Universidade Federal de Campina Grande
Centro de Ciências e Tecnologia
Curso de Mestrado em Informática
Coordenação de Pós-Graduação em Informática

Ferramenta para Aumento da Produtividade no Desenvolvimento de Aplicações Web Sobre a Plataforma J2EE

José Maria Rodrigues Santos Júnior

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Informática do Centro de Ciências e Tecnologia da Universidade Federal de Campina Grande como requisito parcial para a obtenção do grau de Mestre em Ciências (MSc).

Área de Concentração: Ciência da Computação
Linha de Pesquisa: Sistemas de Software

Prof. Dr. Francisco Vilar Brasileiro
(orientador)

Campina Grande, Paraíba, Brasil
Agosto de 2002

Ficha Catalográfica

SANTOS JÚNIOR, José Maria Rodrigues

S237F

Ferramenta para Aumento da Produtividade no Desenvolvimento de Aplicações WEB Sobre a Plataforma J2EE

Dissertação (Mestrado), Universidade Federal de Campina Grande, Centro de Ciências e Tecnologia, Coordenação de Pós-Graduação em Informática, Campina Grande, Paraíba, Agosto de 2002.

140p. Il.

Orientador: Francisco Vilar Brasileiro

Palavras-Chave:

1. Engenharia de Software
2. Ferramenta de Desenvolvimento
3. Produtividade
4. Java 2EE
5. EJB

CDU - 519.683

"FERRAMENTA PARA AUMENTO DA PRODUTIVIDADE NO DESENVOLVIMENTO DE APLICAÇÕES WEB SOBRE A PLATAFORMA J2EE"

JOSÉ MARIA RODRIGUES SANTOS JÚNIOR

DISSERTAÇÃO APROVADA EM 28.08.2002

PROF. FRANCISCO VILAR BRASILEIRO, Ph.D
Orientador

PROF. JACQUES PHILIPPE SAUVÉ, Ph.D
Examinador

PROF. ÁLVARO FRANCISCO DE C. MEDEIROS, D.Sc
Examinador

CAMPINA GRANDE – PB

Ferramenta para Aumento da Produtividade no Desenvolvimento de Aplicações WEB Sobre a Plataforma J2EE

José Maria Rodrigues Santos Júnior

Dissertação aprovada em ___ /___/___

Prof. Dr. Francisco Vilar Brasileiro
(orientador)

Dr. Álvaro F. C. Medeiros

Dr. Jacques P. Sauvé

"The better entrepreneurs understand the Internet – its strengths and limits – the greater their ultimate success. Those who wait may find it increasingly difficult to catch up with the competition later."

Peter Rowley,
IBM General Manager

1. Agradecimentos

Em primeiro lugar agradeço à Divindade, por ter me dado uma vida: saudável, fácil e repleta de felicidades e pessoas especiais;

Aos meus pais José Maria Rodrigues Santos (em memória) e Nilzete Fonseca Santos (em memória), por terem me dado exemplos de vida a serem seguidos e uma excelente educação. Sinto a presença de vocês ao meu lado principalmente nos momentos de dificuldades.

Aos meus irmãos Zenilton, Alcilene, Alciene e Alcione, pelo exemplo de coragem, dedicação e gosto pela vida e pelo carinho e amor, quase materno, dedicado a mim. Vocês são a minha família. Mesmo sem muito contato a presença de vocês é fundamental para minha felicidade.

A minha noiva Leila, por ter trazido serenidade, paz, amor e felicidade a minha vida, em níveis que eu jamais imaginei. Você é meu complemento. Te Amo.

Aos meus amigos de todas as horas Mary e Juan , Kátia e Ricardo (Macambira), Railde e Dudu, Cândida e André (Babão) e Cláudio Mitidieri, Chiquinho e Igor.

Ao ex-aluno e amigo George Leite (Zoião) por ter sempre acreditado no EasyNet e persistido na sua implementação, mesmo quando eu cheguei a desacreditar. Posso garantir que o discípulo superou o mestre. Sem seu talento como programador o EasyNet não seria o que é.

Ao meu orientador Francisco Vilar Brasileiro, pela paciência e confiança, mesmo nos momentos em que não correspondi as suas expectativas. A sua orientação foi fundamental para que os obstáculos da distância e falta de tempo fossem vencidos.

A Eduardo Bernardes, coordenador do curso de Sistema de Informação da Universidade Tiradentes, por ter confiado num recém-formado e me colocado na vida acadêmica e pelos bons aconselhamentos durante toda a minha carreira profissional.

A equipe da Infonet (www.infonet.com.br - Nivaldo, Jailson, Rodrigo, Levi e André), por ter mostrado o caminho das pedras. O pioneirismo, coragem e confiança de vocês, no desenvolvimento do iWorkplace (www.iworkplace.com.br) e da automação bancária do BANESE (www.banese.com.br), mostrou ser possível implementar o EasyNet.

2. Sumário

3.

Lista de Abreviaturas ou Siglas

API - *Application Program Interface*
BMP - *Bean Manager Persistence*
CGI - *Common Gateway Interface*
CMP - *Container Manager Persistence*
CORBA - *Common Object Request Broker Architecture*
CSS - *Cascade Style Sheet*
DD - *Deployment Descriptors*
EJB - *Enterprise Java Beans*
EJBQL - *Enterprise Java Beans Query Language*
HTML - *Hypertext Markup Language*
HTTP - *Hyper-Text Transport Protocol*
IEEE - *Institute of Electrical and Electronics Engineers*
IDE - *Integrated Development Environment*
IDL - *Interface Definition Language*
IETF - *Internet Engineering Task Force*
IIOP - *Internet Inter-Orb Protocol*
J2EE - *Java 2 Enterprise Edition*
J2SE - *Java 2 Standard Edition*
JAAS - *Java Authentication and Authorization Service*
JAF - JavaBeans Activation Framework
JAR - *Java Archive*
JCP - *Java Community Process*
JDBC - *Java Database Connectivity*
JDK - *Java Development Kit*
JMS - *Java Message Service*
JNDI - *Java Naming and Directory Interface*
JSP - *Java Server Pages*
JTA - *Java Transaction API*
JTS - *Java Transaction Service*
JVM - *Java Virtual Machine*
OQL - *Object Query Language*
OTS - *Object Transaction Service*
RMI - *Remote Method Invocation*
SGBD - Sistema Gerenciador de Banco de Dados
SQL - *Struct Query Language*
TCP/IP - *Transmission Control Protocol / Internet Protocol*
URL - *Uniform Resource Locator*
W3C - *World Wide Web Consortium*
WWW - *World Wide Web*
XML - *Extensible Markup Language*

4. Lista de Figuras

5. Lista de Tabelas

6. Lista de Códigos Fonte

7. Resumo

Este trabalho discute a necessidade de estratégias e ferramentas para o desenvolvimento mais produtivo e menos complexo de aplicações Web sobre a plataforma Java 2 Enterprise Edition (J2EE). Inicialmente foi feito um estudo da plataforma J2EE, principalmente da tecnologia Enterprise Java Beans (EJB).

Uma aplicação Web foi implementada utilizando diretamente os recursos da plataforma J2EE, permitindo identificar aspectos a serem explorados para alcançar os objetivos propostos.

Em seguida foi desenvolvida uma ferramenta, denominada de EasyNet, visando explorar esses aspectos. O EasyNet foi utilizado para a re-implementação da aplicação, permitindo que o aumento da produtividade e a redução da complexidade fossem comparados aos da implementação sem a sua utilização.

Foi possível identificar que o EasyNet forneceu um aumento significativo da produtividade e reduziu a complexidade no desenvolvimento de aplicações Web sobre a plataforma J2EE.

8. Abstract

This work discusses the need of strategies and tools for a more productive and less complex development of Web applications on Java 2 Enterprise Edition (J2EE). A study on the J2EE platform was initially made, especially on the Enterprise Java Beans (EJB) technology.

A Web application was implemented using directly the sources from the J2EE platform, allowing the identification of aspects to be explored in order to reach the desired results.

After, a tool denominated EasyNet was developed aiming the exploration of such aspects. The EasyNet was used for the re-implementation of the application, allowing the productivity growth and the decrease of the complexity to be compared to the ones of the implementation made without its use.

It was possible to identify that the EasyNet provided a significant increase of the productivity and reduced the complexity of the development of the Web applications on the J2EE platform.

Capítulo 1 - Introdução

1.1 Introdução

O desenvolvimento de aplicações, historicamente, tem se mostrado uma tarefa bastante desafiadora e grande parte dos esforços da ciência da computação destinou-se a resolver problemas encontrados nesta área. Muita coisa mudou, em termos de tecnologia, desde o tempo em que a programação era feita exclusivamente em linguagem de máquina, onde o desenvolvedor necessitava de conhecimento relativos até mesmo à arquitetura do processador utilizado e os recursos oferecidos para a programação eram bastante limitados.

A primeira grande mudança foi a criação das linguagens de programação, onde a forma de escrita da solução a ser executada pela máquina teve seu nível de abstração elevado, trazendo as soluções para um formato mais humano e de maior facilidade de uso. Com a utilização de uma linguagem de programação, a implementação de aplicações passou a contar com recursos de níveis mais altos e a tarefa de transformação dessa solução em uma solução que o computador pudesse executar passou a ser uma tarefa automatizada através de ferramentas de tradução.

As primeiras linguagens de programação eram bastante limitadas, tanto em nível de abstração como em recursos oferecidos, mas ainda assim, mostravam-se extremamente mais amigáveis que a programação em linguagem de máquina. Assim, as linguagens de programação foram evoluindo. Em termos de desenvolvimento de aplicações comerciais a principal direção tomada na evolução das linguagens de programação foi à facilidade de uso e aumento da produtividade. A demanda e complexidade das aplicações comerciais aumentavam dia a dia. A programação de computadores passou a ser mais popularizada, permitindo que pessoas sem maiores conhecimentos sobre o funcionamento de computadores pudessem construir soluções computacionais.

Mesmo com as linguagens de programação tendo atingido níveis elevados em relação à facilidade de uso, o que aumentou em muito a produtividade na construção de soluções computacionais, a construção de software continuou sendo uma tarefa altamente desafiadora e problemática, o que direcionou a evolução das linguagens de programação para outro rumo, ou seja, foram propostas mudanças no paradigma utilizado nas linguagens de programação. Agora as evoluções estavam voltadas para a forma conceitual como os programas modelavam

as soluções de problemas do mundo real. Assim nasceu o paradigma da programação estruturada e posteriormente o paradigma orientado a objetos. Com a orientação a objetos, ficou mais natural a modelagem de soluções do mundo real no mundo computacional, pois este paradigma representa uma forma mais natural de implementar componentes de software.

Com a utilização de linguagens orientadas a objetos alcançamos níveis de abstração bem elevados nas linguagens de programação, mas outras evoluções foram necessárias, a fim de ajustar os recursos oferecidos pelas tecnologias de desenvolvimento de software às necessidades das empresas e do mercado. Com o surgimento das redes de computadores e principalmente com a Internet, as empresas passaram a interagir de forma mais automatizada criando uma nova economia: a economia em rede, onde a troca de informações é tão importante quanto a troca de bens e serviços.

Nessa nova economia as empresas têm a necessidade de interagir eletronicamente com clientes, fornecedores e parceiros, onde estes também são empresas, as quais possuem suas próprias aplicações. Diante dessa necessidade de integração entre empresas, a evolução das tecnologias relacionadas ao desenvolvimento de aplicações foi direcionada para a arquitetura utilizada na construção das aplicações.

Pois agora, as aplicações de uma empresa necessitam interagir diretamente com outras aplicações da própria empresa ou com aplicações de outras empresas, usando assim, o conceito de aplicações distribuídas. Essas aplicações tornam-se extensões umas das outras, funcionando como um sistema unificado para atender a demanda de sistemas comerciais nessa nova economia.

Num primeiro momento, podemos imaginar antes do uso em larga escala da Internet como ferramenta de integração comercial, as empresas preocupavam-se apenas em “arrumar a própria casa”, assim foram desenvolvidas diversas aplicações com o objetivo de automatizar as tarefas internas à empresa, tais como: folha de pagamento, controle de estoque, contas a pagar/receber, etc. Diante dessa necessidade foi criada a arquitetura de aplicações denominada de Cliente/Servidor, onde a solução de software foi dividida em duas camadas, tipicamente, uma camada desempenhava a tarefa de interagir com o usuário da aplicação e executar regras de negócios e a outra era responsável pelo gerenciamento dos dados.

A arquitetura cliente servidor mostrou-se bastante adequada para a solução de problemas dentro do escopo das empresas. Assim, diversas ferramentas surgiram baseadas em tecnologias que implementavam a arquitetura Cliente/Servidor e algumas delas atingiram grande maturidade e facilidade de uso. Os requisitos para o desenvolvimento de aplicações internas às empresas eram adequadamente atendidos por tais ferramentas.

Recentemente, com a crescente utilização da Internet como ferramenta na realização de transações comerciais, as necessidades computacionais das empresas, mais uma vez, foram expandidas, pois na era da Internet e da economia da informação as empresas tiveram que implementar soluções que iam além de suas fronteiras, permitindo interação com clientes, parceiros, fornecedores e funcionários. Assim as tecnologias de desenvolvimento de software tiveram que, mais uma vez, evoluir a fim de atender essa nova dimensão dos requisitos de soluções computacionais exigidos pelas empresas nessa nova economia.

Ficou evidente que a arquitetura Cliente/Servidor, a qual era perfeitamente adequada para a solução de problemas internos à empresa, não era adequada para esse novo ambiente, a Internet, o qual possui características e requisitos bastante diferenciados. Diante dessas novas necessidades: interligar-se em rede; interação eficiente com clientes, funcionários, fornecedores e parceiros; integração com sistemas legados; regras de negócios mais complexas e integração com diversos tipos de software clientes, as tecnologias de desenvolvimento de aplicações evoluíram da arquitetura Cliente/Servidor para a arquitetura Multicamadas **[SUN01]**.

Na arquitetura Multicamadas os componentes de software são divididos em camadas, tipicamente tendo-se três camadas denominadas: Camada Cliente (responsável pela interface com o cliente), Camada de Regra de Negócio (executa a lógica das transações) e a camada de Dados (realiza a persistência e gerência dos dados).

Na arquitetura Cliente/Servidor a ênfase da tecnologia de informação foi dada ao gerenciamento de dados. Os softwares responsáveis por essa gerência, denominados de Sistemas Gerenciadores de Banco de Dados – SGBDs **[ULL01]**, evoluíram muito, alcançando altos níveis de confiabilidade, performance, produtividade e facilidade de uso. Na arquitetura Multicamadas, embora os SGBDs continuem a desempenhar um papel fundamental, a ênfase da tecnologia de informação passa para as aplicações, onde os principais fatores determinantes dessa mudança são:

- Acesso a dados em locais muitas vezes remotos e em formatos diferentes;
- Utilização de software clientes variados;
- Problemas com escala na utilização da aplicação.

A arquitetura Multicamadas foi evoluindo e diversos fabricantes de soluções criaram ferramentas com o objetivo de facilitar o desenvolvimento de aplicações nessa arquitetura. A complexidade das aplicações distribuídas aumentou em relação à arquitetura Cliente/Servidor.

Esse aumento da complexidade se deu principalmente pela própria natureza das aplicações distribuídas. A aplicações distribuídas utilizam processos executados em máquinas

distintas que colaboram na resolução de um problema comum. Aplicações distribuídas rodam em vários sistemas de computador ao mesmo tempo, geralmente como processos em filas ou camadas.

As aplicações distribuídas normalmente envolvem transações que necessitam manipular e atualizar dados a partir de diversas fontes, com operações que precisam ser concluídas como um todo ou canceladas como um todo, ou até mesmo transações que precisam acessar recursos compartilhados. Outro fator que contribui para o aumento da complexidade no desenvolvimento de aplicações distribuídas é a comum necessidade de uma infra-estrutura para o seu funcionamento, onde essa infra-estrutura deve fornecer serviços tais como:

- Serviço de nomes e diretório;
- Serviço de mensagens;
- Controle transacional;
- Balanço de carga;
- Acesso a dados em diversos formatos.

Assim, caso o desenvolvedor não conte com uma tecnologia que já forneça toda essa infra-estrutura, o desenvolvimento de aplicações distribuídas/multicamadas será uma tarefa extremamente complexa, pouco produtiva e sujeita a erros.

Embora o desenvolvimento de aplicações multicamadas/distribuídas seja uma atividade relativamente nova, muitos avanços já foram realizados. Diversos fabricantes criaram novas tecnologias e ferramentas. Algumas dessas ferramentas conseguiram atender às necessidades do desenvolvimento de aplicações nesse novo contexto. Porem, ainda continuou existindo um grande problema no uso dessas tecnologias: *a falta de padronização*, já que as mesmas eram implementadas com soluções proprietárias, fazendo com que a empresa cliente ficasse dependente do fornecedor da tecnologia. Também existia uma grande dificuldade na interação com soluções de outras empresas usuárias de outras tecnologias.

Assim nasceu a necessidade de uma padronização da tecnologia, utilizada na implementação de aplicações multicamadas/distribuídas. Quanto mais uma empresa puder usar padrões para se conectar de modo eficaz com seus clientes, fornecedores e parceiros – e até mesmo concorrentes, maior a eficácia com que poderá participar no mercado e mais competitiva ela será na economia em rede. A adoção em grande escala de uma plataforma para o desenvolvimento de aplicações ajudará a criar um mercado grande e diversificado para bens e serviços relacionados.

Percebendo toda essas oportunidades a Sun Microsystems e uma séria de parceiros criaram o padrão Java 2 Enterprise Edition – J2EE **[SUN02]**, sob a forma de um conjunto de padrões, serviços e uma plataforma de desenvolvimento, com a finalidade de padronizar o desenvolvimento e a distribuição de aplicações distribuídas/multicamadas. A Plataforma J2EE fornece um conjunto de serviços padronizados que reduzem a complexidade e aumentam a produtividade nesse desenvolvimento.

Desde a sua introdução, há dois anos, o padrão J2EE alcançou um impulso significativo entre os fornecedores de produtos de tecnologia de informação para empresas. Diversas empresas, em parceria com a Sun Microsystems, licenciados (empresas que aplicam seus produtos a testes de compatibilidade com a plataforma J2EE) em J2EE já distribuíram produtos comerciais com base nesse padrão e muitos dos seus clientes desenvolveram e distribuíram aplicações usando esses produtos.

1.2 Contribuição

A Plataforma J2EE é realmente um padrão aceito pelo mercado e mundialmente utilizado, no desenvolvimento de aplicações multicamadas/distribuídas. Esse fato é evidenciado pelo grande número de *downloads* da plataforma, feitos através do site *www.javasoft.com*, da Sun Microsystems (aproximadamente um milhão até 2001), desde a sua liberação, sem falar nos milhares de clientes que utilizam produtos compatíveis com o padrão J2EE.

Por conta disso, a plataforma J2EE foi escolhida como alvo para este trabalho de pesquisa, onde o objetivo é aprender e identificar os recursos oferecidos e propor uma estratégia que possa elevar ainda mais a produtividade na implementação de aplicações Web.

A plataforma oferece uma excelente infra-estrutura, mas ainda existem aspectos, os quais podem ser explorados a fim de elevar a produtividade e reduzir a complexidade na construção de aplicações sobre a referida plataforma. Principalmente em funcionalidades de níveis mais altos na construção de aplicações, tais como atividades que possam ser automatizadas (ex. geração automática de componentes da plataforma J2EE e/ou da camada de interface, etc) e recursos comuns a várias aplicações (ex. validação e autenticação de usuários/transações).

A contribuição deste trabalho foi a construção de uma ferramenta de desenvolvimento de aplicações Web sobre a plataforma J2EE visando aumentar a produtividade e reduzir a complexidade na utilização da referida plataforma.

1.3 Estrutura da Dissertação

No capítulo 2 apresentaremos um estudo da plataforma Java 2EE focando a especificação Enterprise Java Beans 2.0. O entendimento da plataforma e dos seus principais recursos para construção de soluções é de fundamental importância para alcançar os objetivos propostos.

Uma aplicação foi implementada com o objetivo de aumentar a compreensão da plataforma (**Capítulo 3**), assim como evidenciar os aspectos a serem explorados no aumento da produtividade e na redução da complexidade no desenvolvimento de aplicações sobre a plataforma J2EE. A aplicação desenvolvida foi um portal para inscrição em vestibulares, onde, via Internet, instituições poderão publicar seus vestibulares e candidatos poderão realizar suas inscrições.

Com base na observação das limitações e dificuldades encontradas no desenvolvimento da aplicação, a qual foi implementada apenas com recursos da plataforma J2EE, foi possível levantar os requisitos para a ferramenta.

Em seguida a ferramenta foi projetada, implementada e testada, com a re-implementação da aplicação portal de vestibular (**Capítulo 4**), o que foi de grande ajuda na avaliação das limitações e benefícios alcançados com a utilização da ferramenta.

Ficou evidente a necessidade da utilização de estratégias e ferramentas para o desenvolvimento de aplicações sobre a plataforma J2EE. A estratégia de plataforma aberta adotada pela Sun Microsystems conta com o surgimento dessas estratégias e ferramentas para adequar a plataforma as mais diversas necessidades de mercado. Várias estratégias e ferramentas foram criadas e outras tantas se encontram em processo de criação.

Capítulo 2 - A Plataforma Java 2 Enterprise Edition

"Atualmente, as empresas necessitam expandir seus mercados, reduzir custos e diminuir o seu tempo de resposta, fornecendo facilidade de acesso aos seus serviços a clientes, parceiros, funcionários e fornecedores" [SUN04]. Com esta frase a Sun Microsystem inicia a definição da Plataforma **Java 2 Enterprise Edition**. Deixando claro que o objetivo da mesma é atender aos requisitos do ambiente coorporativo, entendendo como ambiente coorporativo aquele onde os seguintes requisitos são fundamentais: **Integração com Sistemas Legados**; **Robustez**, **Alta Disponibilidade**, **Segurança**, **Confiabilidade**, e **Escalabilidade**.

A Plataforma Java 2EE define um padrão para o desenvolvimento de aplicações coorporativas multicamadas, simplificando o desenvolvimento dessas aplicações através de uma padronização baseada em componentes, fornecendo um conjunto completo de serviços para esses componentes, e da manipulação automática de vários detalhes inerentes ao ambiente coorporativo, tais como: Controle Transacional; Ajuste de Performance; Persistência; Segurança; Escalabilidade; Gerencia de Recursos (Processos, Threads, Memória, Conexões com Banco de Dados e Sessões com Usuários).

A Plataforma J2EE foi projetada para suportar os rígidos requisitos dos ambientes coorporativos atuais, através de uma arquitetura de aplicação multicamada, com enfoque no lado servidor e baseado em componentes. Na arquitetura multicamadas o software é construído através de componentes distribuídos em várias camadas, normalmente incluindo uma Camada Cliente, uma ou mais Camadas Intermediárias e uma Camada de Dados. Conceitualmente cada camada possui atribuições bem definidas, onde uma camada é cliente da outra, existindo uma sucessão de relações Cliente/Servidor entre as camadas.

A **Camada Cliente**, também chamada de Camada de interface, é responsável pela interação com o usuário, sendo composta por componentes que interagem com as regras de negócio do ambiente coorporativo. Os componentes dessa camada normalmente são responsáveis pela criação da interface com o usuário e normalmente são implementados como aplicações ou páginas *html* geradas dinamicamente.

A **Camada Intermediária**, também chamada de Camada de Regra de Negócio, é responsável por encapsular a complexidade do ambiente coorporativo, fornecendo de forma simples recursos para a construção da camada cliente e integração com Sistemas Legados. Os

componentes dessa camada são hospedados no lado servidor, necessitando de um servidor de aplicações para a execução dos mesmos.

A **Camada de Dados** é responsável pela persistência e gerência dos dados, normalmente implementada por um Sistema Gerenciador de Banco de Dados Relacional.

A Plataforma J2EE fornecer os recursos necessários ao desenvolvimento de aplicações coorporativas através da definição de uma arquitetura padronizada.

2.1 Tecnologias da Plataforma J2EE

A Plataforma Java 2EE foi construída sobre e portanto herda uma série de vantagens já disponibilizadas por esta, tais como: independência de plataforma, através da Máquina Virtual Java; independência de servidor de banco de dados, através da API JDBC e tecnologia CORBA para a interação com sistemas legados. Em adição às tecnologias da Plataforma Standard Edition foram criadas uma série de novas tecnologias para servir de suporte à Plataforma J2EE:

- **ECperf** [SUN08] - Composto por uma especificação e um kit de software tendo como finalidade medir a performance e escalabilidade de servidores de aplicação J2EE. Foi desenvolvido através do *Java Community Process* em parceria com fabricantes de servidores J2EE.
- **Enterprise JavaBeans** (EJB) [SUN09] - Representa um modelo de componentes/objetos distribuídos permitindo a criação de componentes do lado servidor, simplificando a implementação das camadas intermediárias fornecendo automaticamente controle transacional, escalabilidade, conexão com banco de dados, segurança e portabilidade. Esses serviços são fundamentais para a implementação das camadas intermediárias.
- **JavaMail API** [SUN12] - Framework para manipulação de email. Necessita da tecnologia JAF.
- **Java Message Service API** (JMS) [SUN13] - Serviço de mensagens assíncronas fornecendo uma API e um *Framework* para a implementação de aplicações com baixo acoplamento.
- **JavaServer Faces** [SUN14] - Tecnologia para a implementação de interfaces para aplicações servidoras Java. Permitindo o isolamento entre a camada de interface e a camada de regra de negócio através da definição de um modelo de programação, o qual permite a reutilização dos componentes, associação destes com fontes de dados e associação de eventos no lado cliente com rotinas no lado servidor.

- **Java Server Pages** (JSP) [SUN15] - Mecanismos para geração de conteúdo Web, baseado na tecnologia Servlet, trazendo maiores facilidades na construção de interface Web, pois associa conteúdo HTML com código Java, através de rótulos pré-definidos. Assim, o JSP já possui o conteúdo HTML a ser retornado ao cliente, permitindo a execução de código Java para composição dinâmica do conteúdo web. Todo JSP é convertido em um Servlet para então ser executado.
- **Servlet** [SUN16] - API representando a extensão para servidores Web e permitindo a geração dinâmica de páginas Web através de programas Java. Permite interação com o usuário via protocolo HTTP. É um programa Java o qual estende a API servlet e é executado no lado servidor e permitindo acesso a toda tecnologia e API Java. O resultado da sua execução é uma página web a ser enviada ao cliente. É uma alternativa Java a solução CGI (Commom Gateway Interface).
- **JDBC** (Data Access API) [SUN17] - API que fornece à tecnologia Java independência de Sistema Gerenciador de Banco de Dados. Representada pela padronização do acesso a Banco de Dados pela linguagem Java, assim, qualquer fabricante de SGBD que implementar um *driver* JDBC permitirá que programas Java acessem o seu servidor de forma independente. Foi uma iniciativa fundamental para a conquista do ambiente coorporativo pela tecnologia Java, pois agregou à independência de plataforma, a independência de SGBD.
- **Connector Architecture** [SUN18] - Definição de uma arquitetura para conexão entre a Plataforma J2EE e sistemas heterogêneos (Sistemas de Informações Coorporativas), tais como Mainframes, Sistemas de Banco de Dados e aplicações legadas não escritas em Java. Essa arquitetura permite que fornecedores de Sistemas Coorporativos possam fornecer mecanismos padronizados de acesso aos seus sistemas, denominados de adaptadores de recursos (*resource adapter*). Os adaptadores de recursos funcionam como um *plugin* para os servidores de aplicação fornecendo conectividade entre os servidores de aplicação e as aplicações coorporativas.
- **Transactions** [SUN19] - Serviço para o gerenciado distribuído de transações. A Plataforma J2EE fornece suporte a transações através de duas especificações: *Java Transaction API* (**JTA**) e *Java Transaction Service* (**JTS**). JTA é uma API de alto nível, independente de implementação e protocolo que permite a Aplicações e Servidores de Aplicações acessar e gerenciar transações. JTS especifica a implementação de um

gerenciador de transações que suporta JTA e implementa o mapeamento Java para Serviço Transacional de Objetos (Object Transaction Service - OTS) utilizando o protocolo IIOP.

2.2 Arquitetura de Aplicação J2EE

A Plataforma J2EE foi projetada para suportar a arquitetura multicamada no desenvolvimento de aplicações. Dentre as tecnologias oferecidas pela plataforma, o centro das atenções está na tecnologia EJB. Como visto anteriormente, a tecnologia EJB permite a construção de componentes da camada intermediária fornecendo toda a infra-estrutura necessária à construção de diversos modelos de aplicações, através da utilização e integração com outras tecnologias da plataforma. As demais tecnologias servem basicamente para fornecer a infra-estrutura necessária ao desenvolvimento dos componentes EJB e em alguns casos ficam transparentes ao desenvolvedor.

Um conceito muito importante para o entendimento da arquitetura de aplicações J2EE é o de ***Container***. Os Container quais são ambientes operacionais responsáveis por executar os componentes da plataforma J2EE.

2.2.1 Container J2EE

A plataforma J2EE é baseada no conceito de componentes, os quais representam unidades de software reutilizáveis, sendo necessário a utilização de container compatível com a especificação do tipo de componentes para publicação e execução destes. Os componentes representam as funcionalidades exigidas pelas aplicações coorporativas e a combinação destes, forma a aplicação em si, sendo que alguns são reutilizáveis em várias aplicações e até mesmo por empresas diferentes.

Desta forma é possível desenvolver aplicações J2EE a partir da combinação de componentes, reduzindo a complexidade e aumentando a produtividade no desenvolvimento de software. Os *containers* representam um ambiente operacional, fornecendo os recursos e infra-estrutura necessária à publicação e execução de componentes, assim encapsulam a complexidade do ambiente coorporativo e fornece independência de plataforma aos componentes.

Os containeres representam a interface entre os componentes e serviços de baixo nível fornecidos pela plataforma e necessários à execução do componente, sendo assim, os componentes devem ser implementados seguindo as especificações do container. Para que um componente possa ser utilizado, o mesmo deve ser publicado em um container, necessitando de uma operação chamada *deploy* (configuração e preparação para execução). Na referida

operação, o componente será configurado, através de arquivos XML **[XML01]** e/ou da geração de classes de serviços. Esses arquivos XML irão conter informações relativas à execução do componente tais como: segurança, controle transacional, persistência de dados, escalabilidade, associação com outros componentes, serviço de nomes, etc.

Após a operação de *deploy* o componente pode então ser publicado no container e a partir deste momento o container é responsável pela execução e gerência do componente. A comunicação com os componentes passa pelo container, fornecendo assim uma arquitetura flexível onde os componentes podem ser configurados de acordo com as necessidades, permitindo que um mesmo componente possa ter configurações diferentes, bastando para isso realizar uma nova operação de *deploy* ajustando os parâmetros de configuração.

O resultado da operação de *deploy* é um arquivo *jar (java archieve)* contendo o(s) componente(s), os arquivos XMLs com suas configurações, e as classes de serviços geradas automaticamente.

A plataforma J2EE está organizada nos seguintes tipos de containeres:

- **Container EJB** – gerencia a execução dos componentes EJB;
- **Container WEB** – gerencia a execução de componentes web Servlets e JSPs;
- **Application** – gerencia a execução de aplicações Java, ficando na camada cliente;
- **Applet** – gerencia a execução de applets Java, ficando na camada cliente associado ao programa de navegação (*browser*).

2.3 Enterprise Java Beans

A tecnologia EJB define um modelo de componentes transacionais, baseados no lado servidor, os quais permitem a implementação de objetos de negócios, representando o recurso principal para o desenvolvimento das camadas intermediárias, na arquitetura de aplicação multicamadas.

A utilização de EJB facilita o desenvolvimento desses componentes. Aspectos como segurança, persistência, escalabilidade e controle transacional são tratados automaticamente pelo container EJB, livrando o desenvolvedor de tarefas tão complexas quase sempre mais complexas que as próprias regras de negócio necessárias às aplicações coorporativas. O padrão EJB foi rapidamente aceito pelo mercado e tornou-se a principal tecnologia de desenvolvimento de aplicações multicamadas distribuídas para ambientes coorporativos.

2.3.1 Componentes Enterprise Java Beans

Os componentes EJB são denominados de *Beans*, dando a idéia de unidades funcionais e independentes. Os componentes EJB devem ser implementados de acordo com as especificações do *container* EJB e têm as seguintes características:

- Contêm as regras de negócio operando sobre os dados do ambiente coorporativo;
- Suas instâncias são criadas e gerenciadas em tempo de execução por um *container* EJB;
- Podem ser configurados em tempo de publicação através da edição de arquivos XML;
- Aspectos relativos ao funcionamento do componente, tais como controle transacional, segurança, persistência, etc, são definidos separadamente das classes e interfaces do componente, permitindo que esses atributos sejam alterados sem precisar alterar a implementação do componente;
- São acessados pelos clientes através do container;
- Podem ser distribuídos em qualquer servidor compatível com a especificação EJB;
- Fornecem a visão do ambiente coorporativo à camada cliente, escondendo sua complexidade e realizando a interface com a camada de dados.

2.3.2 Arquitetura Enterprise Java Beans

No modelo de programação orientado a objetos convencional (não distribuído) quando uma referência a um objeto é utilizada, o próprio objeto é manipulado, ou seja, quando uma referência é utilizada para a invocação de um método, a solicitação é passada diretamente ao objeto, o qual é responsável pela execução do método solicitado. Já no modelo de objetos distribuídos o funcionamento da invocação de métodos não é assim tão simples, pois a invocação de métodos é feita através do ambiente remoto, necessitando de toda uma infra-estrutura para propagar a solicitação até o objeto propriamente dito e retornar o resultado da execução do método.

No caso do modelo de objetos distribuídos da plataforma J2EE, o container é responsável por gerenciar a utilização dos beans. Assim, o mesmo será responsável por gerar e fornecer à camada cliente um conjunto de classes de serviços responsáveis pela transparência da utilização do ambiente distribuído. Os elementos dos componentes EJB (interfaces e classe abstrata) serão utilizados pelo container para geração dessas classes de serviços e efetivamente essas classes de serviços é que irão fornecer a transparência do ambiente distribuído.

Para que uma aplicação cliente solicite a execução de um método de uma componente EJB é necessário que a mesma consiga a referência para a interface remota do mesmo, para

então, através da interface remota, solicitar a execução do método e receber o resultado. A chamada a métodos de componentes EJB funciona da seguinte maneira:

1. A aplicação cliente obtém um objeto que implementa a interface remota, o qual foi gerado automaticamente e fornecido pelo container;
2. Através desse objeto, o método desejado é invocado;
3. O objeto que implementa a interface remota, via RMI **[SUN22]**, repassando a chamada ao método para o container EJB, o qual por sua vez, efetivamente solicita ao Bean a execução do método;
4. O bean executa o método solicitado e devolve o resultado ao container;
5. O container devolve o resultado da execução do método ao objeto que implementa a interface remota;
6. O objeto que implementa a interface remota devolve o resultado do método à aplicação cliente.

Desta forma, através da geração automática dessas classes de serviços denominadas **STUB** (as fornecidas à camada cliente) e **SKELETON** (as responsáveis pela comunicação com o container no lado servidor) a utilização da tecnologia EJB torna transparente a utilização dos objetos remotos.

2.3.3 Elementos dos Componentes Enterprise Java Beans

De acordo com a especificação, todos os componentes EJB precisam possuir os seguintes elementos: *Interface Remota*, *Interface Home* e *Classe Abstrata do Bean*. Esses elementos representam conceitos básicos no modelo de objetos distribuídos e servirão de modelo para a geração de classes de serviços. Por este motivo, os elementos dos componentes EJB foram definidos como interfaces (não podem possuir implementação de métodos) e classes abstratas (não podem ser instanciadas) e servirão como base, para que, na operação de *Deploy* classes de serviços sejam implementadas estendendo as classes abstratas e implementando as interfaces.

Normalmente a nomenclatura dos elementos dos *Beans* EJB segue as seguintes regras:

- A interface Remota terá o nome do conceito que o *Bean* representa;
- A interface Home terá o nome do conceito, acrescido de "**Home**";
- A classe abstrata terá o nome do conceito, acrescido de "**Bean**";

Por exemplo, um *Bean* EJB representando o conceito Funcionário teria os seguintes nomes para a interface Remota e Home e Classe Abstrata, respectivamente: **Funcionário**, **FuncionárioHome** e **FuncionárioBean**.

Todos os métodos das interfaces Home e Remota e os da Classe Abstrata que forem disponibilizados para o ambiente remoto devem indicar na sua assinatura a possibilidade do lançamento da exceção `java.rmi.RemoteException`. Essa exceção será responsável por transportar a exceção ocorrida na execução do método para o ambiente remoto, onde o método foi invocado.

Opcionalmente é possível definir interfaces locais com o objetivo de permitir que componentes publicados no mesmo servidor EJB não precisem utilizar os mecanismos de comunicação como se estivessem em ambientes remotos, evitando assim um sobre esforço na comunicação entre *Beans* executados na mesma Máquina Virtual Java. Neses caso, os objetos gerados pelo container para implementar as interfaces locais não iriam utilizar a camada de rede (RMI) para comunicação. Os nomes das interfaces locais para um bean representando o conceito Funcionário seriam: **FuncionárioLocal** e **FuncionárioLocalHome**.

Para que um componente EJB seja publicado no servidor EJB é necessário que o mesmo seja configurado através da operação de *deploy*, para isso são necessários arquivos XML denominados *deployment descriptors*, os quais contêm as configurações dos componentes EJB.

2.3.3.1 Interface Home

A interface Home irá fornecer métodos para manutenção do ciclo de vida dos componentes EJB. Os seus métodos irão permitir criar, remover e localizar *Beans*. Essa interface deve estender a interface `javax.ejb.ejbHome`. É através da interface Home que a camada cliente irá obter acesso à interface remota do *Bean*. Os métodos de localização da interface Home retornam à interface Remota do *Bean*, ou uma coleção (`java.util.Collection`), justamente para que a camada cliente possa utilizar os métodos ofertados pelo Bean à camada cliente.

2.3.3.2 Interface Remota

A Interface Remota representa a visão do Bean ao ambiente remoto. A interface Remota deve estender a interface `javax.ejb.EJBObject` e deve possuir a assinatura de todos os métodos a serem ofertados, à camada cliente. Todos os seus métodos devem indicar na assinatura a possibilidade do lançamento da exceção `java.rmi.RemoteException`. A interface Remota irá servir como base para geração da classe STUB, a qual é gerada automaticamente através da operação de deployment. Esta classe gerada faz a comunicação entre a camada cliente e o container EJB.

2.3.3.3 Classe Abstrata

A Classe Abstrata irá conter a implementação de todos os métodos a serem fornecidos pelo Bean, alguns dos quais serão métodos de serviços e não serão ofertados à camada cliente, ou seja, suas assinaturas não estarão presentes na interface Remota. A Classe Abstrata deve implementar a interface `javax.ejb.EntityBean`, `javax.ejb.SessionBean` ou `javax.ejb.MessageDrivenBean`, a depender do tipo de Bean. O container irá gerar classes automaticamente para estender a Classe Abstrata do bean, adicionando os métodos necessários à utilização dos recursos oferecidos pelo container.

2.3.3.4 Deployment Descriptors

O Deployment Descriptors são arquivos XML que permitem a configuração dos componentes EJB em tempo de publicação. Possibilitando realizar ajustes sem necessitar alterar o código fonte dos componentes, fornecendo flexibilidade na utilização dos componentes, uma vez que o mesmo componente EJB pode ser configurado de maneira diferente a depender da necessidade. Os *deployment descriptors* configuram as características dos componentes EJB, tais como: segurança, controle transacional, persistência de dados, escalabilidade, associação com outros componentes e serviço de nomes

2.3.4 Tipos de componentes EJB

A plataforma EJB oferece três tipos de componentes, sendo eles: **Entity Bean** (Bean de Entidade – acesso a dados), **Session Bean** (Bean de Sessão – regras de negócio) e **Message Driven Bean** (Bean Orientado a Mensagens – resposta a mensagens assíncronas). A seguir descreveremos cada um deles.

2.3.4.1 Entity Bean

De uma forma simplificada Entity Bean é objeto de negócio representando um conceito sobre o qual é necessário persistir dados, por exemplo uma tabela num SGBD relacional. O Entity Bean será o componente responsável por representar a fonte de dados na camada cliente ou intermediária.

