser incluídos no arquivo *JAR*. O diretório de pacote *jhtp3beans* contém os arquivos *.class* para o *LogoAnimator* e suas classes de suporte, bem como as imagens utilizadas na animação (Deitel et. al., 2001).

4.2.3 JAVABEANS BRIDGE FOR ACTIVEX

Os componentes de JavaBeans podem ser usados como objetos embutidos nos componentes de ActiveX dentro de aplicações Microsoft como Microsoft Office, Internet Explorer, e Visual Basic e outros, utilizando do Java Plug-in (*JavaBeans Bridge for ActiveX*) que funciona como uma ponte e contêm instruções para empacotar e registrar JavaBeans como componentes de *ActiveX*. Para usar um *JavaBean* como um objeto embutido, empacota-se o *JavaBean* e então poderá usar como um componente de *ActiveX* (SUN,2004).

Um arquivo de registro (*registry file*) e uma biblioteca de tipo é criada pelo *packager* (JavaBeans Bridge for ActiveX). Este registro contém um identificação de objeto, o caminho executável para o componente, informação de ponte, e um caminho de biblioteca de tipo. O arquivo de *TypeLib* é um arquivo binário que descreve as propriedades de cada componente, eventos, e métodos (SUN, 2004). A figura 4.3 mostra a *packager*.

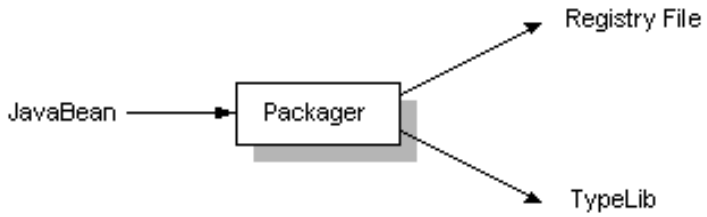


Figura 24.3 : Após utilizar a packager cria-se um "Registry File" e "TypeLib"

4.2.4 EMPACOTANDO E REGISTRANDO JAVABEANS

O primeiro passo é inicializar o *JavaBeans Bridge for ActiveX* (*packager*), para isso utiliza-se o seguinte comando no diretório de instalação do *Java Plug-in* que na maioria dos casos, será "*c:\program files\javasoft\jre\1.2 *" (SUN, 2004).

bin \java.exe -cp lib\rt.jar;lib\swingall.jar;lib\jaws.jar sun.beans.ole.Packager

Após digitar o comando acima dentro do diretório do Java Plug-in, o *packager* o guiará por cinco passos para empacotar seu *JavaBeans*, abaixo estão os 5 passos para empacotar o *JavaBeans*:

O primeiro passo é especificar o arquivo de JAR que contém o *JavaBean* ser empacotado e clik em Next (SUN, 2004).

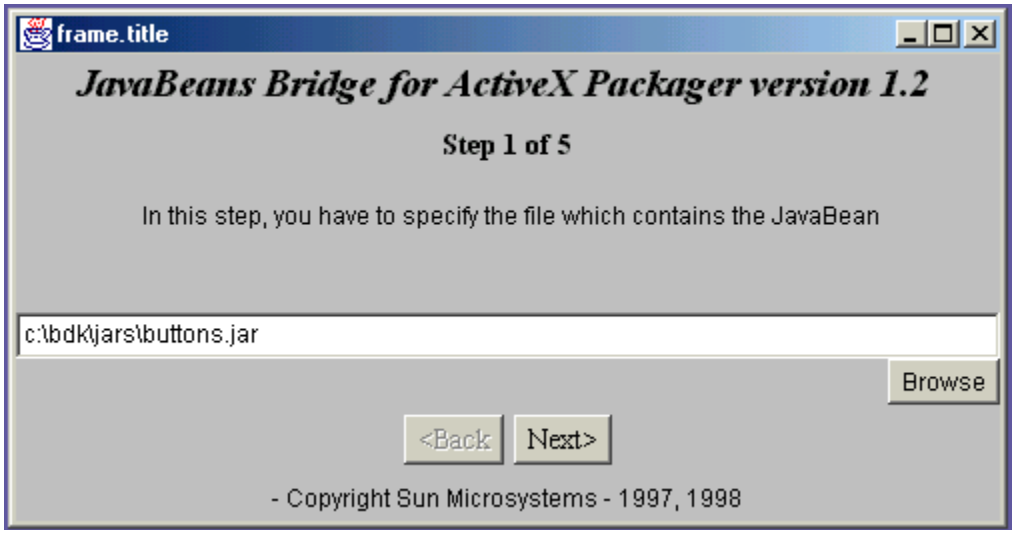


Figura 25.4 : JavaBeans Bridge for ActiveX, 1º passo.

