UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
DEPARTAMENTO DE ELETRÔNICA E COMPUTAÇÃO
CURSO DE CIÊNCIA DA COMPUTAÇÃO

GERAÇÃO DE OPERAÇÕES CRUD A PARTIR DE METADADOS

TRABALHO FINAL DE GRADUAÇÃO

Felipe Gabriel Arend

Santa Maria, RS, Brasil
2011

GERAÇÃO DE OPERAÇÕES CRUD A PARTIR DE METADADOS

Felipe Gabriel Arend

Monografia apresentada ao Curso de Ciência da Computação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Bacharel em Ciência da Computação.**

Orientador: Prof. Dr. Eduardo Kessler Piveta

Trabalho de Graduação Nro. 328
Santa Maria, RS, Brasil
2011

Universidade Federal de Santa Maria
Centro de Tecnologia
Departamento de Eletrônica e Computação
Curso de Ciência da Computação

A Comissão Examinadora, abaixo assinada,
aprova o Trabalho de Graduação

GERAÇÃO DE OPERAÇÕES CRUD A PARTIR DE METADADOS

elaborado por
Felipe Gabriel Arend

como requisito parcial para obtenção do grau de
Bacharel em Ciência da Computação

COMISSÃO EXAMINADORA:

_____
Prof. Dr. Eduardo Kessler Piveta
(Presidente/Orientador)

_____
Prof. Dr. Osmar Marchi dos Santos

_____
Profa. Dra. Lisandra Manzoni Fontoura

Santa Maria, 20 de dezembro de 2011.

RESUMO

Trabalho de Graduação
Curso de Ciência da Computação
Universidade Federal de Santa Maria

GERAÇÃO DE OPERAÇÕES CRUD A PARTIR DE METADADOS

Autor: Felipe Gabriel Arend
Orientador: Eduardo Kessler Piveta
Data e Local da Defesa: Santa Maria, 20 de dezembro de 2011

No desenvolvimento de aplicações para Web geralmente faz-se necessária a criação de páginas que permitam a realização de operações de inserção, atualização, remoção e visualização de dados para diversas entidades presentes no sistema. Esse processo de criação acaba tornando-se repetitivo e, desta maneira, propenso a erros.

Nesse contexto, e assumindo o desenvolvimento de sistemas com JavaServer Faces, a geração automática desse código facilita o desenvolvimento das aplicações, além de permitir um ganho de tempo na realização das tarefas.

Este trabalho apresenta a proposta de uma ferramenta para geração de código para operações de inserção, consulta, atualização e exclusão de dados a partir de metadados de uma base de dados informada em sistemas Web utilizando JavaServer Faces.

Palavras-chave: geração de código, operações CRUD, desenvolvimento Web, JavaServer Faces, metadados, banco de dados.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

AJAX	*Javascript* e XML Assíncronos (*Asynchronous Javascript and XML*)
API	Interface de Programação de Aplicativos (*Application Programming Interface*)
BD	Banco de Dados
BDR	Banco de Dados Relacional
CRUD	Criar, Obter, Atualizar e Apagar (*Create, Retrieve, Update and Delete*)
CSS	Folhas de Estilo em Cascata (*Cascade Style Sheet*)
HTML	Linguagem de Marcação de Hipertexto (*HyperText Markup Language*)
IDE	Ambiente de Desenvolvimento Integrado (*Integrated Development Environment*)
JDBC	*Java DataBase Connectivity*
JPA	*Java Persistence API*
JSF	*Java Server Faces*
MVC	Modelo-Visualização-Controle (*Model-View-Controller*)
ORM	Mapeamento Objeto-Relacional (*Object-Relational Mapping*)
POJO	Objetos Java Simples e Antigos (*Plain Old Java Objects*)
POO	Programação Orientada a Objetos
SGBD	Sistema de Gerenciamento de Banco de Dados
URL	Localizador Padrão de Recursos (*Uniform Resource Locator*)
VTL	*Velocity Template Language*
W3C	*World Wide Web Consortium*
XHTML	Linguagem de Marcação de Hipertexto Extensível (*eXtensible HyperText Markup Language*)
XML	Linguagem de Marcação Extensível (*Extensible Markup Language*)

SUMÁRIO

1 INTRODUÇÃO

A facilidade de acesso à internet tem aumentado o número de usuários e os serviços disponíveis na rede mundial de computadores. Desta maneira, cresce a disponibilização e a evolução de sistemas de software para a Web, tanto para uso na Internet como na intranet.

No desenvolvimento de aplicativos de software, geralmente existe uma preocupação com os prazos a serem cumpridos e, dessa forma, é necessário executar as atividades dentro dos limites de tempo planejados. Para melhorar a produtividade das equipes durante o desenvolvimento, uma tendência natural é procurar automatizar os processos realizados.

Em sistemas de informação que acessam dados, existe, comumente, a necessidade de realizar quatro operações básicas: criação, obtenção, atualização e exclusão de dados (CRUD, do inglês *create, retrieve, update and delete*) para cada entidade do sistema.

Ao criar um sistema Web para uma biblioteca, por exemplo, é preciso realizar estas operações básicas para as entidades alunos, professores, livros, multas, reservas, exemplares, dentre outras. O projeto e implementação de tais operações torna-se repetitivo e propenso a erros.

O objetivo deste trabalho é possibilitar, com a criação de uma ferramenta, a geração de páginas Web para a realização de operações CRUD através de metadados obtidos a partir da base de dados da aplicação. Desta forma, a ferramenta possibilita o auxílio no desenvolvimento de um sistema Web através da geração de código.

Dado que no desenvolvimento de aplicações Web dinâmicas é comumente necessário o uso de linguagens dinâmicas, escolheu-se a plataforma Java (ORACLE, 2011a) para a execução deste trabalho. Nesse caso, utiliza-se a linguagem de programação Java juntamente com *JavaServer Faces* (JSF) (ORACLE, 2011b), a linguagem de visualização padrão de *Java Plataform, Enterprise Edition* desde a versão 6.0.

A linguagem de programação Java, uma das linguagens atuais de grande utilização em diversas plataformas, permite o desenvolvimento de páginas dinâmicas

para Web através de JSF, que possibilita a construção de interfaces de aplicação Web usando componentes.

Na criação de sistemas Web com JSF seguidamente é preciso criar código para as operações CRUD para diversas entidades. Utilizando um motor de *templates* rápido e simples como, por exemplo, o *Apache Velocity* e metadados de uma base de dados podem ser geradas páginas e arquivos para estas operações. Com este trabalho, pretende-se automatizar a criação destas páginas através de modelos (*templates*) e metadados de bancos de dados e gerar código Java e JSF para operações.

O restante do texto está organizado em cinco capítulos. O Capítulo 2 apresenta uma revisão bibliográfica com os conceitos utilizados e as ferramentas existentes. São abordados tópicos sobre o uso de motores de *templates* na geração de código, as operações CRUD e o mapeamento objeto-relacional, as bases de dados e metadados, o desenvolvimento para a Web e ferramentas existentes que realizam a geração de código para operações CRUD.

O Capítulo 3 apresenta o funcionamento da ferramenta proposta e seu desenvolvimento. Inicialmente mostra uma visão geral e em seguida aborda mais detalhadamente as configurações necessárias, os metadados obtidos, o funcionamento da ferramenta e o uso de modelos (*templates*).

O Capítulo 4 apresenta um experimento realizado mostrando a execução da ferramenta e os resultados obtidos. Mais especificamente, um exemplo de criação de um sistema Web para uma empresa é ilustrado.

O Capítulo 5 apresenta as conclusões sobre o desenvolvimento deste trabalho, abordando as dificuldades encontradas. Além disso, apresenta sugestões de trabalhos futuros para melhorias no processo de geração. E finalizando, o Capítulo 6 apresenta as referências utilizadas para o desenvolvimento do trabalho.

2 REVISÃO DE LITERATURA

Este capítulo apresenta a definição de alguns conceitos utilizados no desenvolvimento Web para uma melhor compreensão dos fundamentos estudados e utilizados na aplicação proposta, bem como uma análise de ferramentas que realizam a geração de código para as operações de inserção, de consulta, de atualização e de exclusão de dados a partir de uma base de dados informada.

2.1 Motor de *templates*

No contexto de aplicações Web, um modelo ou *template* é um documento no qual várias marcações foram inseridas e um motor de *templates* é um aplicativo de software que substitui essas marcações por conteúdo dinâmico retirado da aplicação (GARCÍA; CASTANEDO; FUENTE, 2007).

Esses motores geralmente disponibilizam características similares as de linguagens de programação de alto nível como, por exemplo, a utilização de variáveis e funções, a avaliação de decisões e uso de laços, porém apresentando foco na substituição de conteúdo sem formatação, também chamado de texto puro.

Parr (2004) afirma que no desenvolvimento para a Web, a necessidade de criação de páginas dinâmicas fez surgir diversos aplicativos de software de motor de *templates* para facilitar o desenvolvimento, aumentar a flexibilidade e reduzir custos de manutenção. Segundo Bergen (2007), os motores de *template* são muito utilizados para tarefas como criação de páginas, documentos e e-mails dinâmicos e para gerar código fonte.

Dentre os motores de *templates* que surgiram, o *Apache Velocity* destaca-se pela sua simplicidade e facilidade de uso, sendo escolhido para utilização no trabalho. No desenvolvimento utilizando a IDE Netbeans, a utilização do *plug-in Velocity Editor Support* facilita na criação dos modelos, pois destacam em cores as marcações realizadas.

2.1.1 *Apache Velocity*

Apache Velocity (APACHE SOFTWARE FOUNDATION, 2010) é um projeto de software de código fonte aberto mantido pela *Apache Software Foundation* que disponibiliza um motor de *templates* baseado em Java e mais algumas ferramentas como, por exemplo, um transformador de documentos de Linguagem de Marcação Extensível (XML) e um utilitário de geração de texto de propósito geral.

O *Velocity* realiza a substituição das marcações através do uso de contextos seguindo, geralmente, algumas etapas. Primeiramente, inicia-se o *Velocity* e cria-se um contexto vazio. Em seguida, o contexto é preenchido com variáveis e objetos e um modelo ou *template* é escolhido e, no final, são combinados gerando um arquivo de saída. Essas etapas podem ser vistas na Figura 2.1 que mostra um exemplo básico do uso do Velocity.

VelocityTeste.java

```
1
2  import java.io.StringWriter;
3  import java.util.ArrayList;
4  import java.util.List;
5  import org.apache.velocity.VelocityContext;
6  import org.apache.velocity.Template;
7  import org.apache.velocity.app.Velocity;
8
9  public class VelocityTeste {
10     public static void main(String[] args) throws Exception {
11         Velocity.init(); //Inicialização do Velocity
12
13         //Criação de contexto
14         VelocityContext context = new VelocityContext();
15
16         List<String> stringList = new ArrayList<String>();
17         stringList.add("123456789");
18         stringList.add("Hello World!!!");
19
20         //Adicionando valores/objetos ao contexto criado
21         context.put("name", "Velocity");
22         context.put("strings", stringList);
23
24         //Definindo um modelo/template
25         Template template = null;
26         try {
27             template = Velocity.getTemplate("mytemplate.vm");
28         } catch (Exception e) {
29             System.err.println("Erro: " + e.toString());
30         }
31
32         //Realizando a "mistura"/substituição
33         StringWriter sw = new StringWriter();
34         template.merge(context, sw);
35     }
36 }
```

Figura 2.1 - Exemplo de código de uso do *Apache Velocity*.

O motor de *templates Velocity* utiliza uma linguagem de modelos própria denominada *Velocity Template Language* (VTL) que apresenta uma sintaxe simples dividida em referências e diretivas. As referências permitem o acesso a objetos de um contexto definido e as diretivas são um conjunto de declarações para controle como, por exemplo, o uso de laços e de tomadas de decisão.

Dentre as principais características, pode-se destacar a sua simplicidade, a separação de interesses e a adaptação a varias áreas de aplicações. Além disso, outra vantagem é permitir que os modelos acessem qualquer método público de um objeto de dados, significando que é possível reutilizar classes existentes sem a necessidade de ter objetos estruturados de certa maneira como, por exemplo, os *JavaBeans*.

2.1.2 *Velocity Editor Support*

No desenvolvimento de aplicações utilizando o motor de *templates* Velocity no ambiente de desenvolvimento integrado (IDE, do inglês *Integrated Development Environment*) Netbeans (NETBEANS, 2011b), existe um *plug-in* chamado *Velocity Editor Support* (NETBEANS, 2011c) que facilita a edição dos modelos ou *templates* realizando o destaque colorido dos termos da VTL, como mostrado na Figura 2.2.

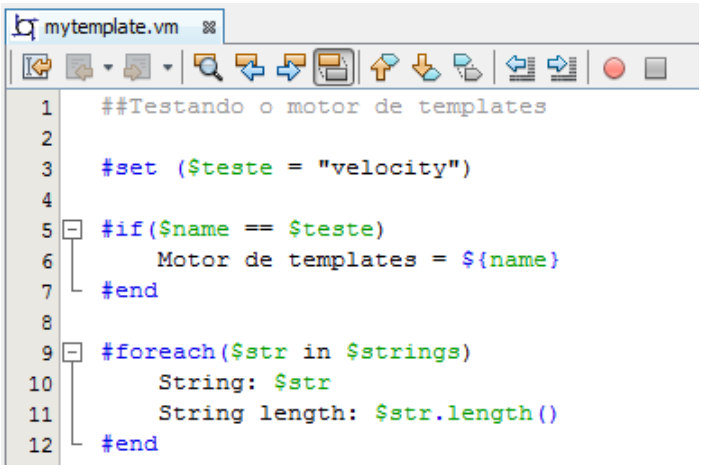

Figura 2.2 - Modelo sendo visualizado no Netbeans utilizando o *plug-in Velocity Editor Support*.

2.2 Operações CRUD e mapeamento objeto-relacional

CRUD, acrônimo de *Create, Retrieve, Update and Delete* em língua inglesa, são as quatro operações básicas de inserção, de consulta, de atualização e de exclusão de dados, utilizadas em bancos de dados relacionais (BDR). Uma das diversas maneiras de realizar a persistência de dados e permitir que a aplicação realize estas operações é a utilização do mapeamento-objeto relacional (ORM, da língua inglesa *Object-Relational Mapping*).

Na programação orientada a objetos (POO), a representação dos dados é realizada como um grafo interconectado de objetos, onde cada entidade de um determinado sistema é um objeto com métodos e atributos e que geralmente manipulam valores não escalares. Porém nos bancos de dados relacionais, a representação de dados é realizada no formato tabular, onde as tabelas armazenam valores de maneira escalar.