O principal objetivo na utilização de Entity Beans é trazer para a camada intermediária um mecanismo de acesso aos dados, permitindo uma maior escalabilidade, através de objetos na camada intermediária representando os dados. Através dos Entity Beans é possível aumentar a capacidade de processamento (escalabilidade horizontal) adicionando novos

pontos de processamento associados a fonte de dados. A camada intermediária irá distribuir a carga de processamento que seria unicamente realizada na camada de dados.

Objetivos secundários alcançados com a utilização de Entity Beans são: possibilidade de acessar fonte de dados heterogêneos e visão orientadas a objetos dos dados. Para uma aplicação, os Entity Beans devem representar uma visão abstrata e orientada a objetos dos dados e as transações sobre os dados devem ser realizadas através dos Entity Beans. O container EJB irá cuidar para que os Entity Beans sejam utilizados dentro de um contexto transacional, onde o nível do controle transacional pode ser definido, por Entity Bean, na operação de deploy.

A maioria das aplicações comerciais utiliza SGBD relacionais. Assim, para cada tabela do Banco de Dados será necessário criar um Entity Bean e este será responsável pelas operações sobre esta tabela. Cada linha da tabela será representada por uma instância do Entity Bean no pool de objetos do servidor EJB. Um Entity Bean além de representar uma tabela no banco de dados, também representa os seus relacionamentos e operações, lembrando que os Entity Beans são classes/objetos agrupando dados, relacionamentos e operações.

O Entity Bean pode possuir um componente adicional, o qual é responsável por representar a chave primária do Entity Bean, sendo fundamental para que o container organize, ordene e controle a repetição dos Entity Bean no seu Pool de objetos. O nome da classe chave primária, normalmente, é formado pelo nome do conceito que o bean representa acrescido de "**PK**".

Exemplo: FuncionárioPK.

2.3.4.1.1 Tipos de Entity Beans

Existem dois tipos de Entity Beans de acordo com os mecanismos através dos quais é feita a gerência da persistência dos seus dados:

Container-Managed Persistence (CMP)

Nesse mecanismo a persistência dos dados é gerenciada pelo container, livrando o desenvolvedor da implementação do código necessário para realizar tal operação. A utilização de Entity Beans CMP fornece uma maior produtividade e simplicidade, pois o container irá gerenciar automaticamente a persistência dos dados. A persistência dos dados dos Entity Beans CMP, normalmente, é realizada através da API JDBC, assim, para todas as fontes de dados que exista um *Driver JDBC* [SUN17] é possível utilizar Entity Beans CMP. Os Entity Beans CMP fornecem também os relacionamentos entre eles, sendo necessário definir a cardinalidade e navegabilidade dos relacionamentos a serem fornecidos pelo container.

Bean-Managed Persistence (BMP)

Nesse mecanismo a persistência dos dados é gerenciada pelo próprio Entity Bean. Esse tipo de Entity Bean permite uma grande flexibilidade na utilização de fontes de dados diversas, pois o desenvolvedor tem total liberdade para codificar o mecanismo para persistência de dados.

O código relativo à persistência dos dados será escrito em métodos específicos redefinidos da interface `javax.ejb.EntityBean`, sendos eles: `ejbCreate`, `ejbPosCreate`, `ejbLoad`, `ejbStore`, `ejbActivate`, `ejbPassivate` e `ejbRemove`. Estes métodos funcionam como eventos no ciclo de vida do Entity Bean. A tabela a seguir indica quando os métodos são invocados pelo container.

Tabela 1. Métodos utilizados pelo container

Método	Quando o método é chamado pelo container
`ejbCreate`	Quando o método `create` da interface Home é chamado. Para cada método `create` da interface home é necessário ter um método `ejbCreate`.
`ejbPostCreate`	Logo após a execução do método `ejbCreate`. Para cada método `ejbCreate` é necessário ter um método `ejbPostCreate`.
`ejbLoad`	Quando o container precisa carregar os dados da fonte de dados para o bean.
`ejbStore`	Quando o container precisa armazenar os dados do bean na fonte dados.
`ejbActivate`	Quando o container estiver trazendo o bean para o pool de beans.
`ejbPassivate`	Quando o container estiver retirando o bean do pool de beans.
`ejbRemove`	Quando remove da interface Remota é chamado.

2.3.4.2 Enterprise Java Beans Query Language (EJBQL)

EJBQL é uma implementação da OQL (Object Query Language) **[ULL01]**. EJBQL representa uma forma de escrever consultas sobre o modelo abstrato dos Entity Beans CMP, sendo uma linguagem de consulta orientada a objetos. O Container interpreta as consultas em EJBQL e gera uma consulta SQL para o SGBD, essa geração é feita em tempo de *deployment*, isso permite que as consultas sejam independente de SGBD.

EJBQL é utilizada para construir os métodos `finders` da Interface Home dos Entity Beans, os quais somente podem atuar sobre o próprio Entity Bean, e métodos genéricos de consultas, podendo atuar sobre todo o esquema abstrato dos Entity Beans, desde que estes estejam publicados no mesmo container.

Os métodos de consultas são definidos na classe do Entity Bean e devem seguir a seguinte nomenclatura: `ejbSelect`**`NomeDaConsulta`**`(`*`lista de parâmetros`*`)`. Na classe do Entity Bean fica apenas a definição do método. O comando em EJBQL que implementa a consulta fica no *deployment descriptor* do Entity Bean. O método `findBy`**`PrimaryKey`** é o único que não precisa ter sua consulta em EJBQL declarada.

2.3.4.3 Criando um Entity Bean CMP (Vestibular)

Na criação de componentes EJB é necessário seguir a especificação EJB para que o container possa tratar automaticamente os Beans. Nos itens a seguir serão demonstradas as regras para criação dos elementos de Entity Beans CMP, seguindo a especificação EJB 2.0. Os exemplo serão demonstrados com o Entity Bean Instituição, implementado para a aplicação de Portal de Inscrição On-Line, o qual representa a tabela Instituição de um banco de dados relacional. A tabela a seguir demonstra o comando SQL utilizado para criar a tabela "Instituicao".

Código Fonte 1. Comando SQL para criação da tabela Instituicao

```
CREATE TABLE Instituicao (
  Codigo int NOT NULL,
  CNPJ char (14) NOT NULL,
  Nome varchar (200) NOT NULL,
  Rua varchar (50) NOT NULL,
  Numero varchar (5) NOT NULL,
  Complemento varchar (50) NOT NULL,
  Bairro varchar (50) NOT NULL,
  CEP char (8) NOT NULL,
  Cidade varchar (50) NOT NULL,
  Estado char (2) NOT NULL,
  Telefones varchar (30) NOT NULL,
  Email varchar (50) NOT NULL,
  Senha varchar (10) NOT NULL,
  Contato varchar (50) NULL,
  Status char (1) NOT NULL,
  BoletoNomeBanco varchar (30) NULL,
  BoletoCodBanco varchar (4) NULL,
  BoletoCodAgencia varchar (5) NULL,
  BoletoCodConta varchar (9) NULL,
  BoletoCarteira varchar (2) NULL,
  BoletoConvenio varchar (6) NULL
)

ALTER TABLE Instituicao ADD
  CONSTRAINT PK_Instituicao PRIMARY KEY
  (Codigo)  ON PRIMARY

ALTER TABLE Instituicao ADD
  CONSTRAINT IX_Instituicao_CNPJ UNIQUE
  (CNPJ)  ON PRIMARY,
  CONSTRAINT IX_Instituicao_Email UNIQUE
  (Email)  ON PRIMARY
```

2.3.4.3.1 Elementos do Entity Bean

Todo Entity Bean deve possuir sua interfaces Home e Remota e uma Classe Abstrata. O Entity Bean pode possuir uma classe chave primária, sendo obrigatória no caso de chave primária da fonte de dados ser formada por múltiplos campos. Para o Entity Bean Instituição, os seus componentes foram denominados da seguinte maneira: **Instituição** (interface remota), **InstituiçãoHome** (interface home), **InstituiçãoBean** (classe abstrata) e **InstituiçãoPK** (chave primária).

2.3.4.3.2 Interface Home

A interface Home deve estender `javax.ejb.EJBHome`, e implementar os métodos `create` e `finder`. A escolha desses métodos fica a critério do desenvolvedor, sendo obrigatório apenas um método `create` com todos os campos obrigatórios da fonte de dados e o método `findByPrimaryKey`. Ambos devem retornar a interface Remota do bean. Os métodos `finders` devem retornar a interface remota do bean, quando a sua consulta EJBQL retornar apenas um único bean e devem retornar uma coleção do tipo `java.util.Collection` quando o resultado da consulta EJBQL for uma coleção de beans.

Os métodos `create` devem possuir na sua cláusula `throws` a exceção `java.ejb.CreateException` para quando ocorrer um erro na criação dos dados na fonte de dados associada ao bean e os métodos finders a exceção `javax.ejb.FinderException` para quando o retorno da consulta for um conjunto vazio.

Código Fonte 2. Interface Home do Entity Bean Instituição

```
package br.com.easyNet.vestibular.ejb.instituicao;

import java.math.BigDecimal;
import java.util.*;
import java.rmi.RemoteException;
import javax.ejb.*;

public interface InstituicaoHome extends EJBHome {

   public Instituicao create (BigDecimal codigo,
                              String cNPJ,
                              String nome,
                              String rua,
                              String numero,
                              String bairro,
                              String cEP,
                              String cidade,
                              String estado,
                              String telefones,
                              String email,
                              String senha,
                              String contato,
                              String status,
                              String boletoNomeBanco,
                              String boletoCodBanco,
                              String boletoCodAgencia,
                              String boletoCodConta,
                              String boletoCarteira,
                              String boletoConvenio)
        throws CreateException, RemoteException;

   public Instituicao findByPrimaryKey (InstituicaoPK pk)
        throws FinderException, RemoteException;

   public Collection findAll ()
        throws FinderException, RemoteException;

   public Instituicao findByCNPJ (String value)
```

```
      throws FinderException, RemoteException;

   public Instituicao findByEmail (String value)
        throws FinderException, RemoteException;
}
```

2.3.4.3.3 Interface Remota

A interface remota deve possuir a assinatura de todos os métodos acessíveis à camada cliente. O Entity Bean é um objeto de negócios devendo possuir métodos `set` e `get` para manipular os valores de seus atributos. Apenas os campos que formam a chave primária não devem possuir os métodos `set` na interface remota, pois mesmo que estes sejam ofertados sua execução não acarretará na alteração dos campos.

Outros métodos devem ser ofertados à camada cliente, pois além de acesso aos dados os Entity Beans devem oferecer operações sobre estes. No caso do Entity Bean Instituição foram oferecidos métodos para manipular os Entity Beans relacionados Curso e Vestibular, assim como um método para retornar uma representação XML do Entity Bean.

Todos os métodos da interface Remota devem possuir a exceção `javax.ejb.RemoteException` na sua cláusula `throws`.

Código Fonte 3. Interface Remota do Entity Bean Instituição

```
package br.com.easyNet.vestibular.ejb.instituicao;

import java.math.BigDecimal;
import java.sql.Timestamp;
import java.util.*;
import javax.ejb.*;
import java.rmi.RemoteException;
import br.com.easyNet.vestibular.ejb.curso.Curso;
import br.com.easyNet.vestibular.ejb.vestibular.Vestibular;

public interface Instituicao extends EJBObject {

   public abstract BigDecimal getCodigo () throws RemoteException;

   public abstract void setCNPJ (String value) throws RemoteException;
   public abstract String getCNPJ () throws RemoteException;

   public abstract void setNome (String value) throws RemoteException;
   public abstract String getNome () throws RemoteException;

   public abstract void setRua (String value) throws RemoteException;
   public abstract String getRua () throws RemoteException;

...

   public Curso createCurso(BigDecimal codigo,String descricao)
      throws Exception, RemoteException;

   public Set getCursos ()
      throws RemoteException;

   public void removeCurso (BigDecimal codigo)
      throws Exception, RemoteException;
```

```
   public Vestibular createVestibular(BigDecimal codigo,
                                      String descricao,
                                      Timestamp inscricaoInicio,
                                      Timestamp inscricaoFim,
                                      BigDecimal numeroOpcoes,
                                      BigDecimal valorInscricao)
      throws Exception, RemoteException;

   public Set getVestibulares ()
      throws RemoteException;

   public void removeVestibular (BigDecimal codigo)
      throws Exception, RemoteException;

   public String toXml() throws RemoteException;

}
```

2.3.4.3.4 Classe Abstrata

A classe abstrata representa a implementação do Entity Bean propriamente dito, sendo que, nos beans CMP não é necessário implementar os métodos `set` e `get` para os atributos, os quais serão implementados pelas classes geradas pelo container. Apenas os métodos de negócio, propriamente dito, devem ser implementados na classe abstrata.

A classe abstrata deve implementar a interface javax.ejb.EntityBean e os seus métodos não devem lançar exceção do tipo `java.rmi.RemoteException`, pois serão executados localmente no container EJB. Os Entity Bean CMP devem possuir um construtor padrão e um atributo privado do tipo `javax.ejb.EntityContext`, representando o objeto através do qual o Entity Bean irá comunicar-se com o container, esse objeto será manipulado pelos métodos `setEntityContext` e `unsetEntityContext`.

Código Fonte 4. Classe abstrata do Entity Bean Instituição

```
package br.com.easyNet.vestibular.ejb.instituicao;

import java.math.BigDecimal;
import java.sql.Timestamp ;
import java.util.*;
import javax.ejb.*;
import java.rmi.RemoteException;
import javax.naming.*;

import br.com.easyNet.vestibular.ejb.curso.Curso;
import br.com.easyNet.vestibular.ejb.curso.CursoHome;
import br.com.easyNet.vestibular.ejb.curso.CursoPK;

import br.com.easyNet.vestibular.ejb.vestibular.Vestibular;
import br.com.easyNet.vestibular.ejb.vestibular.VestibularHome;
import br.com.easyNet.vestibular.ejb.vestibular.VestibularPK;

public abstract class InstituicaoBean implements EntityBean {

   private EntityContext ctx;

   public InstituicaoBean () {}
```

```
   public abstract void setCodigo (BigDecimal value);
   public abstract java.math.BigDecimal getCodigo ();

   public abstract void setCNPJ (String value);
   public abstract String getCNPJ ();

   public abstract void setNome (String value);
   public abstract String getNome ();

...

   public abstract void setCurso (Set value);
   public abstract Set getCurso ();

   public abstract void setVestibular (Set value);
   public abstract Set getVestibular ();

   public Curso createCurso(java.math.BigDecimal codigo,String descricao)
      throws Exception {
      try {
        Context ctx = new InitialContext();
        CursoHome ejbHome
        = (CursoHome) ctx.lookup("vestibularEasyNet.CursoEJB");
        Curso ejb = ejbHome.create(codigo,descricao);
        getCurso().add(ejb);
        return ejb;
      } catch (Exception e) {
        throw new Exception(e.getMessage());
      }
   }

   public Set getFromRelationShipCurso () {
      HashSet hs = new HashSet();
      Iterator i = getCurso().iterator();
      while (i.hasNext()) {
        hs.add(i.next());
      }
      return (Set) hs;
   }

   public void removeCurso (java.math.BigDecimal codigo)
      throws Exception {
     try {
      Context ctx = new InitialContext();
      CursoHome ejbHome
      = (CursoHome) ctx.lookup("vestibularEasyNet.CursoEJB");
      CursoPK pk = new CursoPK(codigo);
      Curso ejb = ejbHome.findByPrimaryKey(pk);
      ejb.remove();
     } catch (Exception e) {
      throw new Exception(e.getMessage());
     }
   }

   public Vestibular createVestibular(BigDecimal codigo,
                                      String descricao,
                                      Timestamp inscricaoInicio,
                                      Timestamp inscricaoFim,
                                      BigDecimal numeroOpcoes,
                                      BigDecimal valorInscricao)
   throws Exception {
   try {
      Context ctx = new InitialContext();
      VestibularHome ejbHome
      = (VestibularHome) tx.lookup("vestibularEasyNet.VestibularEJB");
      Vestibular ejb
             = ejbHome.create(codigo, descricao, inscricaoInicio,
                              inscricaoFim,numeroOpcoes,valorInscricao);
```

```
      getVestibular().add(ejb);
      return ejb;
   } catch (Exception e) {
       throw new Exception(e.getMessage());
   }
   }

   public Set getFromRelationShipVestibular () {
      HashSet hs = new HashSet();
      Iterator i = getVestibular().iterator();
      while (i.hasNext()) {
        hs.add(i.next());
      }
      return (Set) hs;
   }

   public void removeVestibular (java.math.BigDecimal codigo)
      throws Exception {
   try {
      Context ctx = new InitialContext();
      VestibularHome ejbHome
       =(VestibularHome) ctx.lookup("vestibularEasyNet.VestibularEJB");
      VestibularPK pk = new VestibularPK(codigo);
      Vestibular ejb = ejbHome.findByPrimaryKey(pk);
      ejb.remove();
   } catch (Exception e) {
       throw new Exception(e.getMessage());
   }
   }
   public String toXml() {
      StringBuffer sb = new StringBuffer();
      sb.append("<Instituicao>\n");
      sb.append("\t<codigo>" + getCodigo() + "</codigo>\n");
      sb.append("\t<cNPJ>" + getCNPJ() + "</cNPJ>\n");
      sb.append("\t<nome>" + getNome() + "</nome>\n");
      sb.append("\t<rua>" + getRua() + "</rua>\n");
      sb.append("\t<numero>" + getNumero() + "</numero>\n");
      sb.append("\t<complemento>" + getComplemento() + "</complemento>\n");
      sb.append("\t<bairro>" + getBairro() + "</bairro>\n");
      sb.append("\t<cEP>" + getCEP() + "</cEP>\n");
      sb.append("\t<cidade>" + getCidade() + "</cidade>\n");
      sb.append("\t<estado>" + getEstado() + "</estado>\n");
      sb.append("\t<telefones>" + getTelefones() + "</telefones>\n");
      sb.append("\t<email>" + getEmail() + "</email>\n");
      sb.append("\t<senha>" + getSenha() + "</senha>\n");
      sb.append("\t<contato>" + getContato() + "</contato>\n");
      sb.append("\t<status>" + getStatus() + "</status>\n");
      sb.append("\t<boletoNomeBanco>" + getBoletoNomeBanco() +
                     "</boletoNomeBanco>\n");
      sb.append("\t<boletoCodBanco>" + getBoletoCodBanco() +
                     "</boletoCodBanco>\n");
      sb.append("\t<boletoCodAgencia>" + getBoletoCodAgencia() +
                     "</boletoCodAgencia>\n");
      sb.append("\t<boletoCodConta>" + getBoletoCodConta() +
                     "</boletoCodConta>\n");
      sb.append("\t<boletoCarteira>" + getBoletoCarteira() +
                     "</boletoCarteira>\n");
      sb.append("\t<boletoConvenio>" + getBoletoConvenio() +
                     "</boletoConvenio>\n");
      sb.append("</Instituicao>\n");
      return sb.toString();
   }
   public InstituicaoPK ejbCreate (BigDecimal codigo,
                                   String cNPJ,
                                   String nome,
                                   String rua,
                                   String numero,
                                   String bairro,
```

```
                                    String cEP,
                                    String cidade,
                                    String estado,
                                    String telefones,
                                    String email,
                                    String senha,
                                    String contato,
                                    String status,
                                    String boletoNomeBanco,
                                    String boletoCodBanco,
                                    String boletoCodAgencia,
                                    String boletoCodConta,
                                    String boletoCarteira,
                                    String boletoConvenio)
      throws CreateException {
   setCodigo(codigo);
   setCNPJ(cNPJ);
   setNome(nome);
   setRua(rua);
   setNumero(numero);
   setBairro(bairro);
   setCEP(cEP);
   setCidade(cidade);
   setEstado(estado);
   setTelefones(telefones);
   setEmail(email);
   setSenha(senha);
   setContato(contato);
   setStatus(status);
   setBoletoNomeBanco(boletoNomeBanco);
   setBoletoCodBanco(boletoCodBanco);
   setBoletoCodAgencia(boletoCodAgencia);
   setBoletoCodConta(boletoCodConta);
   setBoletoCarteira(boletoCarteira);
   setBoletoConvenio(boletoConvenio);
   return null;
}
public void ejbPostCreate (BigDecimal codigo,
                           String cNPJ,
                           String nome,
                           String rua,
                           String numero,
                           String bairro,
                           String cEP,
                           String cidade,
                           String estado,
                           String telefones,
                           String email,
                           String senha,
                           String contato,
                           String status,
                           String boletoNomeBanco,
                           String boletoCodBanco,
                           String boletoCodAgencia,
                           String boletoCodConta,
                           String boletoCarteira,
                           String boletoConvenio)
   throws CreateException {}

public void setEntityContext (EntityContext ctx) {
   this.ctx = ctx;
}
public void unsetEntityContext() {
   this.ctx = null;
}
public void ejbLoad() { }
public void ejbStore() { }
public void ejbActivate() { }
```

```
   public void ejbPassivate() { }
   public void ejbRemove() { }
}
```

2.3.4.3.5 Deployment Descriptors (Descritores de publicação)

Os *deployment descriptors* são arquivos XML que configuram aplicações e componentes, assim como os componentes EJB. De acordo com a especificação EJB 2.0 apenas o arquivo `ejb-jar.xml` tem o seu conteúdo rigorosamente definido, sendo que, para cada servidor EJB, o seu fabricante define outros arquivos, assim como o seu conteúdo. A especificação EJB não define completamente como devem ser os *deployment descriptors* deixando esta responsabilidade com cada fabricante de servidor EJB. Isso impossibilita que um container uma vez gerado seja publicado em servidores EJB distintos, necessitando de uma nova operação de deploy específica para o servidor EJB alvo da publicação do container.

O Entity Bean Instituição usado nos exemplos foi desenvolvido como parte da aplicação de Vestibular. Esta aplicação foi publicada no servidor EJB Weblogic 6.0, na época o único compatível com a especificação EJB 2.0

Para o servidor EJB Weblogic os seguintes arquivos completam os *deployment descriptors* para Entity Beans CMP: `weblogic-ejb.jar` e `weblogic-cmp-rdbms.jar`. A tabela abaixo resume o conteúdo de cada desses:

Tabela 2. Deployment Descriptors

Arquivo	Conteúdo
`ejb-jar.xml`	Informações sobre o componente, tais como: nome e tipo do componente, suas interfaces e classes, versão da especificação, consultas EJBQL e definições de segurança e controle transacional.
`weblogic-cmp-rdbms.jar`	Informações para a associação entre o Entity Bean e Banco de Dados, tais como: conexão com a fonte de dados, nome da tabela e mapeamento entre atributos e campos e relacionamentos.
`weblogic-ejb.jar`	Informações para publicação no servidor, tais como: Nome JNDI, estratégia de controle no aceso concorrente, configuração do pool, etc.

Código Fonte 5. Arquivo ejb-jar.jar

```
<?xml version="1.0"?>
<!DOCTYPE ejb-jar PUBLIC '-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans
2.0//EN' 'http://java.sun.com/j2ee/dtds/ejb-jar_2_0.dtd'>
<ejb-jar>
   <display-name>VestibularEasyNetEJB</display-name>
   <enterprise-beans>
      <entity>
        <ejb-name>InstituicaoEJB</ejb-name>
        <home>br.com.easyNet.vestibular.ejb.instituicao.InstituicaoHome</home>
        <remote>br.com.easyNet.vestibular.ejb.instituicao.Instituicao</remote>
        <ejb-class>br.com.easyNet.vestibular.ejb.instituicao.InstituicaoBean</ejb-
class>
        <persistence-type>Container</persistence-type>
        <prim-key-
```

```
class>br.com.easyNet.vestibular.ejb.instituicao.InstituicaoPK</prim-key-class>
        <reentrant>False</reentrant>
        <cmp-version>2.x</cmp-version>
        <abstract-schema-name>InstituicaoEJB</abstract-schema-name>
        <cmp-field>
           <field-name>codigo</field-name>
        </cmp-field>
        <cmp-field>
           <field-name>cNPJ</field-name>
        </cmp-field>
        <cmp-field>
           <field-name>nome</field-name>
        </cmp-field>
        <cmp-field>
           <field-name>rua</field-name>
        </cmp-field>
        <cmp-field>
           <field-name>numero</field-name>
        </cmp-field>
        <cmp-field>
           <field-name>complemento</field-name>
        </cmp-field>
        <cmp-field>
           <field-name>bairro</field-name>
        </cmp-field>
        <cmp-field>
           <field-name>cEP</field-name>
        </cmp-field>
        <cmp-field>
           <field-name>cidade</field-name>
        </cmp-field>
        <cmp-field>
           <field-name>estado</field-name>
        </cmp-field>
        <cmp-field>
           <field-name>telefones</field-name>
        </cmp-field>
        <cmp-field>
           <field-name>email</field-name>
        </cmp-field>
        <cmp-field>
           <field-name>senha</field-name>
        </cmp-field>
        <cmp-field>
           <field-name>contato</field-name>
        </cmp-field>
        <cmp-field>
           <field-name>status</field-name>
        </cmp-field>
        <cmp-field>
           <field-name>boletoNomeBanco</field-name>
        </cmp-field>
        <cmp-field>
           <field-name>boletoCodBanco</field-name>
        </cmp-field>
        <cmp-field>
           <field-name>boletoCodAgencia</field-name>
        </cmp-field>
        <cmp-field>
           <field-name>boletoCodConta</field-name>
        </cmp-field>
        <cmp-field>
           <field-name>boletoCarteira</field-name>
        </cmp-field>
        <cmp-field>
           <field-name>boletoConvenio</field-name>
        </cmp-field>
        <query>
```

```
            <query-method>
               <method-name>findAll</method-name>
               <method-params/>
            </query-method>
            <ejb-ql>
               <![CDATA[SELECT OBJECT(o) FROM InstituicaoEJB AS o]]>
            </ejb-ql>
         </query>
         <query>
            <query-method>
               <method-name>findByCNPJ</method-name>
               <method-params>
                 <method-param>java.lang.String</method-param>
               </method-params>
            </query-method>
            <ejb-ql>
            <![CDATA[SELECT OBJECT(o) FROM InstituicaoEJB AS o WHERE o.cNPJ = ?1 ]]>
            </ejb-ql>
         </query>
         <query>
            <query-method>
               <method-name>findByEmail</method-name>
               <method-params>
                 <method-param>java.lang.String</method-param>
               </method-params>
            </query-method>
            <ejb-ql>
            <![CDATA[SELECT OBJECT(o) FROM InstituicaoEJB AS o WHERE o.email = ?1 ]]>
            </ejb-ql>
         </query>
      </entity>
      <ejb-relation>
         <ejb-relation-name>Instituicao-Curso</ejb-relation-name>
         <ejb-relationship-role>
            <ejb-relationship-role-name>instituicao</ejb-relationship-role-name>
            <multiplicity>one</multiplicity>
            <relationship-role-source>
               <ejb-name>InstituicaoEJB</ejb-name>
            </relationship-role-source>
            <cmr-field>
               <cmr-field-name>curso</cmr-field-name>
               <cmr-field-type>java.util.Set</cmr-field-type>
            </cmr-field>
         </ejb-relationship-role>
         <ejb-relationship-role>
            <ejb-relationship-role-name>curso</ejb-relationship-role-name>
            <multiplicity>many</multiplicity>
            <relationship-role-source>
               <ejb-name>CursoEJB</ejb-name>
            </relationship-role-source>
            <cmr-field>
               <cmr-field-name>instituicao</cmr-field-name>
            </cmr-field>
         </ejb-relationship-role>
      </ejb-relation>
      <ejb-relation>
         <ejb-relation-name>Instituicao-Vestibular</ejb-relation-name>
         <ejb-relationship-role>
            <ejb-relationship-role-name>instituicao</ejb-relationship-role-name>
            <multiplicity>one</multiplicity>
            <relationship-role-source>
               <ejb-name>InstituicaoEJB</ejb-name>
            </relationship-role-source>
            <cmr-field>
               <cmr-field-name>vestibular</cmr-field-name>
               <cmr-field-type>java.util.Set</cmr-field-type>
            </cmr-field>
         </ejb-relationship-role>
```

```
        <ejb-relationship-role>
           <ejb-relationship-role-name>vestibular</ejb-relationship-role-name>
           <multiplicity>many</multiplicity>
           <relationship-role-source>
              <ejb-name>VestibularEJB</ejb-name>
           </relationship-role-source>
           <cmr-field>
              <cmr-field-name>instituicao</cmr-field-name>
           </cmr-field>
        </ejb-relationship-role>
     </ejb-relation>
   <assembly-descriptor>
     <container-transaction>
        <method>
           <ejb-name>InstituicaoEJB</ejb-name>
           <method-name>*</method-name>
        </method>
        <trans-attribute>Required</trans-attribute>
     </container-transaction>
     <container-transaction>
        <method>
           <ejb-name>OpcoesEJB</ejb-name>
           <method-name>*</method-name>
        </method>
        <trans-attribute>Required</trans-attribute>
     </container-transaction>
   </assembly-descriptor>
</ejb-jar>
```

Código Fonte 6. Arquivo weblogic-cmp-rdbms.jar

```
<?xml version="1.0"?>
<!DOCTYPE weblogic-rdbms-jar PUBLIC '-//BEA Systems, Inc.//DTD WebLogic 6.0.0 EJB
RDBMS Persistence//EN' 'http://www.bea.com/servers/wls600/dtd/weblogic-rdbms20-
persistence-600.dtd'>
<weblogic-rdbms-jar>
   <weblogic-rdbms-bean>
     <ejb-name>InstituicaoEJB</ejb-name>
     <data-source-name>vestibular.DataSource</data-source-name>
     <table-name>Instituicao</table-name>
     <field-map>
        <cmp-field>codigo</cmp-field>
        <dbms-column>codigo</dbms-column>
     </field-map>
     <field-map>
        <cmp-field>cNPJ</cmp-field>
        <dbms-column>cNPJ</dbms-column>
     </field-map>
     <field-map>
        <cmp-field>nome</cmp-field>
        <dbms-column>nome</dbms-column>
     </field-map>
     <field-map>
        <cmp-field>rua</cmp-field>
        <dbms-column>rua</dbms-column>
     </field-map>
     <field-map>
        <cmp-field>numero</cmp-field>
        <dbms-column>numero</dbms-column>
     </field-map>
     <field-map>
        <cmp-field>complemento</cmp-field>
        <dbms-column>complemento</dbms-column>
     </field-map>
     <field-map>
        <cmp-field>bairro</cmp-field>
        <dbms-column>bairro</dbms-column>
     </field-map>
```

```
      <field-map>
        <cmp-field>cEP</cmp-field>
        <dbms-column>cEP</dbms-column>
      </field-map>
      <field-map>
        <cmp-field>cidade</cmp-field>
        <dbms-column>cidade</dbms-column>
      </field-map>
      <field-map>
        <cmp-field>estado</cmp-field>
        <dbms-column>estado</dbms-column>
      </field-map>
      <field-map>
        <cmp-field>telefones</cmp-field>
        <dbms-column>telefones</dbms-column>
      </field-map>
      <field-map>
        <cmp-field>email</cmp-field>
        <dbms-column>email</dbms-column>
      </field-map>
      <field-map>
        <cmp-field>senha</cmp-field>
        <dbms-column>senha</dbms-column>
      </field-map>
      <field-map>
        <cmp-field>contato</cmp-field>
        <dbms-column>contato</dbms-column>
      </field-map>
      <field-map>
        <cmp-field>status</cmp-field>
        <dbms-column>status</dbms-column>
      </field-map>
      <field-map>
        <cmp-field>boletoNomeBanco</cmp-field>
        <dbms-column>boletoNomeBanco</dbms-column>
      </field-map>
      <field-map>
        <cmp-field>boletoCodBanco</cmp-field>
        <dbms-column>boletoCodBanco</dbms-column>
      </field-map>
      <field-map>
        <cmp-field>boletoCodAgencia</cmp-field>
        <dbms-column>boletoCodAgencia</dbms-column>
      </field-map>
      <field-map>
        <cmp-field>boletoCodConta</cmp-field>
        <dbms-column>boletoCodConta</dbms-column>
      </field-map>
      <field-map>
        <cmp-field>boletoCarteira</cmp-field>
        <dbms-column>boletoCarteira</dbms-column>
      </field-map>
      <field-map>
        <cmp-field>boletoConvenio</cmp-field>
        <dbms-column>boletoConvenio</dbms-column>
      </field-map>
   </weblogic-rdbms-bean>
   <weblogic-rdbms-relation>
      <relation-name>Instituicao-Curso</relation-name>
      <weblogic-relationship-role>
        <relationship-role-name>curso</relationship-role-name>
        <column-map>
           <foreign-key-column>codigoInstituicao</foreign-key-column>
           <key-column>codigo</key-column>
        </column-map>
      </weblogic-relationship-role>
   </weblogic-rdbms-relation>
   <weblogic-rdbms-relation>
```

```
      <relation-name>Instituicao-Vestibular</relation-name>
      <weblogic-relationship-role>
         <relationship-role-name>vestibular</relationship-role-name>
         <column-map>
            <foreign-key-column>codigoInstituicao</foreign-key-column>
            <key-column>codigo</key-column>
         </column-map>
      </weblogic-relationship-role>
   </weblogic-rdbms-relation>
</weblogic-rdbms-jar>
```

Código Fonte 7. Arquivo weblogic-ejb-jar.jar

```
<?xml version="1.0"?>
<!DOCTYPE weblogic-ejb-jar PUBLIC '-//BEA Systems, Inc.//DTD WebLogic 6.0.0
EJB//EN' 'http://www.bea.com/servers/wls600/dtd/weblogic-ejb-jar.dtd'>
<weblogic-ejb-jar>
   <weblogic-enterprise-bean>
      <ejb-name>InstituicaoEJB</ejb-name>
      <entity-descriptor>
         <entity-cache>
            <max-beans-in-cache>20</max-beans-in-cache>
            <read-timeout-seconds>600</read-timeout-seconds>
            <concurrency-strategy>Database</concurrency-strategy>
         </entity-cache>
         <lifecycle>
            <passivation-strategy>Default</passivation-strategy>
         </lifecycle>
         <persistence>
            <persistence-type>
               <type-identifier>WebLogic_CMP_RDBMS</type-identifier>
               <type-version>6.0</type-version>
               <type-storage>META-INF/weblogic-cmp-rdbms-jar.xml</type-storage>
            </persistence-type>
               <db-is-shared>True</db-is-shared>
            <persistence-use>
               <type-identifier>WebLogic_CMP_RDBMS</type-identifier>
               <type-version>6.0</type-version>
            </persistence-use>
         </persistence>
      </entity-descriptor>
      <enable-call-by-reference>True</enable-call-by-reference>
      <jndi-name>vestibularEasyNet.InstituicaoEJB</jndi-name>
   </weblogic-enterprise-bean>
</weblogic-ejb-jar>
```

2.3.4.3.6 A Operação de Deployment (Preparação para Publicação)

Para que o Entity Bean gerado seja publicado no servidor Weblogic **[BEA02]** é necessário que o mesmo passe pela operação de *Deployment*. A operação necessita dos elementos do Entity Bean (interfaces, classes e *deployment descriptors*) e fornece como resultado um arquivo jar contendo os elementos do Entity Bean e um conjunto de classes de serviços. Essas classes de serviço irão utilizar a infra-estrutura da plataforma para o funcionamento do Bean.

A operação necessita de uma série de atividades, tais como: compilação das interfaces e classes do componente, agrupamento destas em um único arquivo, geração das classes de

serviços realizadas através de utilitários do servidor EJB e finalmente o agrupamento final de todos os elementos. Para realização da operação, normalmente, é utilizada a API ant **[APA02]**, a qual faz parte do projeto Apache Jakarta **[APA01]**. A API ant tem a mesma funcionalidade dos arquivos *makefile*, muito utilizados na compilação de programas, sendo que a sua entrada é um arquivo XML (Veja figura abaixo) contendo as operações que devem ser realizadas, tais como criação de diretórios, cópia de arquivos, compilação, agrupamento de arquivos, etc.

Após a operação de *deployment* o arquivo jar gerado deve ser publicado no servidor EJB. Cada servidor EJB fornecer suas ferramentas para publicação. No caso do Weblogic existe uma ferramenta web que permite toda a sua administração, inclusive publicação de beans EJB.

Código Fonte 8. Arquivo build.xml

```
<project name="VestibularEasyNetEJB" default="all" basedir=".">
   <property name="JAVAC" value="modern"/>
   <property name="WL_HOME" value="D:/DevTools/Bea/wlserver6.1"/>
   <property name="source" value="."/>
   <property name="javasources" value="${source}/.."/>
   <property name="build" value="${source}/../build"/>
   <property name="dist" value="${source}/../dist"/>
   <target name="all" depends="clean, init, compile_ejb, jar_ejb, ejbc"/>
   <target name="init">
      <tstamp/>
      <mkdir dir="${build}"/>
      <mkdir dir="${build}/META-INF"/>
      <mkdir dir="${dist}"/>
      <copy todir="${build}/META-INF">
        <fileset dir="${source}">
           <include name="*.xml"/>
           <exclude name="build.xml"/>
        </fileset>
      </copy>
   </target>
   <target name="compile_ejb">
      <javac srcdir="${javasources}" destdir="${build}"
        includes="
              br/com/easyNet/vestibular/ejb/instituicao/Instituicao.java,
              br/com/easyNet/vestibular/ejb/instituicao/InstituicaoHome.java,
              br/com/easyNet/vestibular/ejb/instituicao/InstituicaoBean.java,
              br/com/easyNet/vestibular/ejb/instituicao/InstituicaoPK.java
      "/>
   </target>
   <target name="jar_ejb" depends="compile_ejb">
      <jar jarfile="${dist}/VestibularEasyNetEJBClient.jar"
        basedir="${build}">
      </jar>
   </target>
   <target name="ejbc" depends="jar_ejb">
      <java classname="weblogic.ejbc" fork="yes">
        <sysproperty key="weblogic.home" value="${WL_HOME}"/>
        <arg line="-compiler javac ${dist}/VestibularEasyNetEJBClient.jar
${dist}/VestibularEasyNetEJBServer.jar"/>
        <classpath>
           <pathelement
path="${WL_HOME}/lib/weblogic_sp.jar;${WL_HOME}/lib/weblogic.jar"/>
```

```
        </classpath>
      </java>
   </target>
   <target name="clean">
      <delete dir="${build}"/>
      <delete dir="${dist}"/>
   </target>
</project>
```

2.3.4.3.7 Implementando Relacionamentos em Entity Beans CMP

Os Entity Beans também representam os seus relacionamentos existentes entre os dados. De acordo com a especificação 2.0 para Entity Beans CMP é possível definir as propriedades e atributos dos relacionamentos e deixar que o container faça o gerenciamento automaticamente dos mesmos.

O Entity Bean utilizado como exemplo possui dois relacionamentos com outros Entity Beans, sendo eles: Curso e Vestibular, os quais representam os Cursos e Vestibulares cadastrados pela Instituição. A seguir serão listados os passos necessários para a implementação dos relacionamentos:

1. Na Classe Abstrata é necessário definir métodos `set` e `get` para os Entity Beans relacionados, sendo que, quando a cardinalidade do relacionamento for 1, o parâmetro para o método `set` e o retorno do método `get` deve ser do tipo da Interface Remota do Entity Bean relacionado e quando a cardinalidade for N, N > 1, deve ser do tipo `java.util.Set` ou `Java.util.Collection`;
2. Os métodos `set` e `get` dos relacionamentos não podem ser ofertados diretamente à camada cliente;
3. Os relacionamentos devem ser definidos nos *deployment descriptors*.

2.3.4.4 Utilizando o Entity Beans a partir da Camada Cliente

A utilização de objetos remotos exige uma codificação diferenciada da programação orientada a objetos convencional, sendo que o aumento da complexidade adicionado por conta disso é mínimo e a produtividade e re-aproveitamento da orientação a objetos continuam bem evidentes.

A seguir serão demonstradas as principais operações a serem realizadas através do Entity Bean Vestibular, criado como exemplificação nas sessões anteriores.

2.3.4.4.1 Obtendo o contexto EJB para o container

A primeira operação para utilizar componentes EJB é a aquisição de uma referência para o serviço de nomes disponibilizado no servidor EJB. Lembrando que todos os

componentes EJB são publicados no servidor EJB e seu acesso remoto é feito através da tecnologia JNDI.

O trecho de programa abaixo demonstra a codificação necessária para a obtenção do contexto JNDI, onde o objeto `InitialContext` representa o acesso ao serviço de nomes JNDI e a sua construção irá buscar uma referência para o serviço JNDI no servidor EJB. Neste caso o servidor Weblogic. O objeto `Context` representará o acesso efetivo ao serviço de nomes e caso ocorra algum erro na construção do objeto será lançada a exceção `javax.naming.NamingException`.

