

# WebCasER - Gerador automático de código Java(JSP)

Felipe Ramos, Francine Bica

Curso de Análise e Desenvolvimento de Sistemas

Faculdade de Tecnologia Senac RS (FATEC/RS)

Porto Alegre – RS – Brasil

[felipe\\_ramos@sicredi.com.br](mailto:felipe_ramos@sicredi.com.br), [frbica@gmail.com](mailto:frbica@gmail.com)

**Abstract.** This article approaches the development of an automatically generator of source code in Java(JSP) to WEB systems based in a template ER. The automatic generation based in ER models is a technique that turns possible to improve development process productivity. The template ER contains informations sufficient to development automatically the big part of source code of system. The WebCasER is a tool CASE capable of extract by one template ER created into the DBDesiner the datas necessary for creating of the codes Java/JSP, supporting relations type 1-1 and 1-N, with the finality of optimizing the process of development of software.

**Resumo.** Este artigo aborda o desenvolvimento de um gerador automático de código fonte em Java(JSP) para sistemas Web a partir de uma modelagem ER. A geração automática de código a partir de modelos ER é uma técnica que permite aumentar significativamente a produtividade no processo de desenvolvimento de software. O modelo ER contém informações suficientes para a geração automática de grande parte do código do sistema. O WebCasER é uma ferramenta CASE capaz de extrair de um modelo ER criado no DBDesiner os dados necessários para criação de códigos Java/JSP, suportando relacionamento do tipo 1-1 e 1-N com a finalidade de otimizar o processo de desenvolvimento de software.

## 1. Introdução

Um processo de desenvolvimento de *software* é um conjunto de atividades, parcialmente ordenadas, com a finalidade de obter um produto de *software*. Existem diversos modelos de processo criados para auxiliar no desenvolvimento de *software*, como modelos em cascata, espiral e a metodologia ágil (FOWLER, 2001). As atividades dos processos de *software* podem ser apoiadas por ferramentas CASE (*Computer-Aided Software Engineering*). Segundo Sommerville (2003), a tecnologia CASE proporciona apoio ao processo de *software* pela automação de algumas atividades de processo e pelo fornecimento de informações sobre o *software* que está sendo desenvolvido.

Podem ser citadas como exemplos de linguagens de especificação de transformações e mecanismos para geração automática de código o CodeCharge (SANTOS, 2002), uma ferramenta para geração de códigos Web para acesso a banco de dados (SANTOS, 2003) e o Jakarta Velocity (VELOCITY, 2007) do grupo Apache que é uma ferramenta de geração de código Web com Java.

Neste contexto a ferramenta WebCasER foi desenvolvida como forma de otimizar o processo de desenvolvimento de *software* de forma prática, rápida, segura e menos burocrática, podendo ser utilizada para geração de protótipos de teste. O objetivo do WebCasER é auxiliar o trabalho de desenvolvimento de *software* na etapa de criação de código Java(JSP) para que esse atenda o que foi modelado de acordo com as necessidades do cliente. Com isso evitam-se alguns problemas oriundos da etapa de codificação, uma vez que, no momento da codificação pode ocorrer alguns imprevistos, tais como: (1) Divergências entre o desenvolvido e o que realmente foi modelado; (2) Não seguir um padrão de codificação; (3) Falhas na codificação que ocasionarão re-trabalho e (4) Perda de tempo na criação de códigos que poderiam ser padronizados e agilizariam o processo de desenvolvimento (SOMMERVILLE, 2003).

Paralelamente a isso o fato de existirem poucas ferramentas capazes de desenvolver de forma rápida aplicações Java(JSP) a partir de um modelo ER e principalmente, que façam isso de forma gratuita, foram pontos decisivos na definição do WebCasER.

Pelo fato dos modelos ER possuírem uma grande gama de informações úteis ao desenvolvimento de um *software*, é possível utilizar tais informações como forma de automatizar o processo de desenvolvimento, sendo assim, o WebCasER é capaz de compreender o XML (*eXtensible Markup Language*) gerado pelo DBDesigner e gerar a codificação das páginas JSP, evitando erros comuns ocorridos durante a criação do programa, trazendo maior confiabilidade ao resultado final em relação ao que foi inicialmente modelado. Além disso, o tempo para desenvolvimento é reduzido, podendo direcionar um maior esforço em atividades relacionadas ao negócio, documentação e modelagem.

A ferramenta WebCasER utiliza um arquivo de configuração XML onde são relacionados todos os *templates* a serem utilizados para a geração dos códigos, além de *templates* com extensão própria. É uma ferramenta extensível e alterável que atende a geração de relacionamentos do tipo 1-1 e 1-N.

Este artigo está dividido da seguinte forma: na seção 2 são apontados os principais conceitos e técnicas utilizadas no desenvolvimento desta ferramenta. Na seção 3 a ferramenta WebCasER é descrita. Na seção 4 são apresentados os testes realizados e a seção 5 as conclusões e trabalhos futuros são descritos.

## 2. Ferramentas Case

Ferramentas CASE são ferramentas que apóiam as atividades de processo de software, como atividades de planejamento; edição; gerenciamento de mudança; gerenciamento de configuração; prototipação; apoio a métodos; processamento de linguagem; análise de programas; testes; depuração; documentação e reengenharia (SOMMERVILLE, 2003).

Dentre as categorias de ferramentas CASE que prestam apoio às atividades da engenharia de *software*, a ferramenta aqui descrita classifica-se como uma ferramenta de prototipação, pois destina-se a construção de sistemas, possuindo a capacidade de gerar código a partir de um determinado modelo.

Atualmente, existem inúmeras ferramentas CASE que auxiliam o processo de geração de código fonte. Três dessas ferramentas têm forte relação com o presente trabalho, apresentando características importantes. São elas:

- CodeCharge: Gerador de códigos para aplicação Web. O CodeCharge é um a ferramenta *Rapid Application Development* (RAD) desenvolvida pela empresa norte americana Yes Software Corporation que integra a função de geração de código para aplicativos de acesso a banco de dados via Web, em um ambiente IDE bastante simples. Como características possui acesso a diversos bancos de dados, linguagens e diferentes plataformas (SANTOS, 2002).
- Geração automática de código a partir de caso de uso: Efetua a geração automática de código a partir de documentos de casos de uso, efetuando a substituição da linguagem natural existente nos casos de uso produzindo códigos automaticamente (SANTOS, 2003).
- Velocity: um *software* do projeto Jakarta, independente de linguagem e bastante utilizado para geração de código. O Velocity é um *Template-Engine* feito em Java, ele é um conjunto de classes, e não um programa diferente, em outra linguagem. Uma de suas maiores utilidades é no desenvolvimento de aplicações Web, onde o código Java fica totalmente separado do código HTML, tornando assim a aplicação muito mais modularizada e fácil de manter (VELOCITY, 2007).

O WebCasER possui características semelhantes as três ferramentas acima descritas, sendo que destas pode-se destacar a geração de páginas web com acesso a banco de dados e geração a partir de um modelo de entrada, neste caso um modelo ER. Os principais diferenciais do WebCasER são a geração a partir de *templates* pré-definidos; facilidade de adaptação dos *templates* a necessidade do desenvolvedor e facilidade na configuração de conexões de banco, *package* entre outras informações descritas na seção 3.

### **3. Delineamento da Ferramenta**

A ferramenta CASE WebCasER tem como objetivo criar códigos Java (JSP) a partir de diagramas ER gerados pela ferramenta DBDesigner. O DBDesigner é um editor visual para criação de banco de dados que integra a criação, modelagem e manutenção de bancos. É uma ferramenta *opensource* distribuído sobre licença *GPL* (*General Public License*).

O WebCasER foi desenvolvido em Java, utilizando a API Swing para a criação de sua interface gráfica. Alguns fatores relacionados a linguagem Java foram determinantes para a sua escolha, como por exemplo: a geração de páginas JSP; a portabilidade da ferramenta; a robustez e versatilidade da linguagem Java; além de ser uma linguagem amplamente difundida entre os meios acadêmicos (HORSTMANN, 2004).

A ferramenta não possui tabelas relacionais. A configuração e padronizações de códigos, utilizados na transformação do modelo ER em uma aplicação Java, são armazenadas em arquivos XML e uma extensão própria identificada pela sigla WCE. Na interpretação do ER e geração dos códigos, são contemplados relacionamentos 1-1 e 1-N.

A ferramenta é constituída de um executável, um arquivo de configuração denominado *config\_webcaser.xml* e um diretório de *templates*, contendo *templates* para a geração de código. O arquivo *config\_webcaser.xml* possui todas as informações (Tabela 1) necessárias para identificar quais templates serão utilizados assim como

informações necessárias para a geração dos arquivos Java(JSP). Ao efetuar a geração dos códigos a ferramenta abrirá o arquivo *config\_webcaser.xml* para identificar quais serão os arquivos templates que ela deverá utilizar na geração dos códigos. Caso seja criado um novo arquivo de *template*, o mesmo deverá ser incluído neste arquivo de configuração, pois somente desta forma será possível utilizá-lo na geração dos arquivos.

**Tabela 1. Estrutura do arquivo config\_webcaser.xml**

TAG	Descrição
Nome	Nome do <i>template</i>
Descricao	Descrição do arquivo <i>template</i>
Tipo	Os templates podem ser de dois tipos: “ÚNICO” – Indica que será gerado UM arquivo apenas “TABELA” – Indica que o template será gerado para cada uma das tabelas modeladas no ER
arquivoTemplate	Nome do arquivo template (extensão WCE)
nomeAntes	Incremento do nome do arquivo a ser gerado, concatenado anteriormente ao nome da tabela
nomeDepois	Incremento do nome do arquivo a ser gerado, concatenado posteriormente ao nome da tabela
extensoao	Extensão do arquivo gerado

A Figura 1 demonstra o *layout* do arquivo de configuração e a divisão de cada *template* em *tags*. Pode-se visualizar na linha destacada na Figura 1 a configuração de um *template* denominado Servlet, além de identificar algumas informações do mesmo como: *template* gerado para todas as tabelas do modelo ER, nome do arquivo usado como *template* Servlet.wce, extensão .java e complemento de nome “srv”.

```

- <webcaser>
- <templates>
  <template name="Model" descricao="Template de Classe" tipo="TABELA" arquivoTemplate="Model.wce"
    nomeAntes="" nomeDepois="" extensoao="java" />
  <template name="ViewList" descricao="View para Listar Registros" tipo="TABELA"
    arquivoTemplate="JspList.wce" nomeAntes="" nomeDepois="List" extensoao="jsp" />
  <template name="View" descricao="View para Cadastrar/Alterar" tipo="TABELA"
    arquivoTemplate="Jsp.wce" nomeAntes="" nomeDepois="" extensoao="jsp" />
  <template name="Servlet" descricao="Servlet" tipo="TABELA" arquivoTemplate="Servlet.wce"
    nomeAntes="srv" nomeDepois="" extensoao="java" />
  <template name="Dao" descricao="Arquivos Dao" tipo="TABELA" arquivoTemplate="Dao.wce" nomeAntes=""
    nomeDepois="Dao" extensoao="java" />
  <template name="index" descricao="Pagina de Index" tipo="UNICO" arquivoTemplate="Index.wce"
    nomeAntes="" nomeDepois="" extensoao="jsp" />
  <template name="web" descricao="Web.xml" tipo="UNICO" arquivoTemplate="Web.wce" nomeAntes=""
    nomeDepois="" extensoao="xml" />
  <template name="errorPage" descricao="Página para redirecionamento de erros" tipo="UNICO"
    arquivoTemplate="errorPage.wce" nomeAntes="" nomeDepois="" extensoao="jsp" />
  <template name="DBConnection" descricao="Conexão com Banco" tipo="UNICO"
    arquivoTemplate="DBConnection.wce" nomeAntes="" nomeDepois="" extensoao="java" />
</templates>
</webcaser>
```

**Figura 1. Arquivo de configuração config\_webcaser.xml**

Os arquivos com extensão WCE são os *templates* utilizados para geração dos arquivos, e podem ser visualizados no Anexo de Modelagem – Seção 4. São compostos basicamente de códigos Java e palavras reservadas ou *tags*, as quais serão substituídas por informações referentes às tabelas, campos e configurações, detalhados no Anexo de Modelagem seção 6.1, 6.2 e 6.3. Todas as palavras reservadas são de uso exclusivo do WebCasER, não podendo ser utilizadas no código padrão dos *templates*. A ferramenta possui inicialmente nove *templates*, sendo eles:

- Index.WCE: usado na geração do arquivo com extensão JSP que apresentará o índice das páginas geradas;
- ErrorPage.WCE: usado na geração de arquivo com extensão JSP que conterá a página de erros a ser invocada nas *exceptions*;
- web.WCE: usado na geração de arquivo com extensão XML. Utilizado na geração do arquivo web.xml de configuração de aplicações web;
- DBConnection.WCE: usado na geração de arquivo único com extensão JAVA. Utilizado para conexão com banco de dados;
- JspList.WCE e Jsp.WCE: usado na geração de arquivos com extensão JSP para cada uma das tabelas. Utilizado geração de páginas para listar, incluir, excluir e alterar dados;
- Model.WCE: usado na geração de arquivos com extensão JAVA para cada uma das tabelas. Utilizado na geração de classes;
- Servlet.WCE: usado na geração de arquivos com extensão JAVA para cada uma das tabelas. Utilizado na geração de servlets para controle das ações solicitadas pelas páginas;
- Dao.WCE: usado na geração de arquivos com extensão JAVA para cada uma das tabelas. Utilizado na geração de classes de acesso a dados (DAO).