Essa diferença na representação de dados entre a POO e os BDR proporciona alguns problemas de incompatibilidade que podem ser amenizados através de um mapeamento dos objetos para as tabelas, técnica chamada de mapeamento-objeto relacional.

Nesse contexto, o mapeamento objeto-relacional em Java pode ser realizado utilizando a *Java Persistence Application Programming Interface* (JPA ou *Java Persistence API*).

2.2.1 Java Persistence API

Java Persistence Application Programming Interface é uma especificação que padroniza o mapeamento objeto-relacional para o desenvolvimento Java (BISWAS; ORT, 2006). Existem diversas implementações de JPA, dentre elas destacam-se a do Hibernate (JBOSS COMMUNITY, 2011), Toplink (ORACLE, 2011d) e Open JPA (APACHE SOFTWARE FOUNDATION, 2011).

O JPA define uma maneira de mapeamento objeto-relacional para objetos Java simples e comuns (POJO, do inglês *Plain Old Java Objects*) onde cada

entidade persiste em uma tabela no BDR. Esses objetos simples possuem atributos e métodos *de* leitura e escrita e não implementam ou estendem outras classes. Os POJOs são usados juntamente com algumas anotações que representam como o objeto é armazenado no BDR ou através de configuração por XML.

2.3 Base de dados e metadados

Metadados, ou metainformação, são dados que descrevem dados. Qualquer dado pode ser um metadado, para isso basta que ele descreva outro dado do sistema e o seu uso facilita o entendimento dos relacionamentos e a utilidade das informações dos dados presentes.

Cada BDR possui seus próprios mecanismos para armazenar seus metadados. Geralmente armazenam tabelas sobre suas tabelas contendo nome, tamanhos e quantidades de linhas e tabelas de colunas contendo informações sobre os tipos de dados salvos em cada coluna e em que tabelas elas são utilizadas. Nos BDRs o seu conjunto de metadados é chamado de catálogo e a descrição em linguagem formal das tabelas, campos e relações é chamada de esquema.

2.3.1 Metadados em Java

O acesso a metadados em um banco de dados relacional em Java é realizado através da API *Java DataBase Connectivity* (JDBC) (ORACLE, 2011c). Ela fornece uma interface chamada *DatabaseMetaData* que permite buscar diversas informações referentes a determinada base de dados.

Para buscar informações sobre as tabelas de um BDR é preciso realizar uma conexão e chamar algum método. Para descobrir as tabelas, por exemplo, utiliza-se o método *getTables()*, cujo retorno é um *ResultSet*, uma tabela onde cada linha representa informações sobre uma das tabelas presentes na base de dados.

Outras informações que podem ser obtidas sobre as tabelas são as chaves primárias e as chaves importadas e exportadas. Além disso, estão disponíveis

métodos que buscam informações sobre esquemas e catálogos, entre outras diversas informações relativas a base de dados.

2.4 Desenvolvimento para a Web

O desenvolvimento para a Web geralmente está associado ao uso de programação e de marcação de textos no desenvolvimento de *sites*, para uso na Internet ou na intranet. Envolve desde o desenvolvimento de páginas estáticas até *sites* de comércio eletrônico. A seguir, serão descritas as linguagens XHTML, CSS e Java utilizadas no desenvolvimento Web.

2.4.1 XHTML

Linguagem de Marcação de Hipertexto Extensível (XHTML, do inglês *Extensible Hypertext Markup Language*) é uma recomendação da *World Wide Web Consortium* (W3C) que corresponde a uma reformulação de Linguagem de Marcação de Hipertexto (HTML, do inglês *Hypertext Markup Language*) (W3C, 2010), combinando suas *tags* de marcação com as regras de XML.

Em XHTML todas as *tags* e seus atributos devem ser escritos em letras minúsculas, os elementos vazios devem ser fechados, sendo obrigatório o fechamento de todas as *tags*, os documentos devem ser bem formados com a abertura e fechamento de *tags* de maneira correta e os valores de atributos devem estar entre aspas, dentre outras regras.

2.4.2 CSS

Cascading Style Sheets (CSS) (W3C, 2011) é uma linguagem de estilos utilizada para definir a apresentação de documentos escritos em alguma linguagem

de marcação, por exemplo, XHTML, e cujo principal benefício é disponibilizar a separação entre formatação e o conteúdo de um documento.

Possui uma sintaxe simples e consiste numa lista de regras e atributos. Um dos grandes desafios ao se utilizar CSS é garantir que as regras criadas serão representadas de maneira igual, ou pelo menos similar, pois os diversos navegadores Web existentes não implementam o analisador de código CSS de maneira correta.

2.4.3 Java com JSF

JavaServer Faces (JSF) (ORACLE, 2011b) é um *framework* para o desenvolvimento de aplicações Web baseado no padrão Modelo-Visualização-Controle (MVC, do inglês *Model-View-Controller*) e na linguagem de programação Java. É composto por um conjunto de componentes de interface de usuário, por um modelo de programação baseado em eventos, e um modelo de componentes que permite que terceiros forneçam componentes adicionais.

O JSF é a camada de visualização do padrão *Java Enterprise Edition* e possui múltiplas implementações, sendo que cada servidor de aplicação pode desenvolver a sua versão. Utiliza os *beans* gerenciados, um *Java Bean* que pode ser acessado de uma página JSF, para acessar dados que uma página necessite (GEARY, 2010). Normalmente é criada uma página JSF para cada tela a ser mostrada no navegador.

As páginas JSF precisam ser XHTML válidas e suas *tags* apresentam uma pequena variação ao formato HTML. Dentre as principais características destacam-se a possibilidade de validação e tratamento de exceções, a utilização de internacionalização, o uso de componentes customizados e o suporte a *Javascript* e XML Assíncronos (AJAX, acrônimo da língua inglesa de *Asynchronous Javascript and XML*) (GARRETT, 2005).

2.5 Ferramentas para a geração de operações CRUD

Atualmente existem algumas ferramentas que realizam a geração de código para operações CRUD, como a SQLCODEGEN (RAFFEL, 2010), uma ferramenta para linguagem C#. Para a linguagem Java utilizada com JSF, pode-se destacar o uso do Netbeans (NETBEANS, 2011a), que será descrito a seguir.

2.5.1 Netbeans

O IDE Netbeans oferece dois assistentes que realizam a geração de código para operações CRUD: o assistente para Classes de entidade do banco de dados e o assistente para Páginas JSF de classes de entidade.

Após a criação da base de dados e do projeto Web, é utilizado o primeiro assistente que possibilita a geração de entidades a partir de um banco de dados informado. Primeiramente, configuram-se as informações de conexão, como o *driver*, *host*, o BDR, usuário e senha e o Localizador Padrão de Recursos (URL, do inglês *Uniform Resource Locator*) do JDBC, e em seguida escolhem-se as tabelas do BD que serão utilizadas pra gerar as entidades. Por último, podem-se definir nomes para as entidades, o pacote em que serão salvas e algumas opções de mapeamento. A Figura 2.3 mostra a primeira tela desse assistente com algumas informações configuradas.

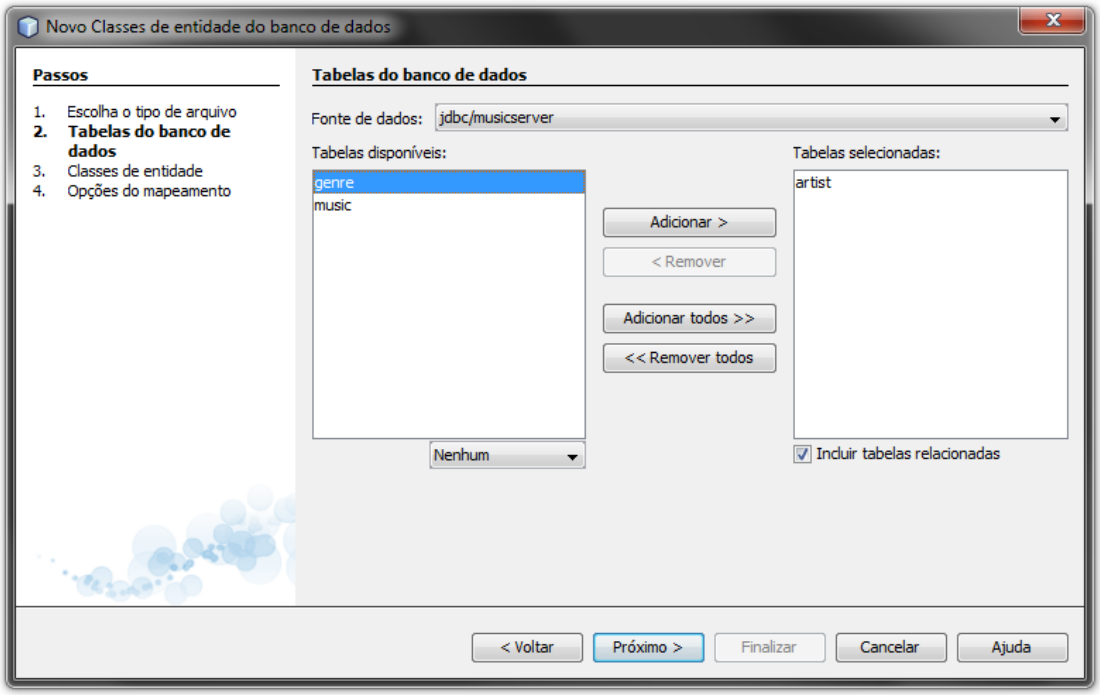

Figura 2.3 - Assistente do Netbeans para criação de classes de entidade a partir de uma base de dados.

Com estas classes criadas, utiliza-se o segundo assistente para criar os *beans* de classes de entidade e as páginas JSF da interface para realizar as operações CRUD. Primeiramente, escolhe-se para quais entidades vão ser criadas as páginas e, em seguida, definem-se os locais onde serão salvos os *beans* e as páginas. Neste ponto podemos utilizar os modelos padrão da IDE ou realizar alterações conforme a necessidade. A Figura 2.4 mostra uma das telas de um teste realizado e apresenta algumas configurações realizadas.

Ao final, todos os arquivos estão gerados e organizados conforme as configurações realizadas, e o projeto está pronto para executar as operações CRUD criadas com os dois assistentes. Porém os assistentes não permitem inserir modelos que o usuário tenha criado, sendo necessário alterar os existentes.

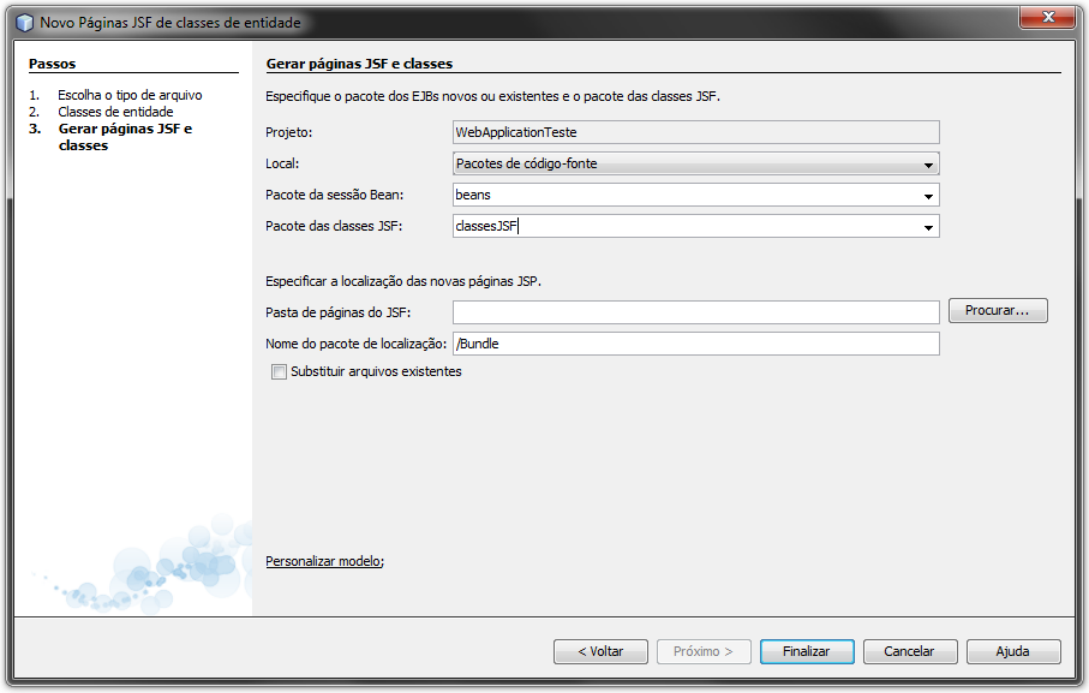

Figura 2.4 - Assistente do Netbeans para geração de páginas JSF a partir de classes de entidade.

3 GERAÇÃO DE OPERAÇÕES CRUD EM SISTEMAS WEB USANDO JSF

Este capítulo destina-se a apresentar a proposta de criação da ferramenta *CRUDerJSF*, que tem como objetivo gerar código para operações CRUD em sistemas Web usando JSF. Na Seção 3.1, descreve-se o funcionamento da ferramenta de maneira geral. Nas seções seguintes, expõem-se, de maneira mais detalhada, as configurações iniciais necessárias, os metadados obtidos, o funcionamento da ferramenta e, por último, os *templates* utilizados.

3.1 Visão Geral

Dijkstra (1982) descreve a separação de interesses como uma técnica disponível para a ordenação eficaz de nossos pensamentos e afirma que é preciso centrar a atenção em um determinado aspecto, sem ignorar os outros, de maneira que sobre o ponto de vista desse, os demais são irrelevantes. Além disso:

> Ao projetar uma aplicação, é sempre importante a correta divisão dos componentes de modo que cada componente ou grupo de componentes realiza uma ou várias tarefas. Esta separação de interesses leva a menos dependências, código mais limpo, e uma aplicação mais manutenível e flexível. (Bond, 2002, p. 817, tradução nossa).

Normalmente no desenvolvimento para Web, a separação de interesse nos sistemas é implementada utilizando uma arquitetura em camadas na qual, segundo Bond (2002), o modelo de três camadas tornou-se de fato a arquitetura padrão para sistemas Web empresarial.

Nessa arquitetura, que pode ser vista na Figura 3.1, o sistema é dividido nas camadas de apresentação, negócios e acesso a dados. A camada de apresentação é responsável pela maneira como as informações e os dados serão disponibilizados ao usuário. A camada de negócios contém a lógica do negócio, ou seja, as

funcionalidades referentes ao domínio da aplicação. Por último, a camada de acesso a dados provém uma maneira de acessar os dados que estão armazenados em algum local, geralmente em um banco de dados.