Código Fonte 9. Obtendo acesso ao serviço de Nomes JNDI

```
...
import javax.naming.Context;
import javax.naming.InitialContext;
...
    Context ctx = new InitialContext();
...
```

O código acima será executado pelo próprio servidor EJB, assim não foi necessário especificar detalhes sobre o serviço de nomes JNDI, pois os valores padrões publicados no servidor EJB serão utilizados. Caso seja necessário especificar o acesso a outro servidor EJB ou o código represente uma aplicação a ser executada remotamente ao servidor EJB, é necessário ajustar as propriedades do sistema (Máquina Virtual Java). O código fonte abaixo demonstra a especificação das propriedades do sistema para o servidor EJB Weblogic. A primeira definem a classe para implementar o contexto JNDI e a segunda especifica o servidor de nomes a ser utilizado, onde **t3** representa o protocolo a ser utilizado.

Código Fonte 10. Especificando os parâmetros JNDI

```
...
    System.setProperty("java.naming.factory.initial",
                       "weblogic.jndi.WLInitialContextFactory");
    System.setProperty("java.naming.provider.url",
                       "t3://localhost:7001");
    Context ctx = new InitialContext();
...
```

2.3.4.4.2 Obtendo a Interface Home do Entity Bean

Após obter a referência ao contexto JNDI é possível obter a referência à Interface Home de componentes EJB. Isso é feito através de uma operação denominada “lookup”, a qual representa a solicitação de um objeto ao serviço de nomes JNDI. No caso de componentes EJB, o objeto recebido será o STUB gerado pelo container na operação de *deployment*.

O Código abaixo demonstra a obtenção da interface Home para o Entity Bean Instituição, onde o parâmetro para o método `lookup` é nome JNDI do Entity Bean.

Código Fonte 11. Obtendo a Interface Home do Entity Bean

```
...
  try {
    Context ctx = new InitialContext();
    String jndiName = "vestibularEasyNet.InstituicaoEJB"
    InstituicaoHome instHome = (InstituicaoHome) ctx.lookup(jndiName);
  } catch (NamingException e) {
    throw new NamingException ("Erro localizando interface remota : " +
                               jndiName);
  }
...
```

2.3.4.4.3 Criando um novo Entity Bean

A Interface Home permite cuidar do ciclo de vida dos componentes EJB. No caso de Entity Beans, os quais são objetos de negócios representando dados persistentes, a criação de um novo Entity Bean representa a inserção de dados na fonte de dados a qual o mesmo representa. O método `create`, o qual é definido pela especificação EJB e foi implementado no Entity Bean Instituição, será o responsável pela criação de um novo Entity Bean e o container irá cuidar da comunicação com o SGBD inserindo uma nova linha na tabela "Instituicao". De acordo com a especificação EJB, o método `create` retorna a Interface Remota do Entity Bean e pode lançar as seguintes exceções `javax.ejb.CreateException` e `javax.ejb.DuplicateKeyException`.

O código abaixo representa a utilização do método `create` da Interface Home do Entity Bean Instituição. O métodos `getChave` foi implementado para obter o próximo valor da chave primária de uma tabela e objeto `sub` faz parte da API do EasyNet e representa os dados fornecidos pelo usuário.

Código Fonte 12. Utilizando a Interface Home

```
...
import javax.naming.NamingException;
import javax.ejb.CreateException;
import javax.ejb.DuplicateKeyException;
import br.com.easyNet.vestibular.ejb.instituicao.InstituicaoHome;
import br.com.easyNet.vestibular.ejb.instituicao.Instituicao;
...
  try {
    Context ctx = new InitialContext();
    String jndiName = "vestibularEasyNet.InstituicaoEJB"
    InstituicaoHome instHome = (InstituicaoHome) ctx.lookup(jndiName);
    Instituicao inst =
      instHome.create(getChave("Instituicao"),
                      (String) sub.getObject("CNPJ"),
                      (String) sub.getObject("Nome"),
                      (String) sub.getObject("Rua"),
                      (String) sub.getObject("Numero"),
```

```
                        (String) sub.getObject("Bairro"),
                        (String) sub.getObject("CEP"),
                        (String) sub.getObject("Cidade"),
                        (String) sub.getObject("Estado"),
                        (String) sub.getObject("Telefones"),
                        (String) sub.getObject("Email"),
                        senha,
                        (String) sub.getObject("Contato"),
                        "C",
                        (String) sub.getObject("boletoNomeBanco"),
                        (String) sub.getObject("boletoCodBanco"),
                        (String) sub.getObject("boletoCodAgencia"),
                        (String) sub.getObject("boletoCodConta"),
                        (String) sub.getObject("boletoCarteira"),
                        (String) sub.getObject("boletoConvenio"));
    inst.setComplemento((String) sub.getObject("Complemento"));
  } catch (NamingException e) {
    throw new NamingException ("Erro localizando interface remota: " +
                               jndiName);
  } catch (DuplicateKeyException e) {
    throw new DuplicateKeyException ("Chave repetida");
  } catch (CreateException e) {
    throw new CreateException ("Erro na inserção da Instituição: " +
                               e.getMessage());
  }
...
```

2.3.4.4.4 Localizando um Entity Bean

Para localizar um Entity Bean é necessário utilizar um dos métodos `finders` da Interface Home, sendo que, quando o retorno do método for a Interface Remota e o Entity Bean não for encontrado a exceção `javax.ejb.ObjectNotFoundException` será lançada e quando o retorno for uma coleção e nenhum Entity Bean for encontrado será retornada uma coleção vazia. Em ambos os caso a exceção `javax.ejb.FinderException` será lançada pelo container quando a pesquisa não puder ser realizada.

O código fonte abaixo demonstra a pesquisa pelo CNPJ para o Entity Bean Instituição.

Código Fonte 13. Localizando Entity Beans

```
...
import javax.naming.NamingException;
import javax.ejb. ObjectNotFoundException;
import br.com.easyNet.vestibular.ejb.instituicao.InstituicaoHome;
import br.com.easyNet.vestibular.ejb.instituicao.Instituicao;
...
  try {
    Context ctx = new InitialContext();
    String jndiName = "vestibularEasyNet.InstituicaoEJB"
    InstituicaoHome instHome = (InstituicaoHome) ctx.lookup(jndiName);
    String cnpj = (String) doc.getObject("CNPJ");
    Instituicao inst = inst.findByCNPJ(cnpj);
...
  } catch (NamingException e) {
    throw new NamingException ("Erro localizando interface remota: " +
                               jndiName);
  } catch (ObjectNotFoundException e) {
    throw new ObjectNotFoundException ("Instituição não encontrada");
  }
...
```

2.3.4.4.5 Utilizando métodos do Entity Bean

Após obter a Interface Remota para um Entity Bean, a utilização de seus métodos é semelhante à utilização de objetos na programação orientada a objetos convencional.

2.3.4.4.6 Removendo o Entity Bean

Para remover um Entity Bean e consequentemente os dados que o mesmo representa, basta chamar o método `remove` da sua Interface Remota. Caso os dados não possam ser removidos a exceção `javax.ejb.RemoveException` será lançada.

2.3.4.5 Session Bean

Os Session Beans são componentes EJB que representam regras de negócios, fazendo assim o papel de uma extensão da camada cliente. Diferentemente dos Entity Beans, os quais representam dados persistentes, os Session Bean representam transações e normalmente utilizam-se dos Entity Beans para executar tarefas mais complexas, normalmente com a participação de vários Entity Beans.

Os Entity Beans podem possuir regras de negócios, mas estas normalmente estarão relacionadas aos dados que os mesmos representam. Os Session Bean representam regras de negócios mais flexíveis, as quais irão esconder as complexas operações do ambiente corporativo. Quanto melhor o projeto e implementação dos Session Beans, mais fácil e produtiva será a implementação da camada cliente.

Normalmente, cada Session Bean irá representar um cliente na camada servidora e eles serão responsáveis por atender todas as solicitações por lógicas de negócios realizadas pelas aplicações clientes. São objetos que seguem a especificação EJB e possuem uma coleção de métodos a ser ofertada à camada cliente. Os métodos de um Session Bean possuem controle transacional garantido pelo container, onde todas as operações com os dados, tanto via Entity Beans como JDBC, serão desfeitas automaticamente caso seja lançada a exceção `javax.ejb.EJBException`.

Um Session Bean, assim como qualquer objeto, pode possuir atributos, sendo que estes, diferentemente dos Entity Beans, não serão persistidos. Existem dois tipos de Session Bean: com estado (*Statefull*) e sem estado (*Stateless*). Essa classificação é feita de acordo com a relação que o Session Bean tem com o cliente. Os Session Beans com estado são alocados pelo cliente e mantêm uma sessão com o mesmo. Já os Session Beans sem estado não mantêm nenhum vínculo com o cliente, a não ser no momento da execução de um dos seus métodos.

2.3.4.6 Session Beans Sem Estado

Quando um cliente interage com um Session Bean sem estado, este simplesmente executa o método solicitado e retorna para a coleção (*pool*) de Session Beans do container. Dessa forma não existe nenhum vínculo entre o Session Bean e o cliente que o está utilizando. Um cliente pode solicitar a mesma operação várias vezes e em cada uma delas poderá ser atendido por uma instância diferente do Session Bean, porque, a cada solicitação o container aloca, de acordo com os seus critérios, um Session Bean para atender ao cliente. Os Session Beans sem estado são mais simples de implementar e fornecem uma grande escalabilidade, pois um conjunto pequeno pode atender a uma quantidade grande de clientes.

2.3.4.7 Session Beans Com Estado

Quando um cliente interage com um Session Bean com estado, este é alocado ao cliente durante todo o seu ciclo de vida. Assim, caso exista uma determinada quantidade de clientes utilizando Session Beans com estado, para cada cliente haverá um Session Bean no pool de beans do container. Essa associação permite que o Session Bean armazene informações necessárias à execução de seus métodos durante a sua utilização pelo cliente.

A utilização de Session Beans com estado aumenta o acoplamento entre a camada cliente e a camada intermediária e o preço pago é a redução da capacidade de escala.

2.3.4.8 Criando um Session Bean Sem Estado

O Session Bean implementado com demonstração é do tipo sem estado e representa uma operação muito comum em aplicações: a geração automática de valores para campos chave primária. Essa operação normalmente é implementada através dos campos auto-incrementáveis fornecidos por alguns SGBDs.

Será criado um Session Bean (Cadastro), o qual irá utilizar um Entity Bean (Chave) para armazenar os valores para a chave primária de cada tabela. O Entity Bean Chave estará associado a uma tabela que possuirá uma linha para cada tabela que se deseja gerar chaves automaticamente.

Sempre que uma nova chave precisar ser gerada, o método `getChave` do Session Bean Cadastro será chamado passando-se como parâmetro o nome da tabela e tendo como retorno o valor da Chave, ambos serão valores do tipo `java.lang.String`. Quando o nome da tabela solicitada não existir na tabela com as chaves será retornado o valor “1” e inserida uma nova linha com o nome da tabela e valor da próxima chave como sendo “2”.

É importante observar que essa operação possui uma região crítica, pois a operação de leitura e escrita do novo valor da chave deve ser atômica e não permitir acesso concorrente para evitar que o mesmo valor da chave seja fornecido a dois clientes. Os atributos do controle transacional para os métodos dos Session Beans são definidos no seu *deployment descriptor*.

2.3.4.8.1 Elementos do Session Bean

Assim como os Entity Beans, os Session Beans devem seguir rigorosamente as especificações EJB e os componentes serão os mesmo, exceto pela inexistência da classe chave primária, pois os Session Beans não representam dados persistentes. O Session Bean utilizado como demonstração possui os seguintes componentes: CadastroHome, Cadastro, CadastroBean.

2.3.4.8.2 Interface Home

A interface Home de Session Beans Stateless necessita apenas do método `create` sem nenhum parâmetro, representando um construtor padrão e segue as mesmas regras para criação da Interface Home dos Entity Beans.

Código Fonte 14. Interface Home para o Session Bean

```
package br.com.easyNet.vestibular.ejb.session.cadastro;

import java.rmi.RemoteException;
import javax.ejb.CreateException;
import javax.ejb.EJBHome;

public interface CadastroHome extends EJBHome {

  Cadastro create() throws CreateException, RemoteException;

}
```

2.3.4.8.3 Interface Remota

Semelhantemente aos Entity Beans, a Interface Remota deve possuir a assinatura de todos os métodos a serem ofertados pelo Session Bean e segue as mesmas regras da Interface Remota dos Entity Beans.

Código Fonte 15. Interface Remota para o Session Bean

```
package br.com.easyNet.vestibular.ejb.session.cadastro;

import java.rmi.RemoteException;
import javax.ejb.EJBObject;

public interface Cadastro extends EJBObject {
```

```
  public String getChave (String tabela) throws Exception, RemoteException;

}
```

2.3.4.8.4 Classe Abstrata

A Classe Abstrata do Session Beans deve implementar a interface: `javax.ejb.SessionBean` e possuir um atributo privado do tipo `javax.ejb.SessionContext`, o qual irá representa a comunicação com o container e deve ser manipulado pelo método: `setSessionContext`. Como os Session Beans não representam dados persistentes, a interface `SessionBean` define apenas os métodos: `ejbActivate`, `ejbPassivate` e `ejbRemove`, representando os eventos gerados pelo container.

Código Fonte 16. Classe Abstrata para o Session Bean

```
package br.com.easyNet.vestibular.ejb.session.cadastro;

import javax.naming.InitialContext;
import javax.naming.Context;

import javax.ejb.SessionBean;
import javax.ejb.SessionContext;
import javax.ejb.ObjectNotFoundException;
import javax.ejb.EJBException;

import br.com.easyNet.vestibular.ejb.entity.chave.ChaveHome;
import br.com.easyNet.vestibular.ejb.entity.chave.Chave;

public class CadastroBean implements SessionBean {

  private SessionContext ctx;

  public String getChave (String tabela) throws Exception {
    ChaveHome chaveHome = null;
    try {
      Context ctx = new InitialContext();
      chaveHome = (ChaveHome) ctx.lookup("vestibular.Chave");
      Chave chave = chaveHome.findByPrimaryKey(tabela);
      String chaveLida = chave.getChave();
      String chaveParaGravar = String.valueOf(Integer.parseInt(chaveLida) + 1);
      chave.setChave(chaveParaGravar);
      return chaveLida;
    } catch (ObjectNotFoundException onfe) {
      try {
        chaveHome.create(tabela, "2");
      } catch (Exception e) {
        String msg = "Erro inserindo nova tabela/chave";
        throw new EJBException(msg);
      }
      return "1";
    } catch (Exception e) {
      String msg = "Erro gerando nova chave";
      throw new EJBException(msg);
    }
  }

  public void setSessionContext(SessionContext ctx) {
     this.ctx = ctx;
  }

  public void ejbCreate () throws CreateException { }
```

```
  public void ejbActivate() { }

  public void ejbRemove() { }

  public void ejbPassivate() { }

}
```

2.3.4.8.5 Deployment Descriptors

Os *Deployment Descriptors* dos Session Beans são mais simples que os dos Entity Beans, pois não necessitam o mapeamento para a fonte de dados.

Código Fonte 17. Arquivo ejb-jar.jar

```
<?xml version="1.0"?>
<!DOCTYPE ejb-jar PUBLIC '-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans
2.0//EN' 'http://java.sun.com/j2ee/dtds/ejb-jar_2_0.dtd'>
<ejb-jar>
  <display-name>VestibularEJB</display-name>
  <enterprise-beans>

...
    <session>
      <ejb-name>CadastroEJB</ejb-name>
      <home>br.com.easyNet.vestibular.ejb.session.cadastro.CadastroHome</home>
      <remote>br.com.easyNet.vestibular.ejb.session.cadastro.Cadastro</remote>
      <ejb-class>br.com.easyNet.vestibular.ejb.session.cadastro.CadastroBean</ejb-
class>
      <session-type>Stateless</session-type>
      <transaction-type>Container</transaction-type>
    </session>

  </enterprise-beans>

...
  <assembly-descriptor>
...
    <container-transaction>
      <method>
        <ejb-name>CadastroEJB</ejb-name>
        <method-name>*</method-name>
      </method>
      <trans-attribute>Required</trans-attribute>
    </container-transaction>
  </assembly-descriptor>
</ejb-jar>
```

Código Fonte 18. Arquivo weblogic-ejb-jar.jar

```
<?xml version="1.0"?>
<!DOCTYPE weblogic-ejb-jar PUBLIC '-//BEA Systems, Inc.//DTD WebLogic 6.0.0
EJB//EN' 'http://www.bea.com/servers/wls600/dtd/weblogic-ejb-jar.dtd'>
<weblogic-ejb-jar>
...
  <weblogic-enterprise-bean>
    <ejb-name>CadastroEJB</ejb-name>
    <jndi-name>vestibular.Cadastro</jndi-name>
  </weblogic-enterprise-bean>
</weblogic-ejb-jar>
```

2.3.4.9 Utilizando o Session Bean Stateless a partir da Camada Cliente

Numa arquitetura multicamada bem projetada as aplicações clientes devem fazer uso dos Session Beans para realizar qualquer operação no ambiente coorporativo, ficando a cargo das aplicações cliente apenas a manutenção da interface com o usuário.

A operações para **obter o contexto EJB para o container** e a **Interface Home do Session Bean** são idênticas às apresentadas anteriormente para os Entity Beans.

Procedimento idêntico na utilização de Entity Bean. Visto anteriormente neste capítulo.

2.3.4.9.1 Obtendo a Interface Remota do Session Bean

Para obter a Interface Remota de um Session Bean é necessário solicitar a sua criação ao Container, sendo que no caso de Session Beans Stateless, o container irá retornar um dos Session Beans disponíveis no seu pool de beans.

Código Fonte 19. Solicitação de um Session Bean ao Container

```
...
  Context ctx = new InitialContext();
  CadastroHome cadastroHome =
    (CadastroHome) ctx.lookup("vestibular.Cadastro");
  Cadastro cadastro = cadastroHome.create();
...
```

2.3.4.9.2 Utilizando os métodos do Session Bean

Após obter a Interface Remota para um Entity Bean, a utilização de seus métodos é semelhante à utilização de objetos na programação orientada a objetos convencional.

2.3.4.9.3 Finalizando a utilização do Session Bean

Para finalizar a utilização do Session Bean basta chamar o método remove da sua Interface Remota, sendo que o caso dos Session Beans Stateless isso não implicará na remoção do Session Beans do pool de beans do container, mas somente a limpeza dos objetos STUB enviados ao cliente.

2.3.4.10 Message Driven Bean

Esse tipo de EJB permite que aplicações J2EE processem mensagens assíncronas geradas pelo serviço de mensagens JMS (Java Message Service). O Message Driven Bean atua como um receptor de mensagens JMS, as quais podem ser geradas por outro bean EJB, uma aplicação cliente ou um componente web. Diferentemente dos Entity Beans e Session Beans, os Message Driver Beans não são acessados via suas interface e sim ativados pelo container quando da chegada de mensagens. Desta forma, os Message Driver Beans possuem

apenas a Classe Abstrata. Quando uma mensagem JMS chega no servidor, o container chama o método `onMessage` da Classe Abstrata do bean para processar a mensagem. O método `onMessage` representa a regra de negócio fornecida pelo bean, podendo fazer uso de outros componentes EJB: Entity Bean e Session Bean.

Message Driver Beans servem para implementar transações com baixo acoplamento, as quais serão ativadas vias serviço de mensagens.

2.4 Conclusões

Observando a tecnologia Enterprise Java Beans é possível perceber que a mesma apresenta uma alta maturidade e visa fornecer o suporte para o desenvolvimento de aplicações utilizando a arquitetura multicamadas. Embora a tecnologias EJB forneça uma excelente infra-estrutura, a implementação e utilização de seus componentes apresentam níveis altos de complexidades e exigem a execução de um grande número de passos levando a uma baixa produtividade na implementação.

Capítulo 3 - Uma Aplicação Web sobre a Plataforma J2EE

Com o objetivo de fixar e por em prática os conhecimentos adquiridos sobre a plataforma J2EE e principalmente sobre a tecnologia EJB, uma aplicação WEB foi implementada. A implementação da aplicação também teve fundamental importância na validação dos aspectos a serem abordados para alcançar os objetivos propostos: Aumentar a produtividade e reduzir a complexidade.

A aplicação escolhida foi "Portal para Publicação e Inscrição de Vestibulares", por possuir regras de negócios previamente conhecidas através da sua implementação utilizando JDBC e JSP, anterior à dissertação de mestrado e por possuir requisitos típicos de aplicações WEB. A atenção pôde ser totalmente voltada para o processo de implementação, pois os requisitos da aplicação e seu funcionamento em produção já eram bem conhecidos.

3.1 Aplicação Estudo de Caso

A aplicação é um portal permitindo que instituições de ensino publiquem seus vestibulares e que candidatos possam realizar as inscrições nestes vestibulares. Toda a interação com a aplicação será feita através de programas de navegação (Internet Explorer ou Netscape Navigator). A aplicação, por ser um portal, irá permitir a publicação de diversos vestibulares por diversas instituições com o objetivo de centralizar num mesmo local a possibilidade de inscrição em vestibulares de instituições diferentes, aumentando assim a sinergia entre a publicação de vestibulares das instituições.

A aplicação é uma solução de baixo custo para instituições que não possuem conexão permanente com a Internet. A aplicação foi desenvolvida de forma que a instituição fornece seus dados e prepara os vestibulares a serem publicados, em seguida candidatos podem realizar suas inscrições, e cada instituição irá obter um arquivo XML contendo as inscrições realizadas nos seus vestibulares.

O pagamento das inscrições nos vestibulares por parte dos candidatos será realizado via boleto bancário, gerado pela aplicação. Será obrigatório que as instituições de ensino possuam convênio de boleto bancário com algum banco.

A aplicação irá possuir duas visões, uma para as instituições, outra para os candidatos.

3.1.1 Visão da Instituição

Nesta visão, a aplicação irá exibir um texto explicativo sobre o portal e permitir a realização das operações de “Cadastro” e “Login”. Em primeiro lugar a instituição deve realizar seu cadastro, fornecendo informações da instituição (Nome, CNPJ, Endereço, etc), do convênio com o banco (Nome e código do Banco, número da agência e da conta, etc) e definindo sua senha de acesso, a qual junto com o CNPJ será o seu *Login* na aplicação.

Assim que a instituição for cadastrada a mesma possuirá um status de “CADASTRADA” para que as suas informações sejam validadas após um contato a ser realizado pela a equipe de administração da aplicação. Esse contato se faz obrigatório a fim de confirmar a veracidade das informações e permitir que a geração do boleto seja adaptada caso a instituição trabalhe com um banco para o qual a aplicação ainda não está preparada para gerar boletos. Uma vez que os dados estejam confirmados o status da instituição será alterado para “ATIVA” e os seus vestibulares serão ofertados pela aplicação.

Após a realização do cadastro a instituição poderá realizar o seu Login e a aplicação irá fornece opções para: Editar seus dados, cadastrar (*incluir, listar, alterar e excluir*) cursos e vestibulares e para cada vestibular será possível ainda realizar as seguintes operações: adicionar cursos, visualizar sumário de inscrições e gerar o arquivo XML com todas as inscrições realizadas. No cadastro de vestibulares serão fornecidas as datas (hora, dia, mês e ano) para o início e fim das inscrições e a aplicação irá utilizar essas datas a fim de publicar os vestibulares apenas no seu período de inscrição.

3.1.2 Visão do Candidato

Na visão do candidato também será exibido um texto explicativo sobre a aplicação portal e serão oferecidas a opções para “Inscrição” e “Login”. Em primeiro lugar o candidato terá que realizar a sua inscrição, a qual terá quatro passos. No primeiro passo o candidato irá fornecer suas informações pessoais (nome, cpf, endereço, email, etc); no segundo passo serão exibidas informações sobre a inscrição, login e a confirmação de inscrição via email; no terceiro passo o candidato irá escolher a instituição, o vestibular e as opções de cursos, representando a sua inscrição; no quarto passo a aplicação irá gerar o boleto bancário. Posteriormente a inscrição, o candidato poderá realizar login na aplicação a fim de re-emitir o boleto bancário.

A aplicação foi desenvolvida sobre a plataforma J2EE e utilizando suas principais tecnologias e arquitetura multicamadas com três camadas.

3.1.3 Camada de Interface

A camada de interface foi implementada através de páginas HTML (*Hyper Text Markup Language*), CSS (*Cascade Style Shet*), JavaScript, Servlet e JSP (*Java Server Pages*). Tecnologias típicas para esta camada e que mostram-se as mais utilizadas no desenvolvimento de aplicações, sendo que HTML, CSS e JavaScript não são tecnologias Java e podem ser utilizadas por qualquer tecnologia de desenvolvimento de aplicações para Web.

3.1.3.1 HTML

As páginas HTML foram utilizadas para montar a estrutura visual da aplicação e fornecer o conteúdo estático, tais como textos explicativos e menus de opções. A estrutura visual da aplicação foi definida numa página HTML disposta em três frames. Um frame esquerdo contendo o menu de opções; um frame superior contendo o cabeçalho da aplicação e um frame direito representando a área de trabalho da aplicação, onde as operações exibem seus resultados.

3.1.3.2 CSS

Os arquivos CSS foram utilizados para definir as propriedades visuais dos elementos HTML. Uma vez definidas essas propriedades, as mesmas são aproveitadas para definir as propriedades visuais (cor, formatação, fonte, tamanho, etc) de todo o conteúdo HTML, facilitando assim em muito a padronização e manutenção da interface da aplicação. Uma vez alterada uma propriedade no arquivo CSS todas as páginas HTML e JSPs que o utilizam terão as suas aparências também alteradas.

3.1.3.3 JavaScript

Rotinas Java Scripts foram utilizadas para formatação e validação de dados em formulários HTML. JavaScript representa a principal tecnologia para realizar processamento no lado cliente (programa de navegação) em aplicações web, permitindo assim cuidar de forma mais apurada da apresentação e entrada de dados. Através de rotinas JavaScript é possível formatar e utilizar máscaras para campos, assim como realizar uma prévia validação, ainda no lado cliente, do conteúdo dos campos. As rotinas JavaScritps são implementadas em arquivos com extensão "`js`" e permitem que páginas HTML e JSPs utilizem-se dessas rotinas associando-as a eventos dos componentes HTML, tais como *clicks* em botões ou o pressionamento de teclas em campos de edição.

Para uma maior produtividade e menor complexidade foi desenvolvida uma estratégia e API para definir as propriedades de formatação e validação dos campos de formulários HTML. Essa estratégia consiste na definição prévia, através de um vetor, das propriedades de todos os componentes de um formulário, onde é possível definir máscara de edição, obrigatoriedade de conteúdo, tipo do conteúdo (Data, CPF, CNPJ, Números, Email, etc) e associar essas definições a um formulário HTML. Assim no momento da solicitação de processamento do formulário (pressionamento do botão de submissão), todos os campos serão validados e uma janela irá exibir todas as mensagens de erro de acordo com as regras definidas.

3.1.3.4 JSP

Páginas JSP foram utilizadas para gerar dinamicamente a interface com o usuário e implementar transações relacionadas com esta interface. O código Java incluído nos JSPs foram os responsáveis pela interação com os componentes da camada intermediária (Session e Entity Beans). A sessão HTTP oferecida pelos JSPs foi utilizada para, após o *login*, armazenar uma referência à interface remota da instituição ou do candidato, permitindo que todos os outros JSPs pudessem obter as informações sobre a instituição ou candidato que realizou o *login*. Isso facilitou em muito a implementação da camada de interface, pois as informações relacionadas ao Entity Bean que está na sessão são facilmente obtidas.

3.1.3.5 Servlet

Foi necessário a implementação de um Servlet com o objetivo de interagir com uma API específica (JBarcode – www.jbarcode.com) para a geração do código de barras do boleto bancário.

3.1.4 Camada de Dados

A camada de dados foi implementada visando a utilização de um SGBD relacional, onde o banco de dados foi criado contendo apenas regras para validação de chaves primárias e chaves estrangeiras, com o intuito de manter a integridade no relacionamento entre as tabelas. Não foram implementadas funções, procedimentos ou *triggers* no banco de dados. Na arquitetura multicamadas é comum deixar que a camada de dados cuide apenas dos dados, evitando a perda do poder de escala pelo fato de concentrar processamento no SGBD. Um dos principais benefícios da arquitetura multicamada é permitir que a camada intermediária concentre o processamento permitindo que mais nós de processamento sejam adicionados de

acordo com a necessidade, permitindo assim uma grande facilidade na escalabilidade horizontal.

3.1.5 Camada Intermediária

Nesta camada foram utilizados os componentes EJB, Session e Entity Beans, os quais foram projetados para servirem como objetos de negócios. Os Entity Beans modelam o acesso aos dados e os Session Beans modelam as transações.

3.1.5.1 Camada de Acesso aos Dados

A estratégia utilizada na camada de acesso aos dados foi a implementação de Entity Beans CMP para todas as tabelas do Banco de Dados, fornecendo assim um modelo abstrato de todos os dados e permitindo que todas as operações (inserção, pesquisa, atualizações e remoções) com as tabelas fossem realizadas através dos Entity Beans. Os relacionamentos dos Entity Beans foram todos implementados com navegabilidade dupla, ou seja, as duas extremidades dos relacionamentos são implementadas. Isso permite que o modelo orientado a objetos fornecidos pelos Entity Beans forneça a navegabilidade em ambos os sentidos dos relacionamentos. O modelo relacional permite essa navegabilidade nos dois sentidos naturalmente através das chaves estrangeiras.

Para os relacionamentos 1:N, nos Entity Beans de cardinalidade 1 foram implementados métodos para permitir a criação, pesquisa e remoção dos Entity Beans relacionados. Por exemplo, no Entity Bean Instituição foram implementados os métodos: `incluirCurso()`, `pesquisarCurso()` e `excluirCurso()`.

Em todos os Entity Beans foram implementados o método `toXML()`, o qual retorna uma representação XML da instância do Entity Bean.

3.1.5.2 Camada de Transações

As regras de negócios, exigidas pela aplicação desenvolvida, são simples, o que facilitou a implementação da camada de transações. Apenas um Session Bean foi desenvolvido contendo três métodos (`getChave`, `IncluirInstituicao` e `IncluirCandidato`), o primeiro para gerar automaticamente a chave primária de cada tabela e os outros dois para incluir Instituições e Candidatos.

O método para geração automática da chave utiliza um Entity Bean chamado Chave. Este armazena o nome de cada tabela e o valor da próxima chave primária a ser utilizada. Recebe como parâmetro o nome da tabela para a qual deseja-se obter a próxima chave

primária. Cada vez que o mesmo é chamado ele retorna o valor atual e incrementa o valor da próxima chave primária da instância do Entity Bean Chave correspondente ao nome da tabela fornecido como parâmetro. O método possui uma região crítica, a qual fica isolada de acesso concorrente e dentro de um contexto transacional oferecido pelo container.

3.2 Considerações sobre o desenvolvimento da Aplicação

A implementação da aplicação foi de fundamental importância para o aprendizado e avaliação da produtividade e complexidade no desenvolvimento de aplicações WEB sobre a Plataforma J2EE. Diante mão é possível afirmar que a produtividade alcançada está abaixo da desejada, assim como o nível de complexidade está acima do desejado. Estamos vivendo a era da economia da informação e a demanda por aplicações WEB e coorporativa está crescendo continuamente, exigindo que a produtividade no desenvolvimento de aplicações cresça em igual proporção.

A análise do esforço no desenvolvimento da aplicação desenvolvida será feita de acordo com a camada: Interface, Intermediária e de Dados.

3.2.1 Camada de Dados

A camada de dados foi a mais simples de ser implementada, até mesmo por representar um processo bastante conhecido e maduro no desenvolvimento de software. Após a definição do projeto do banco de dados a sua implementação em qualquer SGBD é uma tarefa simples, ainda mais de acordo com a estratégia escolhida para a camada de dados, onde o banco de dados foi utilizado apenas como repositório de dados, não existindo a implementação de rotinas no banco de dados. As ferramentas para gerenciamento da camada de dados já alcançaram simplicidade e produtividades satisfatórias aos requisitos exigidos atualmente no desenvolvimento de software.

3.2.2 Camada Intermediária

Nessa camada foi observado o maior esforço e complexidade no desenvolvimento da aplicação, principalmente nos elementos da camada de acesso a dados (Entity Beans). Mesmo estes sendo do tipo CMP, os quais são mais simples de serem implementados por terem a persistência dos dados gerenciada pelo container.

A aplicação possuía apenas **7** tabelas e exigiu a implementação de **25** classes e interfaces (Interface Home, Interface Remota, Classe Abstrata e Classe Chave Primária), totalizando **1772** linhas de código. Assim, a tarefa de implementação dos Entity Beans foi bastante custosa e sujeita a erros, principalmente pelas rígidas regras determinadas pela especificação EJB a serem seguidas rigorosamente.

Ainda foi necessária a implementação dos *deployment descriptors*, onde foram criados três arquivos XML totalizando **1186** linhas. Na criação dos *deployment descriptors* para os

Entity Beans a definição dos relacionamentos (cardinalidade e navegabilidade) foi a tarefa que demandou maior atenção e esforço.

A implementação da camada de transações (Session Bean) foi mais simples que a implementação dos Entity Beans, por não possuir o mapeamento para a camada de dados, permitindo que o esforço fosse direcionando para a implementação dos seus métodos, os quais representam as transações oferecidas à camada cliente.

Foi implementado apenas um Session Bean, possuindo três métodos, a implementação do Session Bean, classes e interfaces, exigiu **280** linhas de código.

Outro detalhe importante é que qualquer alteração nas classes, interfaces ou *deployment descriptors* do Bean, exigia a execução da operação de *deployment*, a qual durava cerca de 5 minutos, na plataforma utilizada.

3.2.3 Camada de Interface

A implementação da camada de Interface apresentou também uma baixa produtividade e alta complexidade, exigindo muita diversidade de conhecimento. Foi necessária a utilização de várias tecnologias: HTML, CSS, JavaScript, Servlet e JSP. Um outro inconveniente encontrado na camada de interface foi a geração de um grande número de componentes, onde foi necessário escrever **13** páginas HTML contendo **229** linhas de código, **29** páginas JSP contendo **2639** linhas de código e **3** arquivos JavaScript, perfazendo um total de **398** linhas de código.

Apesar de possuir quase três vezes mais linhas de código que a camada intermediária, a camada de interface foi mais simples e produtiva de ser implementada. As alterações não necessitavam de operações extras para publicação, sendo assim mais produtiva a implementação de seus componentes.

Em algumas situações os JSPs fizeram utilização direta de recursos da camada de dados, onde a API JDBC foi utilizada para a execução de comandos SQL junto ao SGBD, pelo fato de EJBQL apresentar algumas limitações, tais como: não suportar o uso de `order by`, `group`, e `select` na cláusula `from`.

A maior quantidade de linhas de código desta camada correspondeu à criação de formulários HTML, na validação e formatação dos seus campos e na obtenção das informações dos formulários nos arquivos JSP.

3.3 Conclusões Extraídas do Desenvolvimento da Aplicação

Ficou evidente que a produtividade alcançada no desenvolvimento de aplicações na arquitetura com duas camadas (Cliente/Servidor) é superior ao desenvolvimento de aplicações na arquitetura com multicamadas. Uma aplicação equivalente à desenvolvida, caso fosse desenvolvida numa arquitetura Cliente/Servidor teria uma quantidade de linhas de código bem inferior, já que não haveria todo o esforço para criação dos objetos distribuídos da camada de negócios, assim como a complexidade de administração do servidor de aplicação J2EE.

Foi despendido maior esforço implementando elementos de infra-estrutura (Session e Entity Beans) que transações relativas ao negócio (Publicação e Inscrição em Vestibular). Mesmo com a plataforma J2EE fornecendo toda uma camada de infra-estrutura de nível mais baixo, o que aumentou a transparência do ambiente remoto e dos objetos distribuídos, a produtividade e complexidade não atingiram os níveis exigidos pela demanda atual na produção de software.

O principal motivo que eleva a complexidade e consequentemente reduz a produtividade é a implementação da camada intermediária, sendo que está é fundamental para fornecer a escalabilidade exigida por aplicações corporativas. Dessa forma é de fundamental importância avaliar os requisitos de escalabilidade para as aplicações desenvolvidas, pois caso esse não seja um requisito determinante é preferível utilizar a arquitetura em duas camadas. Mas caso a escalabilidade seja essencial, a única maneira de alcançar um alto poder de escalabilidade é através da arquitetura em multicamadas.

Outros fatores secundários podem exigir a arquitetura multicamadas, tais como: interação com sistema legado e utilização de diversos tipos de software clientes. Dessa forma a opção pela arquitetura multicamadas não deve ser uma opção baseada em modismo e sim determinada pela avaliação dos requisitos da aplicação a ser desenvolvida. No caso da Plataforma Java 2EE foi possível, como veremos adiante, criar uma estratégia e ferramentas para reduzir a complexidade e aumentar a produtividade, mas ficou claro que o objetivo da Sun Microsystems é oferecer a base tecnológica para o desenvolvimento de aplicações coorporativas e deixar que terceiros trabalhem para fornecer estratégias e ferramentas mais produtivas e menos complexas.

De acordo com as observações no desenvolvimento da aplicação ficou evidente a necessidade de uma estratégia e a utilização de ferramentas para o desenvolvimento de aplicações Web sobre a plataforma J2EE. Quando o desenvolvimento é feito apenas com os

recursos e ferramentas oferecidas pela Sun Microsystems, o desenvolvimento poderá ser visto como uma tarefa complexa e pouco produtiva.

Outra observação importante no desenvolvimento de aplicações WEB multicamadas é a grande interdisciplinaridade de conhecimentos exigida ao desenvolvedor, pois o processo de desenvolvimento abrange tecnologias distintas exigindo grandes habilidades e conhecimentos por parte dos profissionais de informática. Nunca foi tão complexo desenvolver software e a promessa feita em décadas passadas, que os avanços das tecnologias iriam elevar o nível de interação com os computadores permitindo que a sua utilização na construção de soluções computacionais fosse facilitada a cada dia, não passou de uma promessa e tudo indica que estamos caminhando num caminho contrário, exigindo cientistas para a tarefa de criação de software.

Capítulo 4 - EasyNet

O EasyNet é uma de ferramenta de desenvolvimento com a finalidade de aumentar a produtividade e reduzir a complexidade no desenvolvimento de aplicações web sobre a plataforma Java (Standard e Enterprise).

Parece paradoxal procurar atingir esses objetivos sobre a plataforma J2EE, pois os objetivos da SUN quando projetou a Plataforma J2EE foram exatamente esses. A redução na complexidade e aumento da produtividade, fornecidos pela plataforma J2EE, diz respeito a aspectos relativos à infra-estrutura utilizada no desenvolvimento e execução de aplicações coorporativas, tais como: controle transacional, persistência, interoperabilidade, escalabilidade, distribuição, disponibilidade, etc. Já o EasyNet, busca oferecer níveis mais altos de facilidades na construção de aplicações WEB, através da geração automática de componentes das camadas de acesso a dados e interface.

O EasyNet é formado pelos seguintes Módulos:

- **Gerador de Entity Beans CMP** – Aplicação Java para gerar Entity Beans CMP a partir de metadados obtidos do Banco de Dados.
- **Gerenciador de Projetos** – Aplicação WEB para permitir a criação de projetos.
- **Processador de Transações** – Componente WEB responsável por executar as transações do EasyNet.
- **API** – Hierarquia de classes oferecendo integração entre os componentes de projeto do EasyNet e funcionalidade comuns a implementações de aplicações.

4.1 Arquitetura do EasyNet

O EasyNet segue o modelo de programação WEB baseado no protocolo http consistindo em requisições feita pela camada cliente e respostas devolvidas pela camada servidora (*request/response*).