Para os *templates* configurados como tipo ÚNICO, será gerado somente um arquivo efetuando substituições de palavras reservados por dados configurados em tela ou informações de tabelas. Já para os *templates* de tipo TABELA, é gerado um arquivo para cada uma das tabelas modeladas no ER e selecionadas em tela, fazendo a substituição das palavras reservadas por informações configuradas em tela, informações de tabelas e seus campos.

A utilização de XML e *templates* (WCE) se deu por agregar vantagens como:

- O XML é uma linguagem de marcação de dados que provê um formato para descrever dados estruturados. Um arquivo XML possui *tags* que servem para armazenar dados de forma estruturada e organizada, facilitando o acesso às informações (FARIA, 2005);
- O arquivo XML de configuração poderá ser alterado pelo próprio usuário, adaptando-o as suas necessidades. Incluindo e excluindo novos *templates* a serem codificados;
- Os arquivos WCE são constituídos de código Java associado a utilização das palavras reservadas da ferramenta. Isto faz com que possam ser alterados pelos próprios usuários adaptando-os as suas necessidades e padrões de codificação;
- Facilidade em adquirir uma nova configuração, sendo que basta armazenar os *templates* na pasta correspondente e ajustar o arquivo de configuração para começar a codificar novos fontes.

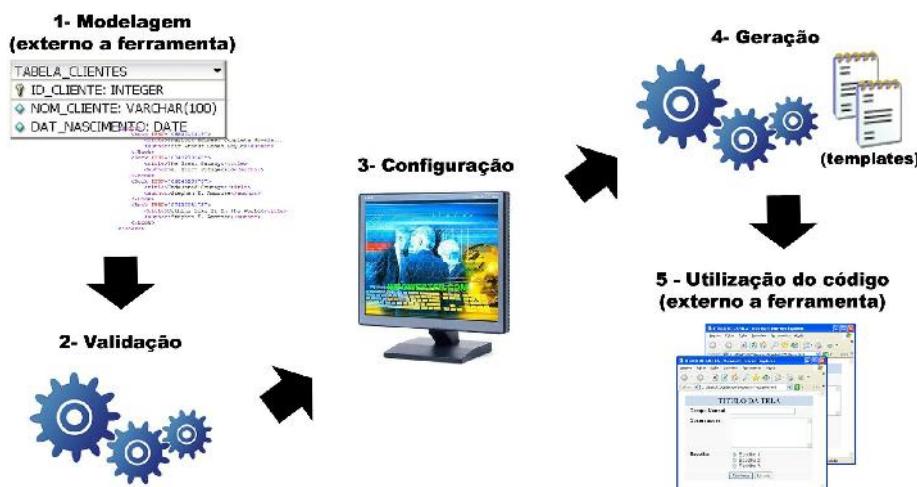
É possível incluir *template*, para isso é necessário seguir os seguintes passos:

- Criar um novo arquivo com extensão .wce
- O conteúdo do arquivo deverá conter os códigos padrões a serem utilizados;

- Nos pontos do código onde são necessárias informações de tabelas e campos, ou algum dado que possa ser parametrizável, conforme possibilidades oferecidas pela ferramenta, devem ser utilizadas as palavras reservadas;
- Para os blocos de código que devem ser repetidos para alguma situação específica, como por exemplo: Repetir um código específico para todas as colunas do tipo FK, devem ser utilizadas as *tags* de repetição atendidas pela ferramenta, conforme Anexo de Modelagem - seção 6.4.
- Após a criação do arquivo de *template* é necessário incluí-lo no arquivo config\_webcaser.xml, para que a ferramenta possa identificá-lo como um *template* a ser utilizado, além de informar os parâmetros descritos na tabela 1.

### 3.1 Do ER ao Código Java(JSP)

O processo de criação de códigos Java(JSP) é composto de cinco passos. A Figura 1 representa o caminho dessa criação, o qual inicia-se pela modelagem no DBDesigner, e, após configurações na ferramenta chega-se ao resultado final .



**Figura 2. Processo de criação de uma página JSP– passos 1 ao 5.**

As seções 3.1.1 a 3.1.5 detalham cada um dos passos ilustrado na Figura 2.

#### 3.1.1 Modelagem

A modelagem das tabelas no DBDesigner é o primeiro passo. Nesse ponto o usuário deve:

- Modelar tabelas e relacionamentos, preferencialmente preenchendo todas as informações descritas as tabelas 1, 2 e 3, como por exemplo: nome da coluna, se é chave primária, nome do relacionamento entre tabelas; Para auxiliar o trabalho do desenvolvedor todo o padrão a ser seguido no momento da criação do modelo ER está explicado no Help da ferramenta, descrito passo a passo na seção 10 do anexo de modelagem. .
- Salvar modelagem.

Para se obter sucesso na modelagem de forma que se possa, a partir do ER, construir um código de qualidade é necessário seguir um padrão de modelagem onde devem ser informados alguns atributos obrigatórios. Esse padrão foi definido levando em consideração a facilidade na interpretação do ER para o correto desenvolvimento das páginas JSP. Com isso obtém-se uma maior confiabilidade no resultado esperado não deixando que alguma informação possa ser codificada de maneira errada.

A Figura 3 destacada as telas do DBDesigner que servem para configuração do ER, como por exemplo a criação dos campos e seus atributos (A) e as relações entre cada tabela (B). Essas configurações são baseadas no modelo ER criado (C).

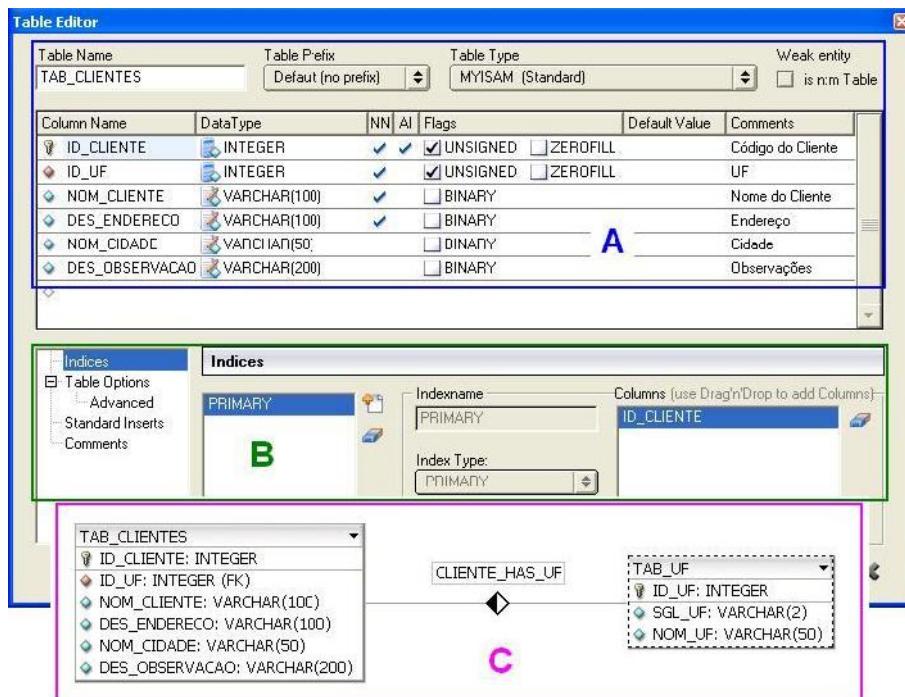


Figura 3. Parametrização de tabelas, campos, índices e joins no DBDesigner

A partir das tags <DATATYPES> e <METADATA> existentes no arquivo XML gerado pelo DBDesigner é possível criar as páginas JSP para acessar as tabelas. O grupo de tags <DATATYPES> (Tabela 2) define quais os tipos de dados podem ser atribuídos aos campos criados. A partir dessa informação é possível definir o tipo de campo a ser criado em tela para receber os dados informados pelo usuário. A Figura 3 (A) apresenta dois exemplos de DATATYPES válidos: INTEGER e VARCHAR.

Como exemplo da tag completa tem-se a seguinte linha:

```
<< <DATATYPE ID="5" IDGroup="0" TypeName="INTEGER" Description="A normal-size integer. The signed range is -2147483648 to 2147483647. The unsigned range is 0 to 4294967295." ParamCount="1" OptionCount="2" ParamRequired="0" EditParamsAsString="0" SynonymGroup="1" PhysicalMapping="0" PhysicalTypeName="">.
```

**Tabela 2. Tags usadas na identificação dos tipos de dados (Tag <DATATYPE>)**

<b>TAG</b>	<b>Descrição</b>
ID	Identificador do tipo. Este ID estará armazenado nas <i>tags</i> de campos a fim de identificar o tipo do campo.
TypeName	Nome da TAG. Utilizado para referenciar um tipo de campo utilizado na criação do banco. Exemplo: INTEGER, VARCHAR

O grupo de *tags* <METADATA> (Tabelas 3, 4 e 5) possui as informações de tabelas, campos, índices e joins modelados no ER. Além de serem usadas para a criação do banco de dados também são as referências para a criação das páginas JSP, desde que seguido o padrão de documentação do ER.

**Tabela 3. Tags usadas na identificação de tabelas (Tag <TABLES>)**

<b>TAG</b>	<b>Descrição</b>
ID	Identificador da tabela. Utilizado para linkar tabela dentro do XML com outros componentes
Tablename	Nome da tabela a ser utilizado dentro do sistema. Será usado nas classes de banco para armazenar dados.
Comments	Comentários que serão utilizados para nomear tela de cadastro e título de janelas.