O segundo passo é selecionar o componente de JavaBeans: A ferramenta JavaBeans Bridge for ActiveX lista todo os JavaBeans dentro do arquivo de JAR especificados no primeiro passo. Selecione o JavaBean ser empacotado e click em Next (SUN, 2004).

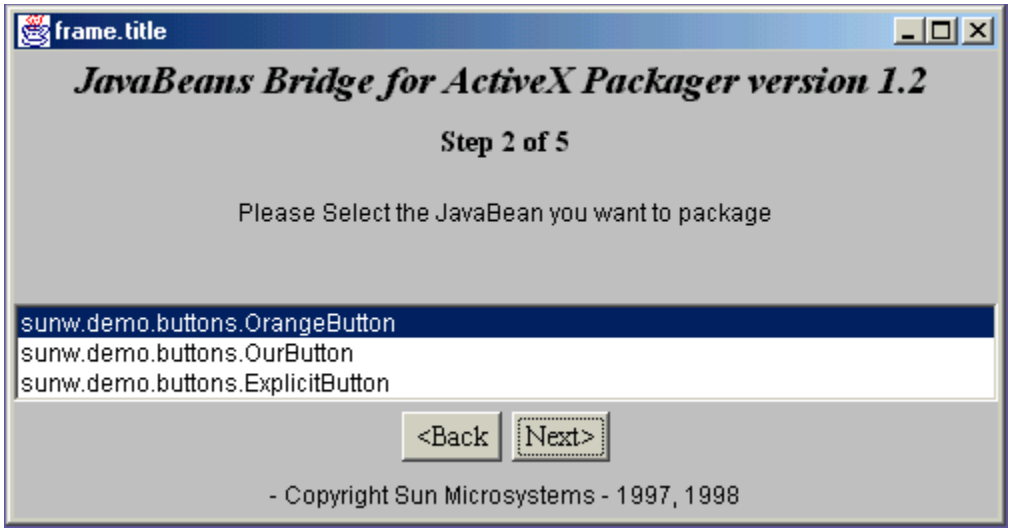


Figura 26.5 : JavaBeans Bridge for ActiveX, 2º passo.

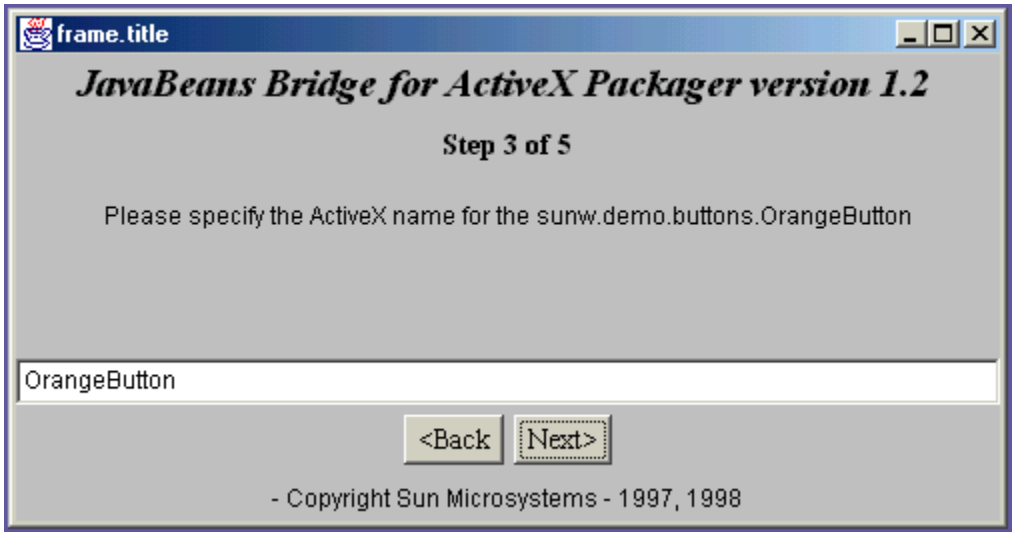


Figura 27.6 : JavaBeans Bridge for ActiveX, 3º passo.