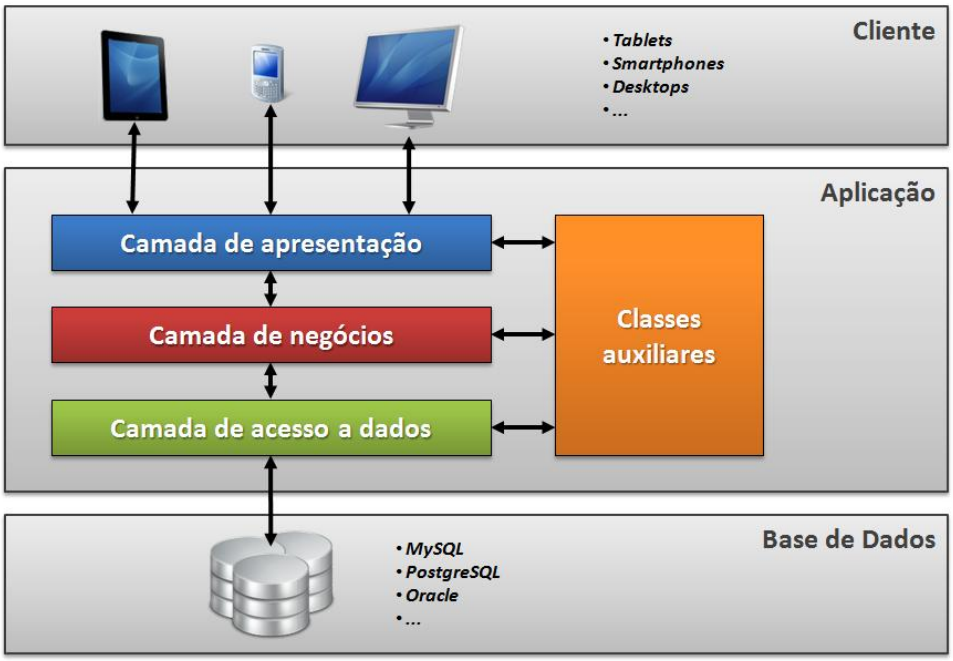

Figura 3.1 - Esquema de arquitetura de três camadas.

Utilizando essa estrutura, podemos criar diversas interfaces de apresentação para os mais variados dispositivos clientes como, por exemplo, *tablets*, *smartphones*, etc., ou substituir a base de dados utilizada por outra praticamente sem alterar a lógica e as funcionalidades da aplicação.

O objetivo da ferramenta proposta é gerar código de uma aplicação Web, seguindo o modelo de arquitetura de três camadas, conforme o processo de geração mostrado na Figura 3.2. Inicialmente são obtidos os metadados da base de dados que são armazenados em memória. Em seguida os criadores são responsáveis por chamar um ou mais geradores e também por criarem, caso necessário, alguns diretórios. Cada gerador recebe um modelo e alguns metadados e, com o auxílio do *Velocity*, gera arquivos *Java* e *xhtml*, por exemplo. Após a execução de todos criadores e geradores a ferramenta finaliza sua execução.

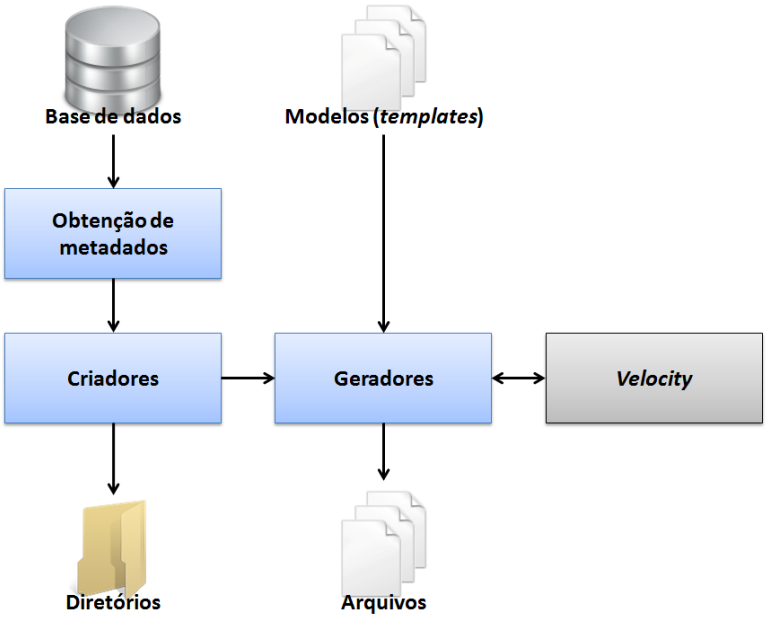

Figura 3.2 - Visão geral do funcionamento da ferramenta proposta.

3.2 Configurações iniciais

Para que a geração de código seja realizada, primeiramente é preciso realizar algumas configurações de entrada de dados como, por exemplo, a definição da conexão com a base de dados e a escolha do diretório onde serão salvos os arquivos. Além disso, é possível definir os nomes dos diretórios específicos onde serão salvas as entidades do sistema, os *beans* ou controladores das páginas JSF, os arquivos de acesso a dados, os arquivos com as regras de negócio e as páginas Web.

A ferramenta apresenta modelos padrões para geração das páginas Web do sistema gerado, mas é possível criar modelos e utilizá-los como entrada. Esses modelos devem utilizar a linguagem *VTL* do *Velocity* para acessar as variáveis disponíveis para a geração do código.

3.3 Metadados obtidos

O processo de obtenção dos metadados da base de dados informada é realizado através da API *Java DataBase Connectivity* (JDBC) (ORACLE, 2011b) utilizando a interface *DataBaseMetaData*. A ferramenta utiliza o *driver* MySQL Connector/J (MYSQL, 2011) , porém é possível adicionar *drivers* de outras bases de dados para uso. Esses metadados são carregados em memória para serem utilizados posteriormente na geração dos arquivos.

Após a conexão com a base de dados, as tabelas presentes no BDR são obtidas utilizando o método *getTables*, responsável por retornar um ResultSet. Cada linha desse *ResultSet*, apresenta informações referentes a uma tabela presente na base de dados.

De posse das tabelas, o próximo passo é adquirir informações referentes aos campos presentes em cada uma delas. Através do nome, são utilizados os métodos *getColumns* para retornar os campos, *getPrimaryKeys* para retornar as chaves primárias, *getImportedKeys* para retornar as chaves que foram importadas de outras tabelas e *getExportedKeys* para retornar as chaves que são exportadas para outras tabelas.

O uso desses métodos permite o acesso a informações referentes a cada campo como, por exemplo, nome, tipo de dado, se pode ser nulo, se é um valor automaticamente incrementado, se é chave primária e se foi exportada para outra tabela.

Na criação de bancos de dados, as tabelas que são definidas podem apresentar diversos conceitos como atributos multivalorados, relacionamentos 1:1, 1:*n*, *n*:1 ou *n*:*n*, auto-relacionamentos, especializações, entre outros.

O mapeamento objeto-relacional, utilizando JPA, inclui o mapeamento de tipos básicos, de chaves primárias e de relacionamentos. Nesse contexto, o mapeamento de relacionamentos apresenta-se mais complexo, envolvendo, por exemplo, o uso de coleções e os mapeamentos unidirecionais e bidirecionais.

Levando em consideração as possibilidades de criação de bancos de dados e de mapeamento objeto-relacional, realizar a geração de código de maneira completa necessitaria tempo e esforço consideráveis. Assim, a ferramenta gera código para as

operações CRUD focando no mapeamento simples, não mapeando as relações entre as entidades.

3.4 Funcionamento da ferramenta

Após a obtenção e carregamento para memória dos metadados da base de dados informada, inicia-se a geração dos arquivos e diretórios do sistema Web através dos criadores e dos geradores seguindo as seguintes etapas:

- Informação dos nomes dos diretórios e caminhos dos arquivos utilizados;
- Informação de conexão com a base de dados a ser utilizada;
- Busca dos metadados;
- Para cada criador:
    - Criação dos diretórios utilizados;
    - Geração de arquivos.

No desenvolvimento de sistemas Web utilizando JSF, os arquivos são geralmente dispostos em dois diretórios, o “*src*”, que contém os códigos *Java*, e o “*web*”, que contém as páginas, os arquivos de estilo, os códigos *javascript*, as imagens, e outros arquivos que são utilizados para visualização do sistema nos navegadores. Os criadores seguem essa estrutura e são responsáveis pela organização do sistema através da criação dos diretórios necessários, como pode ser visto na Figura 3.3.

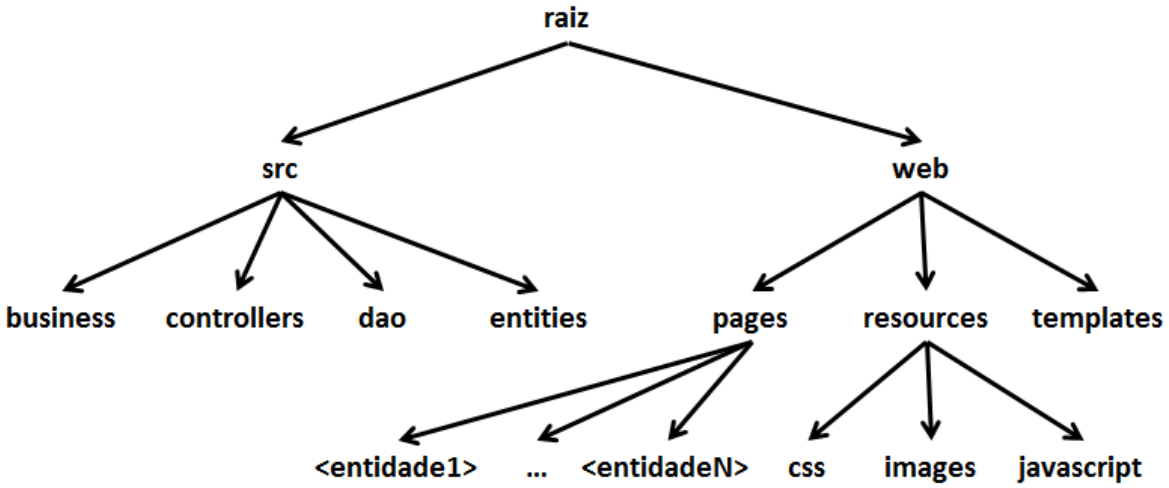

Figura 3.3 - Estrutura básica dos diretórios criados pela ferramenta.

Dentro do diretório "*pages*", são criados diversos diretórios conforme a quantidade de entidades que foram obtidas da base de dados, e cada um desses diretórios apresenta o nome da entidade. Cada criador também é responsável por iniciar a execução de um ou mais geradores de arquivos.

A Figura 3.4 apresenta os criadores presentes na ferramenta. O arquivo *AbstractCreator.java* possui uma classe abstrata de um criador, que é extendida pelos outros criadores. O arquivo *BusinessCreator.java* apresenta o criador responsável pela camada de negócios, o arquivo *CRUDPageCreator.java* apresenta o criador responsável pelas páginas XHTML das operações CRUD, o arquivo *ControllerCreator.java* apresenta o criador responsável pelos controladores ou *beans* e o arquivo *DAOCreator.java* apresenta o criador responsável pela camada de acesso a dados.

Além disso, o arquivo *EntityCreator.java* apresenta o criador responsável pelas entidades do sistema, o arquivo *PageCreator.java* apresenta o criador responsável pelas páginas XHTML inicial e *template* e pelos arquivos CSS e o arquivo *XMLCreator.java* apresenta o criador responsável pelos arquivos XML, principalmente o arquivo que define as configurações de persistência de dados.

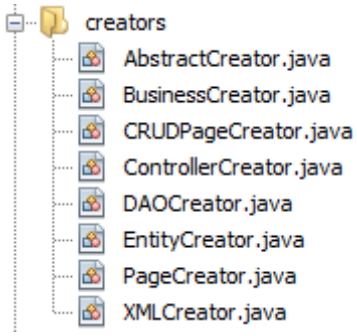

Figura 3.4 - Criadores presentes na ferramenta.

Como motor de *templates Velocity* utiliza contextos para realizar a substituição das marcações presentes nos modelos, cada gerador precisa preencher seu contexto com os metadados obtidos anteriormente. Além disso, cada gerador recebe um modelo como entrada e com a ajuda do *Velocity* realiza as substituições necessárias e gera um arquivo de saída.

A Figura 3.5 apresenta os geradores presentes na ferramenta. No diretório pages estão os geradores responsáveis pela geração das páginas XHTML, que podem ser utilizados ou é possível passar outros geradores criados pelo usuário. São geradas através destes arquivos as páginas de inserção, remoção, atualização e visualização das entidades, a página inicial e o *template* do sistema a ser criado.

Os demais arquivos são responsáveis pela geração dos códigos do sistema. O arquivo *AbstractDAOGenerator* gera uma classe abstrata de acesso a dados, o arquivo *Attribute.java* é uma classe auxiliar para a geração, o arquivo *BussinessGenerator.java* gera classes da camada de negócios e o arquivo *ControllerGenerator.java* gera as classes da camada apresentação, ou seja, os controladores ou *beans*.

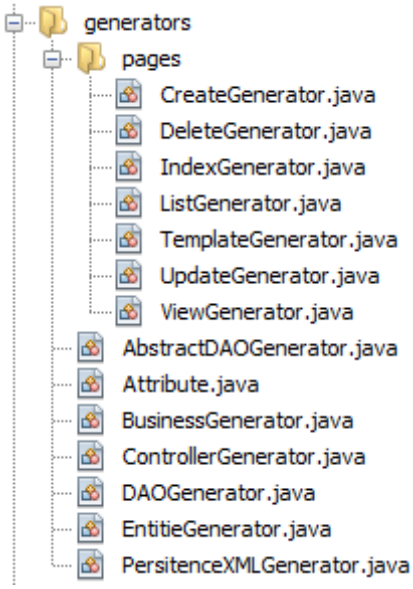

Figura 3.5 - Geradores presentes na ferramenta.

Além disso, o arquivo *DAOGenerator.java* gera classes de acesso a dados, o arquivo *EntitieGenerator.java* gera as entidades do sistema e o arquivo *PersistenceXMLGenerator.java* gera o arquivo XML com as configurações de persistência. A Figura 3.6 apresenta o diretório *velocity,* que possui dois arquivos, o *SimpleContext.java* que apresenta a definição de um contexto simples para ser utilizado por outras classes e o arquivo *SimpleGenerator.java* que é uma classe abstrata de um gerador e é extendida pelos outros geradores.

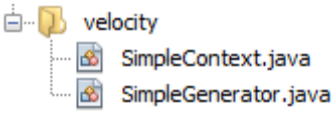

Figura 3.6 - Gerador abstrato e contexto simples.