**Tabela 4. Tags usadas na identificação das Colunas de tabelas (Tag <COLUMNS>)**

<b>TAG</b>	<b>Descrição</b>
ID	Identificador da coluna. Utilizado para linkar coluna dentro do XML com outros componentes
ColName	Nome da coluna dentro da tabela. Será usado nas classes de banco para armazenar dados
idDatatype	Código que indicará o tipo de dado. Existe no início do xml um grupo de <i>tags</i> que define cada tipo
DatatypeParams	Indica o tamanho do campo.
PrimaryKey	0-indica que não é uma primary key 1-indica que é uma primary key Será usado para definir campos de pesquisa
NotNull	Usado para indicar em tela campos obrigatórios de informação. Campos obrigatórios são destacados em tela. 0-Campo pode ser nulo 1-Campo obrigatório.
AutoInc	Toda tabela obrigatoriamente precisa possuir sua <i>primary key</i> como <i>autoinc</i> Usado para auto incrementar campos. Campo não será editável em tela. 0-Campo normal 1-Campo auto incremento
IsForeignKey	0-Campo normal 1-FK Usado para que se for FK, o campo será de busca
DefaultValue	Valor default a ser exibido em tela
Comments	Usado para exibir label de campos em tela

**Tabela 5. Tags usadas na identificação das relações de tabelas (Tag <RELATIONS>)**

<b>TAG</b>	<b>Descrição</b>
ID	Identificador da relação. Utilizado para linkar relação dentro do XML com outros componentes
RElationName	Nome da relação
SrcTable	ID da tabela de origem
DestTable	ID da tabela de destino

### 3.1.2 Validação do XML

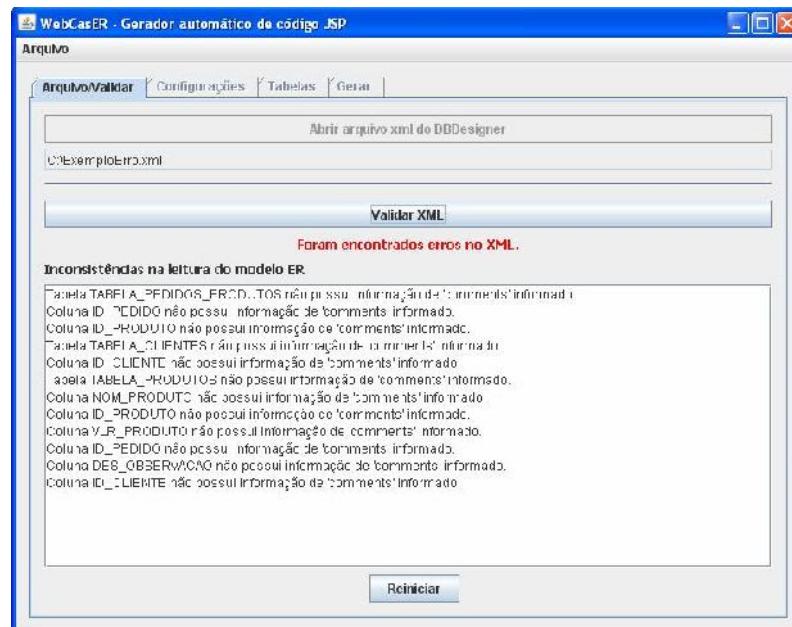
O passo 2 representa a identificação e leitura do arquivo XML (exemplificado na Figura 2). Para tanto o usuário deve abrir o arquivo XML gerado pelo DBDesigner e validá-lo. A Figura 4 exibe um trecho do arquivo XML gerado a partir da modelagem exibida na Figura 3.

```
<?xml version='1.0' standalone='yes'?>
<?MODEL Version='4.0'?>
+ <SETTINGS>
- <METADATA>
  <REGIONS />
- <STARI FS>
  <TABLE ID='1001' Tablename='TAB_CLIENTES' PrevTableName='Table_01' XPos='25' YPos='66' TableType='0' TablePrefix='0' nmTable='0'
    TempId='0' UseStandardInserts='0' StandardInserts='1' TableOptions='DelayKeyTblUpdates=0\!nPackKeys=0\!nRowChecksum=0
    \nRowId=0\!nUserKaid=0\!nKaidType=0\!n' Comments='Cadastro de Clientes' Collapsed='0' IsLinkedObject='0' nLinkedMode='1'
    Obj_d_Lin kod='1' OrderLog='2'>
    - <COLUMNS>
      - <COLUMN ID='1003' ColName='ID_CLIENTE' PrevColName='' Pos='0' idDatatype='5' DatatypeParams='' Width='1' Prec='1'
        PrimaryKey='1' NotNull='1' AutoInc='1' IsForeignKey='0' DefaultValue='' Comments='Código do Cliente'>
        + <OPTIONSELECTED>
      </COLUMN>
      <COLUMN ID='1211' ColName='ID_UF' PrevColName='' Pos='1' idDatatype='5' DatatypeParams='' Width='1' Prec='1' PrimaryKey='0'
        NotNull='1' AutoInc='0' IsForeignKey='1' DefaultValue='' Comments='UF'>
        + <OPTIONSELECTED>
      </COLUMN>
      - <COLUMN ID='1005' ColName='NOME_CLIENTE' PrevColName='NOME_CLIENTE' Pos='1' idDatatype='20' DatatypeParams='(100)'
        Width='1' Prec='1' PrimaryKey='0' NotNull='1' AutoInc='0' IsForeignKey='0' DefaultValue='' Comments='Nome do Cliente'>
        + <OPTIONSELECTED>
      </COLUMN>
      <COLUMN ID='1170' ColName='DES_ENDERECHO' PrevColName='' Pos='3' idDatatype='20' DatatypeParams='(100)' Width='1' Prec='1'
        PrimaryKey='0' NotNull='1' AutoInc='0' IsForeignKey='0' DefaultValue='' Comments='Endereço'>
        + <OPTIONSELECTED>
      </COLUMN>
      - <COLUMN ID='1180' ColName='NOM_CIDADE' PrevColName='' Pos='4' idDatatype='20' DatatypeParams='(50)' Width='1' Prec='1'
        PrimaryKey='0' NotNull='0' AutoInc='0' IsForeignKey='0' DefaultValue='' Comments='Cidade'>
        + <OPTIONSELECTED>
      </COLUMN>
      - <COLUMN ID='1182' ColName='DES_OBSERVACAO' PrevColName='' Pos='6' idDatatype='20' DatatypeParams='(200)' Width='1' Prec='1'
        PrimaryKey='0' NotNull='0' AutoInc='0' IsForeignKey='0' DefaultValue='' Comments='Observação'>
        + <OPTIONSELECTED>
      </COLUMN>
    </COLUMNS>
```

Figura 4. XML do ER gerado pelo DBDesigner

Para a leitura e transformação das *tags* utilizadas do XML é usado um pacote específico do Java para trabalhar com XML. Este pacote (`org.w3c.dom`) disponibiliza os recursos necessários para a busca de informações dentro do XML. Nesse ponto, ilustrado na Figura 5, a ferramenta verifica as *tags* obrigatórias conforme as tabelas 3, 4 e 5 (descritas na seção 3.1.1).

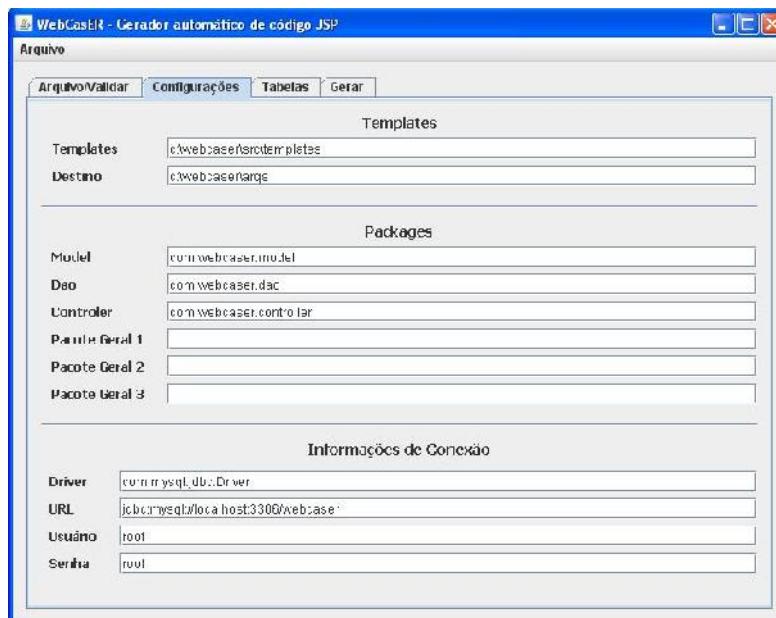
A ferramenta somente permitirá prosseguir caso a validação seja positiva. As informações do XML são lidas e armazenadas em memória para posteriormente serem utilizadas na criação da codificação. Em caso de problemas durante a validação, a ferramenta exibirá os erros encontrados (Figura 5), como por exemplo: nenhuma tabela modelada no ER, diretórios de templates ou destino não informados ou inválidos, etc. A lista completa dos erros está no Anexo de Modelagem – seção 7.



**Figura 5. Abertura e validação do ER.**

### 3.1.3 Configurações

O terceiro passo compreende nas configurações que o usuário poderá fazer antes da geração do código. A Figura 6 apresenta a aba “Configuração” e os dados que devem ser preenchidos. O usuário deverá informar o diretório que contém os *templates* e o config\_webCaser.xml, além do diretório utilizado para a geração dos arquivos Java(JSP).



**Figura 6. Informações de configuração geral**

Também deverá ser informado o nome dos *packages* (pacotes) que serão utilizados na geração dos fontes. São solicitados seis nomes de pacotes, sendo os três primeiros relacionados a *templates* adaptados ao padrão *MVC – Model View and Controller*, além de permitir a informação de mais três novos *packages*, denominados “Pacotes Geral X”. Em java os *packages* servem para agrupar classes relacionadas, dessa forma, a ferramenta possibilita que o usuário defina os nomes dos pacotes que deseja utilizar e com a utilização de palavras reservadas dentro dos *templates*, faça a substituição destas pelo nome dos *pacotes* configurados em tela.

Os dados para conexão como driver, url, usuário e senha para conexão também são informados neste ponto.

A aba “Tabelas”, representado na Figura 7, apresenta todas as tabelas constantes no arquivo XML validado no passo 2. Neste ponto o usuário irá definir quais tabelas pretende gerar, além de poder ajustar algumas informações dos campos da tabela, como: Alterar o *label* do campo e definir qual campo será utilizado como descrição da tabela em relacionamentos com outras tabelas (pontos no qual o registro é uma chave estrangeira em outra tabela).

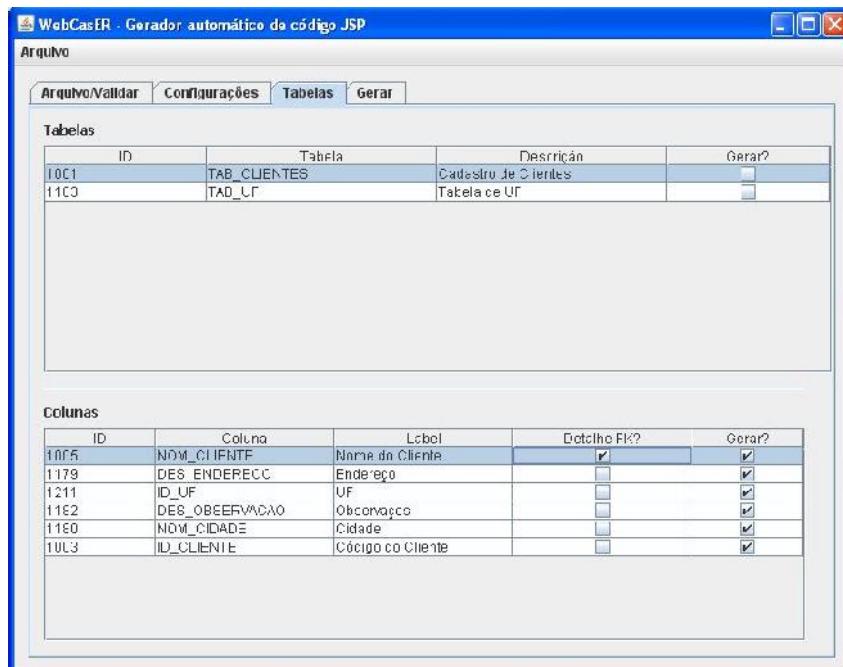


Figura 7. Seleção de tabelas a serem geradas

### 3.1.4 Geração de Código

É no quarto passo que os códigos são gerados. Para isso a ferramenta identifica no arquivo config\_webcaser.xml todos os *templates* a serem utilizados na geração dos arquivos, assim como as configurações de complemento de nome e tipo de arquivo. Para cada um dos *templates* identificados com tipo “ÚNICO”, é gerado apenas um arquivo, já para os *templates* identificados com o tipo “TABELA”, é gerado um arquivo para cada tabela modelada no ER.