O terceiro passo é especificar um nome de ActiveX. A ferramenta JavaBeans Bridge for ActiveX sugestiona um nome de ActiveX para o JavaBean. Este nome será usado no ambiente de ActiveX para se referir ao componente. O *packager* (*JavaBeans Bridge for ActiveX*) lhe dá um nome padrão (o nome do JavaBean menos seu prefixo java) (SUN, 2004).

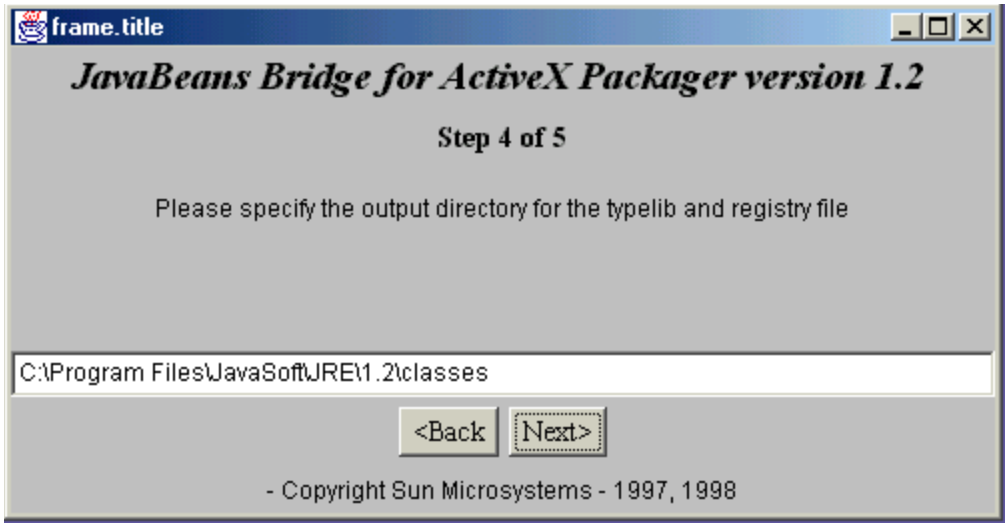


Figura 28.7: JavaBeans Bridge for ActiveX, 4º passo.

O quarto passo é especificar um diretório de *OutPut*. O packager (*JavaBeans Bridge for ActiveX*) cria um arquivo de registro (*.reg*) e uma biblioteca de tipo (*.tlb*) em um diretório que foi especificado. Nota que o arquivo de registro contém informação sobre a

localização do arquivo de biblioteca de tipo. Se o usuário move o arquivo de biblioteca de tipo, o usuário precisará atualizar o arquivo de registro manualmente e registrar isto novamente. A localização do *beans.ocx* (Java Tomada-em componente de ActiveX) é determinado das por padrão no diretório de instalação do *Java Plug-in* (*Packager*, *JavaBeans Bridge for ActiveX*). Se mover o beans.ocx, precisará modificar este arquivo manualmente (SUN, 2004).

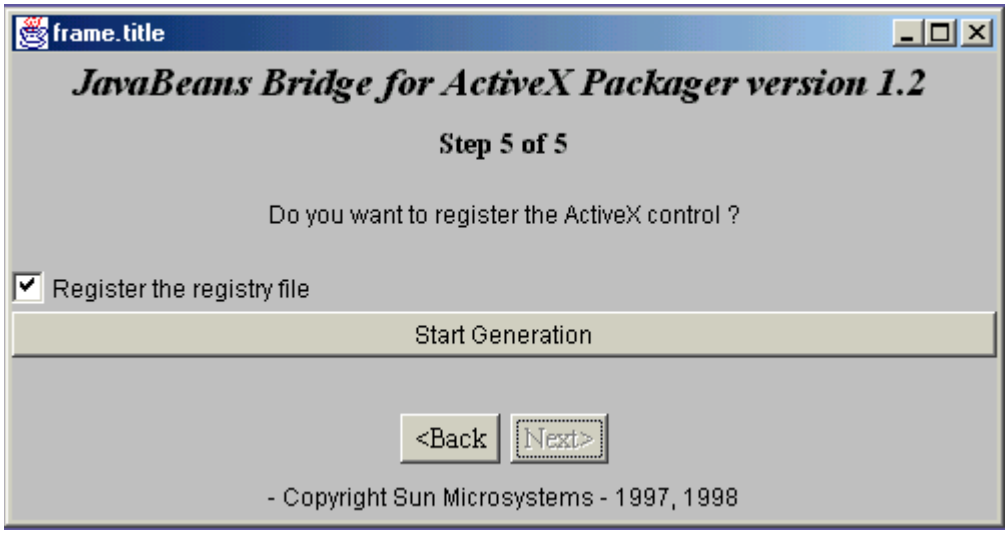


Figura 29.8 : JavaBeans Bridge for ActiveX, 5º passo.