Após a execução dos criadores e dos geradores, os diretórios e os arquivos do sistema Web estão prontos. Tendo em vista que a ferramenta gera código para operações CRUD em sistemas Web com JSF, é possível utilizar esses arquivos juntamente com o IDE Netbeans ou o Eclipse (ECLIPSE, 2011b) e adicionar outras funcionalidades desejadas.

Para realizar a persistência de dados, a aplicação Web gerada utiliza o provedor de persistência EclipseLink (ECLIPSE, 2011a) e, desta forma, caso seja desejado a utilização de outro provedor, faz-se necessária a sua configuração.

3.5 *Templates*

Os *templates* ou modelos são responsáveis por determinar como os arquivos da aplicação Web serão gerados e podem ser divididos em dois grupos na ferramenta, os modelos internos e os externos.

Os modelos internos são utilizados para gerar as classes *Java*, arquivos *XML* e outros arquivos que não podem ser informados pelo usuário. Os modelos externos são utilizados na geração das páginas *JSF* representando arquivos que podem ser informados pelo usuário. A Figura 3.7 apresenta os modelos utilizados pela ferramenta proposta, sendo que no diretório "*pages*" encontram-se *templates* externos, os demais são internos.

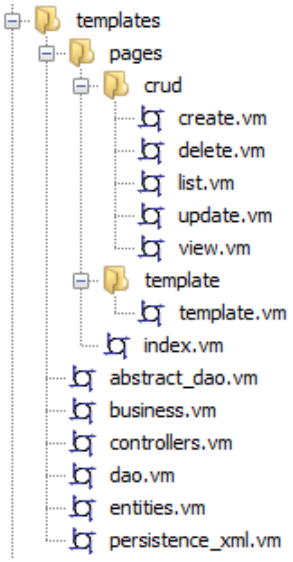

Figura 3.7 - Modelos utilizados pela ferramenta proposta.

Dentro do diretório “*pages*” estão os modelos padrão para as páginas responsáveis pela inserção, remoção, atualização e visualização, sendo esse último representado pelos arquivos “*list.vm*”, que lista todas entidades, e “*view.vm*”, que mostra informações de uma entidade específica. Além disso, estão presentes os modelos para a estrutura das páginas, representado pelo arquivo “*template.vm*” e o modelo da página inicial.

O modelo “*abstract_dao.vm*” é utilizado para gerar uma classe abstrata de acesso a dados, o “*dao.vm*” para gerar as classes específicas de acesso a dados e o modelo “*entites.vm*” gera as entidades básicas do sistema. O modelo “*business.vm*” gera as classes de negócio da aplicação, o modelo “*persistence_xml.vm*” gera o arquivo XML com as configurações de persistência e o modelo “*controlers.vm*” gera os *beans* que controlam as informações das páginas JSF.

O motor de *templates Velocity* é responsável por realizar a substituição das marcações feitas nos modelos pelos metadados obtidos da base de dados. A apresentação do código de um modelo CRUD, mostra-se extensa e de difícil formatação. Assim, optou-se por mostrar a Figura 3.8, que apresenta o código de um modelo simples usado para geração das classes *Java* para acesso aos dados das entidades do sistema Web. Na seção de anexos, é possível visualizar outros modelos e arquivos da ferramenta e para facilitar a visualização dos modelos no IDE Netbeans, foi utilizado o *plug-in Velocity Editor Support*.

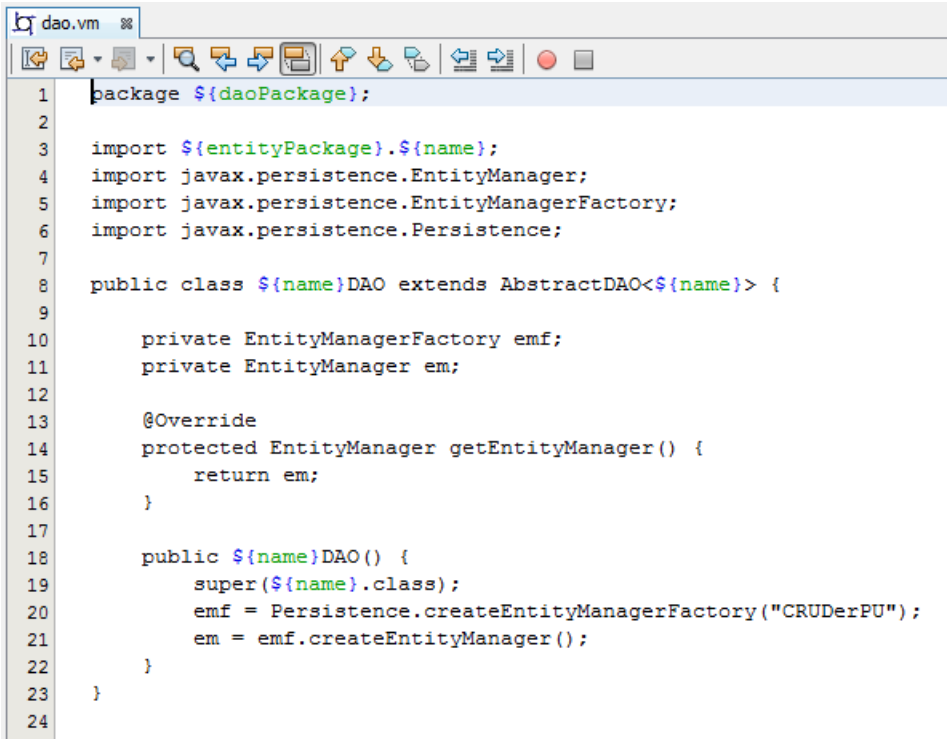

Figura 3.8 - Código de modelo para gerar arquivos de acesso a dados das entidades.

Nesse código, nas linhas de número um até seis são substituídos as variáveis para as importações de outros arquivos gerados, conforme configuração inicial da ferramenta. Da linha oito em diante, a variável *name* vai ser substituída conforme as entidades presentes no sistema.

4 EXPERIMENTO

Este capítulo destina-se a apresentar a utilização da ferramenta através de um exemplo de uso de criação de um sistema Web para uma empresa. Foram utilizados os modelos padrão da ferramenta, que podem ser vistos na seção de anexos juntamente com os arquivos gerados neste exemplo.

4.1 Aplicativo para uma empresa

Este exemplo consiste no desenvolvimento de uma aplicação, a partir da base de dados mostrada na Figura 4.1, para uma empresa fictícia. A empresa possui empregados que trabalham em um determinado departamento. Cada empregado pode ter uma vaga no estacionamento da empresa e participa de um ou mais projetos.

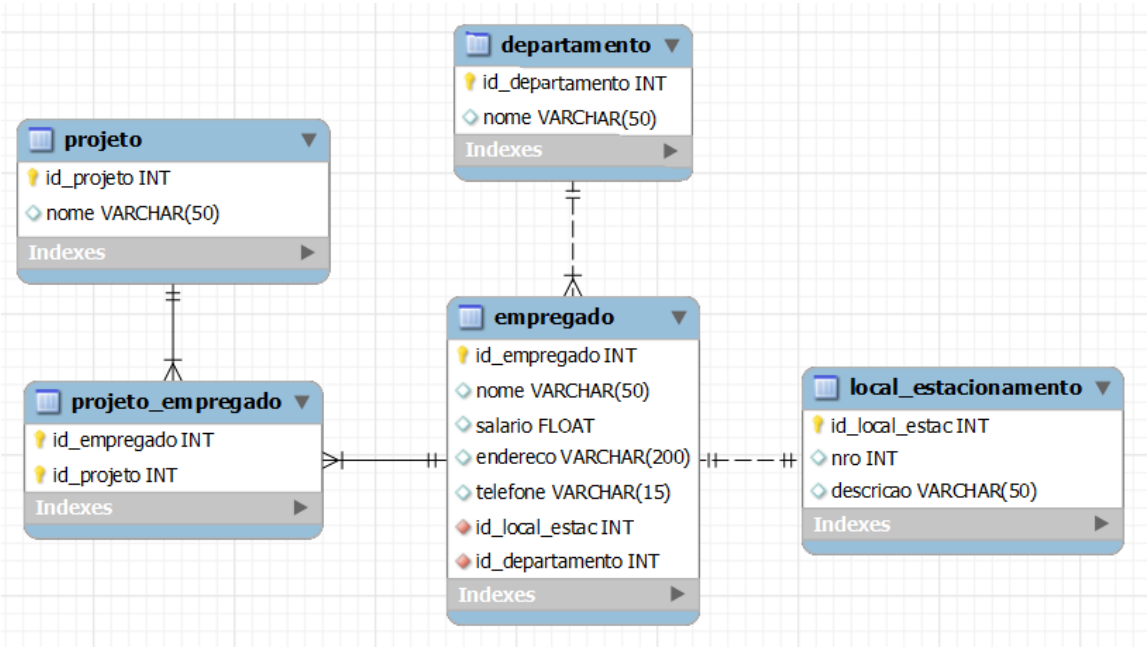

Figura 4.1 - Modelo relacional da empresa.

Neste exemplo, o sistema gerado será utilizado juntamente com o Netbeans e implantado no servidor de aplicação Glassfish (GLASSFISH, 2011). Desta maneira, cria-se uma projeto Web vazio no Netbeans, onde serão salvos os arquivos gerados.

Após a criação da base de dados, inicia-se o processo de geração dos arquivos utilizando as configurações e os modelos padrão presentes na ferramenta, definindo apenas as configurações respectivas à conexão com o banco criado e ao diretório onde serão criados os arquivos. Essa etapa inicial é realizada via console.

Inicia-se a geração e, ao final, temos a estrutura de arquivos apresentada na Figura 4.2. Os diretórios "*build"*, "*dist"*, "*lib"*, "*nbproject"*, "*conf"*, "*java"*, "*WEB-INF"* e os arquivos internos a estes diretórios, juntamente com o arquivo *build.xml* são criados pelo Netbeans. Os demais diretórios e arquivos foram criados ou gerados pela ferramenta.

O banco de dados apresenta cinco entidades, *projeto*, *projeto_empregado*, *departamento, empregado* e *local_estacionamento*. Para cada entidade foi gerado um arquivo da camada de negócios, no diretório "*bussiness"*, um controlador para a apresentação, no diretório "*controllers"*, um arquivo de acesso a dados, no diretório "*dao"*, e um arquivo para a entidade, no diretório "*entities"*.

Além disso foram geradas as páginas XHTML para inserção (*create.xhtml*), remoção (*delete.xhtml*), atualização (*update.xhtml*) e visualização (*list.xhtml* e *view.xhtml*) para cada uma das entidades, nos diretórios com o nome da entidade.

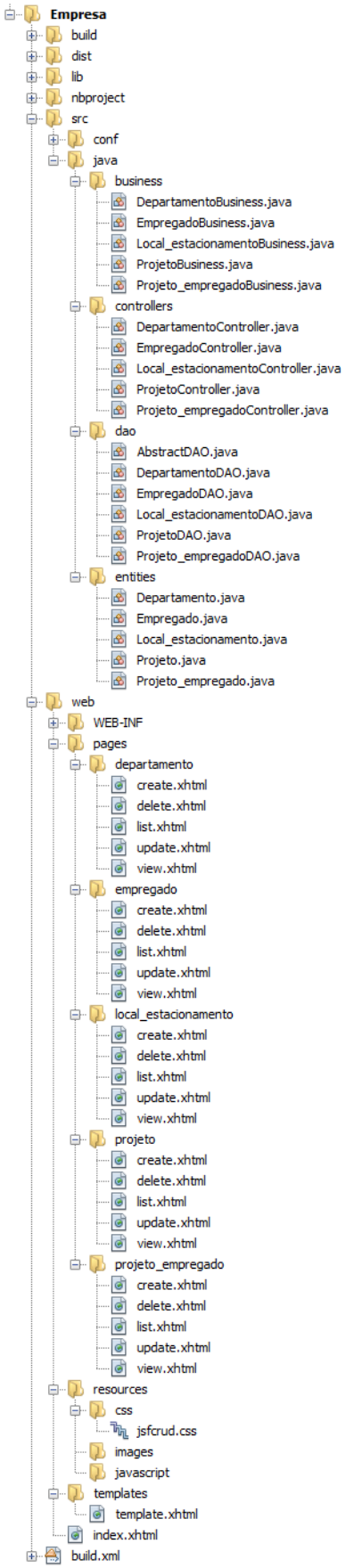

Figura 4.2 - Arquivos e diretórios criados.

Para visualizar o sistema gerado, implanta-se o projeto no servidor de aplicação e executa-se o projeto. Na Figura 4.3, vemos a página inicial gerada, que apresenta as opções para as páginas que listam cada entidade.

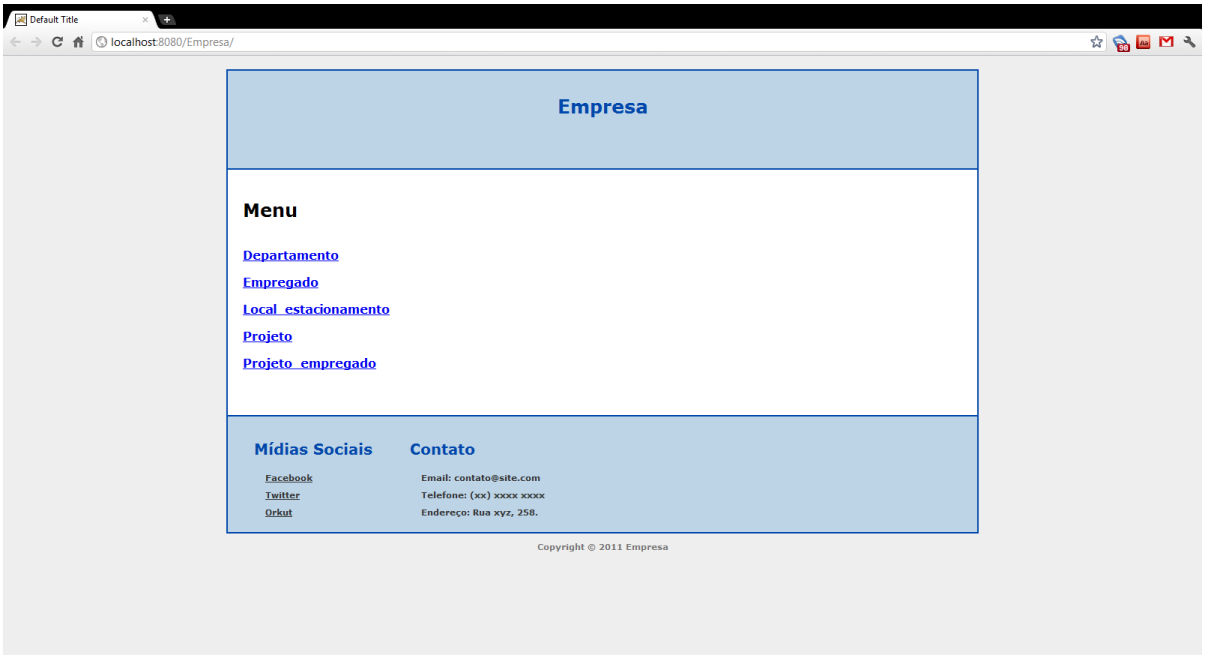

Figura 4.3 - Página inicial do sistema da empresa.