Para todo o arquivo selecionado na Aba “Configurações” a ferramenta abre o arquivo config\_webcaser.xml e identificar todos os *templates* configurados no arquivo, executando os seguintes passos para cada um dos *templates*:

- Abrir o arquivo template (extensão .WCE);
- Ler linha a linha do arquivo *template* e gerar linha a linha o arquivo destino para cada uma das tabelas modeladas;
- O nome do arquivo destino será criado com base no nome da tabela mais a informação das *tags* nomeAntes, nomeDepois e extensão, definidas no arquivo de configuração config\_webcaser.xml;
- Quando encontrada alguma palavra ou tag reservada pelo WebCasER (Figura 8), efetua a substituição/ação.

```
<session-config>
    <session-timeout>
        30
    </session-timeout>
</session-config>
<welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
</welcome-file-list>

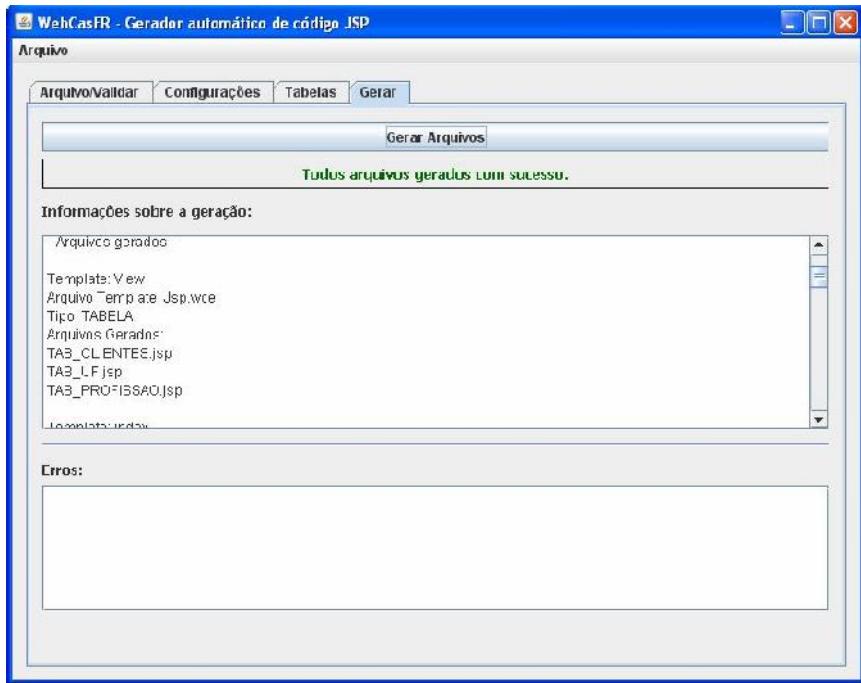
[REPETE_TABELAS]
<servlet>
    <servlet-name>NOME_TABELA</servlet-name>
    <servlet-class>PACOTE_CONTROLLER.srvNOME_TABELA</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>NOME_TABELA</servlet-name>
    <url-pattern>/srvNOME_TABELA</url-pattern>
</servlet-mapping>
[/REPETE_TABELAS]

<error-page>
    <exception-type>java.io.IOException</exception-type>
    <location>/errorPage.jsp</location>
</error-page>
```

**Figura 8. Exemplo de template para geração de um arquivo web.xml**

Conforme a figura 9, após o processo de geração, o sistema exibe as ocorrências de erros que impediram a geração dos arquivos ou em caso de sucesso as informações incluem:

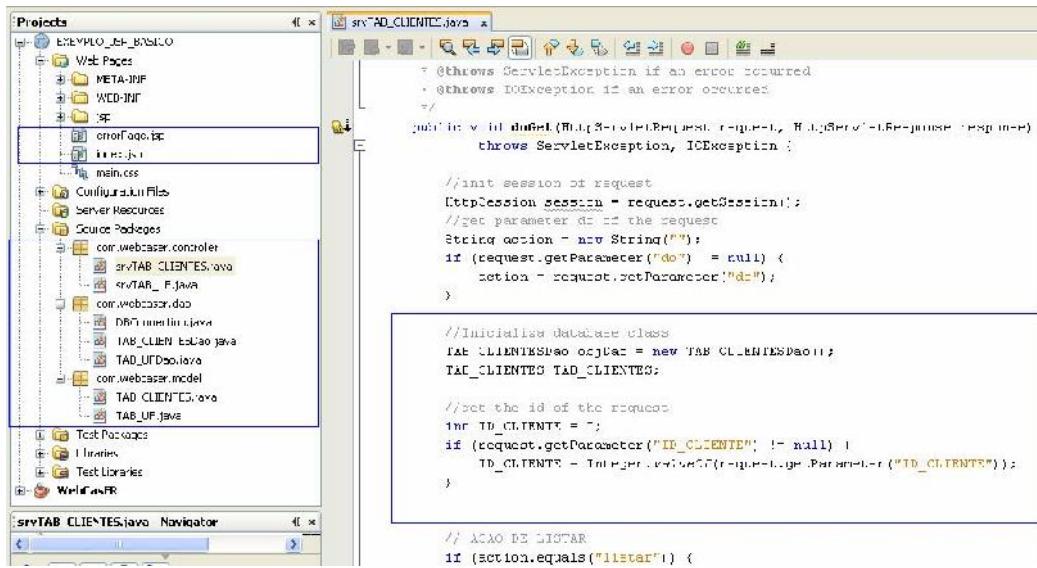
- quais campos não foram parametrizados em tela. Esta informação é importante, pois caso um *template* utilize alguma palavra reservada que deva ser substituída por alguma das informações parametrizadas em tela, no momento da geração esta palavra reservada será substituída por um espaço em branco, ocasionando um possível problema no código gerado;
- quais tabelas do modelo ER não foram utilizadas na geração ;
- quais foram os *templates* utilizados na geração;
- os arquivos gerados.



**Figura 9. Tela de geração dos arquivos**

### 3.1.5 Utilização do código gerado

Para testar o código gerado o usuário deverá criar um novo projeto Java (aplicação web) e distribuir os arquivos/páginas gerados em seus respectivos pacotes, conforme demonstra a Figura 10. A criação do banco de dados não é feita pela aplicação.



**Figura 10. Criação de novo projeto e distribuição dos arquivos gerados**

Caso a aplicação já exista o usuário poderá atualizar diretamente no servidor Web os arquivos gerados, como por exemplo as páginas JSP criadas. Deverá ser tomado

o cuidado de o arquivo gerado ser compatível com as informações existentes na aplicação que já está no servidor, como por exemplo: a aplicação existente trabalha com uma tabela X na qual possui 5 campos, porém o modelo ER sofre alteração sendo incluídos novos campos. Se o usuário efetua a geração dos arquivos para a tabela X, mas sem ajustar a tabela no banco, ocorrerá um erro pelo fato de existir uma divergência entre banco e arquivos gerados.

#### **4. Testes Realizados**

Foram realizados testes com três modelos ER utilizando o WebCasER, sendo que dentre os modelos testados foram identificadas as seguintes situações:

- primeiro modelo possuía três tabelas sem relacionamentos;
- o segundo modelo possuía seis tabelas e relacionamentos entre elas
- o terceiro dez tabelas com relacionamentos entre elas.

Foram testados relacionamentos atendidos pela ferramenta, sendo eles 1-1 e 1-N, e tabelas com média de 5 colunas.

Leva-se em média 1 minuto para abertura e validação do XML do DBDesigner e configuração dos campos de configuração. Também foi identificado uma média de 20 segundos para a configuração das informações de cada tabela na aba “Tabelas” (caso efetuado alterações dos dados apresentados em tela), sendo elas: marcação de tabela para geração; troca de labels de campos; indicação de qual campo servirá para descrição de tabela em relacionamentos FK.

Na etapa de geração, para todos os modelos testados o tempo de geração foi inferior a 5 segundos.

No final os tempos totais da criação de cada modelo ficaram em 2:05, 2:45 e 4:20 respectivamente. Baseando-se em informações coletados de programadores na qual indicam um tempo médio de 2 horas para a criação de um CRUD (Create, Retrieve, Update e Delete) simples, nota-se uma grande redução de tempo na geração dos arquivos.

Nos testes finais foram identificados que todos os campos foram gerados nas telas de consulta, alteração e inclusão, além de validações como tratamentos de obrigatoriedade de campos e relacionamentos FK entre tabelas.

#### **5. Conclusões**

Este artigo descreve uma ferramenta CASE que é capaz de ler um modelo ER feito no DBDesigner e gerar códigos Java(JSP) para aplicações web. Esta ferramenta serve como base para o desenvolvimento de *softwares* reduzindo tempo de desenvolvimento e padronizando código, sendo aplicável a qualquer metodologia de desenvolvimento de software, além de poder ser utilizada para geração de protótipos de teste.

Segundo (FOWLER, 2001) com a diminuição do tempo de desenvolvimento aumenta-se o tempo para análise de negócio e modelagem, obtendo-se ao final um sistema mais confiável.

Como trabalhos futuros sugerem-se:

- Inclusão de novos *templates* para atender diferentes padrões de desenvolvimento, como o *Struts* (FOWLER, 2001), por exemplo, além de inclusão de templates para a criação de classes de persistência;
- Padronização de relacionamentos N-N;
- Aperfeiçoamento das *tags* e palavras reservadas a fim de atender novos *templates* ou o aperfeiçoamento do *templates* existentes
- Uso de novas versões do DBDesigner ou até mesmo outras ferramentas de modelagem, utilizando DTDs para a leitura de modelos ER;
- Melhoria na definição de objetos em tela, principalmente campos de chave estrangeira.

## 5. Referências

FARIA, Rogério Amorin de. Treinamento Avançado em XML: Desvende os poderosos recursos desta linguagem. São Paulo: Digerati Books, 2005

FOWLER,M. *The New Methodology*. Disponível em <http://www.martinfowler.com/articles/newMethodology.html>. 2001. Acesso em nov. 2007.

HORSTMANN, Cay. *Big Java*. Porto Alegre: Bookman, 2004

MACHADO, Felipe; ABREU, Mauricio. *Projeto de Banco de Dados – Uma visão prática*. São Paulo: Érica, 1996.

SANTOS, Edgar Henrique dos.. *CodeCharge: Gerador de Códigos para Aplicações Web*. Comunicado Técnico. Ministério da Agricultura, Pecuária e Abastecimento. Campinas-SP, 2002.

SANTOS, Gustavo Alexandre dos. *Geração automática de código a partir de caso de uso*. Pernambuco: UFPE/PE. Trabalho de graduação do curso de Ciências da computação. Centro de Informática, Universidade Federal de Pernambuco, 2003. fonte. <http://www.cin.ufpe.br/~gas/papers/TrabalhoDeGraduacao-gas.pdf>

SOMMERVILLE, Ian. *Engenharia de Software*. São Paulo: Addison-Wesley, 2003.

VELOCITY. The Apache Jakarta Project. *Máquina de geração baseado em templates*. Disponível em <http://velocity.apache.org/> Acesso em nov. 2007.