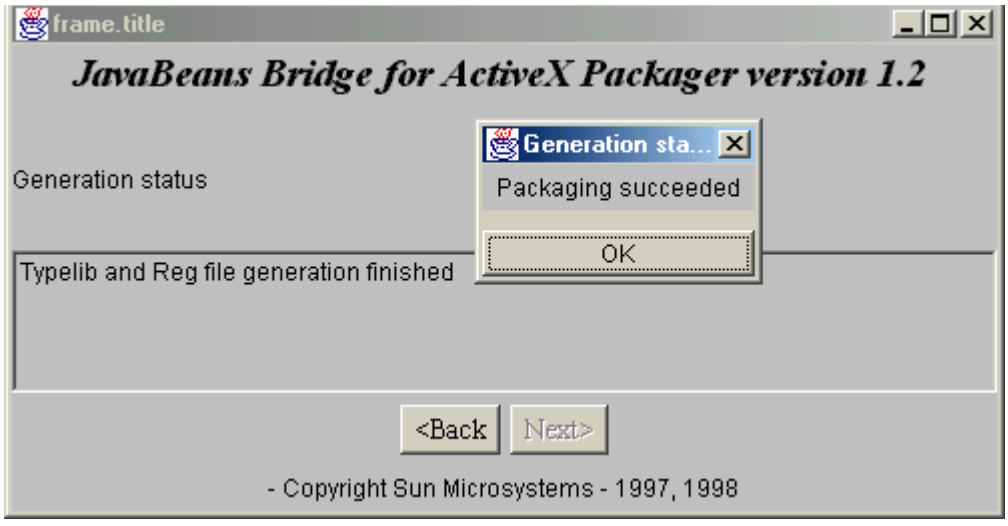


Figura 30.9 : JavaBeans empacotado com sucesso.

O quinto passo é o momento da geração do Beans-ActiveX, (*Start Generation*). O packager (*JavaBeans Bridge for ActiveX*) pode registrar o componente automaticamente na

máquina local. Escolhendo registrar o arquivo de registro, o componente de *JavaBeans* será capaz de ser usado automaticamente em um recipiente de ActiveX. Após o quinto passo aparecerá a janela da figura 3.9 informando que o JavaBeans foi empacotado com sucesso (SUN, 2004).

Após o componente Compra_Venda ser implementado ele deverá ser submetido as 5 etapas deste empacotamento, assim será possível inserir em uma arquitetura ActiveX. Para seu funcionamento em um ambiente ActiveX e comunicação com um outro componente de tecnologia COM se faz necessário vários testes e possivelmente a alterações na implementação do componente, sendo esta a última etapa do processo, o desenvolvimento do componente.

4.3 COMUNNICAÇÃO DO COMPONETE COMPRA_VENDA COM O COMPONENTE LOCAR_RESERVAR_RECURSOS

Para que o Componente Compra_Venda possa se comunicar com o componente Locar_Reservar_Recurso deve-se usar a tecnologia JavaBeans Bridge for ActiveX da forma que foi descrita na seção anterior. As interfaces requeridas do Componente Compra_Venda irá se comunicar diretamente com as interfaces IBase e IRecursos_Envolvidos_Na_Transação providas do Componente Locar_Reservar_Recurso, já as interfaces providas irão se comunicar com a aplicação da Vídeo Locadora desenvolvida por (Melo, 2003) para demonstrar o funcionamento e integração dos componentes Locar_Reservar_Recurso e Compra_Venda na figura 4.10 tem-se um desenho esquemático.