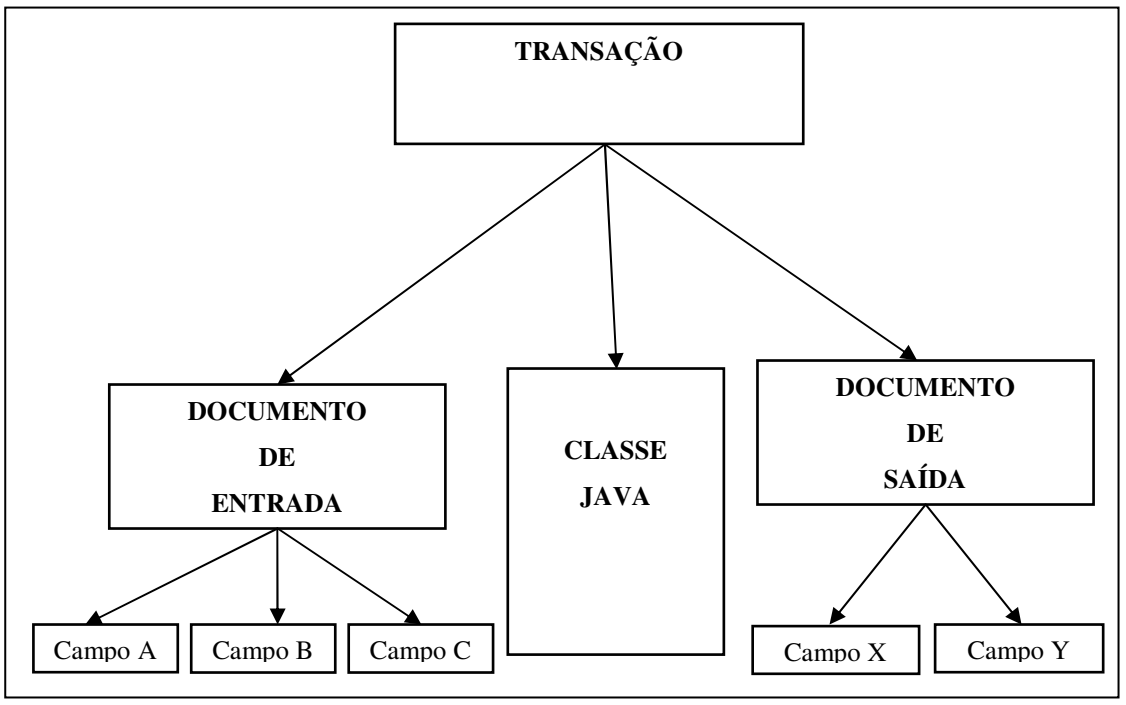
As aplicações no EasyNet são representadas por projetos contendo as definições dos componentes da aplicação. Esses componentes são definidos e manipulados pelo desenvolvedor através do módulo gerenciador de projetos e executados pelo processador de transações. O EasyNet utiliza a arquitetura de três camadas para o desenvolvimento de aplicações e um projeto no EasyNet pode ser constituído por componentes de cada uma das

camadas. Os elementos utilizados pelo EasyNet são: campos, documentos (camada de dados), transações (camada de transação) e querys (camada de acesso a dados).

Uma aplicação no EasyNet é formada por transações. As transações representam unidades de interação com o usuário, consistindo de um documento de entrada, responsável por obter os dados do usuário; uma classe Java de transação, responsável por executar a lógica do negócio e um documento de saída, responsável por exibir as informações processadas pela transação ao usuário. Os documentos por sua vez são compostos por campos representando as unidades de troca de informação com o usuário.

A figura abaixo mostra a relação entre Transação, documentos de entrada e saída, e a classe Java.

Figura 1. Transação, documento e classe Java.



O conceito de transação foi utilizado para encapsular os detalhes das camadas de interface e transação, permitindo que o desenvolvedor defina o conteúdo e propriedades dos elementos da camada de interface e implemente as transações através de classes Java.

A definição dos componentes da camada de interface permite que o Processador de Transações gere dinamicamente, em tempo de execução, o conteúdo HTML e JavaScript para interação com o usuário, evitando que o desenvolvedor precise criar ou manipular tais componentes. Esse processo fornecer maior simplicidade e produtividade na implementação

da camada de interface permitindo que a interação com o usuário seja feita a partir das definições dos documentos e campos.

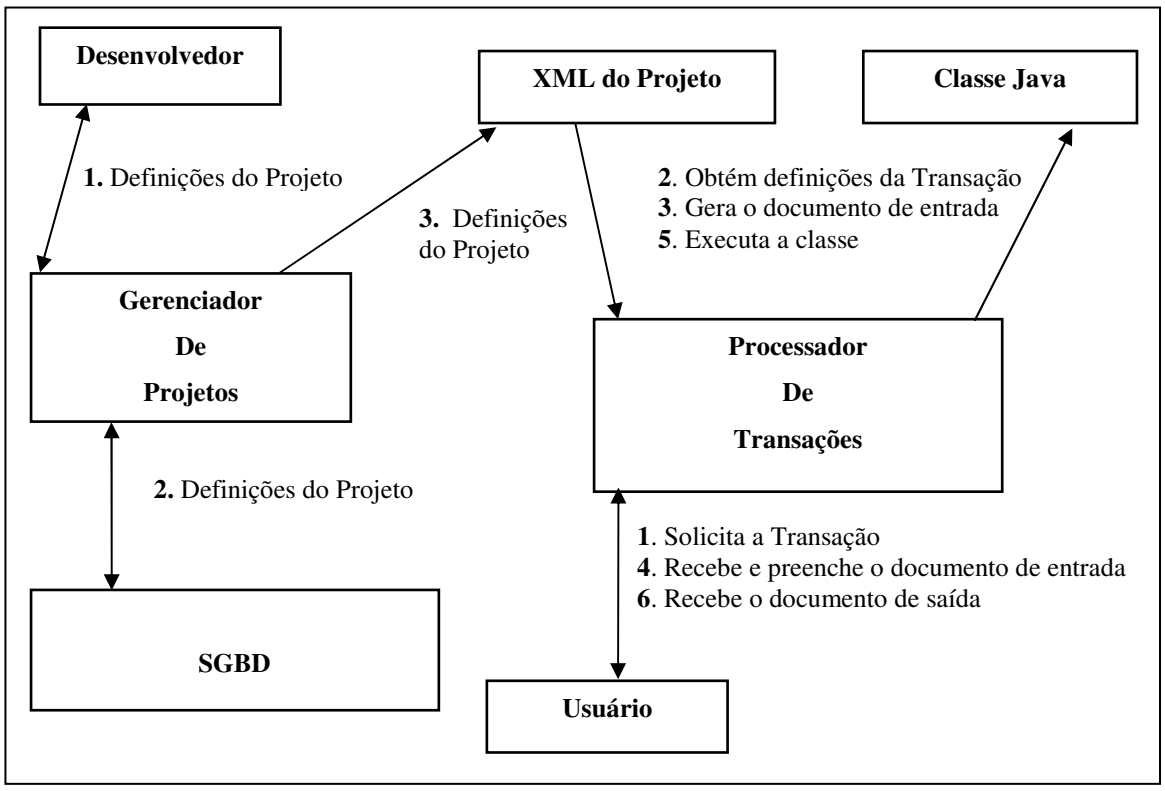
A utilização da classe Java para executar a lógica da transação permite que a camada de transação seja implementada com maior produtividade e facilidade evitando a necessidade de implementação de Session Beans para implementar a camada de transação. As classes Java de transações serão executadas pelo Processador de Transação através da interação com o servidor EJB fornecendo os mesmos recursos de um Session Bean e com a vantagem de não necessitar do mesmo esforço. O Processador de Transações utiliza Session Beans para executar as classes Java de transações.

Através do Gerenciador de Projetos o usuário constrói a aplicação definindo os seus componentes. O Gerenciador de Projetos armazena as definições no banco de dados do EasyNet e posteriormente permite a geração do arquivo XML contendo a definição de todas as transações definidas para a aplicação. Em tempo de desenvolvimento o usuário interage com o Gerenciador de Projetos criando e definindo propriedades dos componentes das camadas de interface (Campos e Documentos) e transação (Transações). Para que a aplicação seja executada é necessário fornecer ao Processador de Transações o arquivo XML da aplicação gerado pelo Gerenciador de Projetos.

A execução da aplicação é realizada pelo Processador de Transações. As unidades de processamento para o EasyNet são as transações. O usuário solicita a execução de uma transação fornecendo ao Processador de Transações o nome da aplicação e da transação. O Processador de Transações recupera as definições da transação e de seus componentes do arquivo XML da aplicação e processa a transação, gerando o documento de entrada, executando a classe Java e gerando o documento de saída.

A figura abaixo fornece uma visão mais clara da arquitetura do EasyNet, demonstrando a relação entre o desenvolvedor e o Gerenciador de Projetos; o usuário e o Processador de Transações e entre o Gerenciador de Projetos e o Processador de Transações.

Figura 2. Arquitetura do EasyNet.



4.2 Gerador de Entity Beans

O processo de criação dos Entity Beans tem como ponto de partida os elementos da camada de dados e para cada Entity Bean sempre os mesmos passos são executados: escolha da tabela, implementação da interface Remota com os métodos `set` e `get` para todos os campos; implementação da interface Home com o método `create` contendo todos os campos obrigatórios e os métodos finders; implementação da classe abstrata contendo os métodos `set` e `get` para os atributos e relacionamentos; implementação da classe representando a chave primária. Ou seja, cada tabela define como os Entity Beans CMP deveriam ser implementados.

A partir das observações realizadas na implementação da aplicação estudo de caso, ficou evidente a possibilidade de gerar automaticamente os Entity Beans, assim como os seus *deployment descriptors*, através de uma aplicação que descobrisse os metadados do bando de dados e aplicasse as especificações EJB. Um outro aspecto que motiva a automatização na geração dos Entity Beans é a constatação que a sua implementação requer um grande esforço de codificação e é bastante suscetível a erros.

Uma vez que a camada de dados tenha sido definida em um SGBD relacional é possível obter essas definições através da linguagem Java usando a API JDBC. Uma vez com os

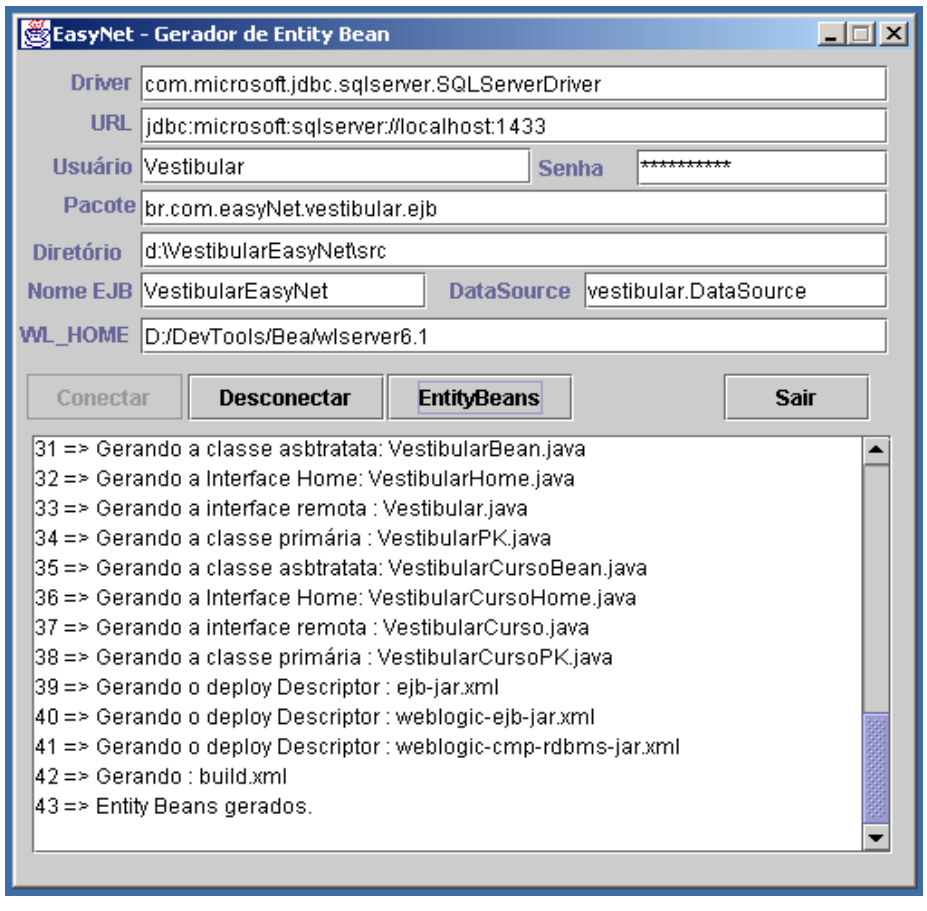
metadados sobre a definição da camada de dados é possível prever como os Entity Beans, para representar essa camada de dados, devem ser implementados.

Foi desenvolvida uma aplicação Java para conectar-se a um SGBD relacional e gerar Entity Beans CMP e os seus *Deployment descriptors* para todas as tabelas do banco de dados conectado. Também é gerado o arquivo `build.xml` que pode ser submetido à API `ant` para gerar o Container para o servidor EJB.

Para que os Entity Beans possam ser gerados automaticamente sem nenhuma intervenção humana, algumas regras deverão ser pré-definidas, tais como: O Entity Bean será do tipo CMP, todos os campos da tabela serão atributos dos Entity Bean, o nome JNDI do Entity Bean, etc.

A figura abaixo mostra a interface da aplicação Geradora de Entity Beans.

Figura 3. Interface da aplicação



A aplicação foi desenvolvida em Java usando a API Swing para criação da interface com o usuário e a API JDBC para a conexão e obtenção dos metadados do SGBD. A interface da aplicação é apresentada na Figura 4.4. A tabela abaixo descreve os elementos da interface da aplicação, contendo os parâmetros necessários para a geração dos EntityBeans :

Tabela 3. Elementos da interface

Campo	Descrição
Driver	Nome do driver JDBC a ser utilizado para conexão como SGBD
URL	URL do banco de dados a ser conectado
Usuário	Usuário do SGBD
Senha	Senha
Pacote	Pacote para ser utilizado na definição `package` das interfaces e classes do Entity Beans, sendo que, para cada tabela será criado um subpacote com o seu nome.
Diretório	Diretório, onde o código fonte e os *Deployments descriptores* serão gravados.
Nome EJB	Nome do arquivo EJB a ser gerado. Será gerado um arquivo EJBServer e outro EJBClient.
DataSource	Nome do DataSource através do qual os Entity Beans irão conectar-se ao banco de Dados.
WL_HOME	Diretório Home para o WebLogic, será utilizado no arquivo XML utilizado pela API ant na operação de deployment.

Características dos Entity Beans gerados

A aplicação irá conectar-se ao Banco de Dados e gerar automaticamente Entity Beans seguindo as seguintes características:

- Todos serão Entity Beans Container Manager Persistence (CMP);
- Para cada tabela do banco de dados será criado um Entity Bean;
- Os Entity Beans irão possuir atributos para todos os campos da tabela;
- Para cada Entity Bean serão criados: suas interfaces Home e Remota, classe Abstrata e classe Chave Primária;
- O pacote de cada Entity Bean será o pacote definido na aplicação acrescido do nome da tabela que o Entity Bean representa. Exemplo: Caso o pacote definido na aplicação seja "`br.com.easyNet.vestibular`" o pacote (cláusula `package`) nas interfaces e classes do Entity Bean para uma tabela chamada **Candidato** será "`br.com.easyNet.vestibular.candidato`";
- Os nomes das interfaces e classes do Entity Bean seguirão o padrão proposto pela própria SUN. Exemplo: Para um Entity Bean representando uma tabela chamada **Candidato**: o nome da **Interface Remota** será **`Candidado`**; para a **Interface Home** será **`CandidatoHome`**; para a **Classe Chave Primária** será **`CandidatoPK`** e para a **Classe Abstrata** será **`CandidatoBean`**;
- Os atributos do Entity Bean terão tipos compatíveis com os tipos dos campos no banco de dados, onde a compatibilidade será feita através da tabela de conversão

entre tipos JDBC e tipos da linguagem Java. Exemplo: Campos do banco de dados com tipo `char` ou `varchar` serão armazenados em atributos no Entity Bean do tipo `java.lang.String`;

- Os atributos que formam chave primária ou que participam de relacionamentos e tiverem tipo primitivo terão seus tipos ajustados para `java.lang.String`, pois segundo a especificação EJB 2.0, campos de tipos primitivos não podem participar de relacionamentos CMP.

Classe Abstrata

As classes abstratas geradas terão as seguintes características:

- Cláusula `import` para cada Entity Bean relacionado;
- Assinatura dos métodos abstratos `set` e `get` para cada atributo do Entity Bean;
- Assinatura dos métodos abstratos `set` e `get` para cada relacionamento do Entity Bean;
- Métodos de negócio para facilitar a manipulação dos relacionamentos. Para cada relacionamento `N` existirão três métodos (`create`, `getFromRelationShip` e `remove`) e para cada relacionamento `1` existirá um método `getFromRelationShip`. Exemplo: A tabela `Instituicao` possui um relacionamento `1..N` com a tabela `Curso`, ou seja uma Instituição pode possuir vários cursos, mas um curso possuirá apenas uma instituição, assim no Entity Bean `Instiuicao` existirão os métodos `create`**`Curso`**, `getFromRelationShip`**`Curso`** e `remove`**`Curso`**; e no Entity Bean `Curso` existirá o método `getFromRelationShip`**`Instituicao`**.
- Os métodos para os relacionamentos N terão as seguintes características: O método `create` terá como parâmetro todos os campos requeridos do Entity Bean relacionado e retornará a interface remota do mesmo; O método `getFromRelationShip` não possuirá parâmetros e irá retornar uma coleção com a interface remota de todos os Entity Beans relacionados, onde essa coleção será criada a partir da coleção CMP que implementa o relacionamento; O método `remove` terá como parâmetros os campos que formam a chave primária do bean relacionado.
- O método para o relacionamento 1 irá retornar a interface remota do Entity Bean relacionado.

- Método `toXml` para retornar uma String com a representação XML para o Enity Bean, onde o rótulo delimitador irá ter o nome do Entity Bean e o valor de cada atributo será colocado em rótulo com seu nome.
- Os métodos `ejbCreate` e `ejbPostCreate` terão como parâmetros todos os campos requeridos do Enity Bean.

A classe abstrata do Entity Bean Curso está no Anexo II.

Interface Remota

As interfaces remotas geradas terão as seguintes características:

- Cláusula `import` para cada Entity Bean relacionado;
- Assinatura dos métodos abstratos `set` e `get` para cada atributo do Entity Bean;
- Assinatura dos métodos de negócio, para manipulação dos relacionamentos, implementados na classe abstrata do Entity Bean;
- Assinatura do método `toXML`;

A interface remota do Entity Bean Curso está no Anexo II.

Interface Home

As interfaces home geradas terão as seguintes características:

- Método `create` com todos os atributos requeridos para o Entity Bean;
- Método `findByPrimaryKey`;
- Método `findAll`;
- Método `findBy` para cada atributo único do Entity Bean.

A interface home do Entity Bean Instituicao está no Anexo II.

Classe Chave Primária

As classes para as chaves primárias terão as seguintes características:

- Atributos para cada campo chave da tabela representada pelo Entity Bean;
- Construtor com todos os atributos que representam a chave primária;
- Método `equals`, o qual irá retornar `true` somente quando todos os atributos correspondentes sejam iguais. A comparação dos respectivos atributos será feita através dos seus próprios métodos `equals`, sendo que, caso o atributo seja de um tipo primitivo, a comparação será realizada através do operador "==";

- Método `hashCode`, o qual irá retornar a soma de todos os métodos `hashCode` dos atributos;

A interface home do Entity Bean Opcoes está no Anexo II.

Deployment Descriptors

A aplicação gera os *deployment descriptors* no subdiretório `meta-inf`, criado a partir do diretório escolhido no campo **Diretório** da aplicação. Os três arquivos XML necessários à operação de *deployment* (`ejb-jar.xml`, `weblogic-ejb-jar.xml`, `weblogic-cmp-rdbms-jar.xml`), onde os dois últimos são específicos para o Weblogic 6.0;

O arquivo ejb-jar.xml

O atributo `<display-name>` do arquivo XML será preenchido com o valor do campo "Nome EJB" da aplicação, acrescido de "`EJB`".

Para cada Entity Bean será criada um rótulo `<entity>` contendo as informações da tabela abaixo.

Tabela 4. Tag <entity> do DD ejb-jar.xml

Informação	Rótulo	Composição
Nome EJB	<ejb-name>	Nome da tabela que o Enity Bean representa, acrescido de "EJB".
Interface Remota	<remote>	Nome da tabela
Interface Home	<home>	Nome da tabela, acrescido de "Home"
Classe abstrata	<ejb-class>	Nome da tabela, acrescido de "EJB"
Tipo de persistência	<persistence-type>	Sempre "Container"
Classe chave primária	<prim-key-class>	Nome da tabela, acrescido de "PK"
Versão cmp	<cmp-version>	Sempre "2.0"
Nome abstrato	<abstract-schema-name>	Nome da tabela, acrescido de "EJB"
Campos	<cmp-field>	Nome do respectivo atributo na tabela, sendo que, com a primeira letra minúscula.
Consultas EJBQL	<query>	Para os métodos: `findByPrimaryKey`, `findAll` e os métodos `find[NomeDoCampo]` para todos os campos únicos.

Todos os relacionamentos do Banco de Dados também serão representados no arquivo XML. Para cada tabela serão obtidos metadados do banco de dados para os seus atributos chaves estrangeiras. Para cada atributo do tipo chave exportada será gerado um relacionamento de cardinalidade N e para cada atributo do tipo chave importada será gerado um relacionamento de cardinalidade 1.

Para cada relacionamento será criado um rótulo `<ejb-relation>` contendo as informações da tabela abaixo:

Tabela 5. Tag <ejb-relation> do DD ejb-jar.xml

Informação	Rótulo	Composição
Nome	<ejb-relation-name>	Nome da tabela que exportou a chave concatenada com o nome da tabela que importou a chave
Regra	<ejb-relationship-role>	Será criada uma para o relacionamento 1 e outra para o relacionamento N, ou seja, o relacionamento sempre será bi-direcional.
Nome da regra	<ejb-relationship-role-name>	Nome da tabela
Mutiplicidade	<multiplicity>	Depende da regra
Fonte	<relationship-role-source>	Nome do Entity Bean
Campo CMP	<cmr-field-name>	Nome do campo cmp para o relacionamento
Tipo do Campo	<cmr-field-type>	java.util.set caso seja uma regra de cardinalidade N, e o tipo da interface remota caso seja uma cardinalidade 1.

Um exemplo de arquivo ejb-jar.xml gerado pelo EasyNet está no Anexo II.

O arquivo weblogic-ejb-jar.xml

Esse arquivo será criado específico para o servidor EJB, servindo para definir as propriedades de publicação dos componentes EJB. Para cada Entity Bean, as informações serão definidas de acordo com a tabela abaixo:

Tabela 6. Deployment Descriptors weblogic-ejb-jar.xml

Informação	Rótulo	Composição
Nome EJB	<ejb-name>	Nome do Entity Bean
Tamanho do Cache	<max-beans-in-cache>	Valor padrão
Time out	<read-timeout-seconds>	Valor padrão
Persistência	<persistence-type>	Informações específicas para o Weblogic 6.0
Nome JNDI	<jndi-name>	Conteúdo do campo "NomeEJB" da aplicação, concatenado com "`.NomeDoEntityBean`"

Um exemplo de arquivo weblogic-ejb-jar.xml gerado pelo EasyNet está no Anexo II.

O arquivo weblogic-cmp-rdbms-jar.xml

Esse arquivo faz o mapeamento do modelo abstrato (Entity Beans) para o modelo relacional (Tabelas), também sendo específico para o servidor EJB.

Para cada Entity Bean existirá uma tag `<weblogic-rdbms-bean>` contendo as seguintes informações:

Tabela 7. Tag <weblogic-rdbms-bean>

Informação	Rótulo	Composição
Nome EJB	<ejb-name>	Nome da tabela acrescido de "EJB"
Data Source	<data-source-name>	Conteúdo do campo "Data Source" da aplicação
Tabela	<table-name>	Nome da tabela
Mapeamento entre atributos e campos	<field-map>	O atributo do Entity bean terá o mesmo nome do campo na tabela

Para cada relacionamento existirá um rótulo `<weblogic-rdbms-relation>` contendo as seguintes informações:

Tabela 8. Tag <weblogic-rdbms-relation>

Informação	Rótulo	Composição
Nome do relacionamento	<relation-name>	Nome da tabela que exportou a chave, concatenado com o nome da tabela que importou a chave
Nome da regra	<relationship-role-name>	Nome da tabela que importou a chave
Mapeamento entre chave primária e chave estrangeira	<column-map>	Nomes dos campos

Um exemplo de arquivo weblogic-cmp-rdbms-jar.xml gerado pelo EasyNet está no Anexo II.

O arquivo build.xml

Para realização do processo de *deployment*, é necessário ter as classes e interfaces dos Entity Beans e os *Deployment descriptors*. O único *Deployment descriptor* contido na especificação EJB é o arquivo `ejb-jar.xml`. Assim, a depender do servidor EJB, a quantidade e conteúdo dos arquivos irá variar. Para facilitar a geração do container contendo os Entity Beans gerados pela aplicação, a mesma gera também o arquivo `build.xml`, o qual servirá como entrada para a API ant e irá realizar as seguintes operações:

- Criação dos diretórios `build` e `dist`. Onde o primeiro será utilizado com diretório de trabalho e segundo irá armazenar o resultado da operação de *deployment*: um arquivo jar cliente com o conteúdo para a camada cliente e outro com o conteúdo para ser publicado no servidor EJB;
- Compilação das classes e interfaces geradas pela aplicação e
- Geração e compilação das classes de serviços necessárias à implementação da camada de infra-estrutura, aquela que fornecer os recursos da arquitetura EJB, provendo transparência ao ambiente remoto, controle transacional, etc.

O conteúdo do arquivo `build.xml` será gerado de acordo com a tabela abaixo:

Tabela 9. Conteúdo do arquivo build.xml

Informação	Rótulo	Composição
Diretório do Weblogic	WL_HOME	Conteúdo do campo WL_HOME da aplicação
Arquivos fontes	srcdir="${javasources}"	Cada classe e interface gerada.
Arquivo jar para o cliente	jarfile	Nome EJB acrescido de "`EJBClient.jar`"
Arquivo jar o servidor	Jarfile	Nome EJB acrescido de "`EJBServer.jar`"

4.3 Gerenciador de Projetos

O gerenciador de projetos consiste de uma aplicação Web, através da qual o desenvolvedor irá criar seus projetos. Cada projeto representa uma aplicação e será composto pelos vários componentes que o EasyNet oferece. O EasyNet fornece componentes para implementação das camadas de interface, transações e acesso a dados. O gerenciador de projetos foi implementado com os próprios componentes do EasyNet.

Para editar um projeto o desenvolvedor precisa efetuar um *login* e selecionar o projeto. O gerenciador de projetos possui um módulo administrativo para realizar o cadastro de contas e projetos. Esse módulo administrativo permite que as contas criadas sejam associadas aos projetos definindo os usuários que terão acesso a cada projeto.

Na edição do projeto o desenvolvedor cria e associa componentes e define suas propriedades e atributos. Os componentes editados através do gerenciador de projetos são campo, documento, transação e query.

4.3.1 Campo

O componente campo representa a unidade de informação na interação com o usuário. Através dos campos o usuário irá fornecer e receber informações (*nome, cpf, cep, telefone*). Cada campo define os atributos dessas unidades de informação definindo o seu conteúdo, aparência e comportamento. A tabela abaixo lista os atributos dos campos.

Tabela 10. Atributos dos Campos

Atributo	Descrição
Nome	Nome do campo.
Label	Rótulo para o campo.
Visão	Indica como o campo será exibido (Edit, Link, Combo, Combo Sql, Label, Memo, Imagem, Senha, Check Box, Radio, Box e Escondido).
Tipo	Define o tipo da informação do campo (Caracter, Número e Data).
Valor	Valor para o conteúdo do campo.
Ação	Utilizado no caso de campos do tipo "Link", para indicar a URL a ser chamada.
Tamanho	Quantidade de caracteres do campo, quando este for do tipo "Edit".
Máscara	Máscara de Edição do campo. Exemplo: 999.999.999-99, para campo CPF
Obrigatório	Indica se o conteúdo do campo é obrigatório.
Visível	Indica se o campo será exibido.
Editável	Indica se o campo será editável.
CSS	Indica qual a classe CSS do campo.
Descrição	Comentário para o campo, será exibido automaticamente.
Altura	Altura do campo, sendo utilizado apenas para os campos com visão do tipo Imagem e Memo.
Largura	Largura do campo, sendo utilizado apenas para os campos com visão do tipo Imagem e Memo.
Alinhamento Horizontal	Alinhamento Horizontal do campo (Esquerda, Direita e Centro).
Alinhamento Vertical	Alinhamento Vertical do campo (Meio, Topo e Abaixo).

4.3.2 Documento

O componente documento representa o agrupamento de campos permitindo que a interface com o usuário seja construída a partir de suas unidades. Os atributos do documento definem o comportamento e a aparência do seu conjunto de campos fornecendo maior flexibilidade na construção da interface. A tabela abaixo lista os atributos dos documentos:

Tabela 11. Atributos dos Documentos

Atributo	Descrição
Nome	Nome do Documento.
Label	Rótulo para o Documento.
Visão	Indica como o documento será exibido (Vertical, Horizontal, Colunas ou Tabela).
Borda	Indica se o documento possuirá bordas.
Ocorrência	Indica se o documento possuirá ocorrências, ou seja, repetição dos campos contidos no documento.
Visível	Indica se o documento será visível.
Editável	Indica se o documento será editável.
CSS	Classe CSS para o documento.

4.3.3 Transação

O componente Transação representa as operações disponibilizadas ao usuário. Uma transação consiste de um agrupamento formador por um documento de entrada, uma classe Java e um documento de saída. O documento de entrada irá obter os dados do usuário, a classe Java irá executar a lógica da transação e o documento de saída irá retornar as informações para o usuário. Uma transação pode não possui documento de entrada, classe

Java ou documento de saída. Por exemplo uma transação pode não precisar de dados de entrada e não irá possuir documento de entrada. A tabela abaixo lista os atributos das Transações:

Tabela 12. Atributos das Transações

Atributo	Descrição
Nome	Nome do Documento.
Label	Rótulo para o Documento.
Ação Cancelar	URL a ser chamado quando a transação for cancelada.
Classe Java	Classe Java a ser executada.
Ação	Caso a transação não deva executar uma classe Java e sim direcionar para uma URL.
JSP Entrada	Quando a transação não utilizar a geração dinâmica do documento de entrada esse atributo especifica qual JSP irá representa o documento de entrada.
JSP Saída	Quando a transação não utilizar a geração dinâmica do documento de saída esse atributo especifica qual JSP irá representa o documento de saída.
Redirecionamento	Indica a URL a ser chamada pela transação após a sua execução.
Mensagem	Mensagem a ser exibida pela transação, após sua execução.

4.3.4 Query

O componente Query representa um comando SQL e será utilizada pela API do EasyNet para interagir com o SGBD. É uma camada construída sobre a API JDBC e permite que comandos SQL sejam utilizados com maior facilidade na implementação das transações do EasyNet. A tabela abaixo lista os atributos das Querys:

Tabela 13. Atributos das Querys

Atributo	Descrição
Nome	Nome da Consulta.
Query	Comando SQL.

4.4 API

A API do EasyNet tem como principal objetivo permitir que a classe Java, associada à transação, possa obter objetos representando os componentes definidos através do gerenciador de projetos, tais como: documento de entrada e saída, campos e querys. Para isso toda classe Java a ser associada a uma transação precisa estender uma classe específica da API do EasyNet, a qual possui métodos para manipular os documentos e interagir com o processador de transações.

O processador de transações fornece uma sessão para as classes Java, com a mesma finalidade da sessão HTTP permitindo que objetos sejam compartilhados por todas as transações executadas por um determinado usuário.

A API fornece também classes de uso genérico para realizar operações comuns na implementação de aplicações, tais como: manipular datas, estruturas de dados, formatar e validar dados.

A tabela abaixo contém a descrição das principais classes da API do EasyNet.

Tabela 14. Principais classes da API do EasyNet.

Classes	Descrição
br.com.easyNet.bancoDados.**Registro**	Registro de consulta SQL
br.com.easyNet.bancoDados.**Registros**	Coleção de registros de uma consulta SQL
br.com.easyNet.bancoDados.**SQL**	Comando SQL
br.com.easyNet.documentos.**Campo**	Campo de documento
br.com.easyNet.documentos.**Documento**	Documento
br.com.easyNet.generico.**RoteadorClient**	Processador de transação
br.com.easyNet.generico.**Transacao**	Transação genérica. Deve ser a superclasse de toda transação
br.com.easyNet.sessao.**Sessao**	Sessão do usuário
br.com.easyNet.util.**Data**	Data
br.com.easyNet.util.**Decimal**	Número decimal
br.com.easyNet.util.**Formatar**	Classe para formatação de valores e tratamento de Strings
br.com.easyNet.util.**Hora**	Hora
br.com.easyNet.util.**SendMail**	Enviar Email

4.5 Processador de Transações

O processador de transações é um componente web responsável por executar as transações definidas através do gerenciador de projetos. Foi implementado através de JSP, Session Beans e da API do EasyNet e representa o ambiente de execução (*Engine*) para os projetos do EasyNet, fazendo o papel de contêiner dentro da arquitetura do EasyNet.

O processador de transações trabalha atendendo solicitações dos usuários para execução de transações. As definições das transações são obtidas através do arquivo XML gerado pelo gerenciador de projetos.

Os passos descritos abaixo detalham o funcionamento mais comum do gerenciador de transações diante das solicitações dos usuários:

1. O usuário solicita, através de uma URL, a execução de uma transação fornecendo o nome do projeto e da transação;
2. O gerenciador de transações busca no arquivo XML as definições da transação solicitada;
3. Verifica a existência da especificação dos documentos de entrada e saída e da classe Java;
4. Caso exista um documento de entrada definido para a transação. O processador de transações irá utilizar este documento para gerar uma página HTML, contendo um formulário seguindo as especificações do documento e dos campos, a ser exibida e preenchida pelo usuário;
5. Quando o usuário submeter o formulário HTML o gerenciador de transações irá ler o seu conteúdo e gerar um objeto representando o documento de entrada.

6. Caso exista uma classe Java definida para a transação. O processador de transações irá executar essa classe Java. A classe Java deverá estender uma classe base de transação da API do EasyNet e poderá obter referências para os objetos representando os documentos de entrada e saída;
7. Após a execução da classe Java, caso exista um documento de saída definido para a transação, o gerenciador de transações irá utilizar esse documento para gerar uma página HTML com o seu conteúdo a ser exibida ao usuário.

4.6 Implementando Aplicações com o EasyNet

O gerenciador de projetos é a interface do EasyNet com o desenvolvedor. Uma aplicação desenvolvida no EasyNet é composta pelos componentes criados através do gerenciador dos projetos e das classes Java implementando as transações definidas no gerenciador de projetos.

Os itens a seguir demonstram os recursos oferecidos pelo EasyNet para a construção de aplicações. Esses recursos são oferecidos pelos componentes criados através do gerenciador de projetos e da API do EasyNet.

4.6.1 Criando Usuário e o Projeto

Para que o desenvolvedor possa utilizar o EasyNet é necessário criar uma conta de acesso para o mesmo e associar a conta a um ou mais projetos. Caso seja um novo projeto o mesmo precisa ser criado.

As figuras abaixo exibem as telas utilizadas para criação do usuário, do projeto e da associação entre o projeto e o usuário.

Figura 4. Criação do usuário

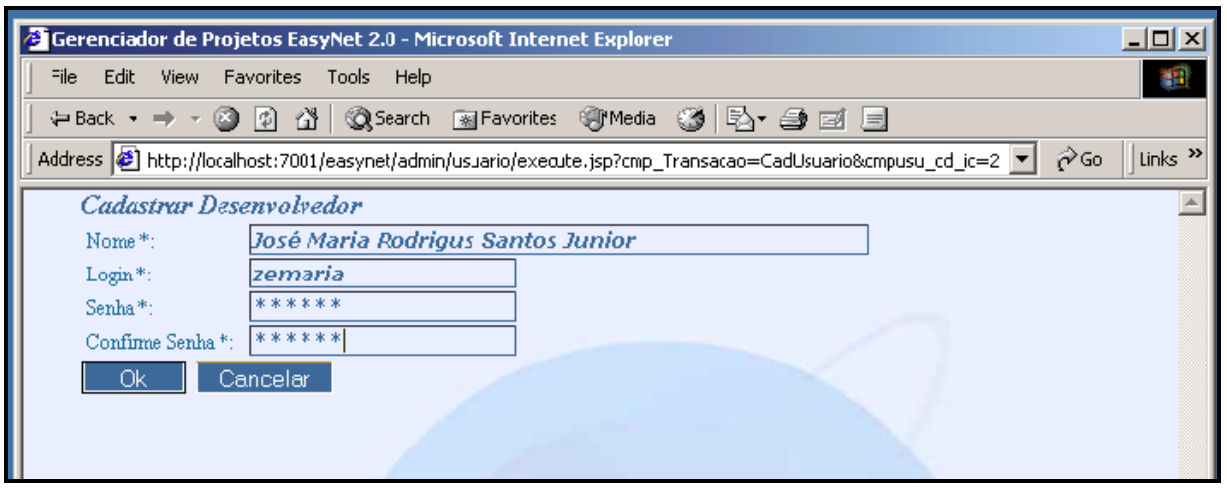


Figura 5. Criação do projeto

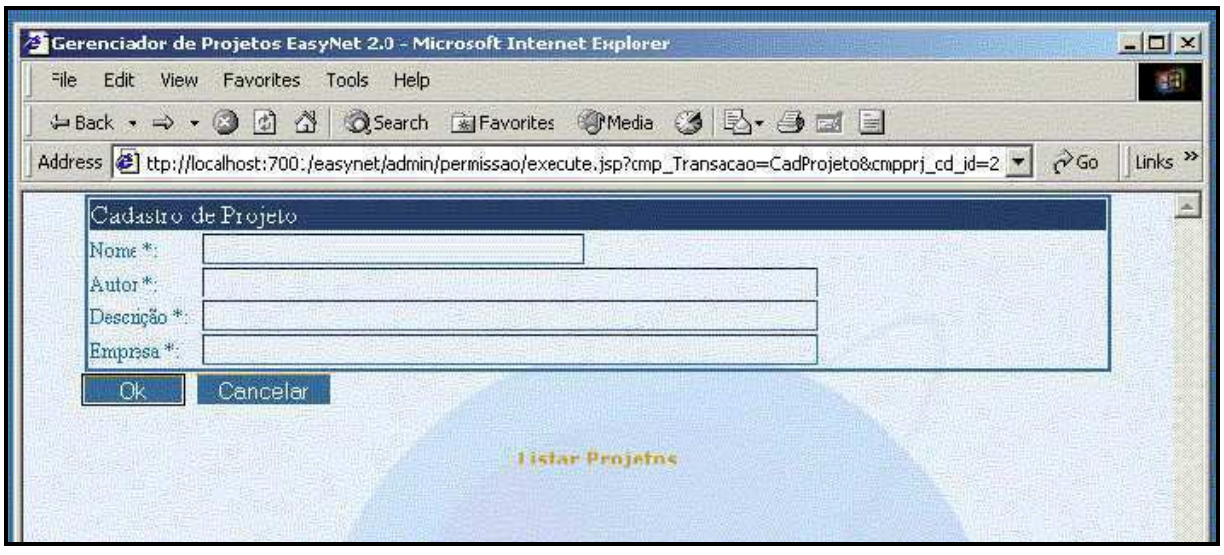


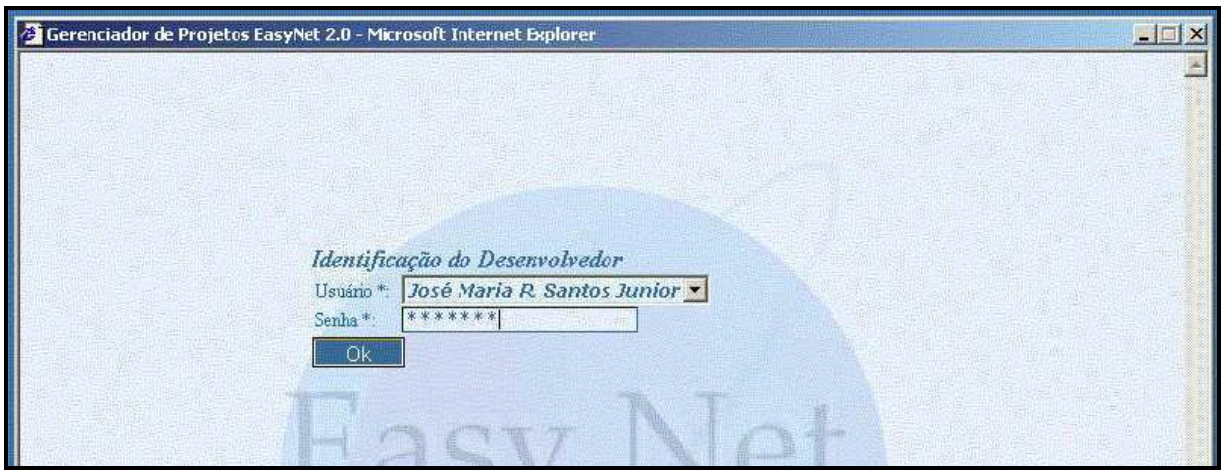
Figura 6. Associação entre o projeto e o usuário

4.6.2 Efetuando Login

Após a criação do usuário, do projeto e da associação entre os dois, o desenvolvedor pode acessar o gerenciador de projetos para editar o projeto. O acesso ao gerenciador de projetos é feito através de um *Login*.