# Relatório da modelagem

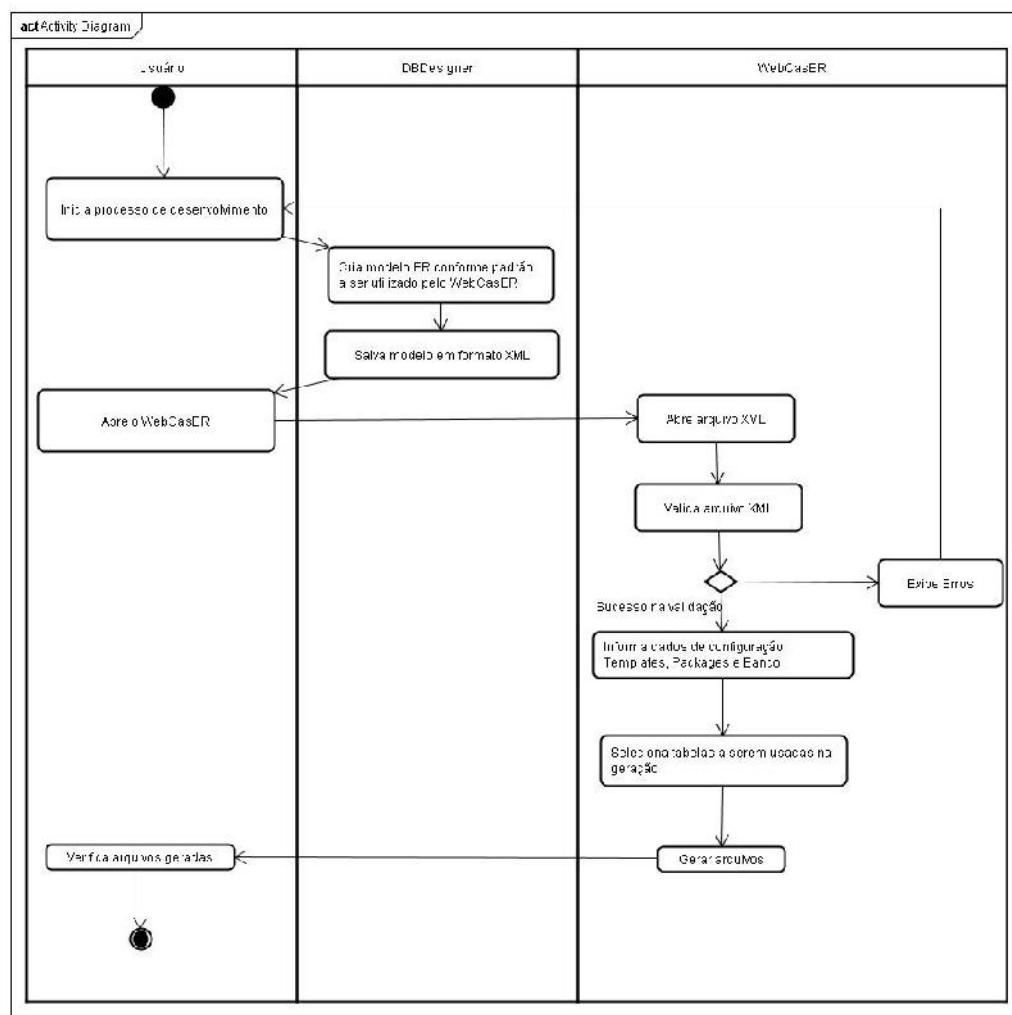
## WebCasER - Gerador automático de código Java(JSP)

Felipe Ramos, Francine Bica

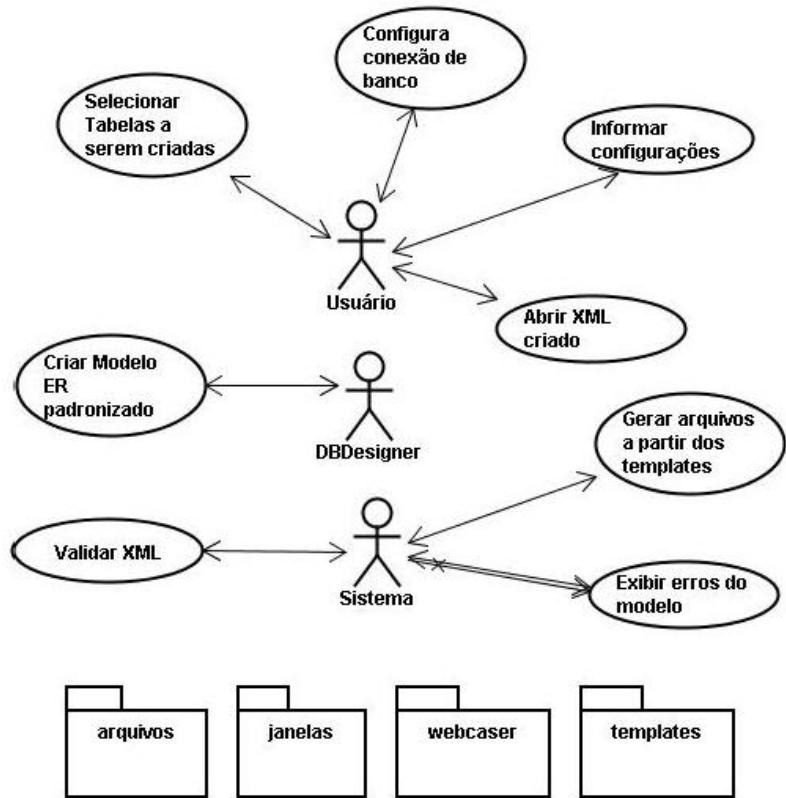
Curso de Análise e Desenvolvimento de Sistemas  
Faculdade de Tecnologia Senac RS (FATEC/RS)  
Porto Alegre – RS – Brasil

felipe\_ ramos@sicredi.com.br, frbica@gmail.com

### 1 - Fluxo de Atividades entre o usuário e a ferramenta WebCasER

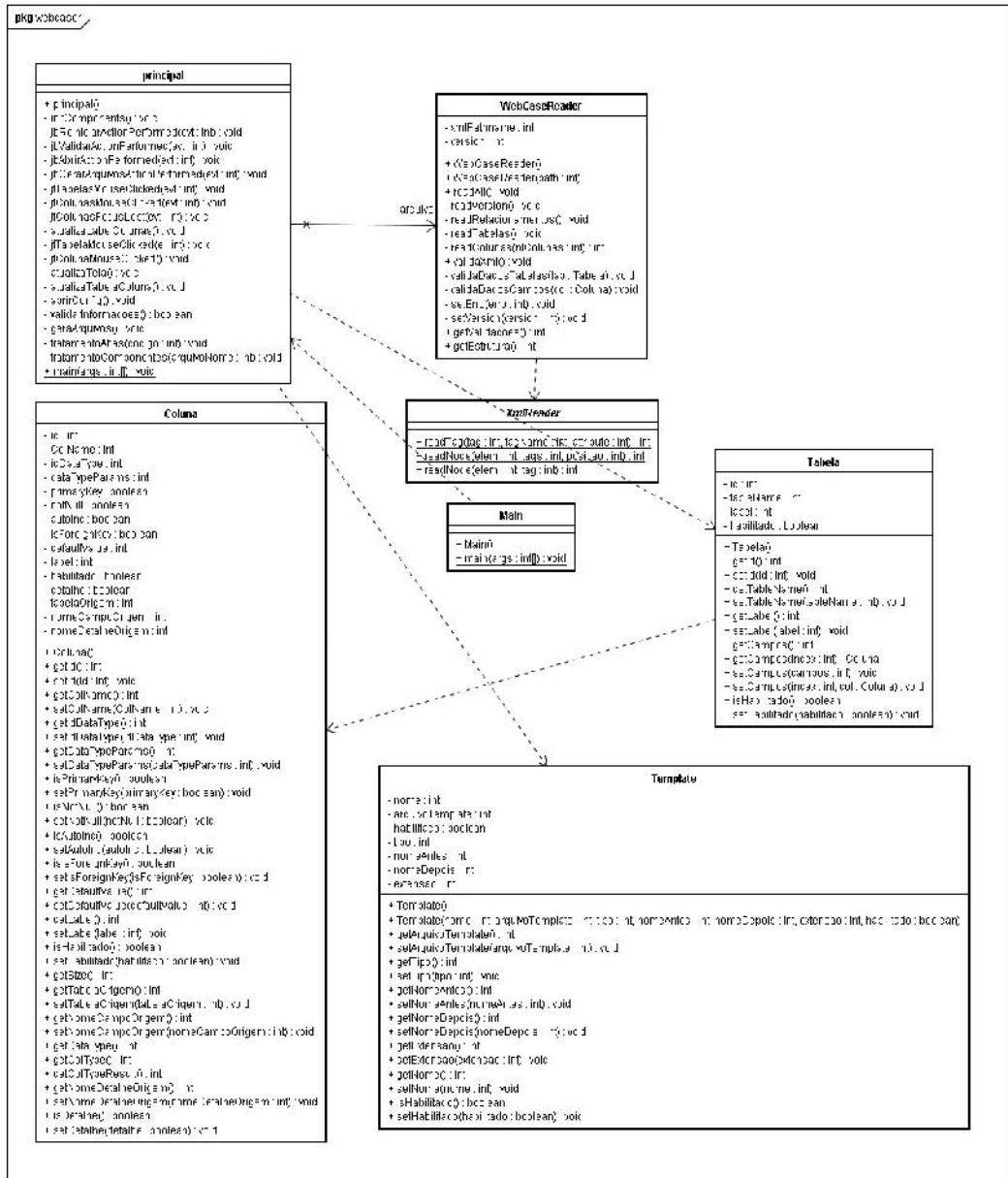


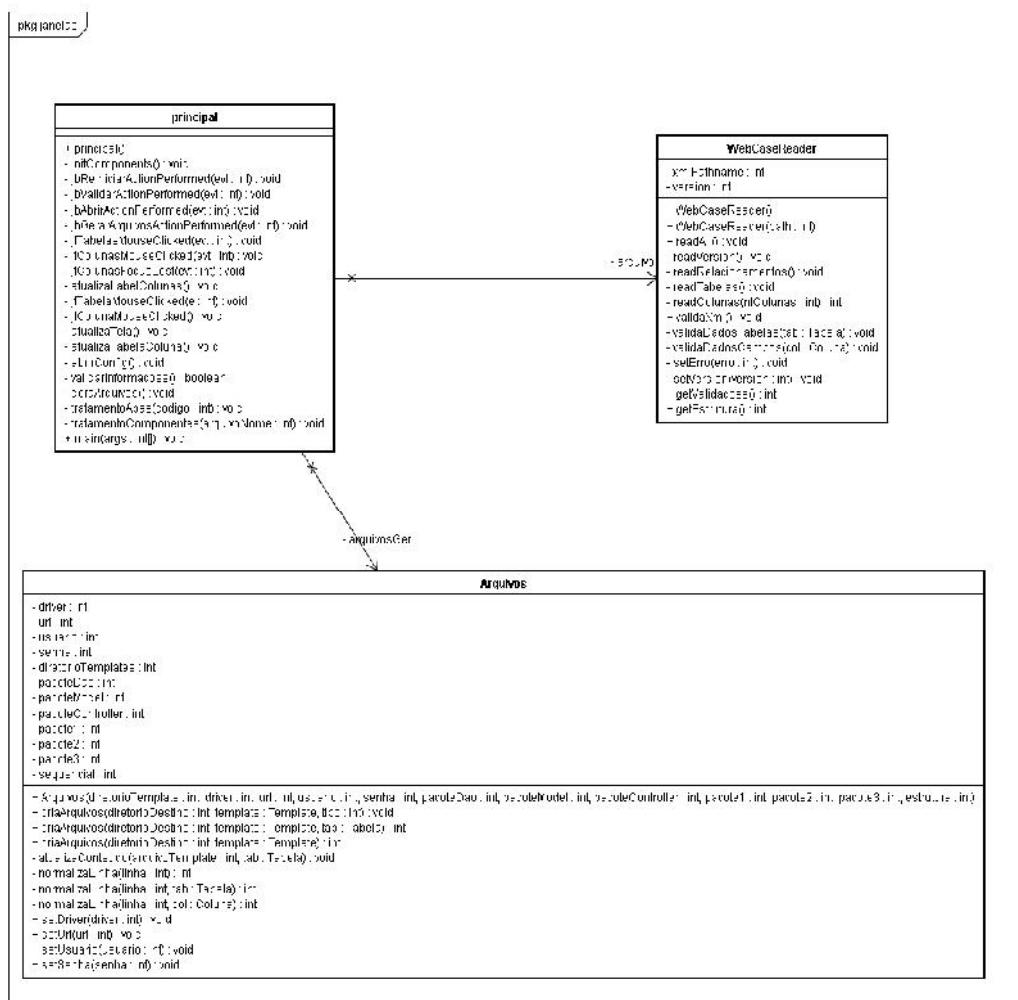
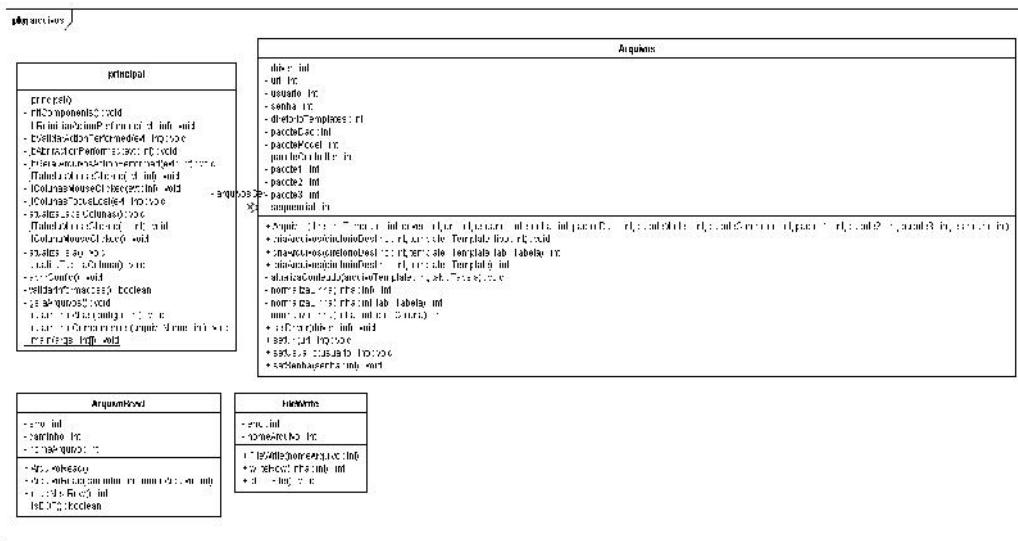
## 2 – Caso de Uso