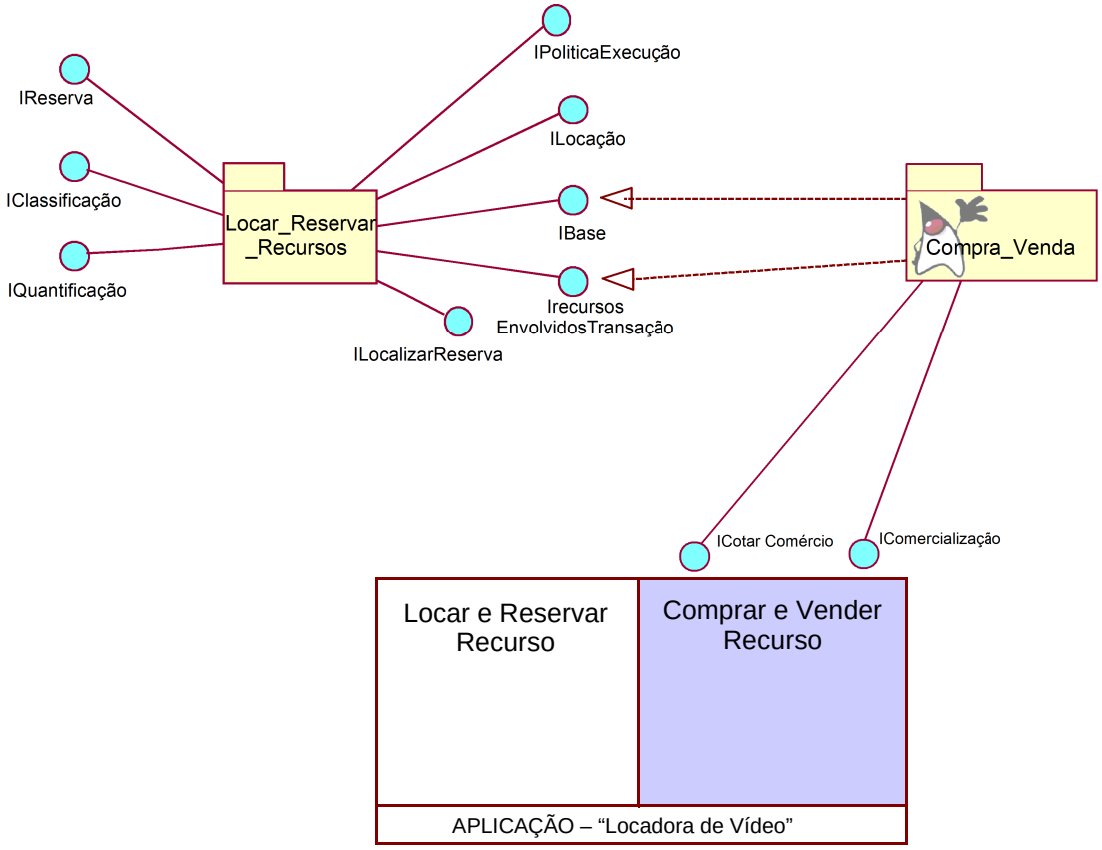


Figura 31.10– Desenho esquemático dos Componentes e a Aplicação

4.4 CARACTERÍSTICAS DO COMPONENTE COMPRA_VENDA

A seguir serão descritas algumas características do Componente Locar_Reservar_Recursos desenvolvido por (Melo, 2003), que são as mesmas para o componente Compra_Venda estruturado usando as mesmas etapas de desenvolvimento (Melo,2003):

Manutenibilidade: O uso dos Padrões de Projeto na estrutura do Componente trouxe mais flexibilidade ao mesmo. Modificações na sua estrutura podem ser feitas sem afetar as aplicações que já utilizavam o Componente, desde que não sejam alteradas as Interfaces que ele possuía. componente Compra_Venda, empacotamento e testes da tecnologia;

Portabilidade: Como o Componente foi desenvolvido com a Tecnologia JavaBeans, ele poderá ser usado em qualquer outro sistema operacional. As aplicações podem ser desenvolvidas em qualquer linguagem de programação, podendo ser utilizado para criar diversas aplicações, desde que sejam especificas ao domínio de Compra e funcionar em um ambiente ActiveX utilizando a ferramenta JavaBeans Bridge for ActiveX.

Reusabilidade: É principal característica do Componente, tendo a possibilidade de ser utilizado em qualquer aplicação que vá realizar compra e venda, fazendo a comunicação através de suas interfaces.