A figura abaixo exibe a tela de *Login* do gerenciador de projetos.

Figura 7. Login do gerenciador de projetos

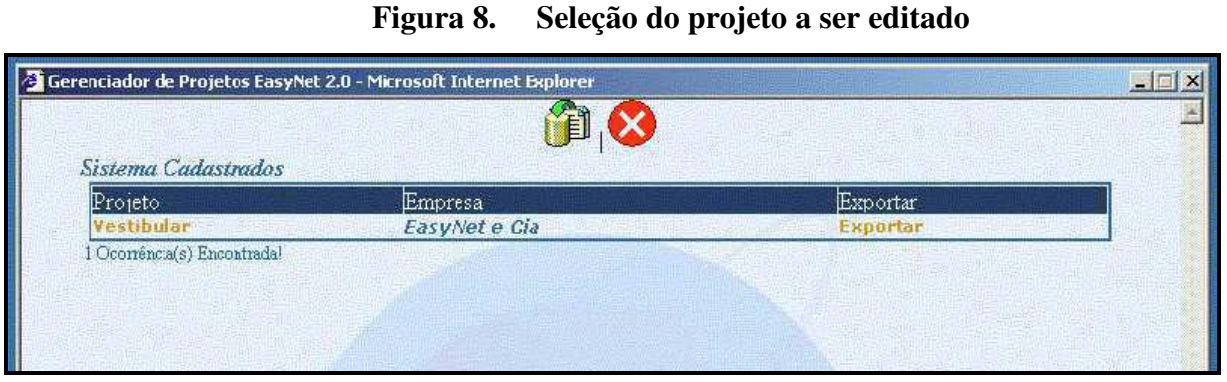


4.6.3 Selecionado o projeto a ser editado

Após o Login no gerenciador de projetos, o desenvolvedor irá escolher o projeto o qual deseja editar.

A figura abaixo exibe a tela para seleção do projeto a ser editado.

Figura 8. Seleção do projeto a ser editado

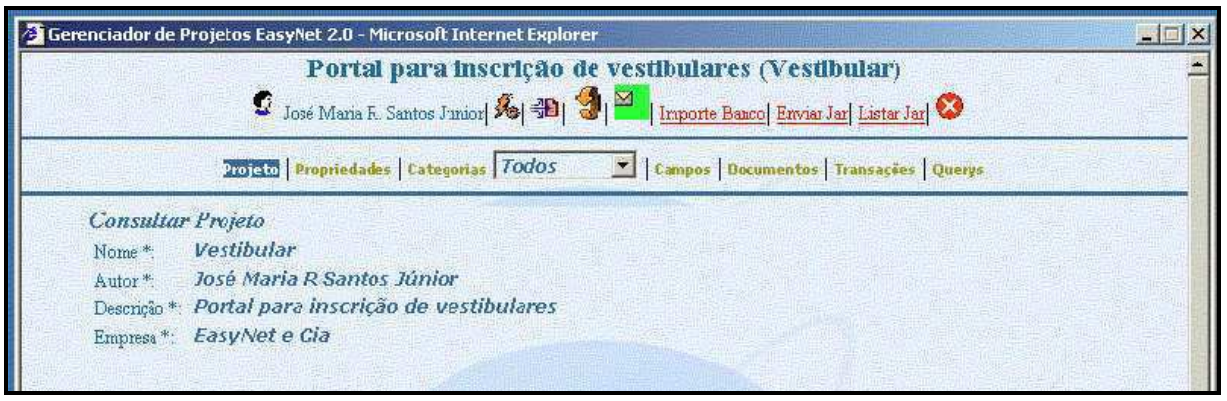


4.6.4 Editando o projeto

Após ter realizado o *Login* e selecionado o projeto para edição o desenvolvedor irá construir a aplicação criando componentes, definindo seus atributos e propriedades e associando os componentes.

A figura abaixo exibe a tela inicial de edição do projeto.

Figura 9. Edição do projeto



Na tela inicial o desenvolvedor tem a opção de editar os dados e as propriedades do projeto (Projeto, Propriedades). As propriedades representam valores a serem utilizador pelo processador de transações. A principal propriedade para um projeto define o nome do pool de conexões do servidor, através do qual o processador de transações irá executar os comandos SQL definidos nas querys.

A figura abaixo exibe a tela para definição das propriedades do projeto.

Figura 10. Definição das propriedades

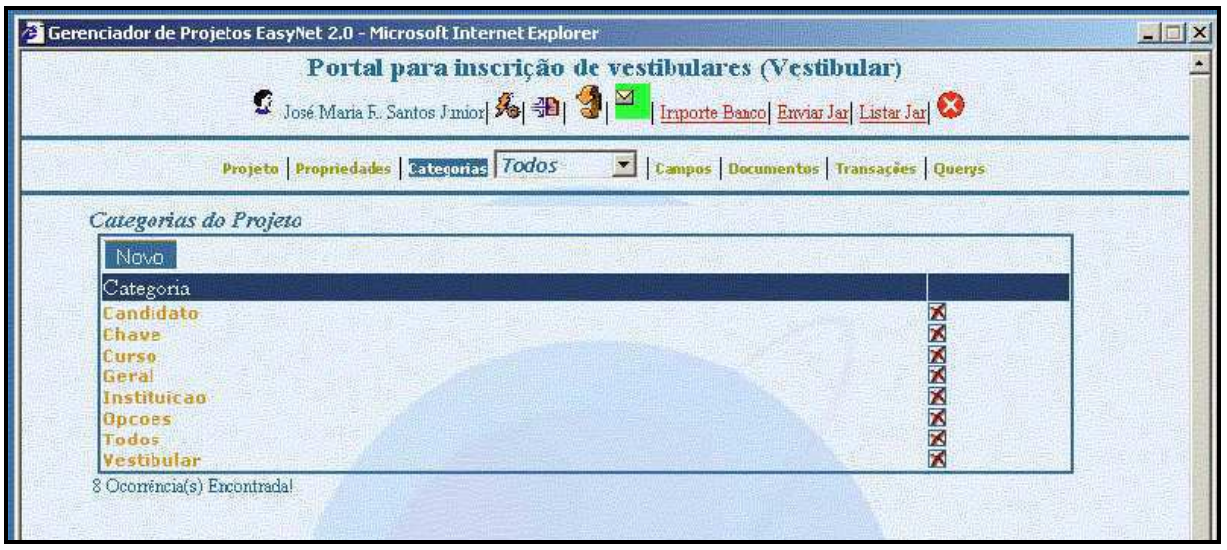
4.6.5 Editando categorias

Todos os componentes do gerenciador de projetos são organizados em categorias com a finalidade de facilitar a edição dos mesmos. As categorias permitem que os componentes sejam agrupados sob um mesmo título e a interface do gerenciador de projetos exibe os componentes sempre através das categorias. Todo projeto já começa com uma categoria denominada “Geral” e essa será utilizada para agrupar todos os componentes caso outras categorias não sejam criadas.

A exibição dos componentes é feita através da seleção de uma categoria na caixa de seleção “categorias”. A categoria “Todos” irá exibir todos os componentes independentemente de categoria. A opção “Categorias” na barra de menu do gerenciador de projetos permite a edição das categorias.

A figura abaixo exibe a tela para edição de categorias.

Figura 11. Edição de categorias



4.6.6 Editando Campos

A edição dos campos é feita através da opção “Campos” da barra de menu, onde todos os campos da categoria selecionada serão exibidos. O botão “Novo” permite criar um novo campo. Para editar um campo basta selecioná-lo e para remover o campo basta pressionar o botão “x” ao seu lado. A tela para edição dos campos permite editar os atributos, definir propriedades e verificar em quais documentos o campo está sendo utilizado.

As figuras baixo exibem as telas listar, editar e criar campos.

Figura 12. Lista de campos

Figura 13. Edição de campo

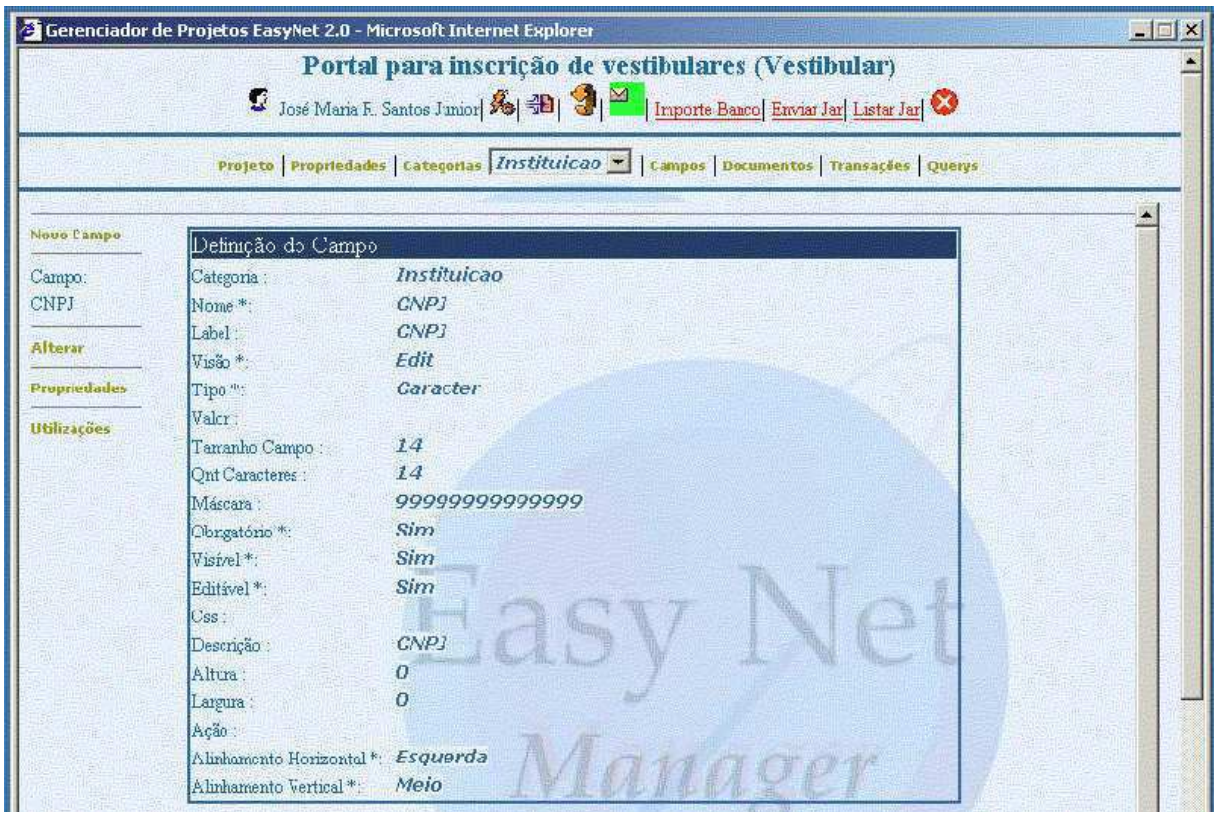
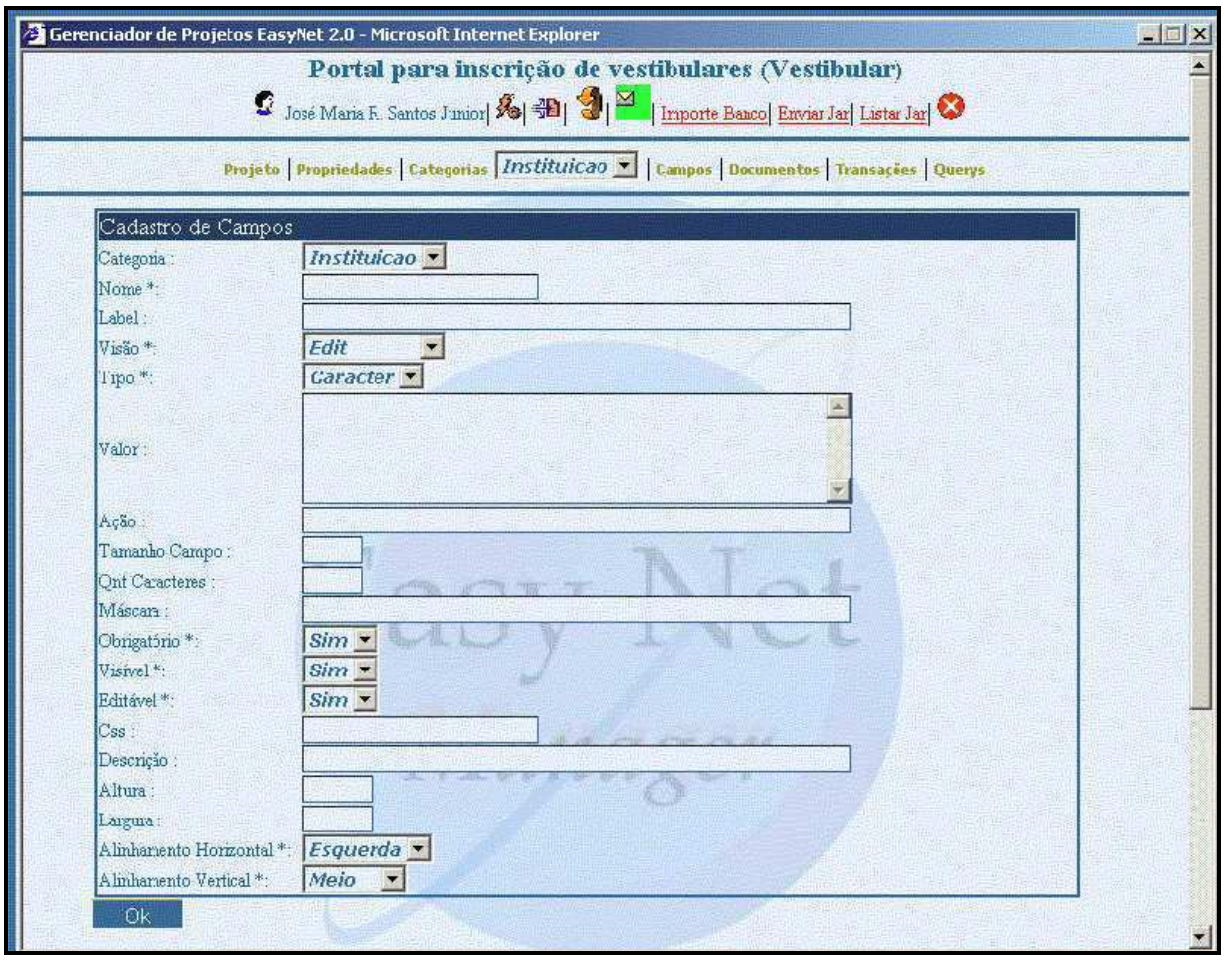


Figura 14. Criação de campo



4.6.7 Editando Documentos

A edição dos documentos é feita através da opção “Documentos” da barra de menu, onde todos os documentos da categoria selecionada serão exibidos. O botão “Novo” permite criar um novo documento. Para editar um documento basta selecioná-lo e para remover o documento basta pressionar o botão “x” ao seu lado. A tela para edição dos documentos permite editar os atributos, definir propriedades, adicionar e listar sub-componentes e verificar em quais transações o documento está sendo utilizado.

As figuras baixo exibem as telas para edição e criação de documentos.

Figura 15. Lista de documentos

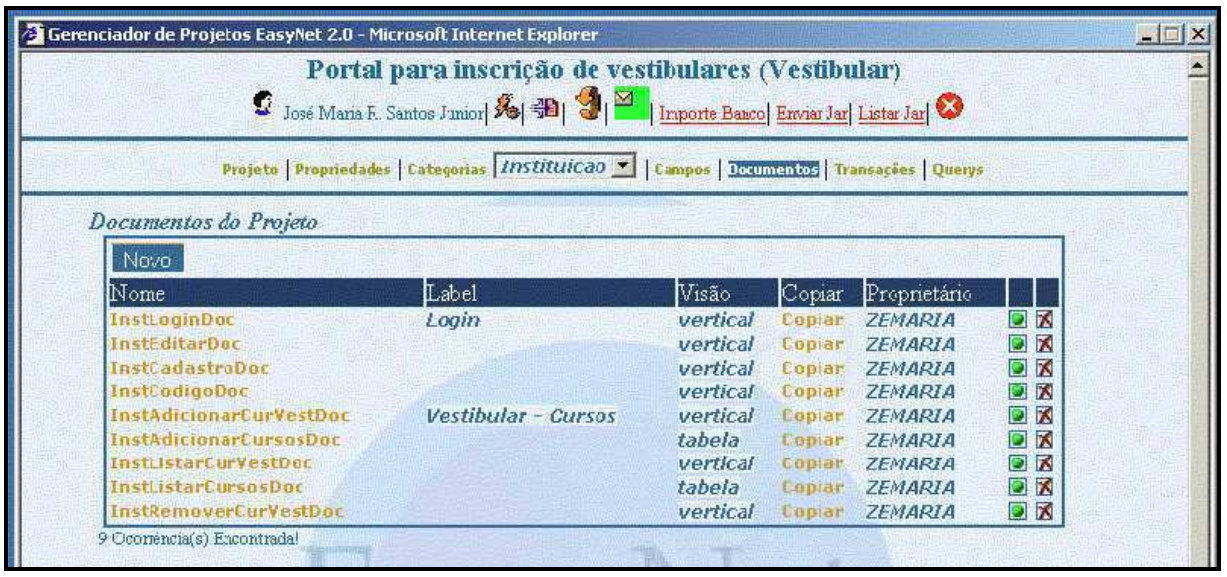


Figura 16. Edição de documento

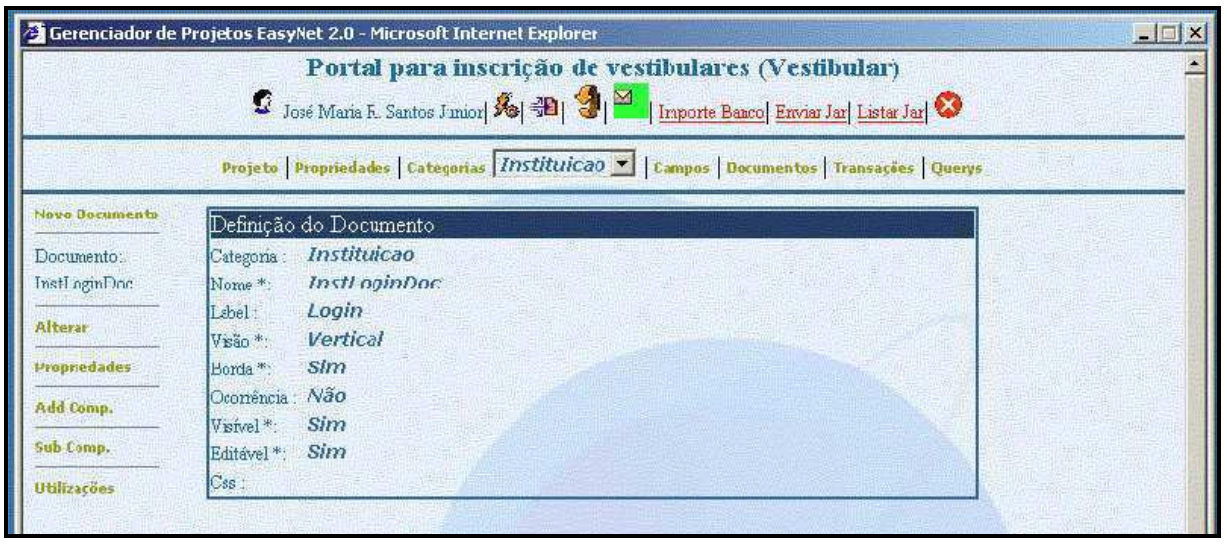
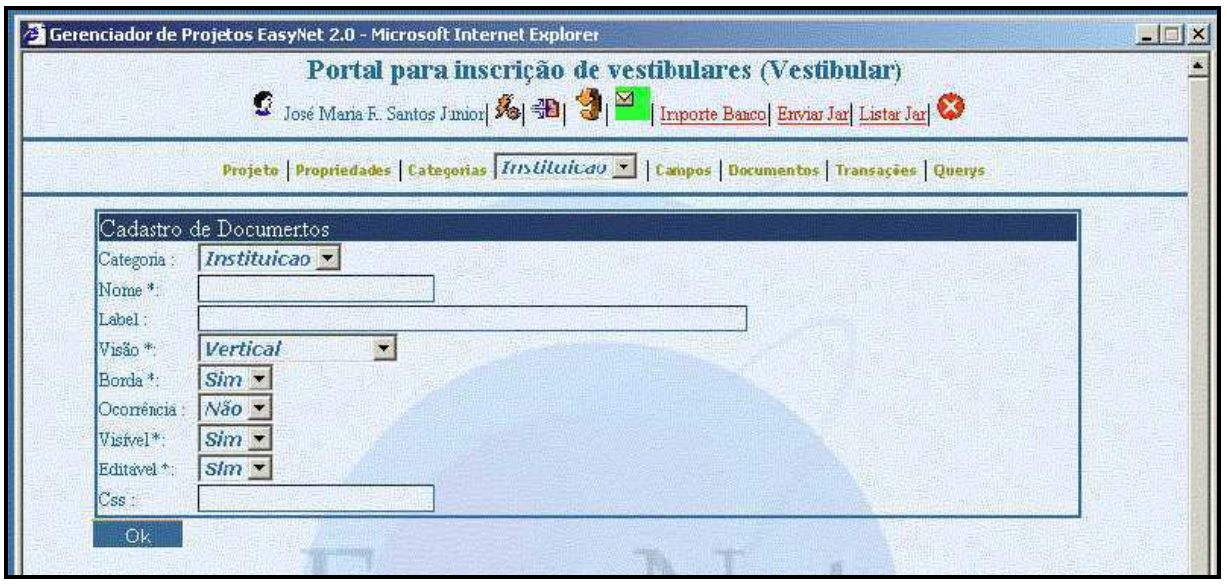


Figura 17. Criação de documento



4.6.8 Adicionando campos ao documento

Os documentos representam contêineres para agrupar os campos. A adição de campos ao documento é feita através da opção "Add Comp." (Adicionar Componentes) da barra de menu de edição do documento e a edição através da opção "Sub Comp." (Sub-Componentes). Para exibir a lista de campos a serem adicionados é necessário escolher a categoria e o tipo de componente "campo". Na tela com os campos do documento é possível remover, definir a ordem e redefinir os atributos e propriedades dos campos no documento. Os atributos e propriedades redefinidas terão valores apenas para o documento.

As figuras abaixo exibem a telas para solicitar a listagem, adicionar, listar e redefinir os campos do documento.

Figura 18. Solicitar a lista de campos

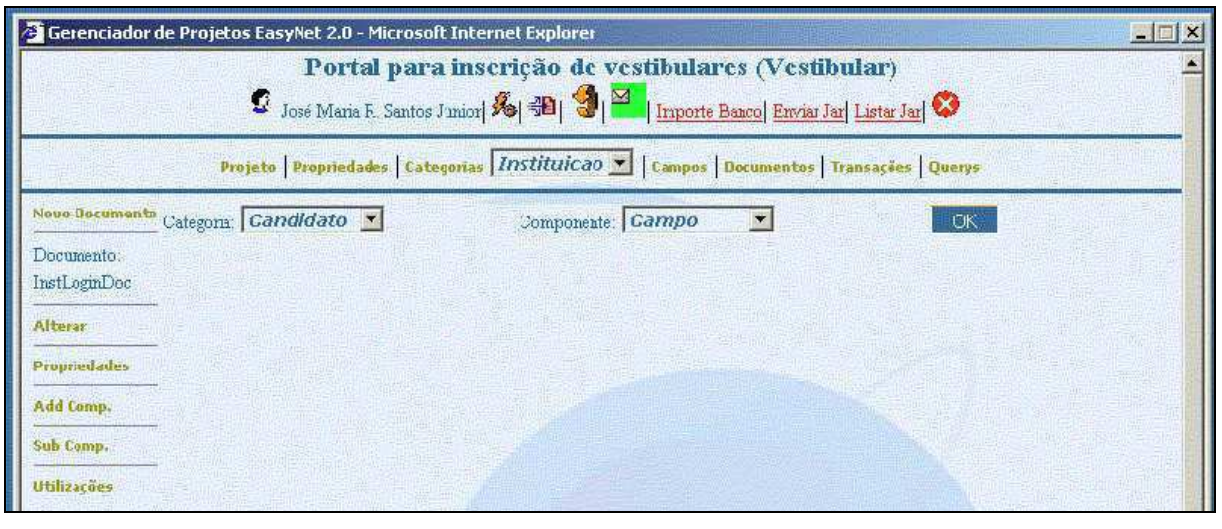


Figura 19. Selecionar o campo a ser inserido

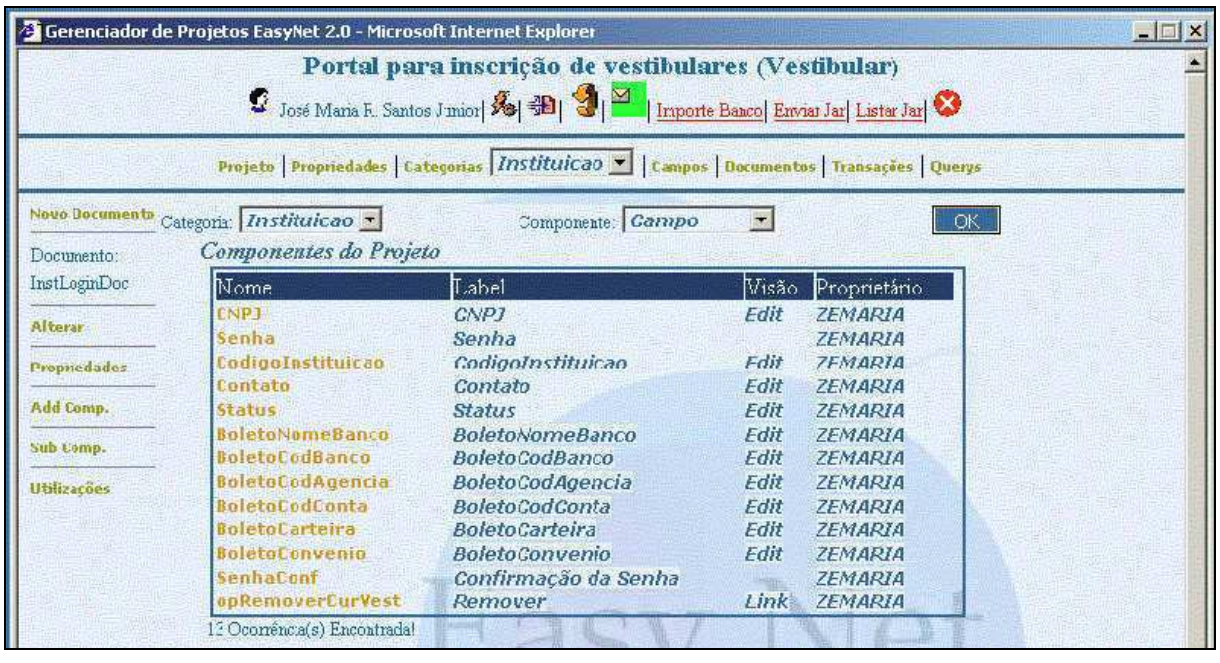


Figura 20. Campos do documento

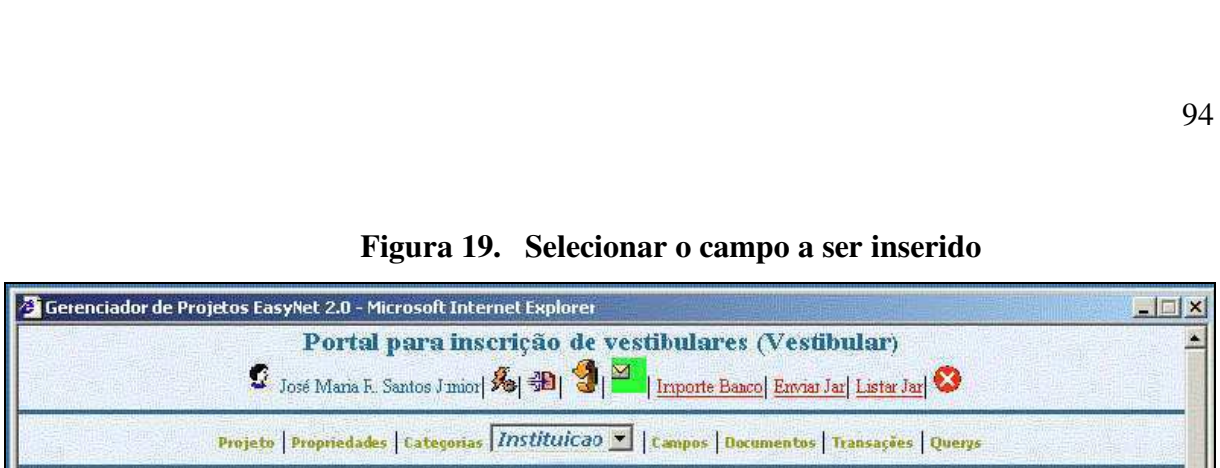
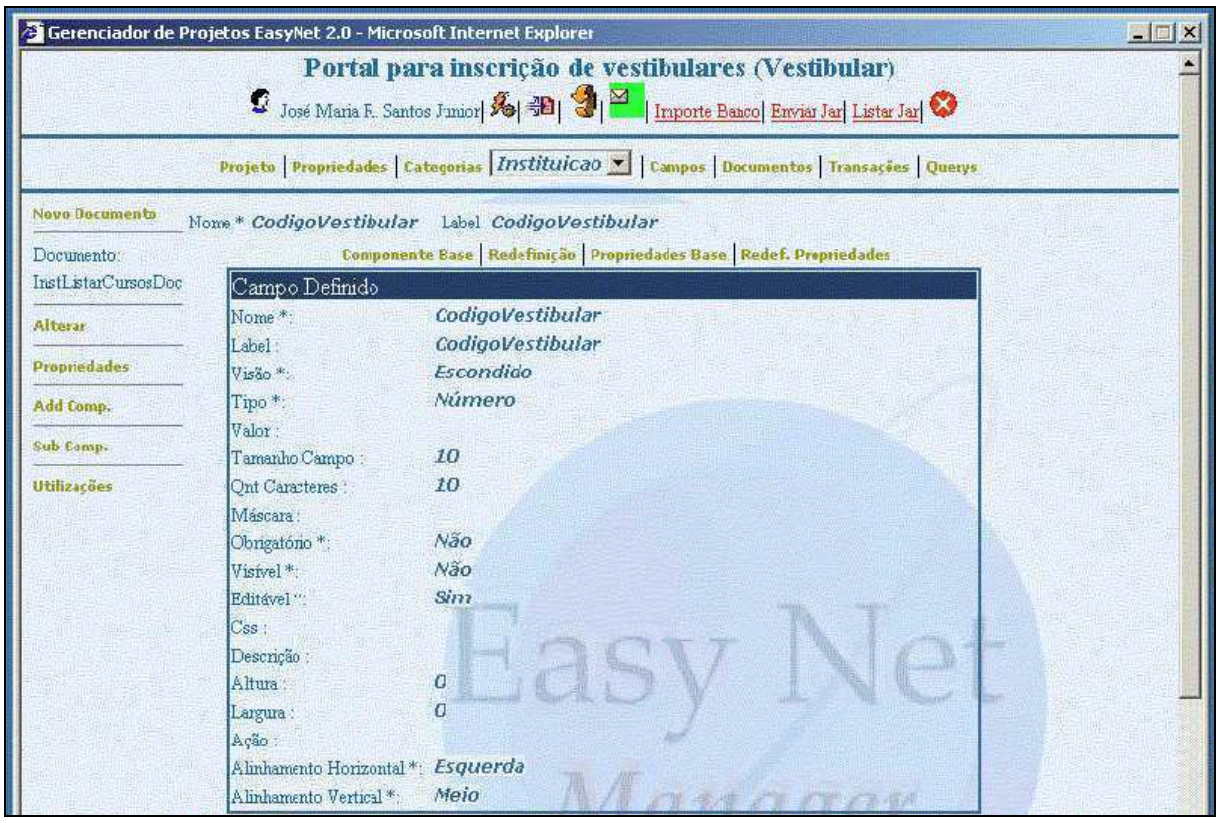


Figura 21. Editar e redefinir o campo



4.6.9 Adicionando sub-documentos ao documento

Um documento pode conter outros documentos. O documento inserido em outro é denominado de sub-documento. A utilização de sub-documentos permite re-aproveitar documentos e fornece maior flexibilidade na criação da interface. A adição de sub-documentos é semelhante à adição de campos, mudando apenas o tipo na caixa de seleção "componente". Os atributos e propriedades dos sub-documentos também podem ser redefinidos.

Figura 22. Tela para seleção de sub-documentos

4.6.10 Importando Campos e documentos do Banco de Dados

O gerenciador de projetos permite que campos e documentos sejam criados automaticamente a partir do banco de dados. Para cada tabela do banco de dados será criado um documento e para cada campo da tabela será criado e inserido um campo no documento. Os atributos nome, tipo e tamanho serão definidos de acordo com os metadados dos no Banco de dados. Os documentos e atributos importados são inseridos em categorias com o mesmo nome da tabela.

A importação é realizada a partir da opção “Importe Banco” do menu de opções e consiste em três etapas, a primeira representando a conexão com SGBD, a segunda que permite selecionar as tabelas a serem importadas e a terceira a importação propriamente dita.

As figuras abaixo demonstram as telas pra conectar ao SGBD e selecionar as tabelas:

Figura 23. Definir a conexão com SGBD

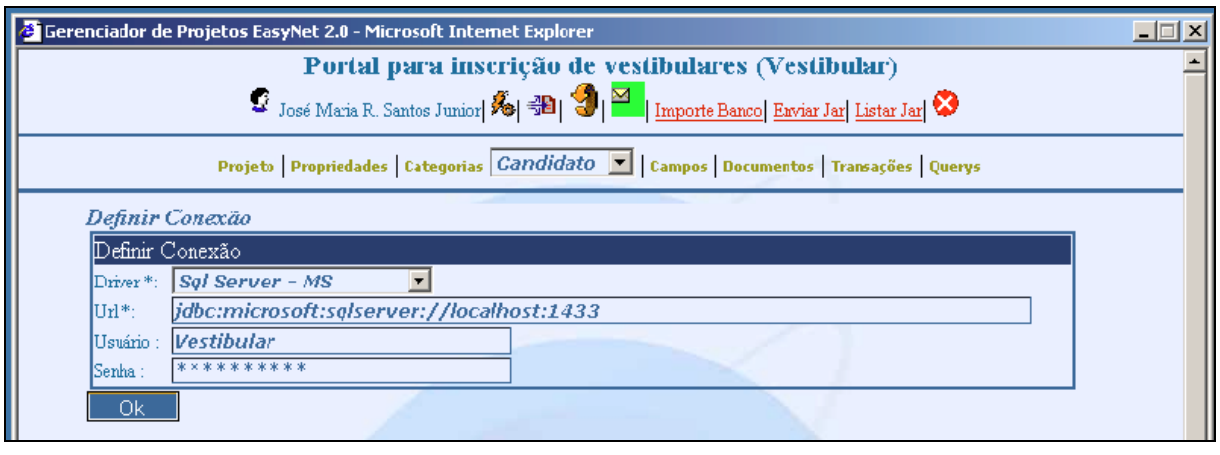
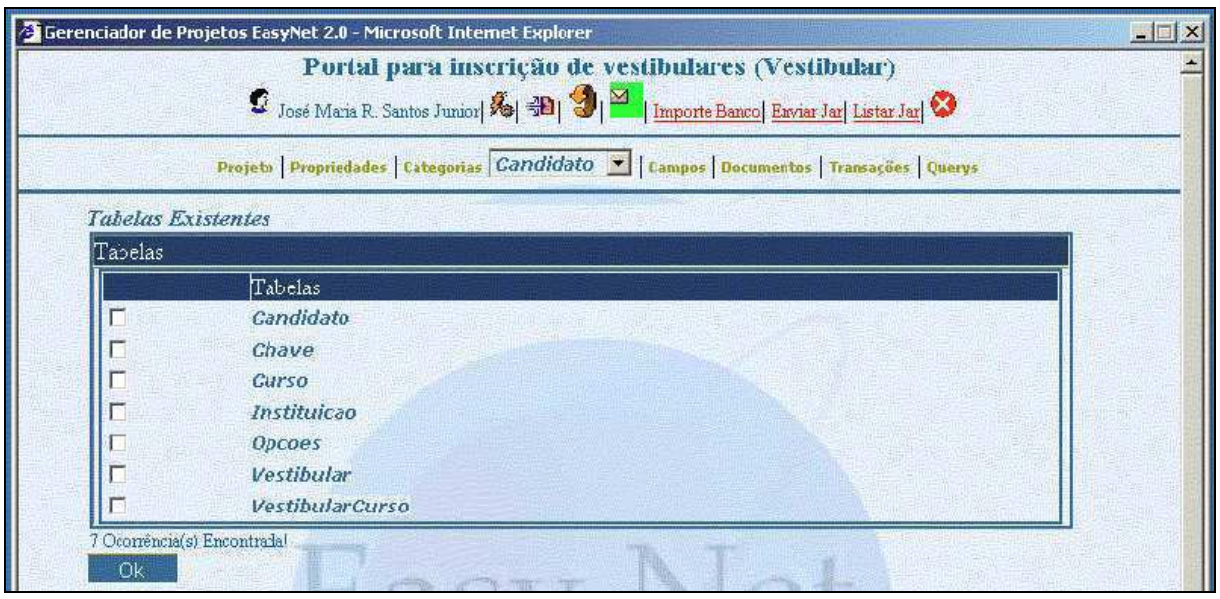


Figura 24. Selecionar as tabelas a serem importadas.



4.6.11 Editando Transações

A edição das transações é feita através da opção “Transações” da barra de menu, onde todos as transações da categoria selecionada serão exibidas. O botão “Novo” permite criar uma nova transação. Para editar uma transação basta selecioná-la e para removê-la basta pressionar o botão “x” ao seu lado. A tela para edição das transações permite editar os atributos, definir propriedades e adicionar os documentos de entrada e saída.

As figuras baixo exibem as telas para edição e criação de transações.