### 3 – Diagramas de Classes

A ferramenta é dividida em quatro *packages*, sendo: webcaser, arquivos, janelas e templates.





#### 4 – Exemplo de um template usado para criação de páginas Jsp (template tipo “TABELA”).

```
<%@ page import="java.util.*" %>
<%@ page import="PACOTE_MODEL.*" %>
<%@ page import="PACOTE_DAO.*" %>

<%
String path = request.getContextPath();
String basePath = request.getScheme() + "://" + request.getServerName() + ":" +
+ request.getServerPort() + path + "/";
%>

<html>
    <head>
        <base href="<%=basePath%>">
        <title> LABEL_TABELA </title>
        <style> p, td { font-family:Tahoma,Sans-Serif; font-size:11pt;
padding-left:15; }
        </style>
        <meta http-equiv="pragma" content="no-cache">
        <meta http-equiv="cache-control" content="no-cache">
        <meta http-equiv="expires" content="0">
        <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">
        <meta http-equiv="description" content="This is my page">
    </head>

    <%
//Pega os dados do registro
NOME_TABELA NOME_TABELA = (NOME_TABELA)
request.getAttribute("NOME_TABELA");
[SOMENTE_FK]
Collection collectionNOME_TABELA_RELATION = ( new
NOME_TABELA_RELATIONDao() ).getAll();
[/SOMENTE_FK]
%>

<body bgcolor="#FFFFFF" text="#000000">

    <form name="edit" action="srvNOME_TABELA" method="post" >

        <center>
            <table cellspacing=2 border=0 >
                <th bgcolor="#CCDDEE" colspan=2 >
                    <font size=5 > LABEL_TABELA </font> </th>
                <tr>
                    <td valign=top> <b>LABEL_COLUNA</b> </td>
                    <td valign=top>
                        <input readonly=true type="text" name="NOME_COLUNA" value="<%=NOME_TABELA.getNOME_COLUNA()%>" size=TAMANHO_COLUNA maxlength=TAMANHO_COLUNA>
                    </td>
                </tr>
                <tr>
                    <td colspan=2> [/SOMENTE_COLUNAS_PK] </td>
                </tr>
                <tr>
                    <td colspan=2> [REPETE_COLUNAS_NAO_PK_FK] </td>
                </tr>
            </table>
        </center>
    </form>
</body>
```

```

[SOMENTE_COLUNAS_FK]
<tr>
    <td valign=top> <b>LABEL_COLUNA</b> </td>
    <td valign=top>
        <select name="NOME_COLUNA" size="1" >
        <%
            for (Iterator iter =
collectionNOME_TABELA_RELATION.iterator();
                    iter.hasNext();) {
                %
                NOME_TABELA_RELATION element =
(NOME_TABELA_RELATION) iter.next();
                %
                <option value=<%= element.getCOLUNA_ORIGEM() %>>
                    <%= element.getCOLUNA_ORIGEM() %> -
                    <%= element.getCOLUNA_ORIGEM()
                %></option>
                %
            }
            %
        </select>
    </td>
</tr>
[/SOMENTE_COLUNAS_FK]

<tr bgcolor="#F7F7F7">
    <td>
        <td colspan="2">
            <input type="submit" name="btnSave"
value="Salvar">
            <input type="reset" name="btnReset"
value="Limpar">
        </td>
    </td>
</tr>
</table>
</form>
</body>
</html>

```

Legenda:

	Dados configurados no no WebCasER
	Informações referentes a Tabela/Colunas utilizados na geração do arquivo
	Tags de ações, como repetições de bloco.

## 5 – Exemplo de um *template* usado para criação de páginas Jsp (template tipo “UNICO”).

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>

    [REPETE_TABELAS]
    <servlet>
        <servlet-name>NOME_TABELA</servlet-name>
        <servlet-class>PACOTE_CONTROLLER.srvNOME_TABELA</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>NOME_TABELA</servlet-name>
        <url-pattern>/srvNOME_TABELA</url-pattern>
    </servlet-mapping>
    [/REPETE_TABELAS]