A página de listagem de cada entidade apresenta uma tabela que lista os elementos existentes na base de dados e permite a opção de adicionar, visualizar, editar ou remover cada elemento. Podemos ver na Figura 4.4, o exemplo da listagem de departamentos.

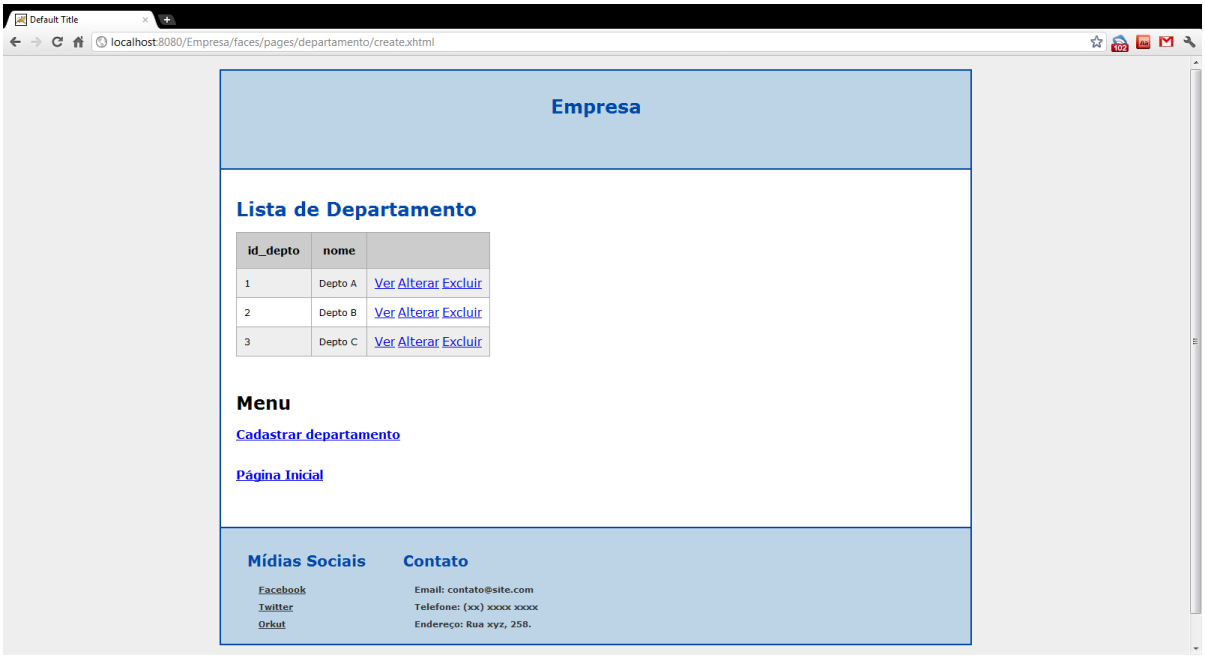

Figura 4.4 - Página de listagem de departamentos.

As páginas de inserção, atualização, visualização de um elemento e remoção são bem similares, apresentando formulários com os campos presentes. Na Figura 4.5, é apresentada a página que permite inserir novos departamentos na empresa.

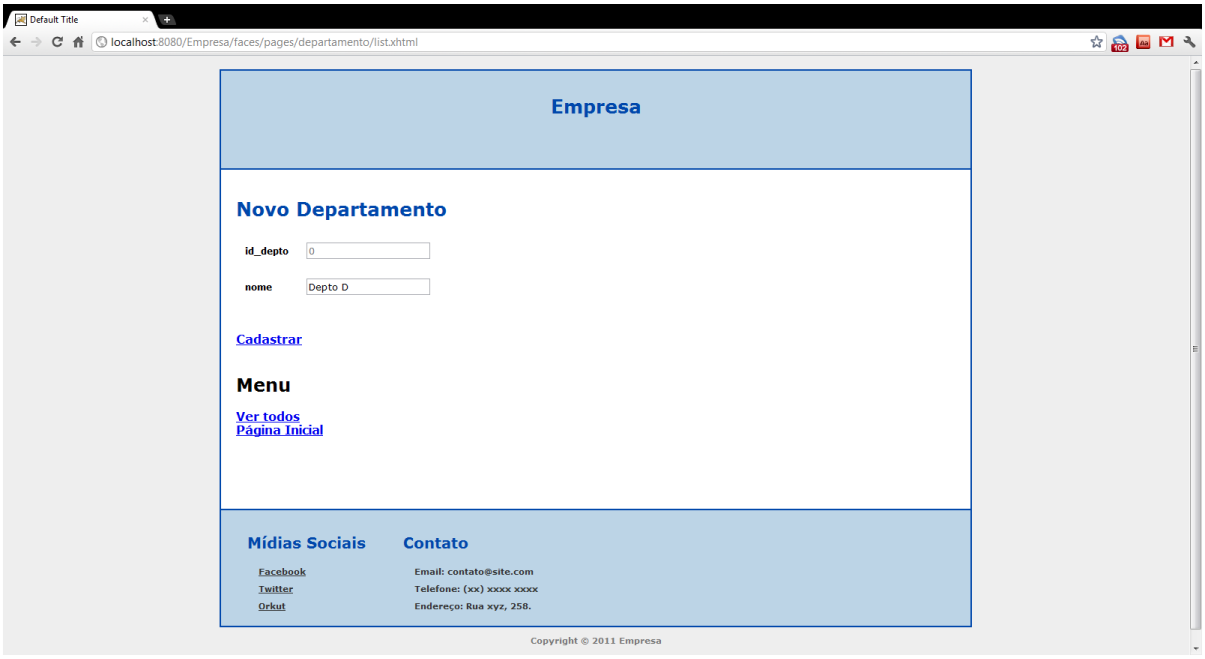

Figura 4.5 - Página de inserção de departamento.

5 CONCLUSÕES

O trabalho propôs a criação de uma ferramenta para geração de código para operações de inserção, atualização, remoção e visualização de dados em sistemas Web desenvolvidos com JSF. Essa geração ocorre através da combinação de informações da base de dados a ser utilizada pelo sistema com alguns modelos através do uso de um motor de *templates*.

Como entrada, a ferramenta obtém os metadados da base de dados informada, que fornecem as informações necessárias para geração das entidades do sistema, e os modelos, que permitem definir como os arquivos são gerados. Com a ajuda do *Apache Velocity*, um motor de *templates*, é gerado os arquivos e diretórios do sistema Web.

Essa ferramenta apresenta-se como uma maneira de facilitar o desenvolvimento de aplicações Web através da geração de código para operações CRUD. Desta forma, ela não tem como objetivo gerar um sistema final, pronto para uso, necessitando a adição de funcionalidades após a geração.

A geração de código para operações CRUD em sistemas Web com JSF mostrou-se complexa na etapa de mapeamento objeto-relacional, principalmente no mapeamento das relações entre as entidades. A dificuldade apresentou-se ao realizar a associação das tabelas através dos metadados obtidos a coleções, somado a pouca experiência de ORM através da API JPA. Nesse contexto de ORM, os arquivos das entidades realizam o mapeamento de maneira bem simples, sendo as relações inseridas por chaves e não pelos objetos.

Levando em conta as dificuldades encontradas no ORM, a proposta da ferramenta de facilitar o desenvolvimento Web e o tempo para execução do trabalho, foi necessária a definição de um mapeamento mais simples.

Apesar de não apresentar uma interface e da necessidade de melhorias no processo de mapeamento objeto-relacional, o objetivo do trabalho foi alcançado permitindo a geração de código para operações CRUD para páginas Web com JSF.

5.1 Trabalhos Futuros

Buscando melhorar a ferramenta desenvolvida, primeiramente sugere-se o desenvolvimento de uma interface gráfica para realização das configurações necessárias para o funcionamento. Nesse contexto, é interessante a integração da ferramenta com algum IDE como, por exemplo, o Netbeans ou o Eclipse através do desenvolvimento de *plug-ins*.

Além disso, na obtenção dos metadados, sugere-se a inclusão de *drivers* e a realização de testes para outros SGBDs possibilitando, desta maneira, um uso mais abrangente para a ferramenta.

Sugere-se também, melhoria no mapeamento objeto-relacional juntamente com a obtenção dos metadados, que devido às dificuldades encontradas nesse trabalho relacionadas à complexidade e ao tempo de execução não foram abordados. Seria interessante definir os metadados necessários a serem obtidos para que a geração das entidades de persistência e o mapeamento das relações entre elas fossem executados.

Por fim, ao configurar a conexão com a base de dados, sugere-se a possibilidade de selecionar quais tabelas da base de dados serão utilizadas para geração do sistema Web e a renomeação das informações vindas da base de dados como, por exemplo, os nomes das entidades e campos presentes.

6 REFERÊNCIAS

APACHE SOFTWARE FOUNDATION. **The Apache Velocity Project**. 2010. Disponível em: <http://velocity.apache.org/>. Acesso em: 10 ago. 2011.

APACHE SOFTWARE FOUNDATION. **Welcome to the Apache OpenJPA project**. 2011. Disponível em: <http://openjpa.apache.org/>. Acesso em: 22 out. 2011.

BERGEN, J. V. **Velocity or FreeMarker?** Two open source Java-based template engines compared. 2007. Disponível em: <http://www.javaworld.com/javaworld/jw-11-2007/jw-11-java-template-engines.html>. Acesso em: 5 out. 2011.

BISWAS, R.; ORT, E. **The Java Persistence API** - A Simpler Programming Model for Entity Persistence. 2006. Disponível em: <http://www.oracle.com/technetwork/articles/javaee/jpa-137156.html>. Acesso em: 29 set. 2011.

BOND, M. et al. **SAMS Teach Yourself J2EE in 21 Days**. 1. Ed. SAMS. 2002.

DIJKSTRA, E. W. **Selected Writings on Computing**: A Personal Perspective. Springer-Verlag. 1982.

ECLIPSE. **EclipseLink Project**. 2011a. Disponível em: < http://www.eclipse.org/eclipselink/>. Acesso em: 3 dez. 2011.

ECLIPSE. **Home**. 2011b. Disponível em: <http://www.eclipse.org/>. Acesso em: 3 dez. 2011.

GARCÍA, F. J.; CASTANEDO, R. I.; FUENTE, A. A. J. **A Double-Model Approach to Achieve Effective Model-View Separation in Template Based Web Applications**. 2007. Springer.

GARRETT, J. J. **Ajax**: A New Approach to Web Applications. 2005. Disponível em: <http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications>. Acesso em: 20 out. 2011.

GEARY, D.; HORSTMANN, C. S. **Core JavaServer Faces**. 3. Prentice Hall, 2010.

GLASSFISH. Glassfish. **Community**. 2011. Disponível em: <http://glassfish.java.net/>. Acesso em: 03 dez. 2011.

JBOSS COMMUNITY. Hibernate. **Home**. 2011. Disponível em: <http://www.hibernate.org/>. Acesso em 22 out. 2011.

MYSQL. **Download Connector/J.** 2011. Disponível em: <http://dev.mysql.com/downloads/connector/j/>. Acesso em: 1 dez. 2011.

NETBEANS. Docs and Support. **Generating a JavaServer Faces 2.0 CRUD Application from a Database**. 2011a. Disponível em: <http://netbeans.org/kb/docs/web/jsf20-crud.html>. Acesso em: 26 out. 2011.

NETBEANS. **Home**. 2011b. Disponível em: <http://netbeans.org/>. Acesso em: 5 out. 2011.

NETBEANS. Plugin Portal. **Velocity Editor Support**. 2011c. Disponível em: <http://plugins.netbeans.org/plugin/9155/velocity-editor-support>. Acesso em 1 dez. 2011.

ORACLE. **Java**. 2011a. Disponível em: <http://www.oracle.com/technetwork/java/index.html>. Acesso em: 1 dez. 2011.

ORACLE. **JavaServer Faces Technology**. 2011b. Disponível em: <http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html>. Acesso em: 10 ago. 2011.

ORACLE. **JDBC Overview**. 2011c. Disponível em: <http://www.oracle.com/technetwork/java/overview-141217.html>. Acesso em: 22 out. 2011.

ORACLE. **Toplink Overview**. 2011d. Disponível em: <http://www.oracle.com/technetwork/middleware/toplink/overview/index.html>. Acesso em: 22 out. 2011.

PARR, T. **Enforcing Strict Model-View Separation in Template Engines**. Publicado em: WWW '04: Proceedings of the 13th international conference on World Wide Web. ACM, 2004.

RAFFEL, M. **Template based C# CRUD code generator**, any database. 2010. Disponível em: <http://crudcodegen.codeplex.com/>. Acesso em: 25 set. 2011.

W3C. **Cascading Style Sheets home page**. 2011. Disponível em: <http://www.w3.org/Style/CSS/>. Acesso em: 16 out. 2011.

W3C. **XHTML™ 1.1** - Module-based XHTML - Second Edition. 2010. Disponível em: <http://www.w3.org/TR/xhtml11/>. Acesso em: 16 out. 2011.

7 ANEXOS

Nesta seção, são apresentados os códigos de alguns arquivos presentes na ferramenta. Além disso, são apresentados os códigos gerados no exemplo mostrado anteriormente.