Figura 25. Lista de transações

Figura 26. Edição de transação

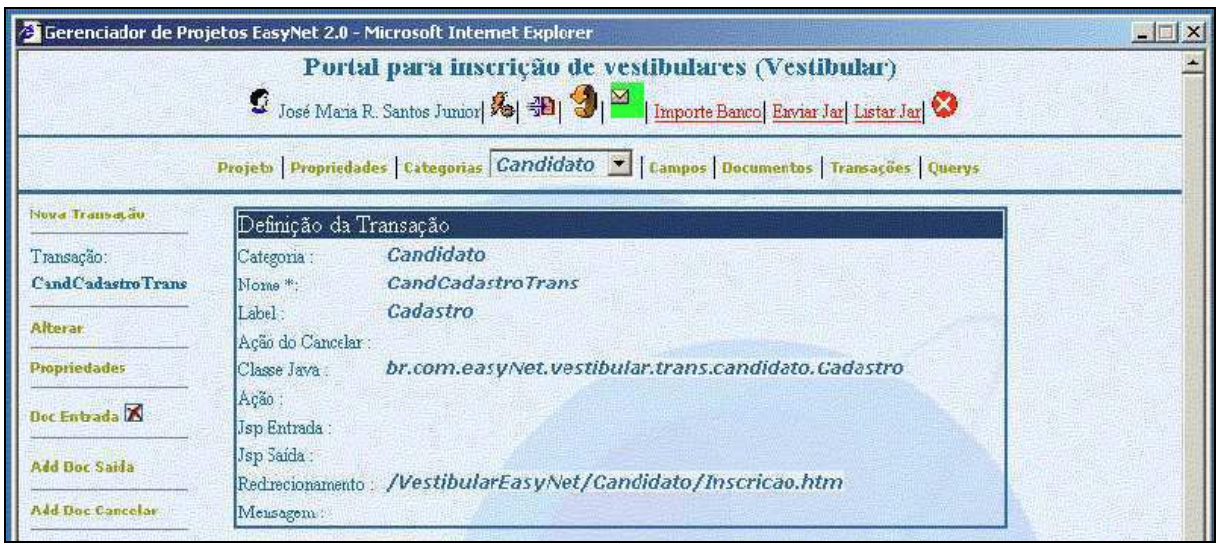
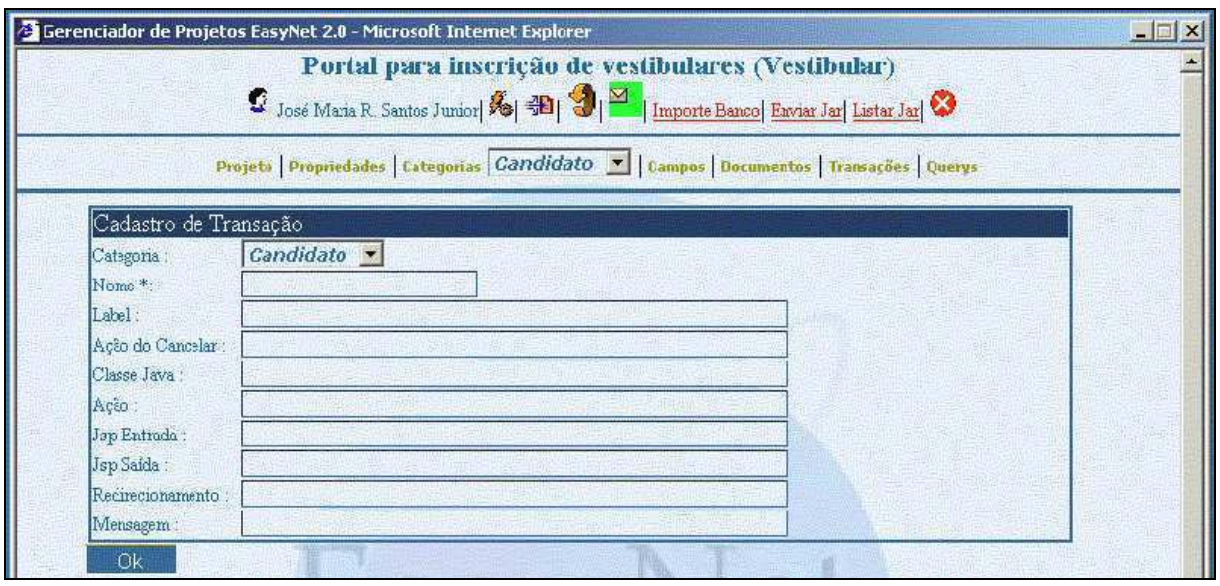


Figura 27. Criação de transação



4.6.12 Associando Documentos de Entrada e Saída à Transação

As transações do EasyNet agrupam três elementos fundamentais, os documentos de entrada e saída e a definição da classe Java que irá executar a sua lógica. A definição da classe Java é feita através do atributo "Classe Java" e os documentos de entrada e saída são definidos através das opções "Add Doc Entrada" e "Add Doc Saída", selecionado um dos documentos do projeto.

4.6.13 Editando Querys

A edição das *querys* é feita através da opção "Querys" da barra de menu, onde todos as querys da categoria selecionada serão exibidas. O botão "Novo" permite criar uma nova

consulta. Para editar uma consulta basta selecioná-la e para removê-la basta pressionar o botão “x” ao seu lado. A tela para edição das querys permite editar a sua categoria, nome e o comando SQL.

As figuras abaixo exibem as telas para edição e criação de querys.

Figura 28. Lista de Querys

Figura 29. Criação de Consulta

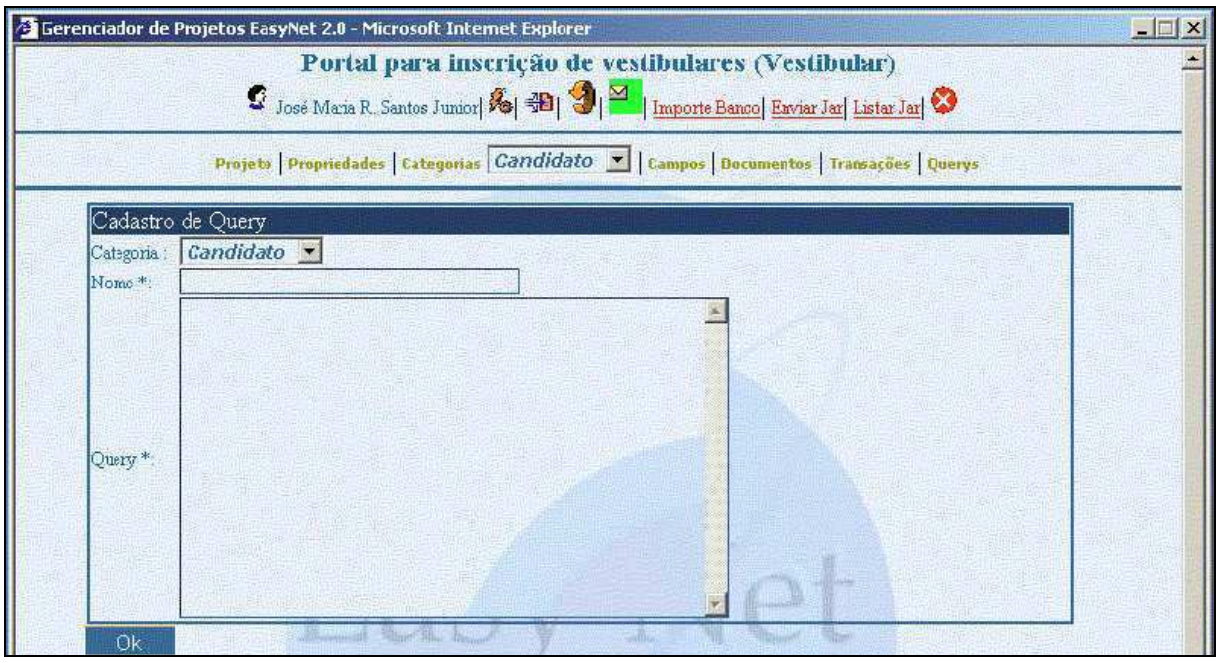
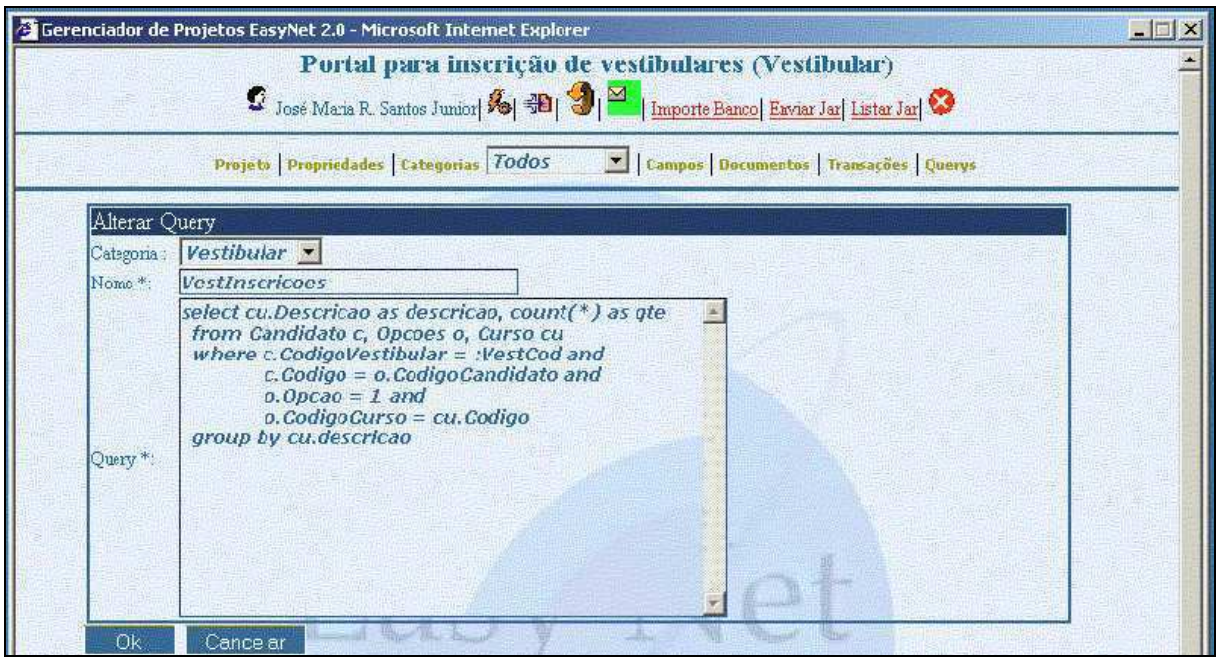


Figura 30. Edição de Consulta

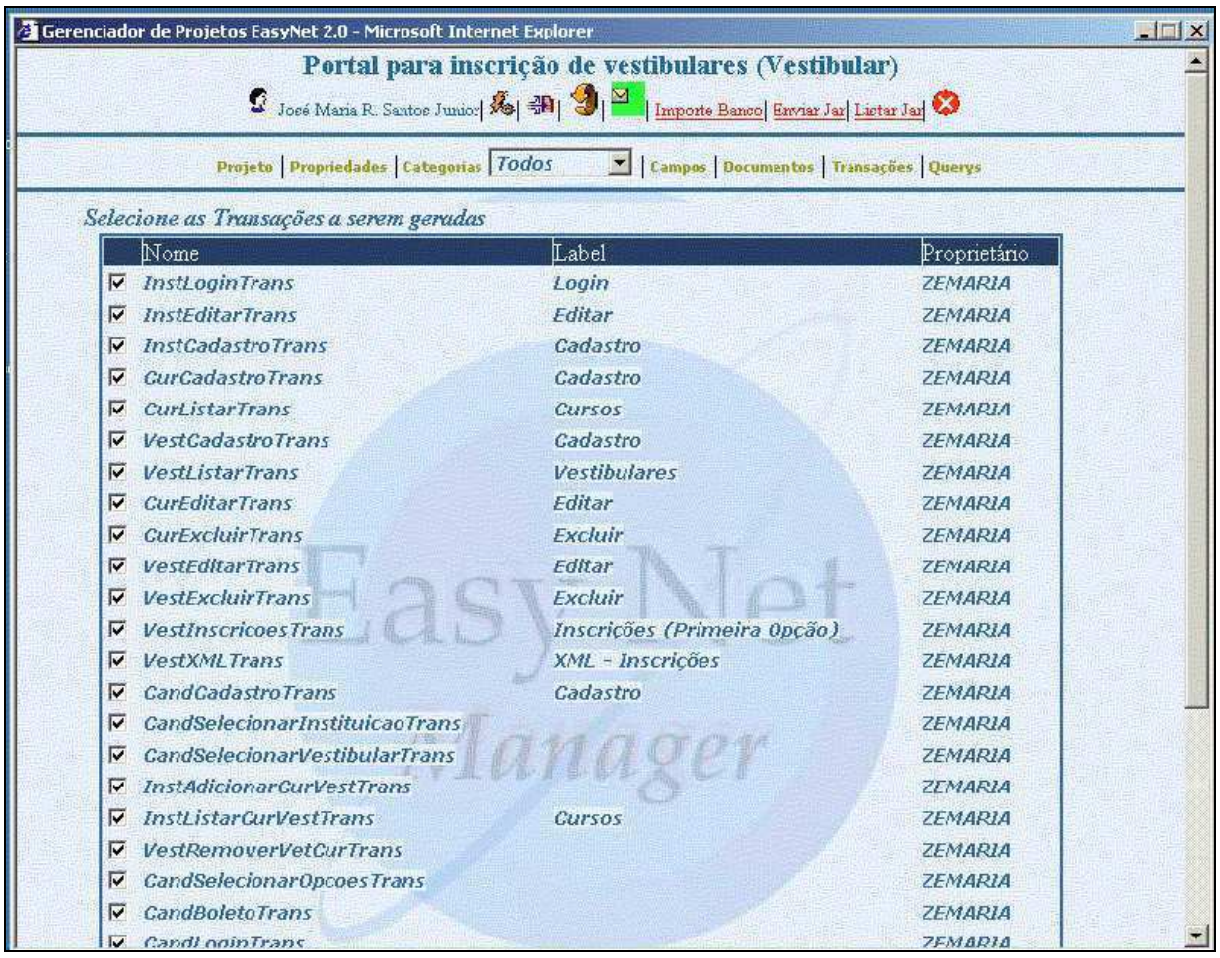


4.6.14 Gerando o arquivo XML

O gerenciador de projetos persiste todas a definições no banco de dados do EasyNet e permite que seja gerado um arquivo XML contendo as definições do projeto. O processador de transações necessita do arquivo XML para executar as transações do projeto. A geração do arquivo XML é feita através da opção  do menu de opções.

A figura abaixo mostra a tela para seleção das transações a serem processadas para gerar o arquivo XML do projeto.

Figura 31. Seleção das transações para publicação



4.6.15 Implementando as classes Java

As transações definidas no gerenciador de projetos representam unidades lógicas de interação com o usuário e a lógica dessas transações são implementadas através de classes Java que estendem a classe de transação (`br.com.easyNet.generico.Transacao`) da API do EasyNet. Esta classe possui a implementação necessária para a sua execução através do processador de transações e fornece ao desenvolvedor recursos para interagir com os elementos do projeto (documentos, campos e querys).

As classes de transação devem redefinir os métodos `inicialize` e `execute`. O método `inicialize` é executado antes do documento de entrada ser exibido ao usuário e o método `execute` é executado após a submissão do documento de entrada. O método `inicialize` é utilizado para definir valores do documento de entrada e o método `execute` realiza a lógica da transação.

O código fonte abaixo demonstra uma classe de transação para efetuar o login de uma instituição. A transação consiste em obter o cnpj e a senha do documento de entrada, verificar

a existência de uma instituição com o cnpj fornecido, comparar as senhas e colocar a interface remota do Entity Bean da instituição na sessão do usuário. Quando o método execute lança uma exceção `br.com.easyNet.util.DadoException` o documento de entrada é re-exibido com a mensagem de erro acima do documento.

Código Fonte 20. Classe Java implementando uma transação

```
package br.com.easyNet.vestibular.trans.instituicao;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.ejb.FinderException;

import br.com.easyNet.generico.Transacao;
import br.com.easyNet.generico.Documento;
import br.com.easyNet.util.TabelaHash;
import br.com.easyNet.bancoDados.Registros;
import br.com.easyNet.util.DadoException;

import br.com.easyNet.vestibular.ejb.instituicao.InstituicaoHome;
import br.com.easyNet.vestibular.ejb.instituicao.Instituicao;

public class Login extends Transacao {

  public void execute () throws Exception {
    Documento doc = this.getDocEntrada();
    String cnpj = (String) doc.getObject("CNPJ");
    String senha = (String) doc.getObject("Senha");

    try {
      Context ctx = new InitialContext();
      InstituicaoHome instituicaoHome =
        (InstituicaoHome) ctx.lookup("vestibularEasyNet.InstituicaoEJB");
      Instituicao instituicao = instituicaoHome.findByCNPJ(cnpj);
      if ( !instituicao.getSenha().equals(senha) ) {
        String msg = "Senha inválida";
        throw new DadoException(msg);
      }
      getSessao().put("Instituicao", instituicao);
    } catch (FinderException e) {
      String msg = "CNPJ não encontrado";
      throw new DadoException(msg);
    } catch (Exception e) {
      throw new DadoException(e.getMessage());
    }
  }
}
```

As principais funcionalidades oferecidas às classes de transações são obter os documentos de entrada e saída, obter *querys e* obter a sessão do usuário.

4.6.15.1 Obtendo e utilizando os documento de entrada e saída.

Os documento de entrada e saída são obtidos através dos métodos `getDocEntrada()` e `getDocSaida()`. Os campos do documento são manipulados através dos métodos `getCampo(campo)` e `setCampo(campo,valor)`. Quando um documento possuir sub-documentos, o método `getSubDocumento(nome)` retornará o sub-documento. Quando o

documento possuir ocorrências os campos serão manipulados através dos métodos `getCampo(i,campo)` e `setCampo(i,campo)` e o método `getNumeroOpcoes()` retornará o número de ocorrências. O método `novaOcorrencia()` cria uma nova ocorrência no documento.

O código fonte abaixo demonstra uma transação que lista todos os curso de uma determinada instituição.

Código Fonte 21. Transação manipulando documentos

```
package br.com.easyNet.vestibular.trans.curso;

import java.util.Iterator;

import br.com.easyNet.generico.Transacao;
import br.com.easyNet.generico.Documento;

import br.com.easyNet.vestibular.trans.Base;
import br.com.easyNet.vestibular.ejb.instituicao.*;
import br.com.easyNet.vestibular.ejb.curso.*;

public class Listar extends Base {

  public void execute () throws Exception {
    Documento doc = this.getDocSaida();
    Instituicao inst = (Instituicao) super.getSessao().get("Instituicao");
    Iterator cursos = inst.getFromRelationShipCurso().iterator();
    while ( cursos.hasNext() ) {
      Curso curso = (Curso) cursos.next();
      doc.novaOcorrencia();
      doc.setCampo("Codigo", curso.getCodigo().setScale(0));
      doc.setCampo("Descricao", curso.getDescricao());
    }
  }
}
```

4.6.15.2 Obtendo e executando querys

As querys são obtidas através do método `getSql()` e os valores dos seus parâmetros são definidos através do método `setParam(nome,valor)`. A query é executada através do método `consultar(query)`, o qual retorna um objeto do tipo `Registros`. A classe Registros implementa a interface `java.util.Iterator` e armazena objetos do tipo `Registro`.

O código fonte abaixo demonstra uma transação que executa um comando SQL para obter todos os inscritos na primeira opção do vestibular de uma instituição.

Código Fonte 22. Transação utilizando query

```
package br.com.easyNet.vestibular.trans.vestibular;

import java.math.BigDecimal;

import br.com.easyNet.generico.Transacao;
import br.com.easyNet.generico.Documento;
import br.com.easyNet.bancoDados.*;

public class Inscricoes extends Transacao {
```

```
  public void execute () throws Exception {
    Documento in = super.getDocEntrada();
    Documento out = super.getDocSaida();
    BigDecimal codigoVest = (BigDecimal) in.getObject("Codigo");
    Sql sql = super.getSql("VestInscricoes");
    sql.setParam("VestCod", codigoVest);
    Registros regs = super.consultar(sql);
    while ( regs.hasNext() ) {
      Registro reg = regs.next();
      String descricao = (String) reg.getValor("descricao");
      BigDecimal qte = (BigDecimal) reg.getValor("qte");
      out.novaOcorrencia();
      out.setCampo("Descricao", descricao);
      out.setCampo("Numero", qte);
    }
  }
}
```

4.6.15.3 Obtendo e utilizando a sessão do usuário

A sessão do usuário é obtida através do método `getSessao()` e permiti inserir e remover objetos através do seus métodos `put(nome,objeto)` e `get(nome)`.

4.7 Conclusão

O EasyNet é formado por um conjunto de ferramentas para o desenvolvimento de aplicações WEB sobre a plataforma Java, objetivando fornecer maior produtividade e menor complexidade. Os requisitos do EasyNet foram encontrados a partir de observações realizadas na implementação de aplicações WEB e sobre a plataforma J2EE.

Os principais pontos observados como críticos, para atingir os objetivos propostos foram: implementação dos componentes EJB e da interface com o usuário.

Em relação à implementação dos componentes EJB foi criada uma ferramenta (*Gerador Automático de Entity Beans*) para gerar automaticamente os Entity Beans a partir do Banco de Dados e uma estratégia para implementar as regras de negócio através de classes Java. Com estas duas ações o esforço para a implementação de componentes EJB foi reduzido a zero.

Em relação à camada de interface foram criadas uma estratégia, ferramenta e API. A estratégia permite que os componentes da camada de interface sejam previamente definidos através da ferramenta (*Gerenciador de Projetos*), gerados dinamicamente (*Processador de transações*) a partir destas definições e utilizados pelas classes de transação Java (*API do EasyNet*).

O EasyNet permite que os recursos da Plataforma J2EE sejam utilizados no desenvolvimento de aplicações WEB típicas, sem a necessidade da implementação de seus componentes. Fornecendo assim um nível mais elevado na utilização da plataforma J2EE no contexto de aplicações WEB típicas.

Capítulo 5 - Validando o EasyNet

Com o objetivo de validar a implementação e verificar o aumento da produtividade e da simplicidade fornecidos pelo EasyNet, a aplicação "Portal para Publicação e Inscrição em Vestibulares", implementada no capítulo 3, foi re-implementada através do EasyNet.

A re-implementação da aplicação permitiu verificar o aumento da produtividade e a redução da complexidade, através da comparação entre as duas implementações.

5.1 Re-Implementando a aplicação com o EasyNet

A seguir serão descritos os passos realizados na implementação da aplicação com o EasyNet. Esses passos serão comparados com a implementação da aplicação sem o EasyNet.

5.1.1 Camada de Acesso a Dados

O esforço para implementação dos Entity Beans, necessário na implementação da aplicação sem o EasyNet, foi eliminado. A geração dos Entity Beans exigiu apenas a configuração da conexão com o Banco de Dados e a aplicação gerou todos os Entity Beans e seus *deployment descriptors*.

A geração automática dos Entity Beans forneceu um aumento significativo na produtividade. A implementação manual dos Entity Beans exigiu a criação de **25** classes e interfaces totalizando **1172** linhas de código e dos *deployment descriptors* exigiu **1186** linhas de código.

Uma observação importante é que o banco de dados da aplicação implementada possuía apenas **7** tabelas e quanto mais tabelas o banco de dados da aplicação possui, maior será o esforço necessário à criação manual dos Entity Beans. Com a utilização do EasyNet esse esforço será sempre constante e próximo de zero, independentemente da quantidade de tabelas.

5.1.2 Camada de Interface

A implementação da camada de interface através do EasyNet foi feita através da definição dos seus elementos: campos e documentos. A utilização desses componentes de interface permite que o processador de transações do EasyNet gere automaticamente, em tempo de execução, os componentes WEB necessários para a interação com o usuário.

A criação de componentes WEB (Páginas HTML e JSP) foi necessária apenas para definir a estrutura básica da interface (página inicial, menus e estrutura de frames) e implementar funcionalidades não fornecidas pela geração dinâmica de interface do EasyNet. Apenas dois JSPs foram implementados: um para exibir o conteúdo XML com as inscrições do vestibular de uma instituição, e outro para exibir o boleto bancário para pagamento da inscrição.

A utilização da estratégia de geração automática dos componentes de interface aumentou a produtividade e reduziu a complexidade. Não foi necessário criar páginas HTML ou JSP para a maioria das operações disponibilizadas, nem implementar e associar aos componentes WEB rotinas JavaScript para validação e/ou formatação de dados.

Na implementação da camada de interface sem o EasyNet foi necessário implementar **13** páginas HTML contendo **229** linhas, **29** páginas JSP contendo **2639** linhas e **3** arquivos JavaScript contendo **398** linhas. Nas páginas HTML e JSP ainda foi necessário definir, nos formulários, a utilização das rotinas JavaScript.

Na implementação da camada de interface com o EasyNet foi necessário implementar **13** páginas HTML contendo **151** linhas e **2** páginas JSP contendo **529** linhas.

Na implementação da aplicação através do Easynet o desenvolvedor não precisou utilizar conhecimentos, nem implementar componentes HTML, JSP e JavaScript na implementação da maior parte da interface.

A estratégia de definição dos componentes de interface utilizada pelo EasyNet permite uma grande re-utilização e quanto maior a interface com o usuário, maior será o ganho de produtividade fornecido pelo EasyNet.

5.1.3 Camada de Transações

Na arquitetura do EasyNet a camada de transações é implementada através das classes Java e a comunicação entre a camada de transações e de interface é feita através da API. Na implementação da aplicação com o EasyNet foi necessário implementar 21 classes Java de transação, totalizando 830 linhas de código. Essas classes Java possuem baixa complexidade e a maior parte do código é relativo à comunicação com a camada de interface, leitura e escrita de dados dos documentos de entrada e saída. As operações com os dados foram na maioria das vezes realizadas através dos Entity Beans e em poucas situações através da Querys.

A utilização das classes Java aumentou a produtividade e reduziu a complexidade, pois o desenvolvedor não precisou implementar as transações em Session Beans. A implementação de Session Beans exige conhecimento da especificação EJB e necessita da implementação de

duas interfaces, uma classe abstrata e do deployment descriptors. A publicação de Session Beans exige a operação de deployment.

Na aplicação implementada sem o EasyNet as regras de transações foram implementadas nos Entity e Session Beans e nos JSPs. A comparação da quantidade de linhas de código entre a implementação da camada de transações das duas implementações não seria um mecanismo adequado para comparar a complexidade.

5.2 Consideração sobe a utilização do EasyNet

A estratégia de geração dinâmica dos Entity Beans e da interface foram os itens que mais contribuíram para aumentar a produtividade e reduzir a complexidade. É possível desenvolver aplicações WEB e EJB sem conhecimentos aprofundados em HTML, JavaScript, JSP e EJB. A interface é gerada dinamicamente através das definições realizadas no gerenciador de projetos e o Entity Beans são gerados automaticamente, sendo necessário conhecimento apenas da sua utilização a partir da camada cliente.

Diante da comparação entre as duas implementações da aplicação, foi observado que a utilização do EasyNet efetivamente reduziu a complexidade e aumentou a produtividade. É importante observar que a aplicação implementada apresenta características típicas de aplicações de comércio eletrônico, voltadas para a interface com o usuário e com baixa complexidade na camada de regras de negócio.

É possível que o mesmo ganho de produtividade e complexidade não seja obtido em aplicações web com características distintas, mas uma grande parcela da necessidade de aplicações WEB pode ser implementada através do EasyNet.

A tabela abaixo mostra a quantidade de componentes e linhas de código necessários em cada camada das duas implementações:

Tabela 15. Quantidade de Componentes e Linhas de Código

Implementação	Camada de Interface	Camada de Transações	Camada de Acesso a Dados
Sem o EasyNet	45/3296	*	25/2368
Com o EasyNet	15/680	21/830	0

* Distribuídos entre Entity Beans, Session Beans e JSP.

5.3 Comparando o EasyNet com outras Tecnologias e Ferramentas

A estratégia de arquitetura aberta da Sun Microsystems para a plataforma J2EE aliada à baixa produtividade e alta complexidade do desenvolvimento de aplicações WEB Multicamadas, proporcionou o surgimento de diversas tecnologias e ferramentas. Um conjunto dessas ferramentas foi analisado com o objetivo de comparar os recursos oferecidos pelo EasyNet aos recursos oferecidos por outras estratégias e ferramentas. A tabela abaixo lista as ferramentas analisadas.

Tabela 16. Ferramentas de desenvolvimento J2EE

Ferramenta	Site
AltoWeb	http://www.altoweb.com
EJB Generator	http://www.beust.com/ejbgen
Forte Enterprise	http://wwws.sun.com/software/sundev
JBanana	http://www.jbanana.com/indexPort.html
JBuilder Enterprise	http://www.borland.com/jbuilder
Struts	http://jakarta.apache.org/struts
Together Control Center	http://www.togethersoft.com/products/controlcenter.jsp

5.3.1 AltoWeb

O AltoWeb é uma plataforma de desenvolvimento sobre a plataforma J2EE. Necessita de um servidor EJB e possui um módulo para execução e outro para edição dos projetos. O módulo de edição de projeto (AltoStudio) é uma aplicação Java que permite a manipulação de elementos representando componentes de todas as camadas (interface, transações e acesso a dados).

O AltoWeb permite a criação completa de uma aplicação, gerenciando todo o seu ciclo de vida e componentes necessários para a sua implementação. Fornece recursos tais como: modelagem através diagramas de processos, criação da interface com o usuário, geração de Entity Beans 1.1 e publicação da aplicação no servidor EJB. Funciona integrado com o servidor EJB, permitindo publicar de forma simplificada seus projetos no servidor EJB.

5.3.2 EJB Generator

O EJB Generator é uma aplicação Java que gera automaticamente as interfaces remotas e home, a classe primária e o *deployment descriptors* para Entity Beans 2.0, a partir da sua classe abstrata. É necessário introduzir comentários *javadoc* especificando propriedades dos elementos a serem gerados, tais como: métodos `finders`, a classe Chave Primária, o nome

`jndi`, os relacionamentos, etc. Não é uma aplicação gráfica, sendo executada através de linha de comando.

5.3.3 Forte Enterprise

O Forte é um IDE da Sun Microsystems. Permite desde a edição de classes Java à publicação de aplicações J2EE nos servidores EJB. Possui uma série de *wizards* para criação de componentes J2EE. Suas principais características são: Suporte avançado para *Web Services*; integração com os principais servidores EJB e ser totalmente compatível com as especificações J2EE 1.3, EJB 2.0, JSP 1.2 e Servlet 2.3.

5.3.4 JBanana

O JBanana é *Framework* MVC (Model View Controler) de código fonte aberto para o desenvolvimento de aplicações web utilizando as tecnologias Java, XML e XSL. É formado por um conjunto de componentes que permitem, através de configurações em documentos XML, a realização automática de tarefas tais como: controle da validação dos dados, controle de navegabilidade entre telas da aplicação e seleção da interface adequada a ser mostrada para o usuário. O mecanismo de comunicação entre a camada de interface e a camada de transações é feita através de arquivos XML.

5.3.5 JBuilder Enterprise

O JBuilder é um IDE da *Borland* para desenvolvimento de aplicações Java. A versão Enterprise é voltada para o desenvolvimento aplicações J2EE, possuindo uma séria de recursos para a implementação de componentes EJB através de *wizards* e integrando-se com os principais servidores EJB. Seus principais recursos são: visualização das classes através de diagrama de classe UML, suporte a *Web Services*, permite a edição de arquivos XML, JSP e SQL e fornecimento de componentes e ferramentas para construção de aplicações GUI e com acesso a banco de dados.

5.3.6 Struts

Struts é um *Framework*, com código fonte aberto, para construção de aplicações WEB. Faz parte do projeto Jakarta da Apache (www.apache.org) e sua arquitetura é baseada nas tecnologias Servlets, JavaBeans e XML. Utiliza a arquitetura MVC (Model View Controler) fornecendo componentes próprios.

Struts permite a definição de componentes da camada de interface e a comunicação destes com classes Java que utilizam sua API para implementar as regras de negócio. O

framework é “instalado” no servidor WEB e controla a execução de seus componentes através de um Servlet que captura as requisições do usuário e ativa os componentes.

5.3.7 Together Control Center

O Together Control Center é uma ferramenta de desenvolvimento que simplifica e integra a análise, projeto, implementação, depuração e publicação de aplicações. Suporta o desenvolvimento de aplicações para as tecnologias: Java, C++, IDL, Visual Basic 6 e .NET e C#. Os principais recursos oferecidos pelo Together Control Center são: Edição dos principais diagramas UML (Classes, Use Case, Atividade, Componente e Distribuição), editor de código fonte para várias linguagens de programação, características CASE gerando código fonte e interligando os modelos ao código fonte, geração de aplicações gráficas, geração automática de módulos para teste dos componentes implementados e fornecimento de wizards para criação de componentes EJB e Web Services.

5.3.8 Comparando as ferramentas

As ferramentas analisadas possuem características distintas, mas todas possuem objetivos em comum: aumentar a produtividade e reduzir a complexidade na implementação de aplicações WEB J2EE. As duas motivações (Geração de componentes EJB e geração de componentes da camada de interface) encontradas no desenvolvimento do EasyNet como sendo as principais para alcançar os objetivos propostos foram às mesmas adotadas pela maioria das ferramentas utilizadas.

Os seguintes recursos foram encontrados nas ferramentas analisadas:

A. Edição de programas

B. Características Case

C. Geração de componentes da camada de acesso a dados

D. Geração de componentes da camada de transações

E. Geração de componentes da camada de interface

F. Geração de Deployment Descriptors;

G. Integração com Servidores EJB;

H. API para a implementação das transações;

I. Geração dinâmica da interface com o usuário;

A tabela abaixo mostra a relação das ferramentas e os recursos oferecidos:

Tabela 17. Ferramentas J2EE x Recursos Oferecidos

Ferramenta	Recursos oferecidos								
	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**
AltoWeb	X	X		X	X		X		
EJB Generator			X			X			
Forte Enterprise	X		X	X		X	X		
JBanana					X		X	X	X
JBuilder Enterprise	X	X	X	X		X	X		
Struts					X		X		X
Together Control Center	X	X	X	X		X	X		
EasyNet			X	X	X	X		X	X

9. Conclusões

9.1. Conclusões

Através do estudo da plataforma J2EE no capítulo 2 e da implementação de uma aplicação web utilizando apenas os seus recursos no capítulo 3, foi possível verificar que a plataforma J2EE fornece uma excelente infra-estrutura para a construção de aplicações robustas, escaláveis, portáveis, estáveis e padronizadas. Mas o desenvolvimento de aplicações diretamente sobre a Plataforma J2EE mostrou-se uma tarefa bastante complexa e pouco produtiva. Isso se deve a estratégia da Sun Microsystems em oferecer uma plataforma aberta fornecendo apenas a infra-estrutura, esperando que terceiros contribuíssem desenvolvendo novas estratégias e ferramentas para oferecer maior produtividade e menor complexidade na implementação de aplicações. Isso pode ser facilmente observado através da quantidade de ferramentas existentes para o desenvolvimento de aplicações sobre a plataforma J2EE. Essa estratégia da Sun Microsystems permitiu que a plataforma J2EE fosse utilizada através de ferramentas para atender requisitos bastante diversificados.

Neste trabalho de pesquisa o objetivo era fornecer uma estratégia e uma ferramenta visando aumentar a produtividade e reduzir a complexidade no desenvolvimento de aplicações Web sobre a Plataforma J2EE. Na implementação da aplicação realizada no capítulo 3 foi possível identificar claramente a existência de aspectos que poderiam ser explorados para alcançar os objetivos propostos. Os principais aspectos identificados como determinantes para a redução da produtividade e aumento da complexidade foram: a implementação da camada de acesso a dados, através de Entity Beans e a construção da camada de interface.

Na implementação dos Entity Beans foi identificado que muita codificação era feita para seguir a especificação EJB e que era possível gerar essa codificação, automaticamente a partir do Banco de Dados. A implementação dos Entity Beans exigiu um enorme esforço e mostrou-se uma tarefa bastante sujeita erros, principalmente pelo nível de detalhamento exigido pela especificação EJB 2.0.

A construção da camada de interface exigiu a criação de uma grande quantidade de componentes e utilizou diversas tecnologias diferentes (HTML, CSS, JavaScript, Servlet e JSP), tornando a sua implementação uma atividade bastante trabalhosa.

Para atender aos requisitos identificados como essenciais para a redução da complexidade e aumento da produtividade foi criada uma estratégia formada por 3 ferramentas denominada de **EasyNet**. As três ferramentas que formam o EasyNet são: *Gerador Automático de Entity Bean*, *Gerenciador de Projetos* e *Processador de Transações*.

O EasyNet permite a geração automática dos Entity Beans para todo um banco de dados, eliminado a complexidade dessa atividade. O EasyNet permite que a interface com o usuário seja definida através do conteúdo e formatação dos dados, utilizando uma estratégia para gerar dinamicamente as páginas HTML. A geração dinâmica da interface reduz a necessidade de construção de componentes da camada de interface (páginas HTML e JSP e rotinas JavaScripts). Para fornecer flexibilidade o EasyNet permite que, nas situações onde a sua estratégia de geração dinâmica da interface não atenda as necessidades, seja possível utilizar diretamente componentes da camada de interface (Páginas HTML e JSP).

No capítulo 5 a aplicação implementada no capítulo 3 foi re-implementada, permitindo comparar a produtividade e complexidade das duas implementações. A implementação da aplicação através do EasyNet foi bem mais produtiva e menos complexa. Assim, o EasyNet mostrou ser possível oferecer maior produtividade e menor complexidade no desenvolvimento de aplicações sobre a plataforma J2EE.

10. Referências Bibliográficas

[AMO01] AMOR, Daniel. **A (R)Evolução do E-Business**. Makron Books.2000.

[APA01] The Jakarta Project. Dispovível em: http://jakarta.apache.org/

[APA02] Apache ant. Disponível em: http://jakarta.apache.org/ant/index.html

[BEA01] Bea. Dev2Dev. Disponível em: http://dev2dev.bea.com/index.jsp

[BEA02] BEA WebLogic 6.1 Documentation. Disponível em: http://edocs.bea.com/wls/docs61/index.html

[BOD01] BODOFF, Stepanie - GREEN, Dale. **J2EE TUTORIAL**. Addison Wesley. 2002

[BUS01] BONNETT, Kendra. **An IBM Guide to Doing Business on the Internet**. McGraw-Hill. 2000.

[CAM01] CAMPIONE, Mary. **The Java Tutorial Second Edition**. The Java Series. 1998.

[DEI01] DEITEL, Harvey M. **Java How to Program**. Prencite HallTerceira Edição. 1999.

[LIV01] LIVINGISTON, Dan - BROWN, Micah. **Essential CSS and DHTML for Web Masters**. Prentice Hall. 1999

[LOU01] LOU, Marco. **EJB & JSP - JAVA ON THE EDGE**, UNLIMITED EDITION. Jrhn Wiley Computer. 2001

[MIC01] Microsoft. JScript Documentation. Disponível em: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/script56/html/js56jsoriJScript.asp

[MON01] MONSON, Richard. **Enterprise JavaBeans**. Second Edition.O´Reilly. 2000.

[NET01] NetScape. Java Script Manuals. Disponível em: http://developer.netscape.com/docs/manuals/javascript.htm

[ULL01] ULLMAN, Jeffrey – WIDOW, Jennifer. **A First Course In DataBase Systems**. Prentice Hall. 1997.

[SUN01] Java 2 Platform, Standard Edition (J2SE). Disponível em: http://java.sun.com/j2se/

[SUN02] Sun Microsystems. Java 2 Plataform Enterprise Edition Specification, v1.3. Final Release. August, 2001 Disponível em: http://java.sun.com/j2ee/download.html#platformspec

[SUN03] Sun Microsystems. Enterprise JavaBeans Specification, v2.0. Final Release. April, 2001 Disponível em: http://java.sun.com/products/ejb/docs.html#specs

[SUN04] Java 2 Platform, Enterprise Edition (J2EE). Disponível em: <http://java.sun.com/j2ee/>

[SUN05] Java Technology and XML. Disponível em: <http://java.sun.com/xml/>

[SUN06] Java Enterprise BluePrints. Disponível em: <http://java.sun.com/j2ee/download.html#blueprints/>

[SUN07] CORBA Technology and Java Plataforma. Disponível em: http://java.sun.com/j2ee/corba/

[SUN08] ECperf. Disponível em: http://java.sun.com/j2ee/ecperf/

[SUN09] Enterprise JavaBeans Technology. Disponível em: http://java.sun.com/products/ejb/

[SUN10] Java Authorization Contract For Containers. Disponível em: http://java.sun.com/j2ee/javaacc/

[SUN11] Java IDL. Disponível em: http://java.sun.com/products/jdk/idl/

[SUN12] JavaMail API. Disponível em: http://java.sun.com/products/javamail/

[SUN13] Java Message Service API. Disponível em: http://java.sun.com/products/jms/

[SUN14] JavaServer Faces Technolgy. Disponível em: http://java.sun.com/j2ee/javaserverfaces/

[SUN15] JavaServer Pages (JSP). Disponível em: http://java.sun.com/products/jsp/

[SUN16] Java Servlet. Disponível em: http://java.sun.com/products/servlet/

[SUN17] JDBC technology. Disponível em: http://java.sun.com/products/jdbc/

[SUN18] J2EE Connector Architecture. Disponível em: http://java.sun.com/j2ee/connector/

[SUN19] Transactions. Disponível em: http://java.sun.com/j2ee/transactions.html

[SUN20] Runnin' With the XML pack. Disponível em: http://java.sun.com/features/2002/06/wspack.html

[SUN21] The Java Web Services Tutorial. Disponível em: http://java.sun.com/webservices/docs/1.0/tutorial/index.html

[SUN22] Remote Method Invocation. Disponível em : http://java.sun.com/products/jdk/rmi/

[TSS01] The Server Side. Disponível em: /www.theserverside.com

[VIJ01] VIJAY, Mukhi. **Java Servlets JSP**. Makron. 2001

[W3C01] World Wide Web Consortium (W3C). Disponível em: http://www.w3.org/

[XML01] The XML Industry Portal. Disponível em: http://www.xml.org/

11. Anexos

Anexo I – CD-ROM

- Projeto da aplicação portal de vestibular;
- Código fonte da aplicação portal de vestibular;
- Projeto do EasyNet;
- Código fonte do EasyNet
    - Gerador de Entity Beans;
    - Gerenciador de Projetos;
    - Processador de Transações;
    - API
- Código fonte da aplicação portal de vestibular, re-desenvolvida no EasyNet.