4.5 CONSIDERAÇÕES FINAIS

O desenvolvimento do Componente Compra_Venda foi suficiente para extrair as conclusões desejadas e verificar as vantagens de desenvolver Softwares utilizando-se de Componentes e, principalmente de Componentes estruturados com Padrões de Projetos.

Do Padrão de Modelagem de Objetos Componentes JavaBeans utilizado, pode-se dizer que é uma Tecnologia eficiente, segura e uma possibilidade muito grande de evolução.

5 CONCLUSÕES

Foram apresentados neste trabalho, o desenvolvimento de um Componente estruturado com Padrões de Projeto, utilizando o padrão JavaBeans, a utilização de uma ferramenta para comunicação deste componente JavaBeans com outro componente de tecnoligia COM com a finalidade de enfatizar vantagens de se estruturar um Componente com Padrões de Projeto e descrito em (Melo, 2003), e verificar a comunicação e operação entre tecnologias diferentes.

As principais conclusões que este experimento proporcionou são:

- O uso de Padrões de Projeto no desenvolvimento de Componentes provou ser uma técnica significativamente vantajosa quando se pretende reduzir a complexidade dos esforços despendidos durante o desenvolvimento de um Software de qualidade como foi comprovado por (Melo, 2003).

- Pode-se averiguar um aumento na facilidade manutenção documentação.

- Pode-se constatar também, que o paradigma da Programação Orientada em Componentes pode-se tornar popular à medida que ferramentas como o Java Plug-in (*JavaBeans Bridge for ActiveX*) se tornarem mais eficientes e imparciais a linguagem de programação e sistema operacional.

Como foi comprovado por (Melo, 2003) as vantagens obtidas são percebidas a médio e a longo prazo, já que a complexidade e o tempo despendido para a criação do Componente ou dos Componentes não é trivial. Mas na medida que aumenta a necessidade de desenvolvimento de aplicações pela web e que terão de funcionar em diferentes plataformas, haverá a necessidade de interligar linguagens, e softwares, sendo assim as vantagens de se desenvolver componentes estruturados com padrões de projetos e que se comunicam com qualquer tecnologia tornará imprescindível e fundamental como visto no decorrer deste trabalho de final de curso.

5.1 CONTRIBUIÇÕES

Dentre as contribuições oferecidas por este projeto, pode-se destacar, um estudo analítico sobre desenvolvimento de sistemas no paradigma da Programação Orientada por Componentes e suas vantagens; a utilização de padrões de projetos aplicados à sua estrutura e a possibilidade do leitor entender tal necessidade dando contribuição e incentivo a programadores a desenvolver componentes que compõem a Arquitetura Estruturada em Componentes utilizando Padrões de Projeto e criação de um Framework que facilite a utilização destes componentes no desenvolvimento de aplicações dentro do domínio de Gestão de Recursos de Negócio. Oferece um incentivo para o desenvolvimento de componentes no padrão JavaBeans com alto grau de portabilidade, viabilizando a criação de outros Frameworks deste porte, dentro de outros domínios, o que será um grande avanço na Engenharia de Software e o fortalecimento do paradigma de Programação Orientada por Componentes.

5.2 TRABALHOS FUTUROS

Trabalhos podem ser feitos para dar continuidade a esta monografia:

- Implementação do Componente Compra_Venda, empacotamento e testes.
- Implementar demais componentes propostos (Coelho,2002) em tecnologias diferentes com CORBA, ENTERPRISE JAVABEANS, DCOM, verificando a comunicação em sistemas distribuídos;
- Desenvolver um sistema que englobem todos os componentes do domínio de Gestão de Recursos Negócios;

REFERÊNCIAS BIBLIOGRÁFICAS

(Melo, 2003) Melo, A.T. Desenvolvimento de um Componente para o Domínio de Gestão de Recursos e Negócios. Dissertação de fim de curso, Universidade Presidente Antônio Carlos, Julho de 2003.

(Júnior, 2001) Júnior, J.N.R.O.; Sistemas Distribuídos CORBA. Universidade de Brasília – Faculdade de Tecnologia.

(Morrison *et. al*., 1999) Morrison, Michael; Weems, Randy; Coffee, Peter; Leong, Jack.; Como Programar em Java Beans, MAKRON Books, 1999.

(Horstmann,1999) Horstmann, Cay S., Cornell, Gary. Core Java, VolII..