7.1 Modelo *create.vm* para geração das páginas XHTML de criação de dados

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core">

    <ui:composition template="/templates/template.xhtml">

        <ui:define name="header">
            <h1>Empresa</h1>
        </ui:define>

        <ui:define name="body">

            <h1><h:outputText value="Novo
${nameCapitalized}"></h:outputText></h1>

            <h:panelGroup id="messagePanel" layout="block">
                <h:messages errorStyle="color: red" infoStyle="color:
green" layout="table"/>
            </h:panelGroup>

            <h:form>
                <h:panelGrid columns="2">
#foreach($field in $fields)
                    <h:outputLabel styleClass="field" value="${field.name}"
for="${field.name}" />
                    <h:inputText id="${field.name}" value="${field.value}"
title="" required="${field.isRequired}"
                            requiredMessage="O campo ${field.name} é
obrigatório." #if($field.isAutoIncrement())readonly="true" #end/>
#end
                </h:panelGrid>

                <br />
                <br />
                <h:commandLink styleClass="menu-link"
action="#{${nameCapitalized}Controller.create()}" value="Cadastrar" />
```

```
                <br />
                <h2>Menu</h2>

                <h:commandLink styleClass="menu-link"
action="#{${nameCapitalized}Controller.prepareList()}" value="Ver todos"
immediate="true"/>

                <h:commandLink styleClass="menu-link"
action="#{${nameCapitalized}Controller.index()}" value="Página Inicial"
immediate="true" />
                <br />
                <br />

            </h:form>

        </ui:define>
    </ui:composition>
</html>
```

7.2 Modelo *delete.vm* para geração das páginas XHTML de remoção de dados

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core">

    <ui:composition template="/templates/template.xhtml">

         <ui:define name="header">
            <h1>Empresa</h1>
        </ui:define>

        <ui:define name="body">

            <h1><h:outputText value="Excluir
${nameCapitalized}?"></h:outputText></h1>

            <h:panelGroup id="messagePanel" layout="block">
                <h:messages errorStyle="color: red" infoStyle="color:
green" layout="table"/>
            </h:panelGroup>

            <h:form>

                <h:panelGrid columns="2">
#foreach($field in $fields)
                    <h:outputText styleClass="field" value="${field.name}"
/>
                    <h:outputText value="${field.value}" />
#end
                </h:panelGrid>
```

```
                <br />
                <br />
                <h:commandLink styleClass="menu-link"
action="#{${nameCapitalized}Controller.delete()}" value="Excluir"/>
                <h:commandLink styleClass="menu-link"
action="#{${nameCapitalized}Controller.prepareList()}" value="Cancelar"/>

                <br />
                <h2>Menu</h2>

                <br />
                <h:commandLink styleClass="menu-link"
action="#{${nameCapitalized}Controller.prepareUpdate()}" value="Editar"/>

                <br />
                <h:commandLink styleClass="menu-link"
action="#{${nameCapitalized}Controller.prepareList()}" value="Ver todos"/>

                <br />
                <br />
                <h:commandLink styleClass="menu-link"
action="#{${nameCapitalized}Controller.index()}" value="Página Inicial"
immediate="true" />

            </h:form>
        </ui:define>
    </ui:composition>
</html>
```

7.3 Modelo *list.vm* para geração das páginas XHTML de visualização (listagem) de dados

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core">

    <ui:composition template="/templates/template.xhtml">

        <ui:define name="header">
            <h1>Empresa</h1>
        </ui:define>

        <ui:define name="body">

            <h1><h:outputText value="Lista de ${nameCapitalized}" /></h1>

            <h:form styleClass="${name}_list_form">

                <h:panelGroup id="messagePanel" layout="block">
```

```
                    <h:messages errorStyle="color: red" infoStyle="color:
green" layout="table"/>
                </h:panelGroup>

                    <h:outputText value="Sem informações cadastradas."
rendered="#{!${nameCapitalized}Controller.getItems().isRowAvailable()}" />

                    <h:dataTable value="${actionGetItems}" var="item"
rendered="#{${nameCapitalized}Controller.getItems().isRowAvailable()}"
border="0" cellpadding="0" cellspacing="0" styleClass="table"
headerClass="header" rowClasses="odd,even" rules="all">

#foreach($field in $fields)
                        <h:column>
                            <f:facet name="header">
                                <h:outputText value="${field.name}"/>
                            </f:facet>
                            <h:outputText value="#{item.${field.name}}"/>
                        </h:column>
#end

                        <h:column>

                            <f:facet name="header">
                                <h:outputText value=" "/>
                            </f:facet>
                            <h:commandLink
action="#{${nameCapitalized}Controller.prepareView(item)}" value="Ver"/>

                            <h:outputText value=" "/>
                            <h:commandLink
action="#{${nameCapitalized}Controller.prepareUpdate(item)}"
value="Alterar"/>

                            <h:outputText value=" "/>
                            <h:commandLink
action="#{${nameCapitalized}Controller.prepareDelete(item)}"
value="Excluir"/>

                        </h:column>

                    </h:dataTable>

                <br />
                <h2>Menu</h2>

                <h:commandLink styleClass="menu-link"
action="${actionPrepareCreate}" value="Cadastrar ${name}"/>

                <br />
                <br />
                <h:commandLink styleClass="menu-link"
action="${actionIndex}" value="Página Inicial" immediate="true" />

            </h:form>
        </ui:define>
    </ui:composition>

</html>
```

7.4 Modelo *view.vm* para geração das páginas XHTML de visualização de dados

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core">

    <ui:composition template="/templates/template.xhtml">

        <ui:define name="header">
            <h1>Empresa</h1>
        </ui:define>

        <ui:define name="body">

            <h1><h:outputText
value="${nameCapitalized}"></h:outputText></h1>

            <h:panelGroup id="messagePanel" layout="block">
                <h:messages errorStyle="color: red" infoStyle="color:
green" layout="table"/>
            </h:panelGroup>

            <h:form>

                <h:panelGrid columns="2">
#foreach($field in $fields)
                    <h:outputText styleClass="field" value="${field.name}"
/>
                    <h:outputText value="${field.value}" />
#end
                </h:panelGrid>

                <br />
                <h2>Menu</h2>

                <h:commandLink styleClass="menu-link"
action="#{${nameCapitalized}Controller.prepareDelete()}" value="Excluir" />

                <br />
                <h:commandLink styleClass="menu-link"
action="#{${nameCapitalized}Controller.prepareUpdate()}" value="Alterar" />

                <br />
                <h:commandLink styleClass="menu-link"
action="#{${nameCapitalized}Controller.prepareCreate()}" value="Novo
${nameCapitalized}" />

                <br />
                <h:commandLink styleClass="menu-link"
action="#{${nameCapitalized}Controller.prepareList()}" value="Listar todos"
/>
```

```
                <br />
                <br />
                <h:commandLink styleClass="menu-link"
action="#{${nameCapitalized}Controller.index()}" value="Página Inicial"
immediate="true" />

            </h:form>

        </ui:define>

    </ui:composition>

</html>
```

7.5 Modelo *update.vm* para geração das páginas XHTML de atualização de dados

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core">

    <ui:composition template="/templates/template.xhtml">

        <ui:define name="header">
            <h1>Empresa</h1>
        </ui:define>

        <ui:define name="body">

            <h1><h:outputText value="Alterar
${nameCapitalized}"></h:outputText></h1>

            <h:panelGroup id="messagePanel" layout="block">
                <h:messages errorStyle="color: red" infoStyle="color:
green" layout="table"/>
            </h:panelGroup>

            <h:form>

                <h:panelGrid columns="2">
#foreach($field in $fields)
                    <h:outputLabel value="${field.name}" for="$field.name"
/>
                    <h:inputText id="${field.name}" value="${field.value}"
title="${field.name}" required="${field.isRequired}" requiredMessage="O
campo ${field.name} é obrigatório."
#if($field.isAutoIncrement())readonly="true" #end/>
#end
                </h:panelGrid>
```

```
                <br />
                <br />
                <h:commandLink styleClass="menu-link"
action="#{${nameCapitalized}Controller.update()}" value="Salvar
alterações"/>

                <br />
                <h2>Menu</h2>

                <br />
                <h:commandLink styleClass="menu-link"
action="#{${nameCapitalized}Controller.prepareView()}" value="Ver"
immediate="true"/>

                <br />
                <h:commandLink styleClass="menu-link"
action="#{${nameCapitalized}Controller.prepareList()}" value="Listar todos"
immediate="true"/>

                <br />
                <br />
                <h:commandLink styleClass="menu-link"
action="#{${nameCapitalized}Controller.index()}" value="Página Inicial"
immediate="true" />

            </h:form>

        </ui:define>

    </ui:composition>

</html>
```

7.6 Modelo *business.vm* para geração das classes da camada de negócios

```
package ${businessPackage};

import ${entityPackage}.${capitalizedName};
import ${daoPackage}.${capitalizedName}DAO;
import java.util.List;

public class ${capitalizedName}Business {

    private ${capitalizedName}DAO ${name}DAO = null;

    public ${capitalizedName}Business() {
        ${name}DAO = new ${capitalizedName}DAO();
    }

    public void create(${capitalizedName} ${name}) throws Exception {
        ${name}DAO.create(${name});
    }

    public void update(${capitalizedName} ${name}) throws Exception {
        ${name}DAO.update(${name});
```

```
    }

    public void delete(${capitalizedName} ${name}) throws Exception {
        ${name}DAO.remove(${name});
    }

    public ${capitalizedName} find(${capitalizedName} ${name}) throws
Exception {
        return ${name}DAO.find(${name});
    }

    public List<${capitalizedName}> findAll() throws Exception {
        return ${name}DAO.findAll();
    }

}
```

7.7 Modelo *controllers.vm* para geração das classes de controladores ou *beans*

```
package ${controllerPackage};

import ${entityPackage}.${capitalizedName};
import ${businessPackage}.${capitalizedName}Business;
import java.io.Serializable;
import javax.faces.application.FacesMessage;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;
import javax.faces.context.FacesContext;
import javax.faces.model.DataModel;
import javax.faces.model.ListDataModel;

@ManagedBean(name = "${capitalizedName}Controller")
@SessionScoped
public class ${capitalizedName}Controller implements Serializable {

    private ${capitalizedName} current = null;
    private DataModel items = null;

    private ${capitalizedName}Business ${name}Business = null;

    public ${capitalizedName}Controller() {
        ${name}Business = new ${capitalizedName}Business();
    }

    public String create() {
        try {
            ${name}Business.create(current);

            FacesContext contexto = FacesContext.getCurrentInstance();
            FacesMessage mensagem = new FacesMessage("Cadastrado com
sucesso.");
            mensagem.setSeverity(FacesMessage.SEVERITY_INFO);
            contexto.addMessage(null, mensagem);
```

```
        } catch (Exception e) {
            FacesContext contexto = FacesContext.getCurrentInstance();
            FacesMessage mensagem = new FacesMessage("Erro ao tentar
cadastrar. Verifique as informações.");
            mensagem.setSeverity(FacesMessage.SEVERITY_ERROR);
            contexto.addMessage(null, mensagem);
        }

        prepareCreate();
        return null;
    }

    public String update() {
        try {
            ${name}Business.update(current);

            FacesContext contexto = FacesContext.getCurrentInstance();
            FacesMessage mensagem = new FacesMessage("Atualizado com
sucesso.");
            mensagem.setSeverity(FacesMessage.SEVERITY_INFO);
            contexto.addMessage(null, mensagem);

        } catch (Exception e) {
            FacesContext contexto = FacesContext.getCurrentInstance();
            FacesMessage mensagem = new FacesMessage("Erro ao tentar
atualizar. Verifique as informações.");
            mensagem.setSeverity(FacesMessage.SEVERITY_ERROR);
            contexto.addMessage(null, mensagem);
        }

        return null;
    }

    public String delete() {
        try {
            ${name}Business.delete(current);

            FacesContext contexto = FacesContext.getCurrentInstance();
            FacesMessage mensagem = new FacesMessage("Excluído com
sucesso.");
            mensagem.setSeverity(FacesMessage.SEVERITY_INFO);
            contexto.addMessage(null, mensagem);

        } catch (Exception e) {
            FacesContext contexto = FacesContext.getCurrentInstance();
            FacesMessage mensagem = new FacesMessage("Erro ao tentar
exluir. Verifique as informações.");
            mensagem.setSeverity(FacesMessage.SEVERITY_ERROR);
            contexto.addMessage(null, mensagem);
        }

        return "/pages/departamento/list";
    }


    public String prepareCreate() {
        current = new ${capitalizedName}();
        return "/pages/${name}/create";
    }
```

```
    public String prepareList() {
        current = null;
        items = null;
        return "/pages/${name}/list";
    }

    public String prepareView() {
        return "/pages/${name}/view";
    }

    public String prepareUpdate() {
        return "/pages/${name}/update";
    }

    public String prepareDelete() {
        return "/pages/${name}/delete";
    }

    public String prepareView(${capitalizedName} ${name}) {
        current = ${name};
        return "/pages/${name}/view";
    }

    public String prepareUpdate(${capitalizedName} ${name}) {
        current = ${name};
        return "/pages/${name}/update";
    }

    public String prepareDelete(${capitalizedName} ${name}) {
        current = ${name};
        return "/pages/${name}/delete";
    }

    public String index() {
        current = null;
        items = null;
        return "/index";
    }

    public DataModel getItems() throws Exception {
        this.items = new ListDataModel(${name}Business.findAll());
        return this.items;
    }

    public ${capitalizedName} getCurrent() {
        if (current == null) {
            current = new ${capitalizedName}();
        }
        return current;
    }

    public void setCurrent(${capitalizedName} current) {
        this.current = current;
    }
}
```

7.8 Modelo *dao.vm* para geração das classes da camada de acesso a dados

```
package ${daoPackage};

import ${entityPackage}.${name};
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

public class ${name}DAO extends AbstractDAO<${name}> {

    private EntityManagerFactory emf;
    private EntityManager em;

    @Override
    protected EntityManager getEntityManager() {
        return em;
    }

    public ${name}DAO() {
        super(${name}.class);
        emf = Persistence.createEntityManagerFactory("CRUDerPU");
        em = emf.createEntityManager();
    }
}
```

7.9 Modelo *entities.vm* para geração das classes das entidades

```
package ${entitiesPackage};

import java.io.Serializable;
import javax.persistence.*;

@Entity
@Table(name = "${tableName}")
public class ${className} implements Serializable {
#foreach($attrib in ${attributes})

#if(${attrib.isPrimaryKey()})
    @Id
#if(${attrib.isAutoIncrement()})
    @GeneratedValue(strategy = GenerationType.IDENTITY)
#end
#end
    @Column(name="${attrib.name}", length=${attrib.size},
nullable=${attrib.nullable})
    private ${attrib.type} ${attrib.name}#if($attrib.haveDefaultValue()) =
$attrib.defaultValue#end;
#end
#foreach($attrib in ${attributes})

    public ${attrib.type} get${attrib.getCapitalizedName()}() {
```

```
        return this.${attrib.name};
    }

    public void set${attrib.getCapitalizedName()}(${attrib.type}
${attrib.name}) {
        this.${attrib.name} = ${attrib.name};
    }
#end
}
```

7.10 Criador *CRUDPageCreator.java*

```
package creators;

import cruderjsf.Paths;
import metadata.Metadata;
import metadata.TableMetadata;
import generators.pages.CreateGenerator;
import generators.pages.DeleteGenerator;
import generators.pages.ListGenerator;
import generators.pages.UpdateGenerator;
import generators.pages.ViewGenerator;

public class CRUDPageCreator extends AbstractCreator {

    private Paths paths = null;

    public CRUDPageCreator(Metadata metadata, Paths paths) {
        super(metadata);
        this.paths = paths;
    }

    @Override
    public void create(String path) {

        createFolder(path);