    <error-page>
        <exception-type>java.io.IOException</exception-type>
        <location>/errorPage.jsp</location>
    </error-page>
</web-app>
```

Legenda:

	Dados configurados no no WebCasER
	Informações referentes a Tabela/Colunas utilizados na geração do arquivo
	Tags de ações, como repetições de bloco.

## 6 – Palavras/Tags reservadas pelo WebCasER.

### 6.1 – Informações de Tabelas

Palavra	Ação	Corresp. ER	UNC	TAB
NOME_TABELA	Substitui palavra pelo nome atribuído a tabela no modelo ER	<METADATA> <TABLES> <TABLE> TableName		X
LABEL_TABELA	Substitui palavra pela descrição atribuída a tabela no modelo ER	<METADATA> <TABLES> <TABLE> Comments		X

### 6.2 – Informações de Colunas\*

Palavra	Ação	Corresp. ER	UNC	TAB
NOME_COLUNA	Substitui palavra pelo nome atribuído a coluna no ER	<METADATA> <TABLES> <TABLE> <COLUMNS> <COLUMN> ColName		X
TIPO_RESULT_COLUNA	Substitui palavra pelo tipo da coluna a ser usado para completar a chamada do método para Get de atributo do resultSet <i>Ex:</i> <i>rs.getTIPO_RESULT_COLUNA</i> <i>Substitui por:</i> Int, String, Date			X
TIPO_COLUNA	Substitui palavra pelo tipo da coluna. (String, int, Date)			X
LABEL_COLUNA	Substitui palavra pela descrição atribuída a coluna no modelo ER	<METADATA> <TABLES> <TABLE> <COLUMNS> <COLUMN> Comments		X
TAMANHO_COLUNA	Substitui palavra pelo tamanho atribuído a coluna no modelo ER (Campos VarChar2)	<METADATA> <TABLES> <TABLE> <COLUMNS> <COLUMN> DatatypeParams		X
TAMANHO_MAX_COLUNA	Substitui palavra pelo tamanho atribuído a coluna no modelo ER (Campos VarChar2), porém limitando em 50	<METADATA> <TABLES> <TABLE> <COLUMNS> <COLUMN> DatatypeParams		X
DEFAULT_COLUNA	Substitui palavra pelo conteúdo default atribuído a coluna no modelo ER	<METADATA> <TABLES> <TABLE> <COLUMNS> <COLUMN> DefaultValue		X
NOME_TABELA_RELATION**	Substitui palavra pelo nome da tabela estrangeira correspondente ao conteúdo da coluna	<METADATA> <RELATIONS> <RELATION> SrcTable		X
COLUNA_ORIGEM**	Substitui palavra pelo nome da coluna estrangeira correspondente	<METADATA> <RELATIONS>		X

	ao conteúdo da coluna	<RELATION> FKFields		
COLUNA_DETALHE_ORIGEM	Substitui palavra pelo nome da coluna indicada na configuração como “detalhe” para a tabela na qual o campo FK é origem			X
CONVERSOR_VALOR	Utilizado para conversor de valor quando campo String			X
INICIALIZA_COLUNA	Substitui palavra por valor de inicialização de variáveis			X

(\*) As palavras reservadas para colunas sempre deverão ser utilizadas entre TAG's de repetições.

(\*\*) Usados para campos FK

#### 6.3 – Gerais

Palavra	Ação	Corresp. ER	ÚNC	TAB
PACOTE.DAO	Substitui pelo nome configurado para o pacote dos objetos Dao		X	X
PACOTE.MODEL	Substitui pelo nome configurado para o pacote dos objetos Model		X	X
PACOTE_CONTROLLER	Substitui pelo nome configurado para o pacote dos objetos de Controller		X	X
PACOTE_GERAL_1	Substitui pelo nome configurado para o pacote 1		X	X
PACOTE_GERAL_2	Substitui pelo nome configurado para o pacote 1		X	X
PACOTE_GERAL_3	Substitui pelo nome configurado para o pacote 1		X	X
DRIVER_DB	Substitui palavra pelo Driver de conexão informado no WebCasER		X	X
URL_DB	Substitui palavra pela URL de conexão informado no WebCasER		X	X
USUARIO_DB	Substitui palavra pelo usuário de conexão informado no WebCasER		X	X
SENHA_DB	Substitui palavra pela senha de conexão informado no WebCasER		X	X

#### 6.4 – Repetições

TAGs	Ação	Corresp. ER	ÚNC	TAB
[NAO_PRIMEIRA_COLUNA]	Usado em repetições (Tabelas ou Colunas). Indica que o conteúdo da linha na qual a tag se encontra não será criado para a primeira ocorrência do “while” de repetição.		X	X
SEQUENCIAL_INI	Usado em repetições (Tabelas ou Colunas). Tag substituída por um seqüencial iniciado em ZERO.		X	X

SEQUENCIAL_CONT	Usado em repetições (Tabelas ou Colunas). Tag substituída por um seqüencial iniciado pelo último número da última sequencia iniciada.		X	X
[REPETE_CAMPOS] [/REPETE_CAMPOS]	Usado para repetições. Indica que todo o conteúdo encontrado entre o início e final da tag será repetido para todas as colunas da tabela.			X
[SOMENTE_PK] [/SOMENTE_PK]	Usado para repetições. Indica que todo o conteúdo encontrado entre o início e final da tag será repetido para todas as colunas da tabela que sejam PK (somente uma das colunas será).			X
[SOMENTE_FK] [/SOMENTE_FK]	Usado para repetições. Indica que todo o conteúdo encontrado entre o início e final da tag será repetido para todas as colunas da tabela que sejam FK.			X
[REPETE_CAMPOS_NAO_PK] [/REPETE_CAMPOS_NAO_PK]	Usado para repetições. Indica que todo o conteúdo encontrado entre o início e final da tag será repetido para todas as colunas da tabela menos a PK			X
[REPETE_CAMPOS_NAO_PK_FK] [/REPETE_CAMPOS_NAO_PK_FK]	Usado para repetições. Indica que todo o conteúdo encontrado entre o inicio e final da tag será repetido para todas as colunas da tabela menos as PK e FK			X
[REPETE_TABELAS] [/REPETE_TABELAS]	Usado para repetições. Indica que todo o conteúdo encontrado entre o início e final da tag será repetido para todas as tabelas da tabela.			X
[SOMENTE_STRING_OBRIGATORIOS] [/SOMENTE_STRING_OBRIGATORIOS]	Usado para repetições. Indica que todo o conteúdo encontrado entre o inicio e final da tag será repetido para todas as colunas que sejam do tipo VARCHAR e obrigatorias			X

## 7 – Erros na validação do XML.

Tag Validada	Validação
<DBMODEL> version	Ferramenta foi validada com a versão 4.0 do DBDesigner. Por não poder garantir seu correto funcionamento com versões inferiores ou posteriores, valida-se a versão do arquivo XML.
<METADATA> <TABLES> <TABLE> Comments	Demais campos da tabela são preenchidos automaticamente. Neste caso, como o campo de “comentário” é opcional porém utilizado para labes e títulos de janelas, obriga-se o preenchimento do mesmo.
<METADATA> <TABLES> <TABLE> <COLUMNS> <COLUMN> Comments	Demais campos de coluna são preenchidos automaticamente. Neste caso, como o campo de “comentário” é opcional, porém utilizado para labes de campos, obriga-se o preenchimento do mesmo.

## 8 – Erros na geração dos arquivos

Mensagem	Validação
Não foi possível identificar nenhuma tabela no modelo ER.	Verifica se existe pelo menos uma tabela modelada no DBDesigner e existente no XML lido.
Diretório de templates não informado.	Verifica se foi preenchido o campo de diretório de templates.
Diretório de templates não existe.	Verifica se diretório de templates informado é válido.
Diretório de destino dos arquivos não informado.	Verifica se foi preenchido o campo de diretório de destino.
Diretório de Destino não existe.	Verifica se diretório de destino informado é válido.
Tabela: xxxxx. Coluna: xxxxx. Coluna não possui label Informado.	Para cada uma das colunas das tabelas selecionadas, verifica se possui informação de informado.

## 9 – Informações exibidas ao final da geração

Mensagem	Validação
Não informada package Dao.	Quando não informado campo referente a package DAO.
Não informada package Controller.	Quando não informado campo referente a package Controller.
Não informada package Model.	Quando não informado campo referente a package Model.
Não informada package Geral 1.	Quando não informado campo referente a package Geral 1.
Não informada package Geral 2.	Quando não informado campo referente a package Geral 2.
Não informada package Geral 3.	Quando não informado campo referente a package Geral 3.
Não informado Driver para conexão com o banco.	Quando não informado campo referente ao Driver para conexão com o Banco de Dados.
Não informado URL para conexão com o banco.	Quando não informado campo referente a URL para conexão com o Banco de Dados.
Não informado Usuário para conexão com o banco.	Quando não informado campo referente ao usuário para conexão com o Banco de Dados.
Não informado Senha para conexão com o banco.	Quando não informado campo referente a Senha para conexão com o Banco de Dados.
Tabela xxxxx não selecionada.	Exibe uma mensagem para cada uma das tabelas que não foram selecionadas.
Tabela xxxxx não possui campo detalhe informado.	Exibe uma mensagem para cada uma das tabelas que não possuem um campo “detalhe” informado. Neste caso para efetuar os relacionamentos entre tabelas,

	será utilizado como default o campo PK da tabela.
Nenhuma tabela informada. Somente serão gerados arquivos tipo UNICO.	Quando nenhuma tabela selecionada para geração. Neste caso apenas arquivos do tipo ÚNICO serão gerados.

## 10 – Usando a Ferramenta WebCasER - Passo a passo

Inicia-se pela modelagem do ER no DBDesigner. Levando em consideração que somente são atendidos relacionamentos 1-N e 1-1, a Figura 1 exibe um modelo que contém 3 tabelas: TAB\_CLIENTES, TAB\_UF e TAB\_PROFISSAO.

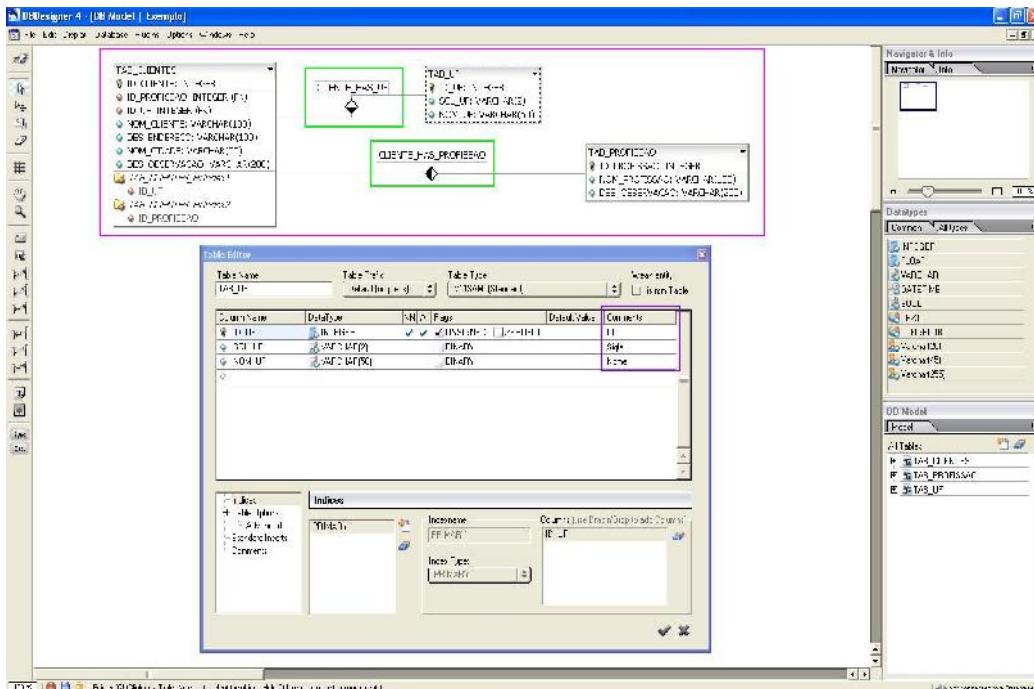


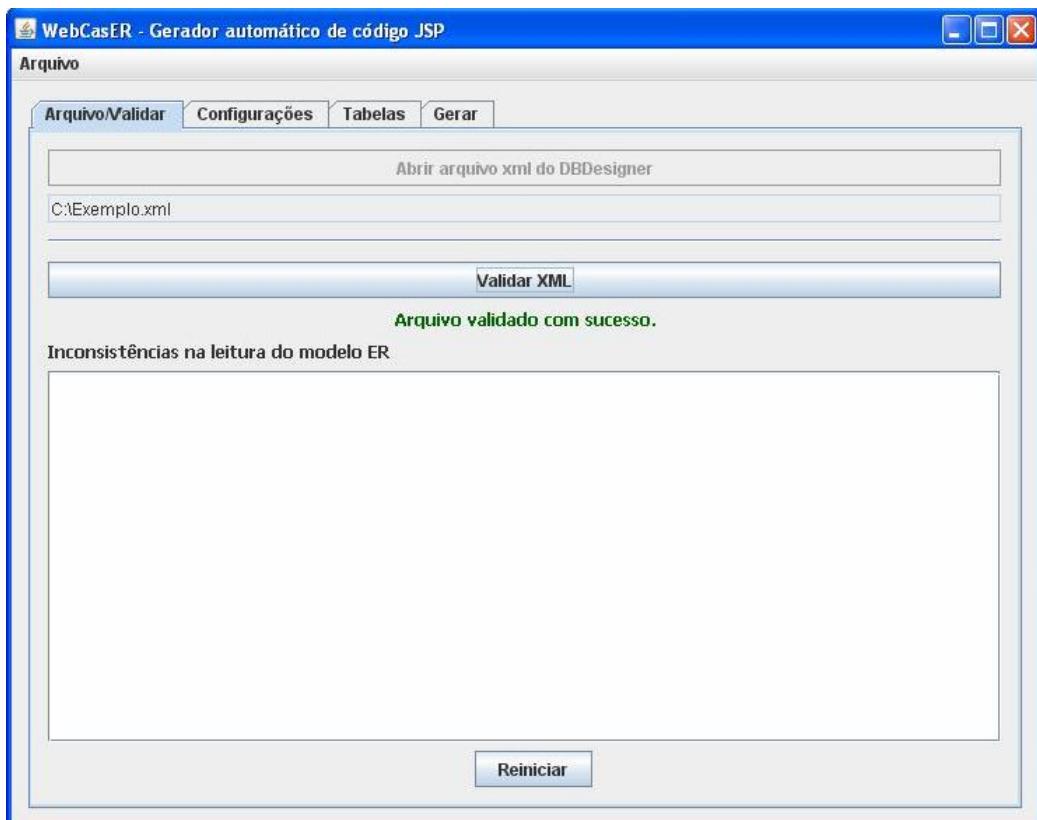
Figura 1. Modelagem de tabelas no DBDesigner

O Modelo ER criado é salvo em disco para posterior utilização. A Figura 2 demonstra parte do XML criado pelo DBDesigner.