(Grei Voss,2004) www.javasoft.com - www.sun.com - Treinamento On Line JavaBeans, Javasoft - Grei Voss

(Vinagreiro,2004) Marcelo Luiz VinagreiroUniversidade de São Paulo – USP - Arquitetura Enterprise JavaBeans http://www.ime.usp.br/~mlv/ejb/

(Jacques,2003) Jacqueline - http://www.dsc.ufpb.br/~jacques/index.htm Visitado em Janeiro de 2004

(Argo Navis,2004) www.argonavis.com.br, Helder da Rocha – Visitado em Junho de 2004

(Sun,2004) http://developer.java.sun.com/developer/onlineTraining/Beans, Visitado em Janeiro de 2004

http://java.sun.com/beans/bridge - http://java.sun.com/beans - http://java.sun.com

(Devx,2004). www.devx.com - Boris Feldman - Visitado em Janeiro de 2004

(Activex,2004) www.activex.com

(JavaMan,2004) http://javaman.com.br http://javaman.com.br/artigos - Visitado em Janeiro de 2004

(Deitel *et. al*., 2001) Deitel, H, M.; Deitel, P. J.; Java Como Programar. 3º Edição Bookman, 2001.

(Brown et. al., 1998) Brown, A.W., Wallnau, K.C. The Current State of CBSE. IEEE Software, September/October 1998.

(Coelho *et. al*., 2002) Coelho, F.M. Uso de Componentes de Software no Desenvolvimento de Frameworks Orientados a Objetos. Instituto de Computação, Universidade Estadual de Campinas, Dezembro de 2002.

(Fayad *et. al*., 1999) Fayad, M. E., Schmidt, D.C., Johnson, R.E. Building Application Frameworks. John Wiley & Sons, 1999.

(Fowler *et. al*., 2000) Martin, F. UML Essencial. Um breve guia para a linguagem-padrão de modelagem de objetos. 2 Ed., Bookman, 2000.

(Johnson. *et. al*., 2000) Ralph, J. Padrões de Projeto. Soluções Reutilizáveis de Software Orientado a Objetos. Bookman, 2000.

(Presmam, 2001) Presmam, RS. Software Engineering – A Practitioner's Approach. Fifth Edition, McGraw Hill, 2001

(Werner *et. al*., 2000) Werner, C. M. L.; Braga,R. M. M.; Desenvolvimento Basedo em Componente – COPPE/UFRJ – Programa de Engenharia de Sistemas.

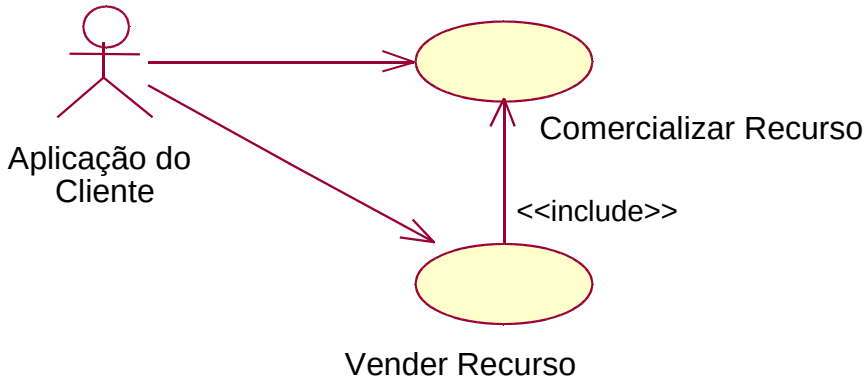
(Gamma. *et. al*., 2000) Gamma, E., Helm, R., R., Johnson, R., Vlissides, J. Padrões de Projeto. Soluções Reutilizáveis de Software Orientado a Objetos. Bookman, 2000.

(Buschmann *et. al.*, 1996) Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M. A System of Patterns: Patterns-Oriented Software. John Wiley & Sons, 1996.

ANEXO A –DIAGRAMA DE CASO DE USO DO COMPONENTE COMPRA_VENDA

Os Diagramas de Caso de Uso mostrados a seguir fornecem uma visão do comportamento do sistema, suas funcionalidades essenciais e a interação dos elementos envolvidos com o sistema.

Pacote: Comercialização_Venda



Aplicação do Cliente: É a aplicação (Software) do Cliente que utiliza o Componente Compra_Venda. Solicita a execução de tarefas específicas através de chamadas às operações das Interfaces do Componente passando os dados necessários à execução dessas tarefas. Requisita uma cotação que resultara em uma compra ou venda.