        //Geradores de páginas
        CreateGenerator createGenerator = null;
        UpdateGenerator updateGenerator = null;
        DeleteGenerator deleteGenerator = null;
        ViewGenerator viewGenerator = null;
        ListGenerator listGenerator = null;

        String fileName = "";

        for (TableMetadata tableMetadata : this.metadata.getTables()) {

            //Cria diretório
            createFolder(path + tableMetadata.getName());

            //CREATE
            fileName = path + tableMetadata.getName() + "/create.xhtml";
            createGenerator = new CreateGenerator(tableMetadata, paths);
            try {
```

```
                createGenerator.generate(fileName);
            } catch (Exception ex) {
                System.err.println("[Create]SQL Exception: " +
ex.toString());
            }

            //UPDATE
            fileName = path + tableMetadata.getName() + "/update.xhtml";
            updateGenerator = new UpdateGenerator(tableMetadata, paths);
            try {
                updateGenerator.generate(fileName);
            } catch (Exception ex) {
                System.err.println("[Update]SQL Exception: " +
ex.toString());
            }

            //DELETE
            fileName = path + tableMetadata.getName() + "/delete.xhtml";
            deleteGenerator = new DeleteGenerator(tableMetadata, paths);
            try {
                deleteGenerator.generate(fileName);
            } catch (Exception ex) {
                System.err.println("[Delete]SQL Exception: " +
ex.toString());
            }

            //LIST
            fileName = path + tableMetadata.getName() + "/list.xhtml";
            listGenerator = new ListGenerator(tableMetadata, paths);
            try {
                listGenerator.generate(fileName);
            } catch (Exception ex) {
                System.err.println("[List]SQL Exception: " +
ex.toString());
            }

            //VIEW
            fileName = path + tableMetadata.getName() + "/view.xhtml";
            viewGenerator = new ViewGenerator(tableMetadata, paths);
            try {
                viewGenerator.generate(fileName);
            } catch (Exception ex) {
                System.err.println("[View]SQL Exception: " +
ex.toString());
            }
        }
    }
}
```

7.11 Criador *EntityCreator.java*

```
package creators;

import cruderjsf.Paths;
import generators.EntitieGenerator;
```

```
import java.sql.SQLException;
import metadata.Metadata;
import metadata.TableMetadata;

public class EntityCreator extends AbstractCreator {

    private Paths paths = null;

    public EntityCreator(Metadata metadata, Paths paths) {
        super(metadata);
        this.paths = paths;
    }

    @Override
    public void create(String path) {

        createFolder(path);

        EntitieGenerator entitieGenerator = null;

        String fileName = "";

        for (TableMetadata tableMetadata : this.metadata.getTables()) {

            fileName = path + tableMetadata.getCapilizedName() + ".java";

            entitieGenerator = new EntitieGenerator(tableMetadata, paths);

            try {
                entitieGenerator.generate(fileName);
            } catch (SQLException sqlex) {
                System.err.println("SQL Exception: " + sqlex.toString());
            } catch (NullPointerException npex) {
                System.err.println("NullPointerException: " +
npex.toString());
            } catch (Exception ex) {
                System.err.println("NullPointerException: " +
ex.toString());
            }
        }
    }
}
```

7.12 Gerador *CreateGenerator.java*

```
package generators.pages;

import cruderjsf.Paths;
import java.util.ArrayList;
import java.util.Collection;
import metadata.Field;
import metadata.FieldMetadata;
import metadata.TableMetadata;
import org.apache.velocity.VelocityContext;
import velocity.SimpleGenerator;
```

```
public class CreateGenerator extends SimpleGenerator {

    TableMetadata tableMetadata = null;
    private Paths paths = null;

    public CreateGenerator(TableMetadata tableMetadata, Paths paths) {
        this.templateName = paths.getCreate();
        this.tableMetadata = tableMetadata;
        this.paths = paths;
    }

    @Override
    protected void setContext(VelocityContext context) {

        context.put("nameCapitalized", tableMetadata.getCapilizedName());

        Collection<Field> fields = new ArrayList<Field>();
        Field newField = null;
        for (FieldMetadata field : tableMetadata.getFields()) {
            newField = new Field();
            newField.setName(field.getColumn_name());
            newField.setType(field.SQLtoJavaType());
            newField.setValue("#{" + tableMetadata.getCapilizedName() +
"Controller.current." + field.getColumn_name() + "}");
            if (field.getIs_autoincrement().equals("YES")) {
                newField.setAutoIncrement(true);
            }
            if(field.getIs_nullable().equals("NO")){
                newField.setIsRequired("true");
            }
            fields.add(newField);
        }

        context.put("fields", fields);

        String var = "item";
        context.put("var", var);

        context.put("actionCreate", "#{" + tableMetadata.getCapilizedName()
+ "Controller.create()}");
        context.put("actionUpdate", "#{" + tableMetadata.getCapilizedName()
+ "Controller.update()}");
        context.put("actionDelete", "#{" + tableMetadata.getCapilizedName()
+ "Controller.delete()}");
        context.put("actionList", "#{" + tableMetadata.getCapilizedName() +
"Controller.list()}");
        context.put("actionView", "#{" + tableMetadata.getCapilizedName() +
"Controller.view()}");

        context.put("actionPrepareCreate", "#{" +
tableMetadata.getCapilizedName() + "Controller.prepareCreate()}");
        context.put("actionPrepareUpdate", "#{" +
tableMetadata.getCapilizedName() + "Controller.prepareUpdate(" + var +
")}");
        context.put("actionPrepareDelete", "#{" +
tableMetadata.getCapilizedName() + "Controller.prepareDelete(" + var +
")}");
        context.put("actionPrepareList", "#{" +
tableMetadata.getCapilizedName() + "Controller.prepareList()}");
```

```
        context.put("actionPrepareView", "#{" +
tableMetadata.getCapilizedName() + "Controller.prepareView(" + var + ")}");

        context.put("actionIndex", "#{" + tableMetadata.getCapilizedName()
+ "Controller.index()}");
    }
}
```

7.13 Gerador *DeleteGenerator.java*

```
package generators.pages;

import cruderjsf.Paths;
import java.util.ArrayList;
import java.util.Collection;
import metadata.Field;
import metadata.FieldMetadata;
import metadata.TableMetadata;
import org.apache.velocity.VelocityContext;
import velocity.SimpleGenerator;

public class DeleteGenerator extends SimpleGenerator {

    TableMetadata tableMetadata = null;
    private Paths paths = null;

    public DeleteGenerator(TableMetadata tableMetadata, Paths paths) {
        this.templateName = paths.getDelete();
        this.tableMetadata = tableMetadata;
        this.paths = paths;
    }

    @Override
    protected void setContext(VelocityContext context) {

        context.put("name", tableMetadata.getName());
        context.put("nameCapitalized", tableMetadata.getCapilizedName());

        Collection<Field> fields = new ArrayList<Field>();
        Field newField = null;
        for (FieldMetadata field : tableMetadata.getFields()) {
            newField = new Field();
            newField.setName(field.getColumn_name());
            newField.setType(field.SQLtoJavaType());
            newField.setValue("#{" + tableMetadata.getCapilizedName() +
"Controller.current." + field.getColumn_name() + "}");
            fields.add(newField);
        }

        context.put("fields", fields);
    }
}
```

7.14 Gerador *UpdateGenerator.java*

```
package generators.pages;

import cruderjsf.Paths;
import java.util.ArrayList;
import java.util.Collection;
import metadata.Field;
import metadata.FieldMetadata;
import metadata.TableMetadata;
import org.apache.velocity.VelocityContext;
import velocity.SimpleGenerator;

public class UpdateGenerator extends SimpleGenerator {

    TableMetadata tableMetadata = null;
    private Paths paths = null;

    public UpdateGenerator(TableMetadata tableMetadata, Paths paths) {
        this.templateName = paths.getUpdate();
        this.tableMetadata = tableMetadata;
        this.paths = paths;
    }

    @Override
    protected void setContext(VelocityContext context) {

        context.put("name", tableMetadata.getName());
        context.put("nameCapitalized", tableMetadata.getCapilizedName());

        Collection<Field> fields = new ArrayList<Field>();
        Field newField = null;
        for (FieldMetadata field : tableMetadata.getFields()) {
            newField = new Field();
            newField.setName(field.getColumn_name());
            newField.setType(field.SQLtoJavaType());
            newField.setValue("#{" + tableMetadata.getCapilizedName() +
"Controller.current." + field.getColumn_name() + "}");
            if (field.getIs_autoincrement().equals("YES")) {
                newField.setAutoIncrement(true);
            }
            if(field.getIs_nullable().equals("NO")){
                newField.setIsRequired("true");
            }
            fields.add(newField);
        }

        context.put("fields", fields);

    }
}
```

7.15 Gerador *ListGenerator.java*

```
package generators.pages;

import cruderjsf.Paths;
import java.util.ArrayList;
import java.util.Collection;
import metadata.Field;
import metadata.FieldMetadata;
import metadata.TableMetadata;
import org.apache.velocity.VelocityContext;
import velocity.SimpleGenerator;

public class ListGenerator extends SimpleGenerator {

    TableMetadata tableMetadata = null;
    private Paths paths = null;

    public ListGenerator(TableMetadata tableMetadata, Paths paths) {
        this.templateName = paths.getList();
        this.tableMetadata = tableMetadata;
        this.paths = paths;
    }

    @Override
    protected void setContext(VelocityContext context) {

        context.put("name", tableMetadata.getName());
        context.put("nameCapitalized", tableMetadata.getCapilizedName());

        Collection<Field> fields = new ArrayList<Field>();
        Field newField = null;
        for (FieldMetadata field : tableMetadata.getFields()) {
            newField = new Field();
            newField.setName(field.getColumn_name());
            newField.setType(field.SQLtoJavaType());
            newField.setValue("#{" + tableMetadata.getCapilizedName() +
"Controller.current." + field.getColumn_name() + "}");
            fields.add(newField);
        }

        context.put("fields", fields);

        String var = "item";
        context.put("var", var);

        context.put("actionGetItems", "#{" +
tableMetadata.getCapilizedName() + "Controller.items}");

        context.put("actionCreate", "#{" + tableMetadata.getCapilizedName()
+ "Controller.create()}");
        context.put("actionUpdate", "#{" + tableMetadata.getCapilizedName()
+ "Controller.update()}");
        context.put("actionDelete",  "#{" +
tableMetadata.getCapilizedName() + "Controller.delete()}");
        context.put("actionList", "#{" + tableMetadata.getCapilizedName() +
"Controller.list()}");
```

```
        context.put("actionView", "#{" + tableMetadata.getCapilizedName() +
"Controller.view()}");

        context.put("actionPrepareCreate", "#{" +
tableMetadata.getCapilizedName() + "Controller.prepareCreate()}");
        context.put("actionPrepareUpdate", "#{" +
tableMetadata.getCapilizedName() + "Controller.prepareUpdate(" + var +
")}");
        context.put("actionPrepareDelete",  "#{" +
tableMetadata.getCapilizedName() + "Controller.prepareDelete(" + var +
")}");
        context.put("actionPrepareList", "#{" +
tableMetadata.getCapilizedName() + "Controller.prepareList()}");
        context.put("actionPrepareView", "#{" +
tableMetadata.getCapilizedName() + "Controller.prepareView(" + var + ")}");

        context.put("actionIndex",  "#{" + tableMetadata.getCapilizedName()
+ "Controller.index()}");
    }
}
```

7.16 Gerador *ViewGenerator.java*

```
package generators.pages;

import cruderjsf.Paths;
import java.util.ArrayList;
import java.util.Collection;
import metadata.Field;
import metadata.FieldMetadata;
import metadata.TableMetadata;
import org.apache.velocity.VelocityContext;
import velocity.SimpleGenerator;

public class ViewGenerator extends SimpleGenerator {

    TableMetadata tableMetadata = null;
    private Paths paths = null;

    public ViewGenerator(TableMetadata tableMetadata, Paths paths) {
        this.templateName = paths.getView();
        this.tableMetadata = tableMetadata;
        this.paths = paths;
    }

    @Override
    protected void setContext(VelocityContext context) {

        context.put("name", tableMetadata.getName());
        context.put("nameCapitalized", tableMetadata.getCapilizedName());


        Collection<Field> fields = new ArrayList<Field>();
        Field newField = null;
        for (FieldMetadata field : tableMetadata.getFields()) {
```

```
            newField = new Field();
            newField.setName(field.getColumn_name());
            newField.setType(field.SQLtoJavaType());
            newField.setValue("#{" + tableMetadata.getCapilizedName() +
"Controller.current." + field.getColumn_name() + "}");
            fields.add(newField);
        }

        context.put("fields", fields);
    }
}
```

7.17 Gerador *EntitiesGenerator.java*

```
package generators;

import cruderjsf.Paths;
import java.sql.Types;
import java.util.ArrayList;
import java.util.Collection;
import metadata.FieldMetadata;
import metadata.PrimaryKeyMetadata;
import metadata.TableMetadata;
import org.apache.velocity.VelocityContext;
import velocity.SimpleGenerator;

public class EntitieGenerator extends SimpleGenerator {

    private TableMetadata tableMetadata = null;
        private Paths paths = null;

    public EntitieGenerator(TableMetadata tableMetadata, Paths paths) {
        this.templateName = "templates/entities.vm";
        this.tableMetadata = tableMetadata;
        this.paths = paths;
    }

    @Override
    protected void setContext(VelocityContext context) {

        String tableName = tableMetadata.getName();
        String className = tableMetadata.getCapilizedName();

        Attribute newAttrib = null;
        Collection<Attribute> attributes = new ArrayList<Attribute>();

        for (FieldMetadata fieldMetadata : tableMetadata.getFields()) {

            newAttrib = new Attribute();

            newAttrib.setName(fieldMetadata.getColumn_name());

newAttrib.setType(SQLtoJavaTypes(fieldMetadata.getData_type()));

            newAttrib.setPrimaryKey(false);
```

```
            for (PrimaryKeyMetadata pksMetadata :
tableMetadata.getPrimaryKeys()){

if(fieldMetadata.getColumn_name().equals(pksMetadata.getColumn_name())){
                    newAttrib.setPrimaryKey(true);
                    break;
                }
            }


newAttrib.setAutoIncrement(fieldMetadata.getIs_autoincrement().equals("YES"
));
            newAttrib.setSize(fieldMetadata.getColumn_size());

newAttrib.setNullable(fieldMetadata.getIs_nullable().equals("YES"));
            newAttrib.setDefaultValue(fieldMetadata.getColumn_def());

            attributes.add(newAttrib);
        }

        context.put("entitiesPackage", paths.getEntityPackage());
        context.put("tableName", tableName);
        context.put("className", className);
        context.put("attributes", attributes);
    }

    public static String SQLtoJavaTypes(int type) {
        switch (type) {
            case Types.CHAR:
            case Types.VARCHAR:
            case Types.LONGVARCHAR:
                return "String";
            case Types.NUMERIC:
            case Types.DECIMAL:
                return "java.math.BigDecimal";
            case Types.BIT:
            case Types.BOOLEAN:
                return "boolean";
            case Types.TINYINT:
                return "byte";
            case Types.SMALLINT:
                return "short";
            case Types.INTEGER:
                return "int";
            case Types.BIGINT:
                return "long";
            case Types.REAL:
                return "float";
            case Types.FLOAT:
            case Types.DOUBLE:
                return "double";
            case Types.BINARY:
            case Types.VARBINARY:
            case Types.LONGVARBINARY:
                return "byte[]";
            case Types.DATE:
                return "java.sql.Date";
            case Types.TIME:
                return "java.sql.Time";
            case Types.TIMESTAMP:
                return "java.sql.Timestamp";
```