Anexo II – Exemplos dos elementos do Entity Beans Gerados pelo EasyNet.

Classe abstrata gerada – Curso

```
package br.com.vestibular.curso;

import java.util.*;
import javax.ejb.*;
import java.rmi.RemoteException;
import javax.naming.*;

import br.com.vestibular.instituicao.Instituicao;
import br.com.vestibular.vestibularCurso.VestibularCurso;
import br.com.vestibular.vestibularCurso.VestibularCursoHome;
import br.com.vestibular.vestibularCurso.VestibularCursoPK;

public abstract class CursoBean implements EntityBean {

  private EntityContext ctx;
  public CursoBean () {}

  public abstract void setCodigo (String value);
  public abstract java.lang.String getCodigo ();

  public abstract void setCodigoInstituicao (java.lang.String value);
  public abstract java.lang.String getCodigoInstituicao ();

  public abstract void setDescricao (java.lang.String value);
  public abstract java.lang.String getDescricao ();

  public abstract void setInstituicao (Instituicao value);
  public abstract Instituicao getInstituicao ();

  public abstract void setVestibularCurso (Set value);
  public abstract Set getVestibularCurso ();

  public Instituicao getFromRelationInstituicao()
    throws FinderException {
    return getInstituicao();
   }

  public VestibularCurso createVestibularCurso(java.lang.String codigoVestibular,
                                               java.lang.String codigoCurso)
    throws Exception {
    try {
      Context ctx = new InitialContext();
      VestibularCursoHome ejbHome =
        (VestibularCursoHome) ctx.lookup("vestibularEasyNet.VestibularCursoEJB");
      VestibularCurso ejb = ejbHome.create(codigoVestibular,codigoCurso);
      getVestibularCurso().add(ejb);
      return ejb;
    } catch (Exception e) {
      throw new Exception(e.getMessage());
    }
  }

  public Set getFromRelationShipVestibularCurso () {
    HashSet hs = new HashSet();
    Iterator i = getVestibularCurso().iterator();
    while (i.hasNext()) {
      hs.add(i.next());
    }
    return (Set) hs;
  }

   public void removeVestibularCurso (java.lang.String codigoVestibular,
                                      java.lang.String codigoCurso)
     throws Exception {
```

```
     try {
       Context ctx = new InitialContext();
       VestibularCursoHome ejbHome =
        (VestibularCursoHome) ctx.lookup("vestibularEasyNet.VestibularCursoEJB");
       VestibularCursoPK pk = new VestibularCursoPK(codigoVestibular,codigoCurso);
       VestibularCurso ejb = ejbHome.findByPrimaryKey(pk);
       ejb.remove();
     } catch (Exception e) {
       throw new Exception(e.getMessage());
     }
   }

   public String toXml() {
     StringBuffer sb = new StringBuffer();
     sb.append("<Curso>\n");
     sb.append("\t<codigo>" + getCodigo() + "</codigo>\n");
     sb.append("\t<codigoInstituicao>" + getCodigoInstituicao() +
"</codigoInstituicao>\n");
     sb.append("\t<descricao>" + getDescricao() + "</descricao>\n");
     sb.append("</Curso>\n");
     return sb.toString();
   }

    public CursoPK ejbCreate (java.lang.String codigo,
                              java.lang.String descricao)
      throws CreateException {
      setCodigo(codigo);
      setDescricao(descricao);
      return null;
    }

    public void ejbPostCreate (java.lang.String codigo,
                               java.lang.String descricao)
      throws CreateException {}

    public void setEntityContext (EntityContext ctx) {
      this.ctx = ctx;
    }

    public void unsetEntityContext() {
      this.ctx = null;
    }
    public void ejbLoad() { }
    public void ejbStore() { }
    public void ejbActivate() { }
    public void ejbPassivate() { }
    public void ejbRemove() { }
}
```

Interface remota - Curso

```
package br.com.vestibular.curso;

import java.util.*;
import javax.ejb.*;
import java.rmi.RemoteException;
import br.com.vestibular.instituicao.Instituicao;
import br.com.vestibular.vestibularCurso.VestibularCurso;

public interface Curso extends EJBObject {

  public abstract void setCodigo (java.lang.String value) throws RemoteException;
  public abstract java.lang.String getCodigo () throws RemoteException;

  public abstract void setCodigoInstituicao (java.lang.String value)
    throws RemoteException;
  public abstract java.lang.String getCodigoInstituicao () throws RemoteException;
```

```
  public abstract void setDescricao (java.lang.String value) throws
RemoteException;
  public abstract java.lang.String getDescricao () throws RemoteException;

  public Instituicao getFromRelationInstituicao() throws FinderException,
RemoteException;

  public VestibularCurso createVestibularCurso(java.lang.String codigoVestibular,
                                                java.lang.String codigoCurso)
    throws Exception, RemoteException;

  public Set getFromRelationShipVestibularCurso ()
     throws RemoteException;

  public void removeVestibularCurso (java.lang.String codigoVestibular,
                                     java.lang.String codigoCurso)
    throws Exception, RemoteException;

  public String toXml() throws RemoteException;
}
```

Interface Home - Instituicao

```
package br.com.vestibular.instituicao;

import java.util.*;
import java.rmi.RemoteException;
import javax.ejb.*;

public interface InstituicaoHome extends EJBHome {

  public Instituicao create (java.lang.String codigo,
                             java.lang.String cNPJ,
                             java.lang.String nome,
                             java.lang.String rua,
                             java.lang.String numero,
                             java.lang.String complemento,
                             java.lang.String bairro,
                             java.lang.String cEP,
                             java.lang.String cidade,
                             java.lang.String estado,
                             java.lang.String telefones,
                             java.lang.String email,
                             java.lang.String senha,
                             java.lang.String status)
     throws CreateException, RemoteException;

  public Instituicao findByPrimaryKey (InstituicaoPK pk)
    throws FinderException, RemoteException;

  public Collection findAll ()
    throws FinderException, RemoteException;

  public Instituicao findByCNPJ (java.lang.String value)
    throws FinderException, RemoteException;

  public Instituicao findByEmail (java.lang.String value)
    throws FinderException, RemoteException;
}
```

Classe Chave Primária - Opcoes

```
package br.com.vestibular.opcoes;

import java.io.Serializable;
```

```
public class OpcoesPK implements Serializable {

  public java.lang.String codigoVestibular;
  public java.lang.String codigoCurso;
  public java.lang.String codigoCandidato;
  public java.lang.String opcao;

  public OpcoesPK () {
  }

  public OpcoesPK (java.lang.String codigoVestibular,
                   java.lang.String codigoCurso,
                   java.lang.String codigoCandidato,
                   java.lang.String opcao) {
    this.codigoVestibular = codigoVestibular;
    this.codigoCurso = codigoCurso;
    this.codigoCandidato = codigoCandidato;
    this.opcao = opcao;
   }

  public boolean equals (Object obj) {
    boolean resp = true;
    if (obj instanceof OpcoesPK) {
      OpcoesPK pk = (OpcoesPK) obj;
      resp = resp && codigoVestibular.equals(pk.codigoVestibular);
      resp = resp && codigoCurso.equals(pk.codigoCurso);
      resp = resp && codigoCandidato.equals(pk.codigoCandidato);
      resp = resp && opcao.equals(pk.opcao);
    } else {
      resp = false;
    }
    return resp;
  }

  public int hashCode () {
    int resp = 0;
    resp = resp + codigoVestibular.hashCode();
    resp = resp + codigoCurso.hashCode();
    resp = resp + codigoCandidato.hashCode();
    resp = resp + opcao.hashCode();
    return resp;
  }
}
```

Arquivo ejb-jar.xml

```
<?xml version="1.0"?>
<!DOCTYPE ejb-jar PUBLIC '-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans
2.0//EN' 'http://java.sun.com/j2ee/dtds/ejb-jar_2_0.dtd'>
<ejb-jar>
   <display-name>VestibularEasyNetEJB</display-name>
   <enterprise-beans>
...
   <entity>
      <ejb-name>InstituicaoEJB</ejb-name>
      <home>br.com.vestibular.instituicao.InstituicaoHome</home>
      <remote>br.com.vestibular.instituicao.Instituicao</remote>
      <ejb-class>br.com.vestibular.instituicao.InstituicaoBean</ejb-class>
      <persistence-type>Container</persistence-type>
      <prim-key-class>br.com.vestibular.instituicao.InstituicaoPK</prim-key-class>
      <reentrant>False</reentrant>
      <cmp-version>2.x</cmp-version>
      <abstract-schema-name>InstituicaoEJB</abstract-schema-name>
      <cmp-field>
        <field-name>codigo</field-name>
      </cmp-field>
```

```
      <cmp-field>
        <field-name>cNPJ</field-name>
      </cmp-field>
      <cmp-field>
        <field-name>nome</field-name>
      </cmp-field>
...
      <query>
        <query-method>
          <method-name>findAll</method-name>
          <method-params/>
        </query-method>
        <ejb-ql>
        <![CDATA[SELECT OBJECT(o) FROM InstituicaoEJB AS o]]>
        </ejb-ql>
      </query>
      <query>
        <query-method>
          <method-name>findByCNPJ</method-name>
          <method-params>
              <method-param>java.lang.String</method-param>
          </method-params>
        </query-method>
        <ejb-ql>
        <![CDATA[SELECT OBJECT(o) FROM InstituicaoEJB AS o WHERE o.cNPJ = ?1 ]]>
        </ejb-ql>
      </query>
      <query>
        <query-method>
          <method-name>findByEmail</method-name>
          <method-params>
              <method-param>java.lang.String</method-param>
          </method-params>
        </query-method>
        <ejb-ql>
        <![CDATA[SELECT OBJECT(o) FROM InstituicaoEJB AS o WHERE o.email = ?1 ]]>
        </ejb-ql>
      </query>
      </entity>
...
   </enterprise-beans>
   <relationships>
      <ejb-relation>
        <ejb-relation-name>Candidato-Opcoes</ejb-relation-name>
        <ejb-relationship-role>
          <ejb-relationship-role-name>candidato</ejb-relationship-role-name>
          <multiplicity>one</multiplicity>
          <relationship-role-source>
              <ejb-name>CandidatoEJB</ejb-name>
          </relationship-role-source>
          <cmr-field>
              <cmr-field-name>opcoes</cmr-field-name>
              <cmr-field-type>java.util.Set</cmr-field-type>
          </cmr-field>
        </ejb-relationship-role>
        <ejb-relationship-role>
          <ejb-relationship-role-name>opcoes</ejb-relationship-role-name>
          <multiplicity>many</multiplicity>
          <relationship-role-source>
              <ejb-name>OpcoesEJB</ejb-name>
          </relationship-role-source>
          <cmr-field>
              <cmr-field-name>candidato</cmr-field-name>
          </cmr-field>
        </ejb-relationship-role>
      </ejb-relation>
...
   </relationships>
```

```
   <assembly-descriptor>
      <container-transaction>
         <method>
            <ejb-name>CandidatoEJB</ejb-name>
            <method-name>*</method-name>
         </method>
         <trans-attribute>Required</trans-attribute>
      </container-transaction>
...
   </assembly-descriptor>
</ejb-jar>
```

Arquivo weblogic-ejb-jar.xml

```
<?xml version="1.0"?>
<!DOCTYPE weblogic-ejb-jar PUBLIC '-//BEA Systems, Inc.//DTD WebLogic 6.0.0
EJB//EN' 'http://www.bea.com/servers/wls600/dtd/weblogic-ejb-jar.dtd'>
<weblogic-ejb-jar>

<weblogic-enterprise-bean>
   <ejb-name>CandidatoEJB</ejb-name>
   <entity-descriptor>
      <entity-cache>
         <max-beans-in-cache>20</max-beans-in-cache>
         <read-timeout-seconds>600</read-timeout-seconds>
         <concurrency-strategy>Database</concurrency-strategy>
      </entity-cache>
      <lifecycle>
         <passivation-strategy>Default</passivation-strategy>
      </lifecycle>
      <persistence>
         <persistence-type>
            <type-identifier>WebLogic_CMP_RDBMS</type-identifier>
            <type-version>6.0</type-version>
            <type-storage>META-INF/weblogic-cmp-rdbms-jar.xml</type-storage>
         </persistence-type>
            <db-is-shared>True</db-is-shared>
         <persistence-use>
            <type-identifier>WebLogic_CMP_RDBMS</type-identifier>
            <type-version>6.0</type-version>
         </persistence-use>
      </persistence>
   </entity-descriptor>
   <enable-call-by-reference>True</enable-call-by-reference>
   <jndi-name>vestibularEasyNet.CandidatoEJB</jndi-name>
</weblogic-enterprise-bean>
...
</weblogic-ejb-jar>
```

Arquivo weblogic-cmp-rdbms-jar

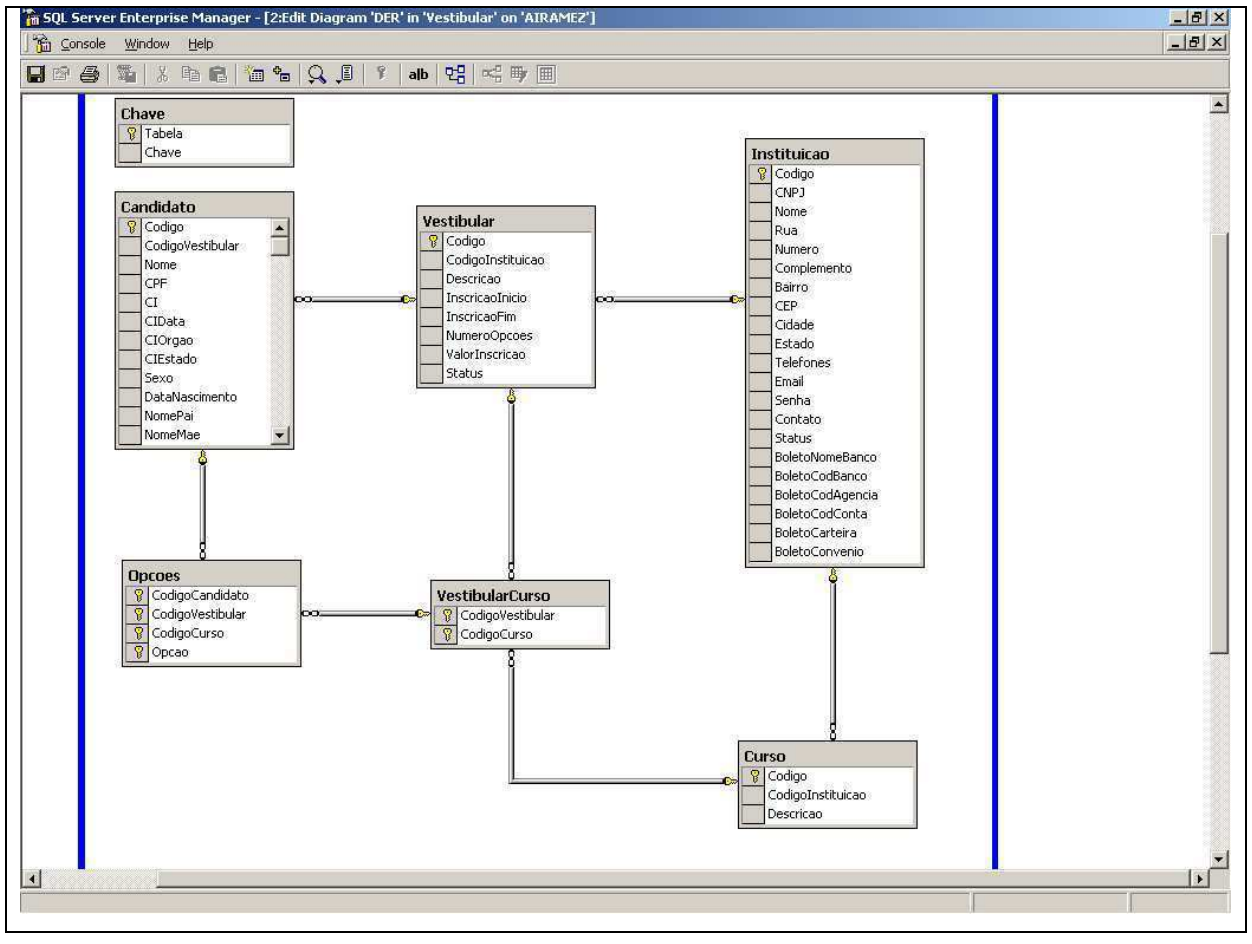
```
<?xml version="1.0"?>
<!DOCTYPE weblogic-rdbms-jar PUBLIC '-//BEA Systems, Inc.//DTD WebLogic 6.0.0 EJB
RDBMS Persistence//EN' 'http://www.bea.com/servers/wls600/dtd/weblogic-rdbms20-
persistence-600.dtd'>
<weblogic-rdbms-jar>
...
   <weblogic-rdbms-bean>
      <ejb-name>InstituicaoEJB</ejb-name>
      <data-source-name>vestibular.DataSource</data-source-name>
      <table-name>Instituicao</table-name>
      <field-map>
         <cmp-field>codigo</cmp-field>
         <dbms-column>codigo</dbms-column>
      </field-map>
      <field-map>
```

```
        <cmp-field>cNPJ</cmp-field>
        <dbms-column>cNPJ</dbms-column>
      </field-map>
      <field-map>
        <cmp-field>nome</cmp-field>
        <dbms-column>nome</dbms-column>
      </field-map>
   <weblogic-rdbms-bean>
...
   <weblogic-rdbms-relation>
      <relation-name>VestibularCurso-Opcoes</relation-name>
      <weblogic-relationship-role>
        <relationship-role-name>opcoes</relationship-role-name>
        <column-map>
           <foreign-key-column>codigoVestibular</foreign-key-column>
           <key-column>codigoVestibular</key-column>
        </column-map>
        <column-map>
           <foreign-key-column>codigoCurso</foreign-key-column>
           <key-column>codigoCurso</key-column>
        </column-map>
      </weblogic-relationship-role>
   </weblogic-rdbms-relation>
</weblogic-rdbms-jar>
```

Anexo III – Projeto da aplicação Portal de Vestibulares.

A aplicação Portal de Vestibulares implementadas em duas situações durante a dissertação seguiu o seguinte projeto de banco de dados e interface.

Projeto do Banco de Dados



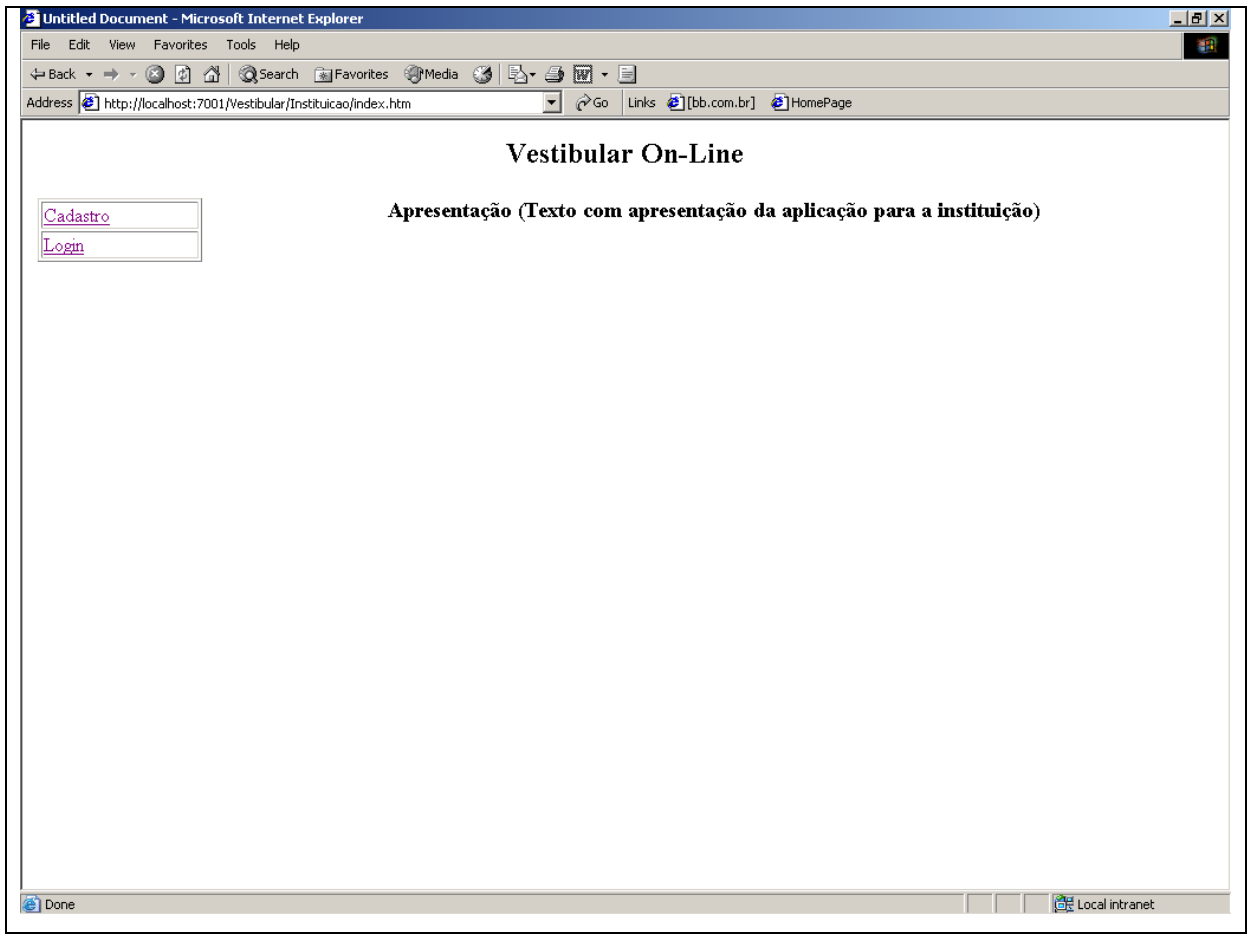
Projeto de interface

Aplicações web devem possuir interfaces simples e leves, desta forma o projeto de interface demonstrado a seguir tem como principal objetivo leveza e simplicidade, sendo que o mesmo possui apenas os requisitos funcionais não contendo os detalhes típicos de webdesign, tais como formatação, cores, textos explicativos, figuras, etc. Com o objetivo de aumentar a capacidade de escala, os formulários de entrada de dados possuirão rotinas em JavaScript para efetuar a validação dos dados, evitando que dados inconsistentes sejam enviados ao servidor, acarretando em desperdício de processamento, assim a validação dos dados será distribuída entre os clientes (navegador), através de rotinas JavaScript.

Projeto de interface da Visão da Instituição

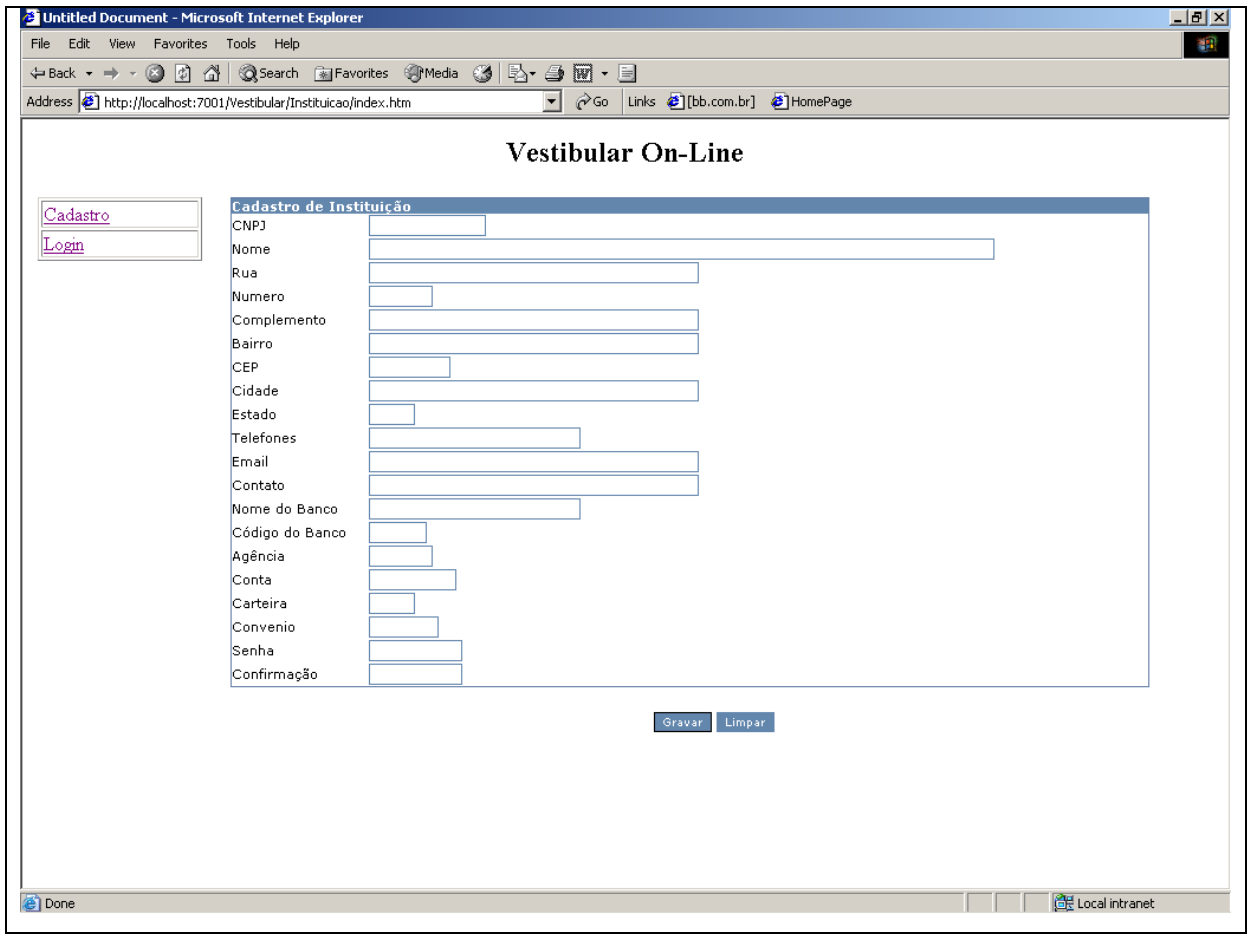
Tela Inicial

A primeira tela da aplicação irá exibir uma mensagem de propaganda e boas vindas à instituição. E terá as opções: [**Cadastro**] e [**Login**]. Para que novas instituições realizem seus cadastro e instituições já cadastradas possam acessar a aplicação.



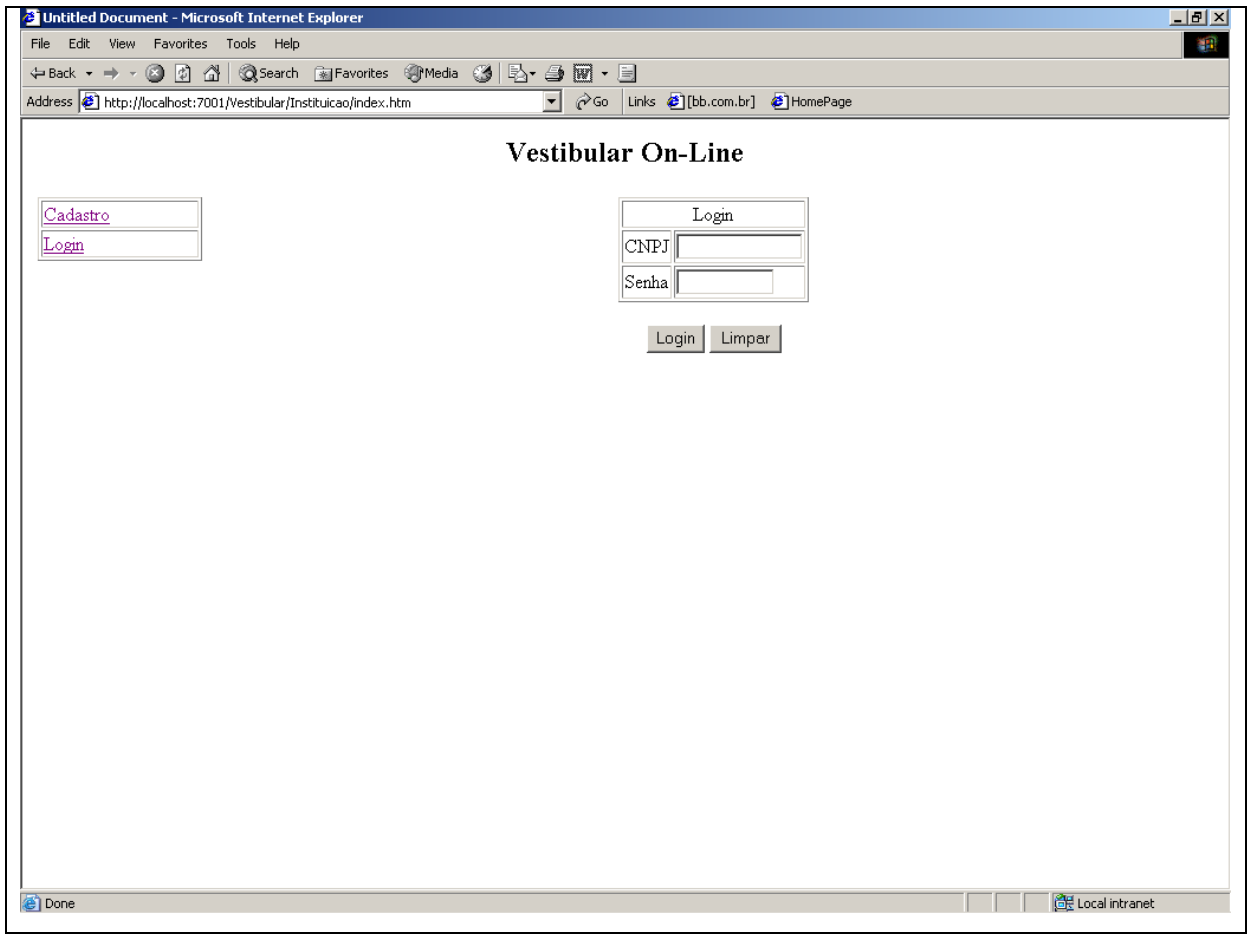
Tela de Cadastro

Formulário de cadastro para a instituição.



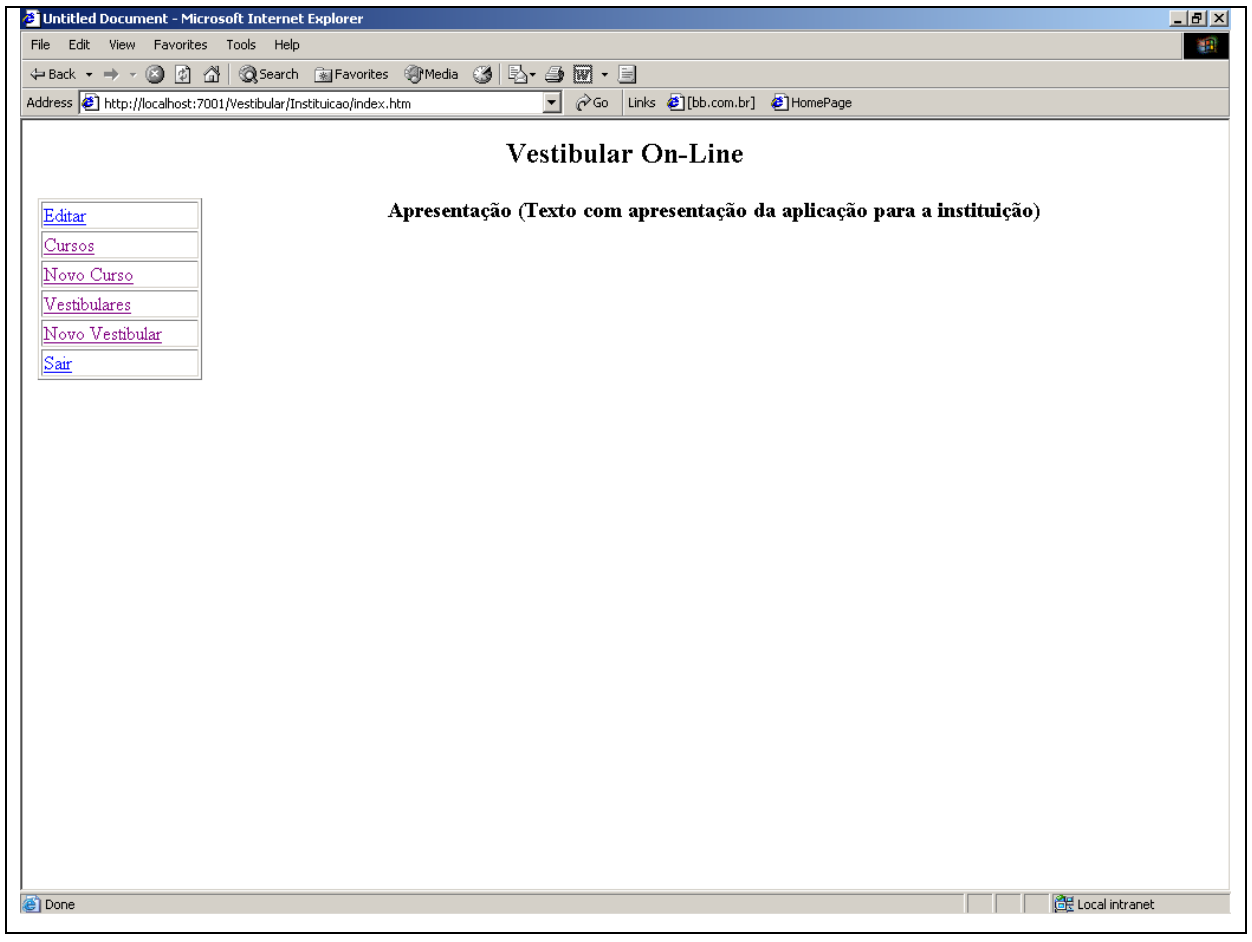
Tela de Login

O login da instituição será o seu CNPJ e a senha definida no cadastro.



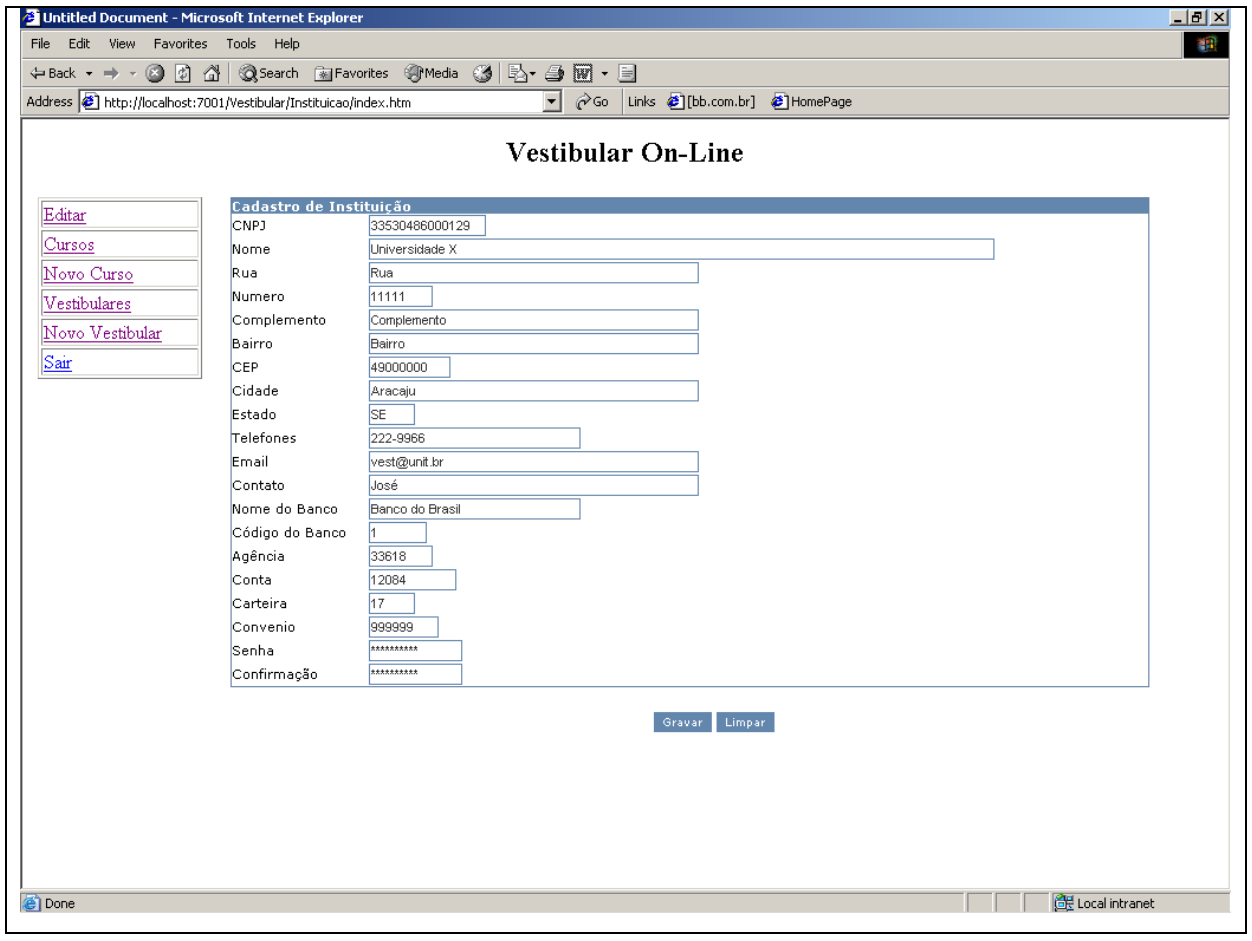
Tela da Instituição após o login

Nesta tela serão exibidos uma mensagem inicial da aplicação e um menu com as opções para a instituição, as quais permitirão: Editar seus dados cadastrais [**Editar**], cadastrar e visualizar seus cursos [**NovoCurso** e **Cursos**], cadastrar e visualizar seus vestibulares [**NovoVestibular** e **Vestibulares**] e encerrar a sessão [**Sair**]



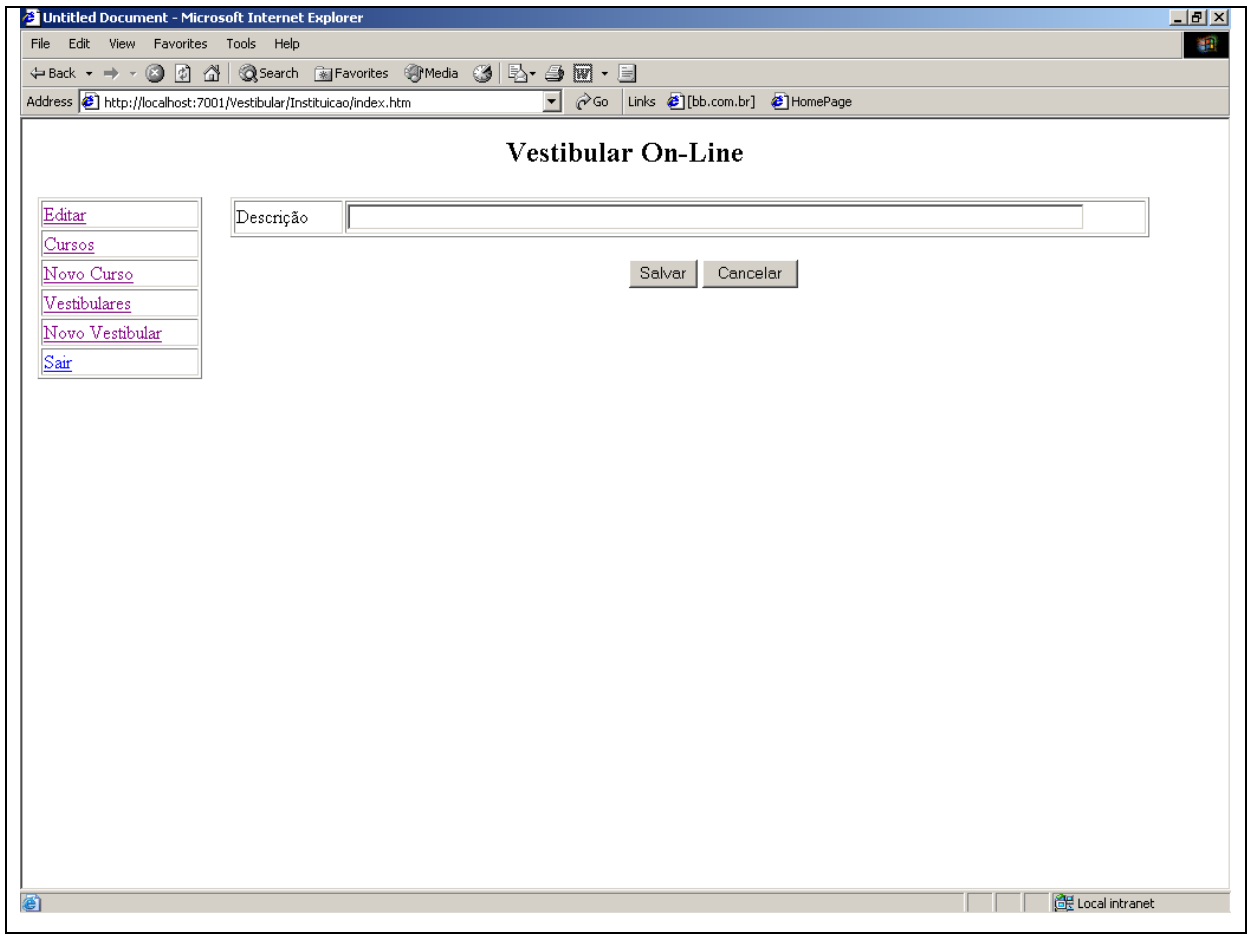
Editar

Permite que a instituição altere seus dados cadastrais.