Pacote: Comercialização_Venda

Caso de uso: Comercializar Recuro

Atores: Aplicação do Cliente

Pré-Condição: O Recurso a ser comercializado deve estar disponível para venda.

Invariante: Não há

Pós-Condição: A comercialização de venda do Recurso estar Cadastrada.

Descrição:

1- A Aplicação do Cliente faz uma chamada à operação de Comercializar recurso passando a descrição da comercialização, a valor da venda, período da venda, etc.

2- O sistema retorna um valor de controle para a Aplicação do Cliente indicando sucesso ou insucesso no processo de comercialização do recurso.

Caso de uso: Vender Recurso

Atores: Aplicação do Cliente

Pré-Condição: O processo de comercialização do recurso já ter sido iniciado.

Invariante: Não há

Pós-Condição: O Recurso estará vendido.

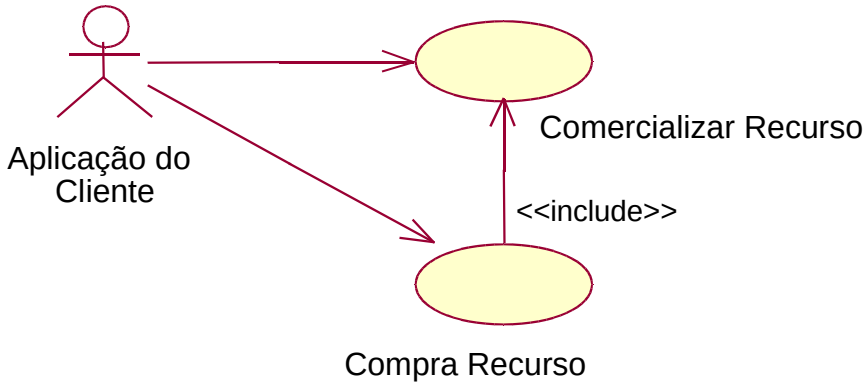
Descrição:

1- A Aplicação do Cliente faz uma chamada à operação de consultar uma comercialização do recurso passando o seu ID respectivo.

2- A Aplicação do Cliente faz uma chamada à operação de vender recurso passando o ID da venda e o ID da comercialização do recurso que está associado.

3- O sistema retorna um valor de controle para a Aplicação do Cliente indicando sucesso ou insucesso na venda do recurso.

Pacote: Comercialização_Compra



Pacote: Comercialização_Compra

Caso de uso: Comercializar Recurso

Atores: Aplicação do Cliente

Pré-Condição: A comercialização do Recurso estar cadastrada e o recurso não exceder o limite máximo em estoque.

Invariante: Não há

Pós-Condição: A comercialização de compra do Recurso ser Cadastrada.

Descrição:

1- A Aplicação do Cliente faz uma chamada à operação de Comercializar recurso passando a descrição da comercialização, a valor da compra, período da compra, etc.

2- O sistema retorna um valor de controle para a Aplicação do Cliente indicando sucesso ou insucesso no processo de comercialização do recurso.

Caso de uso: Comprar Recuro

Atores: Aplicação do Cliente

Pré-Condição: O processo de comercialização do recurso já ter sido iniciado.

Invariante: Não há

Pós-Condição: O Recurso estará comprado.

Descrição:

1- A Aplicação do Cliente faz uma chamada à operação de consultar uma comercialização do recurso passando o seu ID respectivo.

2- A Aplicação do Cliente faz uma chamada à operação de comprar recurso passando o ID da compra e o ID da comercialização do recurso que está associado.

3- O sistema retorna um valor de controle para a Aplicação do Cliente indicando sucesso ou insucesso na compra do recurso.

Pacote: Cotar Recurso

Pacote: Cotar Recurso

Caso de uso: Cotação

Atores: Aplicação do Cliente

Pré-Condição: O Recurso estar cadastrado para comercialização de Compra ou de Venda.

Invariante: Não há

Pós-Condição: A cotação estará Cadastrada.

Descrição:

1- A Aplicação do Cliente faz uma chamada à operação de Cotação passando a descrição do recurso e tipo de comercialização.

2- O sistema retorna um valor da cotação para a Aplicação do Cliente indicando sucesso ou insucesso no processo de cotação do recurso.