```
            case Types.CLOB:
                return "Clob";
            case Types.BLOB:
                return "Blob";
            case Types.ARRAY:
                return "Array";
            case Types.DISTINCT:
                return "mapping of underlying type";
            case Types.STRUCT:
                return "Struct";
            case Types.REF:
                return "Ref";
            case Types.DATALINK:
                return "java.net.URL";
            case Types.JAVA_OBJECT:
                return "underlying Java class";
            default:
                return null;
        }
    }
}
```

7.18 Página XHTML para criação de departamentos gerada no exemplo

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core">

    <ui:composition template="/templates/template.xhtml">

        <ui:define name="header">
            <h1>Empresa</h1>
        </ui:define>

        <ui:define name="body">

            <h1><h:outputText value="Novo
Departamento"></h:outputText></h1>

            <h:panelGroup id="messagePanel" layout="block">
                <h:messages errorStyle="color: red" infoStyle="color:
green" layout="table"/>
            </h:panelGroup>

            <h:form>
                <h:panelGrid columns="2">
                    <h:outputLabel styleClass="field" value="id_depto"
for="id_depto" />
                    <h:inputText id="id_depto"
value="#{DepartamentoController.current.id_depto}" title="" required="true"
requiredMessage="O campo id_depto é obrigatório." readonly="true" />
```

```
                    <h:outputLabel styleClass="field" value="nome"
for="nome" />
                    <h:inputText id="nome"
value="#{DepartamentoController.current.nome}" title=""
required="${field.isRequired}" requiredMessage="O campo nome é
obrigatório." />
                </h:panelGrid>

                <br />
                <br />
                <h:commandLink styleClass="menu-link"
action="#{DepartamentoController.create()}" value="Cadastrar" />

                <br />
                <h2>Menu</h2>

                <h:commandLink styleClass="menu-link"
action="#{DepartamentoController.prepareList()}" value="Ver todos"
immediate="true"/>

                <h:commandLink styleClass="menu-link"
action="#{DepartamentoController.index()}" value="Página Inicial"
immediate="true" />
                <br />
                <br />

            </h:form>

        </ui:define>
    </ui:composition>
</html>
```

7.19 Página XHTML para remoção de departamentos gerada no exemplo

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core">

    <ui:composition template="/templates/template.xhtml">

         <ui:define name="header">
            <h1>Empresa</h1>
        </ui:define>

        <ui:define name="body">

            <h1><h:outputText value="Excluir
Departamento?"></h:outputText></h1>

            <h:panelGroup id="messagePanel" layout="block">
```

```
                <h:messages errorStyle="color: red" infoStyle="color:
green" layout="table"/>
            </h:panelGroup>

            <h:form>

                <h:panelGrid columns="2">
                    <h:outputText styleClass="field" value="id_depto" />
                    <h:outputText
value="#{DepartamentoController.current.id_depto}" />
                    <h:outputText styleClass="field" value="nome" />
                    <h:outputText
value="#{DepartamentoController.current.nome}" />
                </h:panelGrid>


                <br />
                <br />
                <h:commandLink styleClass="menu-link"
action="#{DepartamentoController.delete()}" value="Excluir"/>
                <h:commandLink styleClass="menu-link"
action="#{DepartamentoController.prepareList()}" value="Cancelar"/>

                <br />
                <h2>Menu</h2>

                <br />
                <h:commandLink styleClass="menu-link"
action="#{DepartamentoController.prepareUpdate()}" value="Editar"/>

                <br />
                <h:commandLink styleClass="menu-link"
action="#{DepartamentoController.prepareList()}" value="Ver todos"/>

                <br />
                <br />
                <h:commandLink styleClass="menu-link"
action="#{DepartamentoController.index()}" value="Página Inicial"
immediate="true" />

            </h:form>
        </ui:define>
    </ui:composition>
</html>
```

7.20 Página XHTML para atualização de departamentos gerada no exemplo

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core">
```

```
    <ui:composition template="/templates/template.xhtml">

        <ui:define name="header">
            <h1>Empresa</h1>
        </ui:define>

        <ui:define name="body">

            <h1><h:outputText value="Alterar
Departamento"></h:outputText></h1>

            <h:panelGroup id="messagePanel" layout="block">
                <h:messages errorStyle="color: red" infoStyle="color:
green" layout="table"/>
            </h:panelGroup>

            <h:form>

                <h:panelGrid columns="2">
                    <h:outputLabel value="id_depto" for="id_depto" />
                    <h:inputText id="id_depto"
value="#{DepartamentoController.current.id_depto}" title="id_depto"
required="true" requiredMessage="O campo id_depto é obrigatório."
readonly="true" />
                    <h:outputLabel value="nome" for="nome" />
                    <h:inputText id="nome"
value="#{DepartamentoController.current.nome}" title="nome"
required="${field.isRequired}" requiredMessage="O campo nome é
obrigatório." />
                </h:panelGrid>

                <br />
                <br />
                <h:commandLink styleClass="menu-link"
action="#{DepartamentoController.update()}" value="Salvar alterações"/>

                <br />
                <h2>Menu</h2>

                <br />
                <h:commandLink styleClass="menu-link"
action="#{DepartamentoController.prepareView()}" value="Ver"
immediate="true"/>

                <br />
                <h:commandLink styleClass="menu-link"
action="#{DepartamentoController.prepareList()}" value="Listar todos"
immediate="true"/>

                <br />
                <br />
                <h:commandLink styleClass="menu-link"
action="#{DepartamentoController.index()}" value="Página Inicial"
immediate="true" />

            </h:form>

        </ui:define>

    </ui:composition>
```

```
</html>
```

7.21 Página XHTML para visualização (listagem) de departamentos gerada no exemplo

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core">

    <ui:composition template="/templates/template.xhtml">

        <ui:define name="header">
            <h1>Empresa</h1>
        </ui:define>

        <ui:define name="body">

            <h1><h:outputText value="Lista de Departamento" /></h1>

            <h:form styleClass="departamento_list_form">

                <h:panelGroup id="messagePanel" layout="block">
                    <h:messages errorStyle="color: red" infoStyle="color:
green" layout="table"/>
                </h:panelGroup>

                    <h:outputText value="Sem informações cadastradas."
rendered="#{!DepartamentoController.getItems().isRowAvailable()}" />

                    <h:dataTable value="#{DepartamentoController.items}"
var="item" rendered="#{DepartamentoController.getItems().isRowAvailable()}"
border="0" cellpadding="0" cellspacing="0" styleClass="table"
headerClass="header" rowClasses="odd,even" rules="all">

                        <h:column>
                            <f:facet name="header">
                                <h:outputText value="id_depto"/>
                            </f:facet>
                            <h:outputText value="#{item.id_depto}"/>
                        </h:column>
                        <h:column>
                            <f:facet name="header">
                                <h:outputText value="nome"/>
                            </f:facet>
                            <h:outputText value="#{item.nome}"/>
                        </h:column>

                        <h:column>

                            <f:facet name="header">
```

```
                                <h:outputText value=" "/>
                            </f:facet>
                            <h:commandLink
action="#{DepartamentoController.prepareView(item)}" value="Ver"/>

                            <h:outputText value=" "/>
                            <h:commandLink
action="#{DepartamentoController.prepareUpdate(item)}" value="Alterar"/>

                            <h:outputText value=" "/>
                            <h:commandLink
action="#{DepartamentoController.prepareDelete(item)}" value="Excluir"/>

                        </h:column>

                    </h:dataTable>

                <br />
                <h2>Menu</h2>

                <h:commandLink styleClass="menu-link"
action="#{DepartamentoController.prepareCreate()}" value="Cadastrar
departamento"/>

                <br />
                <br />
                <h:commandLink styleClass="menu-link"
action="#{DepartamentoController.index()}" value="Página Inicial"
immediate="true" />

            </h:form>
        </ui:define>
    </ui:composition>

</html>
```

7.22 Página para visualização de departamentos gerada no exemplo

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core">

    <ui:composition template="/templates/template.xhtml">

        <ui:define name="header">
            <h1>Empresa</h1>
        </ui:define>

        <ui:define name="body">

            <h1><h:outputText value="Departamento"></h:outputText></h1>
```

```
            <h:panelGroup id="messagePanel" layout="block">
                <h:messages errorStyle="color: red" infoStyle="color:
green" layout="table"/>
            </h:panelGroup>

            <h:form>

                <h:panelGrid columns="2">
                    <h:outputText styleClass="field" value="id_depto" />
                    <h:outputText
value="#{DepartamentoController.current.id_depto}" />
                    <h:outputText styleClass="field" value="nome" />
                    <h:outputText
value="#{DepartamentoController.current.nome}" />
                </h:panelGrid>

                <br />
                <h2>Menu</h2>

                <h:commandLink styleClass="menu-link"
action="#{DepartamentoController.prepareDelete()}" value="Excluir" />

                <br />
                <h:commandLink styleClass="menu-link"
action="#{DepartamentoController.prepareUpdate()}" value="Alterar" />

                <br />
                <h:commandLink styleClass="menu-link"
action="#{DepartamentoController.prepareCreate()}" value="Novo
Departamento" />

                <br />
                <h:commandLink styleClass="menu-link"
action="#{DepartamentoController.prepareList()}" value="Listar todos" />

                <br />
                <br />
                <h:commandLink styleClass="menu-link"
action="#{DepartamentoController.index()}" value="Página Inicial"
immediate="true" />

            </h:form>

        </ui:define>

    </ui:composition>

</html>
```

7.23 Entidade departamento gerada no exemplo

```
package entities;

import java.io.Serializable;
```

```
import javax.persistence.*;

@Entity
@Table(name = "departamento")
public class Departamento implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name="id_depto", length=10, nullable=false)
    private int id_depto;

    @Column(name="nome", length=50, nullable=true)
    private String nome;

    public int getId_depto() {
        return this.id_depto;
    }

    public void setId_depto(int id_depto) {
        this.id_depto = id_depto;
    }

    public String getNome() {
        return this.nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }
}
```

7.24 Controlador das páginas XHTML para departamentos gerado no exemplo

```
package controllers;

import entities.Departamento;
import business.DepartamentoBusiness;
import java.io.Serializable;
import javax.faces.application.FacesMessage;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;
import javax.faces.context.FacesContext;
import javax.faces.model.DataModel;
import javax.faces.model.ListDataModel;

@ManagedBean(name = "DepartamentoController")
@SessionScoped
public class DepartamentoController implements Serializable {

    private Departamento current = null;
    private DataModel items = null;

    private DepartamentoBusiness departamentoBusiness = null;

    public DepartamentoController() {
```

```
        departamentoBusiness = new DepartamentoBusiness();
    }

    public String create() {
        try {
            departamentoBusiness.create(current);

            FacesContext contexto = FacesContext.getCurrentInstance();
            FacesMessage mensagem = new FacesMessage("Cadastrado com
sucesso.");
            mensagem.setSeverity(FacesMessage.SEVERITY_INFO);
            contexto.addMessage(null, mensagem);

        } catch (Exception e) {
            FacesContext contexto = FacesContext.getCurrentInstance();
            FacesMessage mensagem = new FacesMessage("Erro ao tentar
cadastrar. Verifique as informações.");
            mensagem.setSeverity(FacesMessage.SEVERITY ERROR);
            contexto.addMessage(null, mensagem);
        }

        prepareCreate();
        return null;
    }

    public String update() {
        try {
            departamentoBusiness.update(current);

            FacesContext contexto = FacesContext.getCurrentInstance();
            FacesMessage mensagem = new FacesMessage("Atualizado com
sucesso.");
            mensagem.setSeverity(FacesMessage.SEVERITY_INFO);
            contexto.addMessage(null, mensagem);

        } catch (Exception e) {
            FacesContext contexto = FacesContext.getCurrentInstance();
            FacesMessage mensagem = new FacesMessage("Erro ao tentar
atualizar. Verifique as informações.");
            mensagem.setSeverity(FacesMessage.SEVERITY_ERROR);
            contexto.addMessage(null, mensagem);
        }

        return null;
    }

    public String delete() {
        try {
            departamentoBusiness.delete(current);

            FacesContext contexto = FacesContext.getCurrentInstance();
            FacesMessage mensagem = new FacesMessage("Excluído com
sucesso.");
            mensagem.setSeverity(FacesMessage.SEVERITY_INFO);
            contexto.addMessage(null, mensagem);

        } catch (Exception e) {
            FacesContext contexto = FacesContext.getCurrentInstance();
            FacesMessage mensagem = new FacesMessage("Erro ao tentar
exluir. Verifique as informações.");
            mensagem.setSeverity(FacesMessage.SEVERITY_ERROR);
```

```
            contexto.addMessage(null, mensagem);
        }

        return "/pages/departamento/list";
    }


    public String prepareCreate() {
        current = new Departamento();
        return "/pages/departamento/create";
    }

    public String prepareList() {
        current = null;
        items = null;
        return "/pages/departamento/list";
    }

    public String prepareView() {
        return "/pages/departamento/view";
    }

    public String prepareUpdate() {
        return "/pages/departamento/update";
    }

    public String prepareDelete() {
        return "/pages/departamento/delete";
    }

    public String prepareView(Departamento departamento) {
        current = departamento;
        return "/pages/departamento/view";
    }

    public String prepareUpdate(Departamento departamento) {
        current = departamento;
        return "/pages/departamento/update";
    }

    public String prepareDelete(Departamento departamento) {
        current = departamento;
        return "/pages/departamento/delete";
    }

    public String index() {
        current = null;
        items = null;
        return "/index";
    }

    public DataModel getItems() throws Exception {
        this.items = new ListDataModel(departamentoBusiness.findAll());
        return this.items;
    }

    public Departamento getCurrent() {
        if (current == null) {
            current = new Departamento();
        }
        return current;
```

```
    }

    public void setCurrent(Departamento current) {
        this.current = current;
    }
}
```