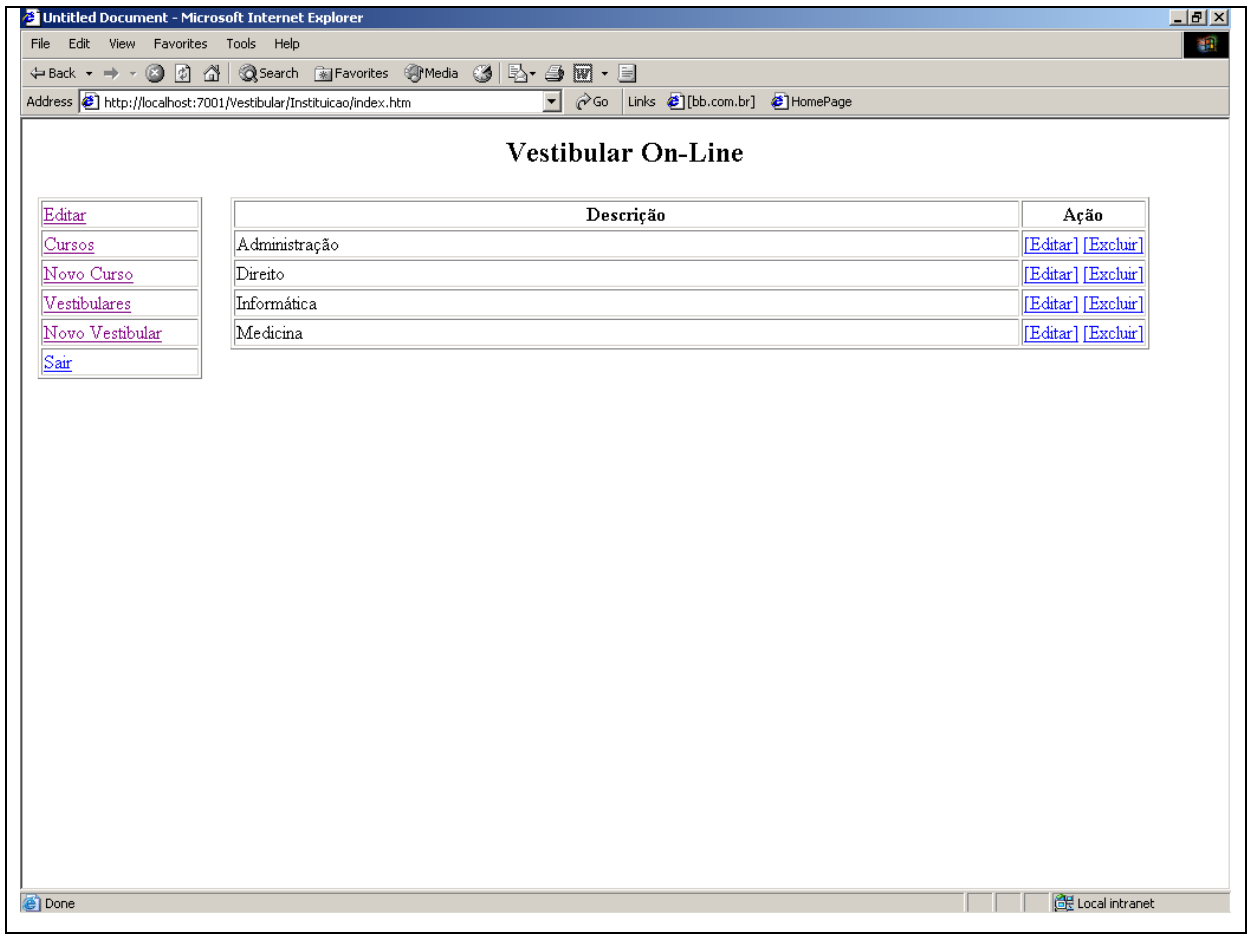
Novo Curso

Inclusão de novo curso.



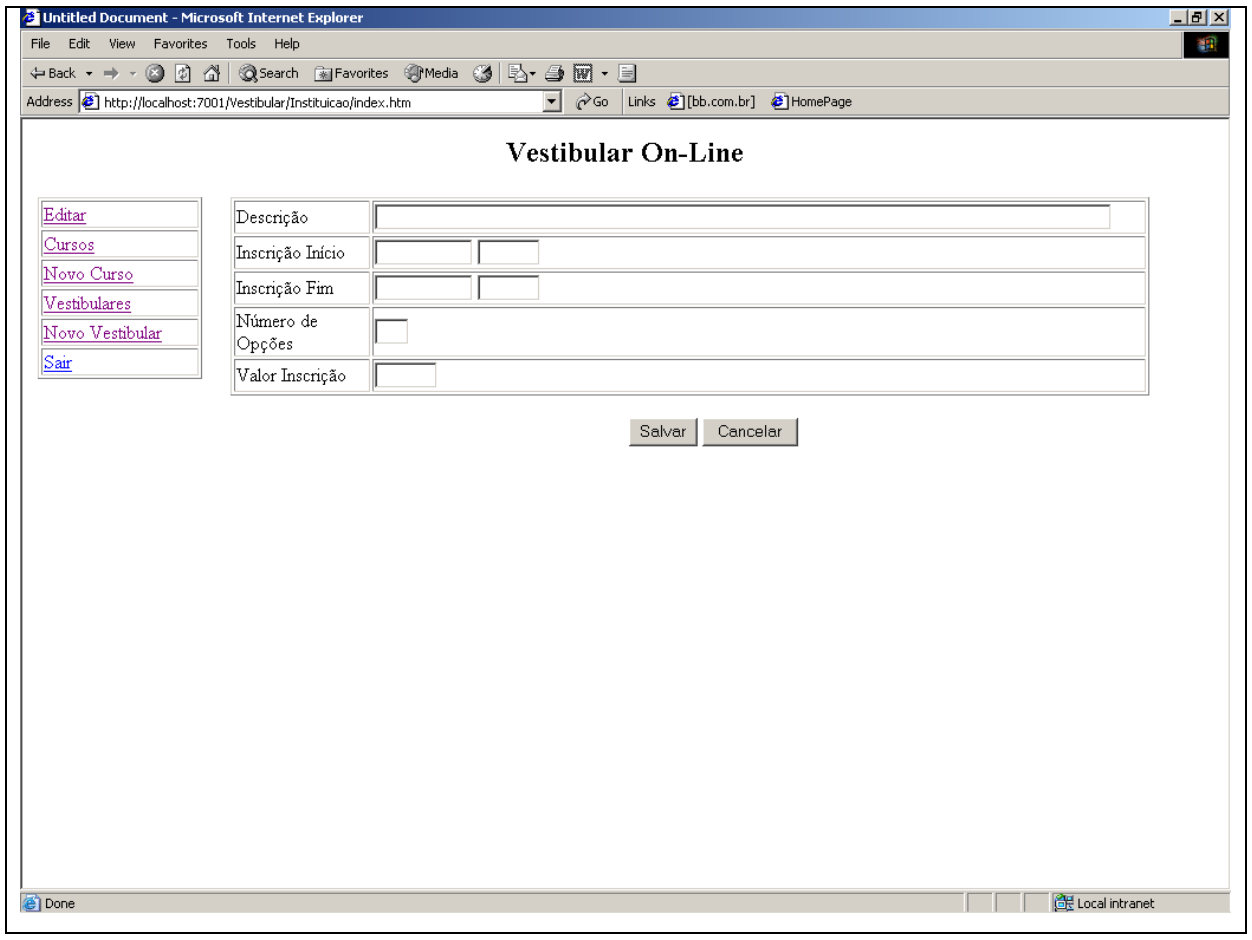
Cursos

Relação de cursos da instituição com as opções de editar e excluir.



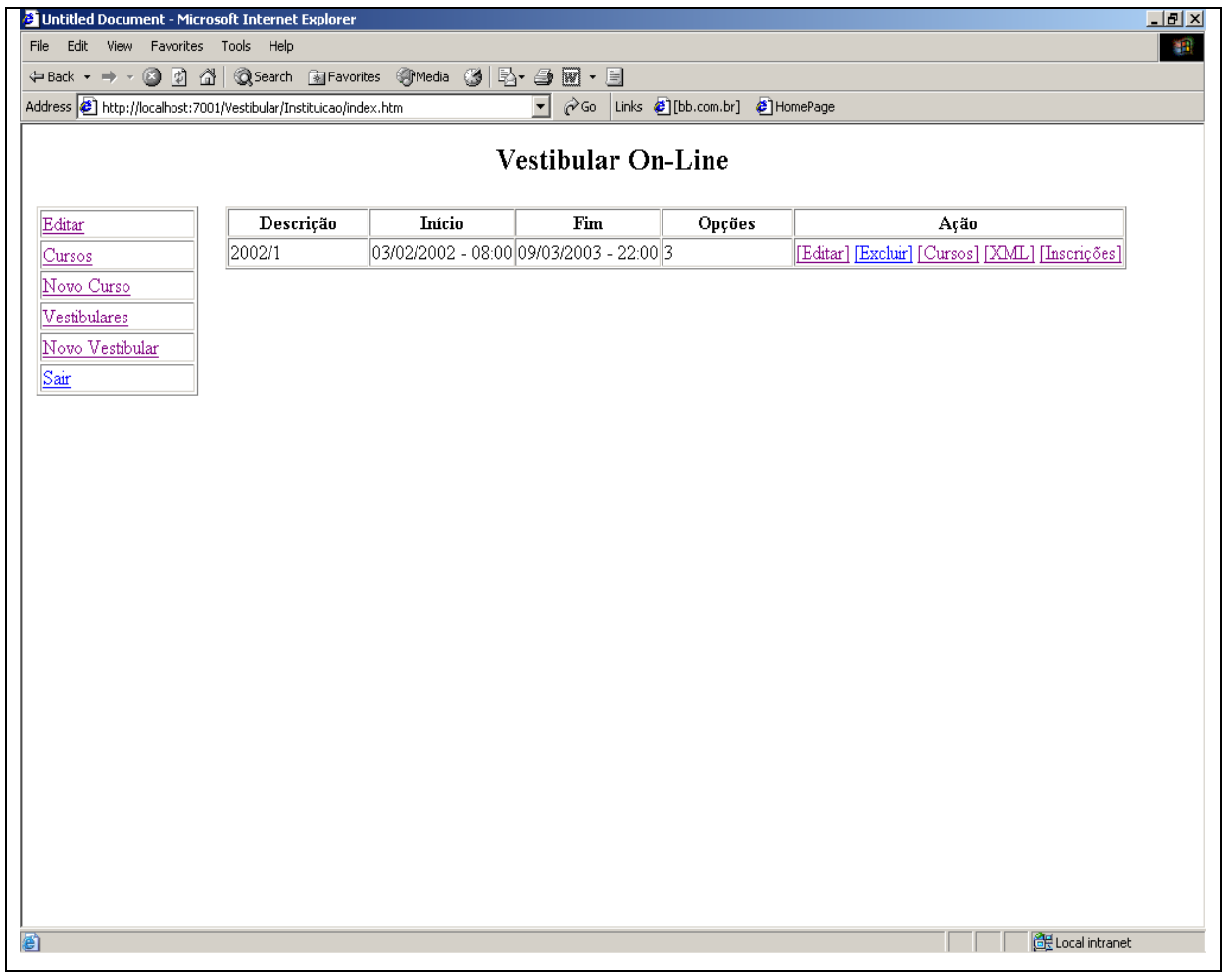
Novo Vestibular

Inclusão de novo vestibular



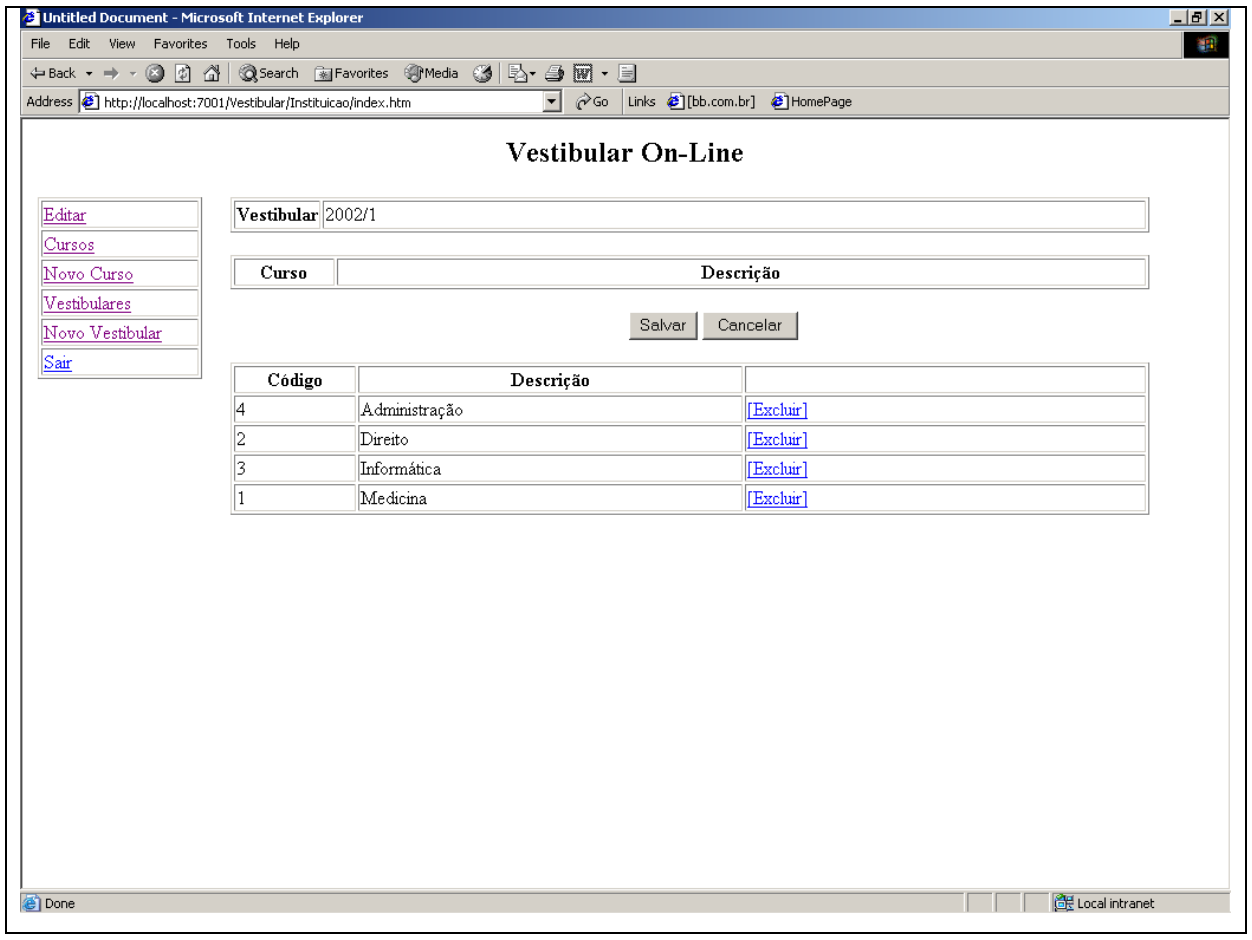
Vestibulares

Relação de vestibulares cadastros pela instituição. Com a opção de **Editar**, **Excluir**, associar aos **Cursos**, gerar arquivo **XML** com as inscrições realizadas e ver estatística de **Inscrições**.



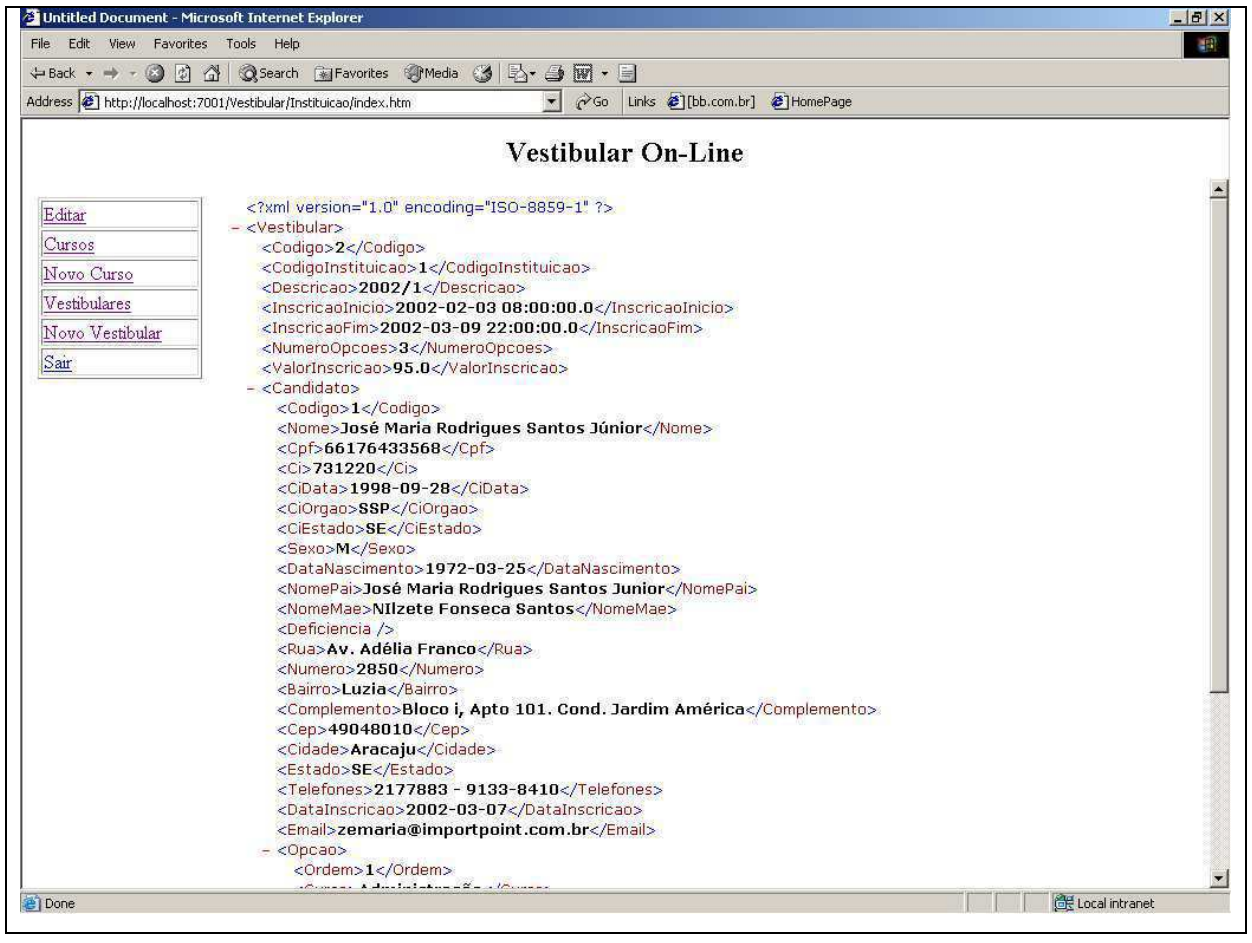
Associação de cursos ao vestibular

Permite a inclusão e remoção de cursos a serem ofertados no referido vestibular.



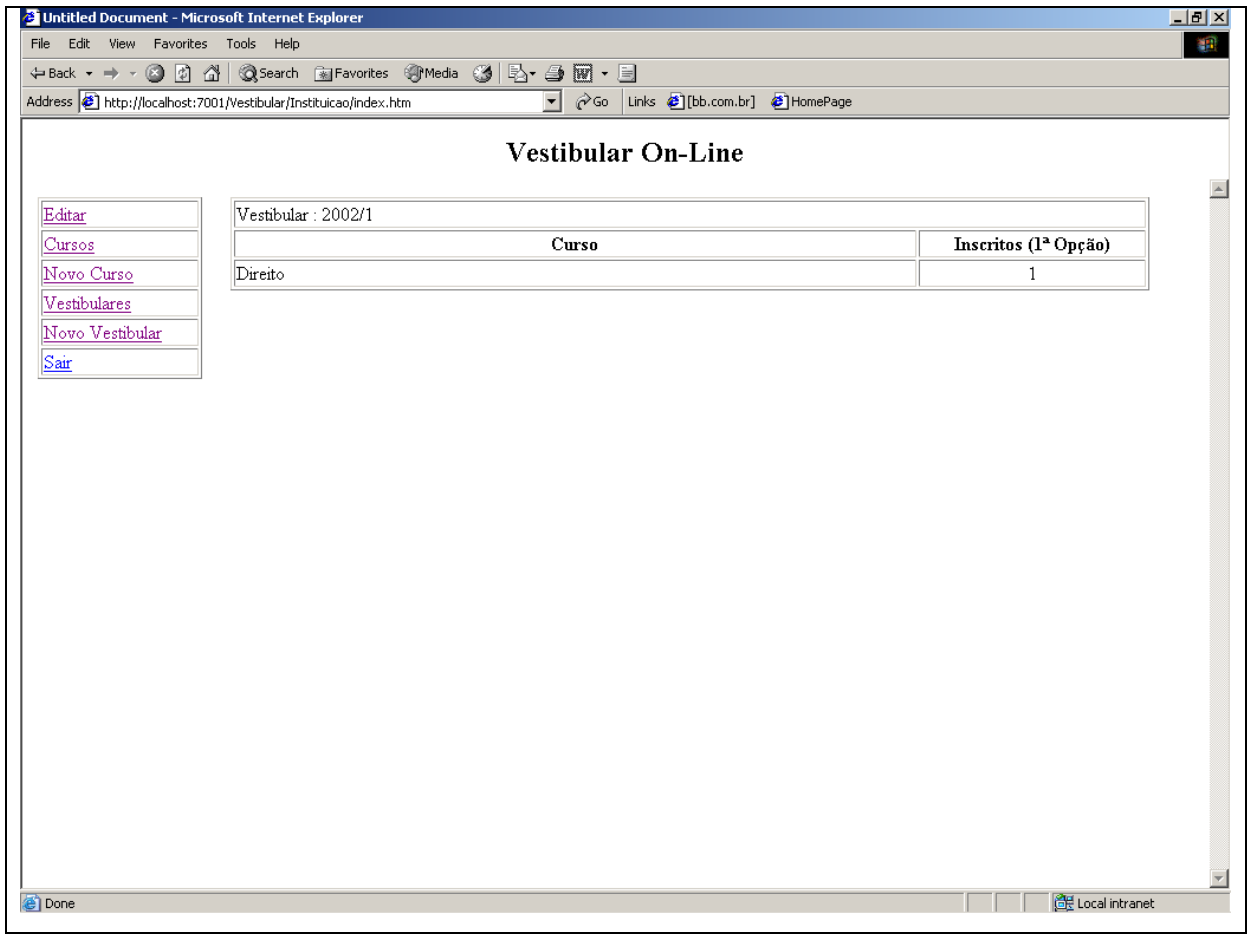
Arquivo XML gerado pela aplicação com as inscrições realizadas.

Arquivo XML contendo todas as inscrições realizadas no referido vestibular da instituição, com o objetivo de facilitar a integração com sistemas legados.



Estatística das inscrições

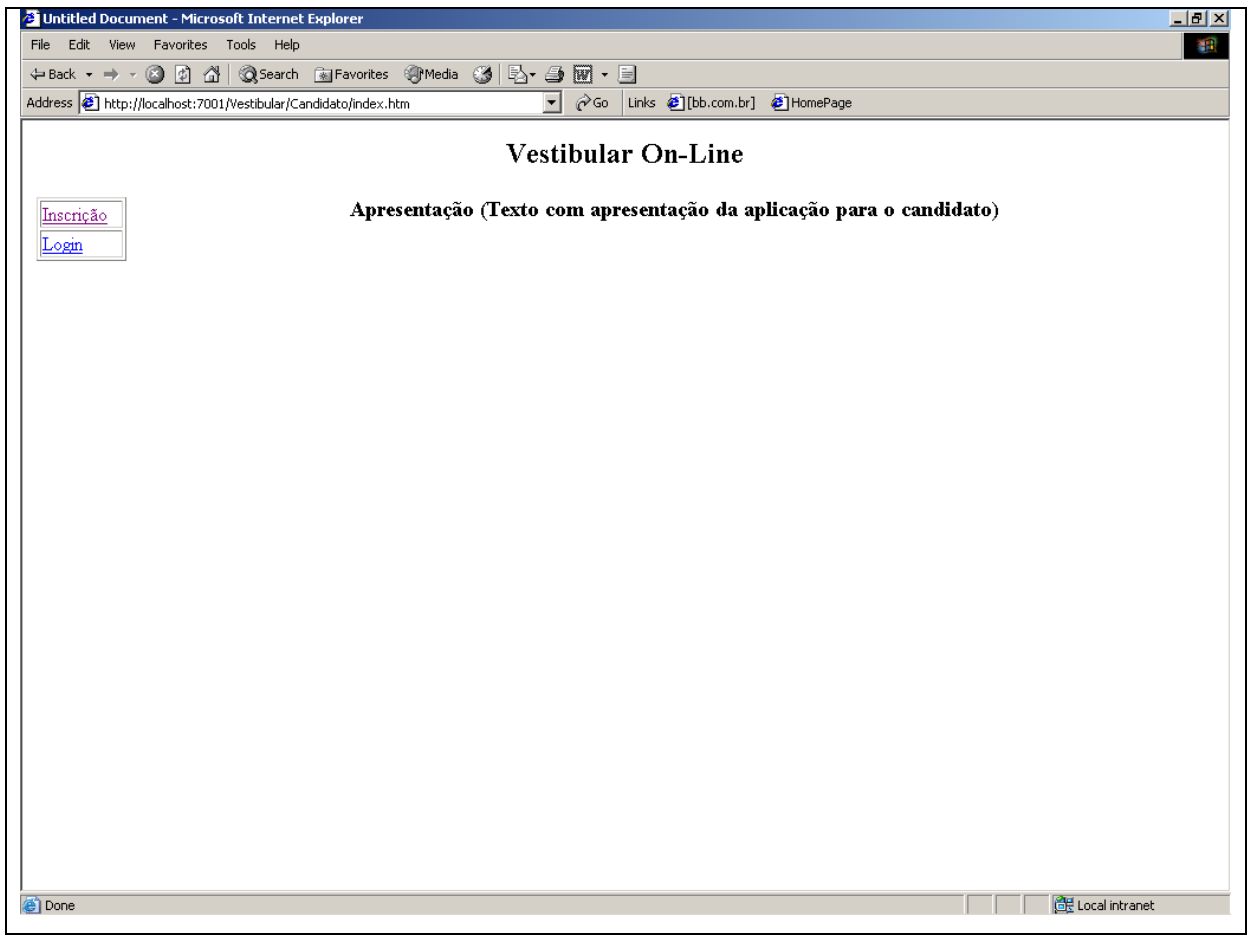
Relatório contendo uma relação de cursos, nos quais foram realizadas inscrições e a quantidade de inscritos na primeira opção.



Projeto de interface da Visão do Candidato

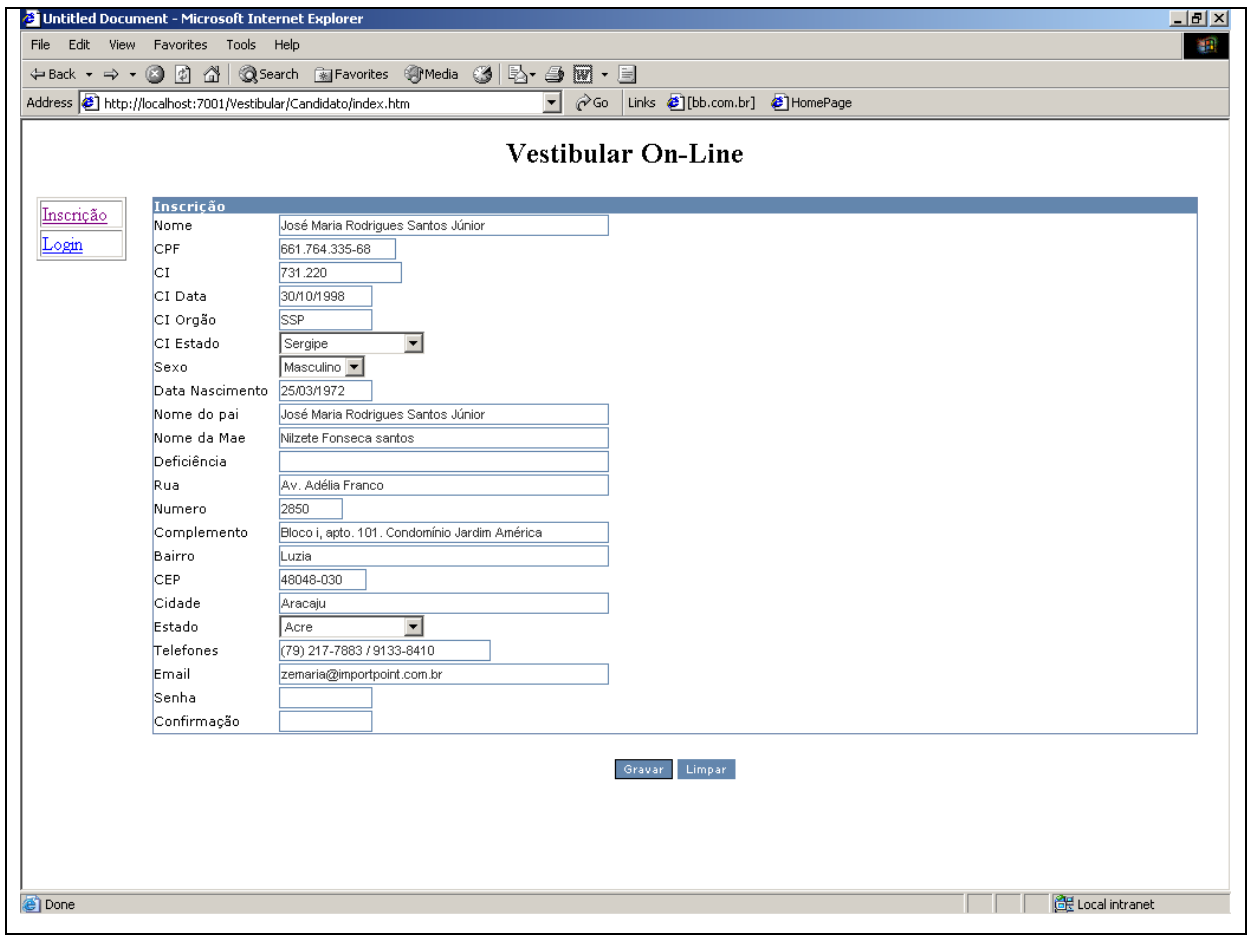
Tela inicial

Apresentação da aplicação ao candidato, com as opções para realizar inscrição [**Inscrição**] e [**Login**].



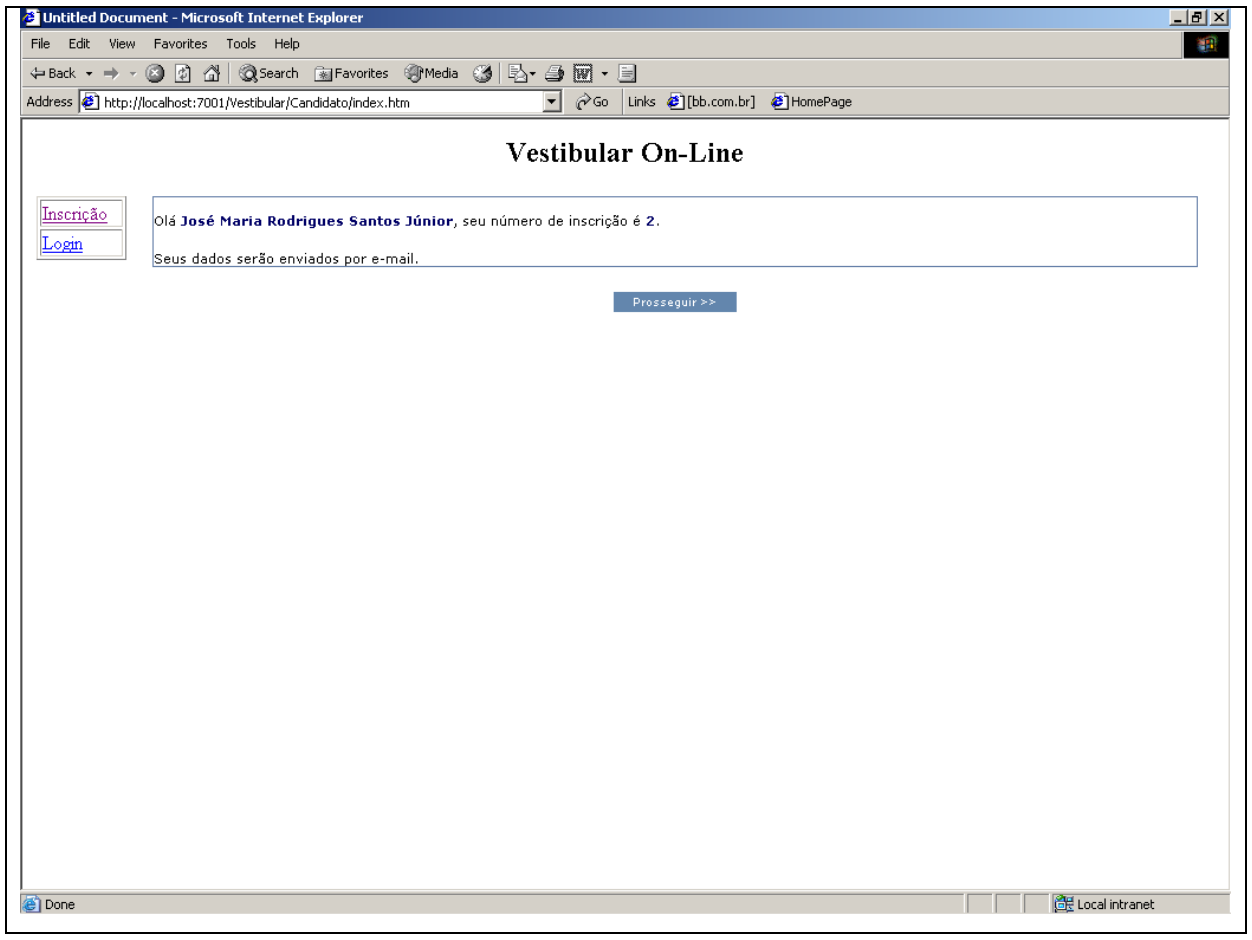
Inscrição

Ficha de inscrição do candidato.



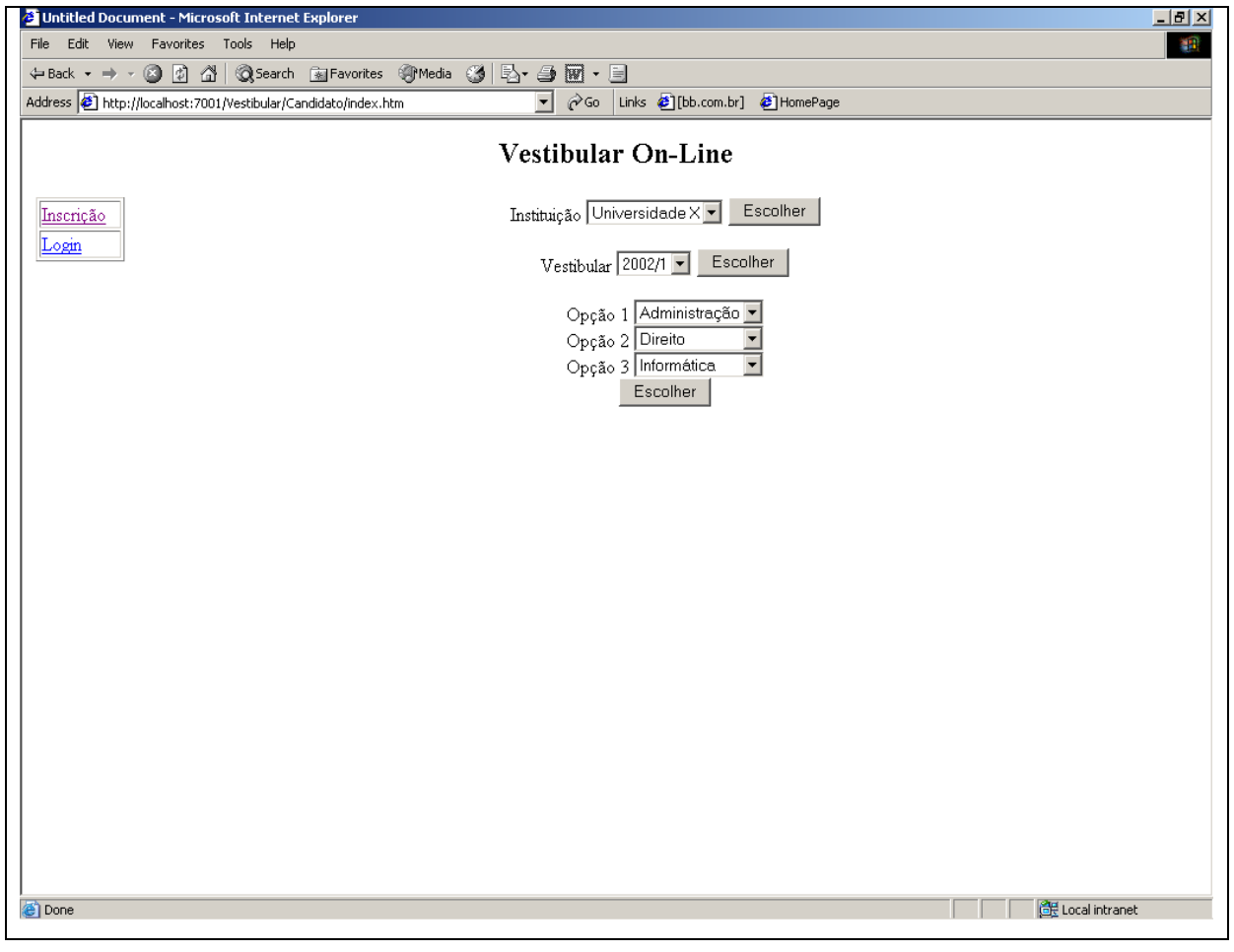
Confirmação do cadastro e início da inscrição

Será exibida uma mensagem de boas vindas, exibindo o número de inscrição do candidato, o qual será utilizado para realizar o login, assim como um aviso sobre o envio de um email de confirmação com os dados para um posterior login.



Tela para escolha da instituição, vestibular e cursos

O candidato irá selecionar a instituição na qual deseja realizar a sua inscrição, em seguida serão exibidos os vestibulares com inscrições abertas para essa instituição e de acordo com a quantidade de opções definida no respectivo vestibular será exibida uma lista para o candidato escolher as opções de cursos que o mesmo deseja se inscrever, sendo que , não serão permitidos cursos repetidos.



Boleto bancário para pagamento da inscrição

Boleto bancário gerado com as informações do convênio entre a instituição e o banco.