```
<xml version="1.0" standalone="yes" />
<!DOCTYPE Version="4.0">
+ <SETTINGS>
- <METADATA>
- <OPTIONS />
- <TABLES>
- <TABLE ID="1001" TableName="TAB_CLIENTES" PrevTableName="Table_01" XPos="25" YPos="66" TableType="0" TablePreface="0" rmTable="0"
  TableFormat="0" UseStandardInserts="0" StandardInserts="" TableOptions="DelayKeyTblUpdates=0\!PackKeys=0\!RowChecksum=0
  \!RowFormat=0\!UseId=0\!LinkType=0\!nComments=Cadastro de Clientes" Collapsed="0" IsLinkedObject="0" IsLinkedMode="1"
  Obj_d_Linked="1" Order="0" >
- <COLUMNS>
  <COLUMN ID="1003" ColName="ID_CLIENTE" PrevColName="" Pos="0" iddatatype="5" DatatypeParams="" Width="-1" Prec="-1"
    PrimaryKey="1" NotNull="1" AutoInc="1" IsForeignKey="0" DefaultValue="" Comments="C\ 243digo do Cliente">
+ <OPTIONSELECTED>
</COLUMN>
- <COLUMN ID="1011" ColName="NOME_CLIENTE" PrevColName="" Pos="1" iddatatype="5" DatatypeParams="" Width="1" Prec="1" PrimaryKey="0"
  NotNull="1" AutoInc="0" IsForeignKey="1" DefaultValue="UF" Comments="UF">
+ <OPTIONSELECTED>
</COLUMN>
- <COLUMN ID="1005" ColName="NOM_CIDADE" PrevColName="NOME_CLIENTE" Pos="2" iddatatype="20" DatatypeParams="(100)"
  Width="1" Prec="1" PrimaryKey="0" NotNull="1" AutoInc="0" IsForeignKey="0" DefaultValue="" Comments="Nome da Cidade">
+ <OPTIONSELECTED>
</COLUMN>
- <COLUMN ID="1179" ColName="DES_ENDERECO" PrevColName="" Pos="3" iddatatype="20" DatatypeParams="(100)" width="-1" Prec="-1"
  PrimaryKey="0" NotNull="1" AutoInc="0" IsForeignKey="0" DefaultValue="" Comments="Endere\231o">
+ <OPTIONSELECTED>
</COLUMN>
- <COLUMN ID="1180" ColName="DES_OBSEVACAO" PrevColName="" Pos="4" iddatatype="20" DatatypeParams="(50)" width="-1" Prec="-1"
  PrimaryKey="0" NotNull="0" AutoInc="0" IsForeignKey="0" DefaultValue="" Comments="Observa\231o\243es">
+ <OPTIONSELECTED>
</COLUMN>
</COLUMNS>
```

Figura 2. XML do ER gerado pelo DBDesigner

O passo seguinte já é feito através do WebCasER. A Figura 3 demonstra a primeira tela corresponde a abertura e validação do modelo ER criado no DBDesigner. Neste ponto é selecionado o arquivo XML gerado no passo anterior e iniciado a validação. Já que todos os campos obrigatórios estão preenchidos são habilitadas as demais abas da janela. Caso houvesse ocorrido algum erro, estes seriam exibidos em tela para que o usuário possa retornar ao passo anterior e ajustar.



**Figura 3. Abertura e validação do ER.**

Foram habilitadas três abas para que o usuário possa efetuar as configurações necessárias para geração dos códigos. Como é demonstrado pela figura 4, a primeira aba está dividida em três partes, sendo estas: *Templates*, *Packages* e Informações de Conexão.

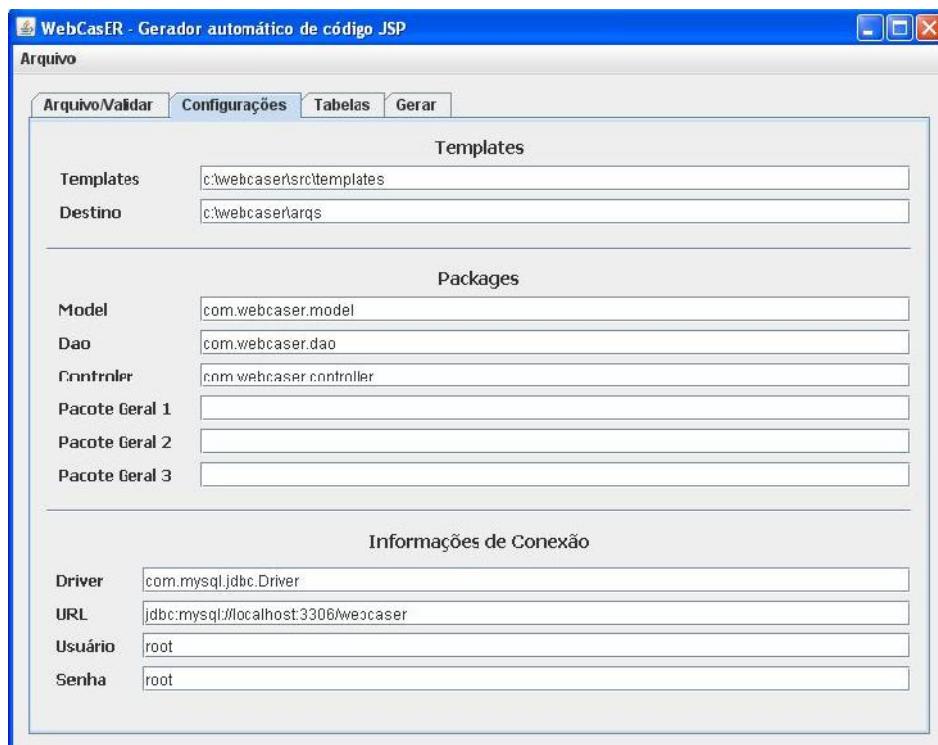
As informação de *Templates* são obrigatórias, pois são estas informações que indicarão o local onde estão armazenados os templates a serem utilizados assim como o local onde os arquivo deverão ser gerados. Os *templates* utilizados estão descritos na tabela 1.

**Tabela 1. Templates utilizados**

Template	Observação
Index.WCE	Template adaptado para a criação do menu. Gera um arquivo.
ErrorPage.WCE	Template adaptado para efetuar tratamentos de erros. Gera um arquivo.
Web.WCE	Template adaptado para a criação de xml de configuração de aplicação web. Gera um arquivo
DBConnection.WCE	Template adaptado para efetuar conexão com banco. Gera um arquivo
JspList.WCE	Template adaptado para exibir consulta de todos os registros da tabela. Para cada tabela modelada no ER, será gerado um arquivo.
Jsp.WCE	Template adaptado para exibir inserção e alteração de registros da tabela. Para cada tabela modelada no ER, será gerado um arquivo.
Model.WCE	Template adaptado para criar classe de objeto para tabela. Para cada tabela modelada no ER, será gerado um arquivo.
Servlet.WCE	Template adaptado para criar um classe de controle. Para cada tabela modelada no ER, será gerado um arquivo.
Dao.WCE	Template adaptado para criar classe de Dao. Para cada tabela modelada no ER, será gerado um arquivo.

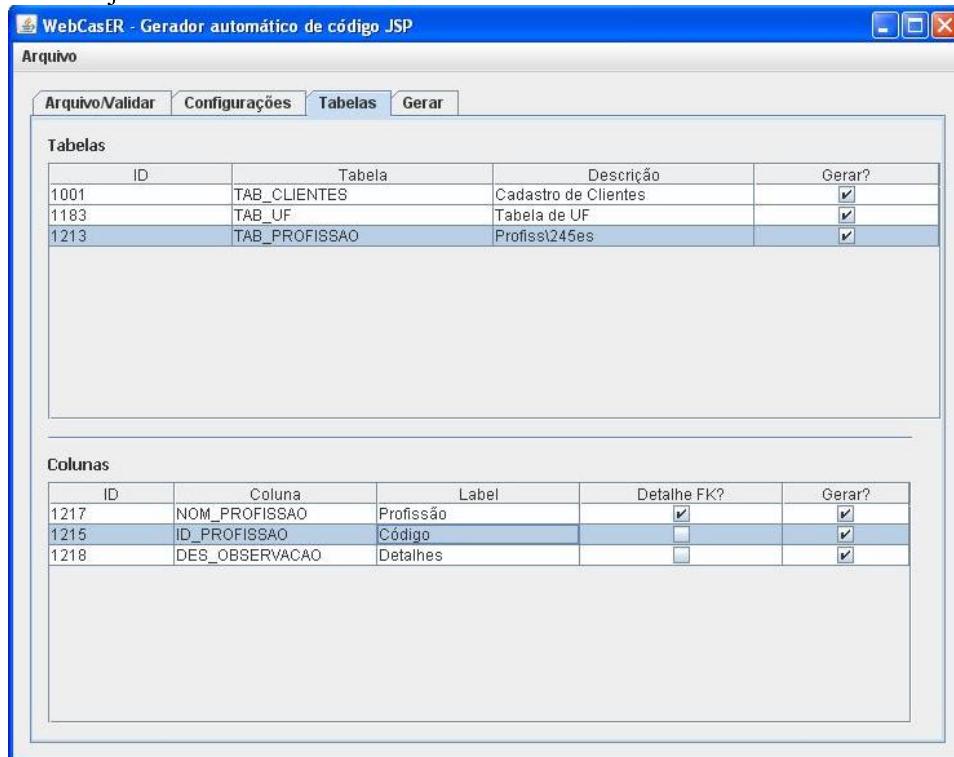
A informação de *Packages* não é obrigatória, porém muito importante para as gerações de arquivos. Em JAVA, as *packages* são um conjunto de classes e interfaces que se relacionam. Sendo assim, pode-se utilizar estas informações em templates, identificando nos arquivos gerados a qual pacote eles pertencem. São disponíveis para informação 6 campos, sendo que os 3 primeiros já indicam que devam ser utilizados para templates adaptados ao padrão MVC. Como os templates utilizados são adaptados ao padrão MVC, são informados somente as *packages* correspondentes.

Por fim, deve ser informado os dados necessários para efetuar a conexão com o banco. Neste caso, utiliza-se um banco MySQL.

**Figura 3. Informações de configuração geral**

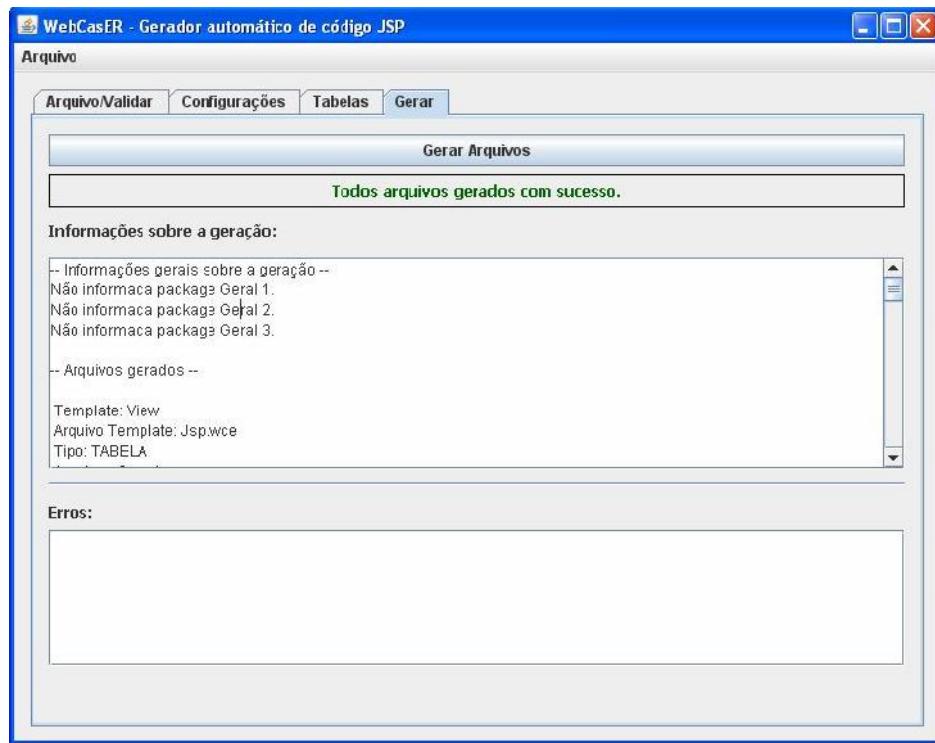
A seguir, conforme demonstrado pela figura 4, são exibidas em tela todas as tabelas modeladas no ER e seus respectivos campos. Neste exemplo são selecionadas todas as tabelas possíveis e indicados os campo que servirão como descrição a serem utilizados nos combobox de tela para os relacionamentos de tabelas.

Os dados que no momento da leitura do XML perderam configuração de acentos podem ser ajustados neste momento.



**Figura 4. Seleção de tabelas a serem geradas**

O último passo feito na ferramenta é a execução da geração, exibido na figura 5. Neste momento são feitos algumas validações e em casos de erros a geração é cancelada e os erros exibidos em tela. Quando a geração é feita com sucesso, são exibidas as informações correspondentes a geração e os arquivos gerados no diretório parametrizado anteriormente.



**Figura 5. Seleção de tabelas a serem geradas**

Para cada *template* utilizado, foram gerados um ou mais arquivos. A tabela 2 exibe o conteúdo gerado após o passo anterior:

**Tabela 2. Detalhes sobre a geração e arquivos gerados**

Informações sobre a geração
<pre>-- Informações gerais sobre a geração -- Não informada package Geral 1. Não informada package Geral 2. Não informada package Geral 3.  -- Arquivos gerados -- Template: View Arquivo Template: Jsp.wce Tipo: TABELA Arquivos Gerados: TAB_CLIENTES.jsp TAB_UF.jsp TAB_PROFISSAO.jsp  Template: index Arquivo Template: Index.wce Tipo: UNICO Arquivos Gerados: index.jsp  Template: Dao Arquivo Template: Dao.wce Tipo: TABELA Arquivos Gerados:</pre>

```
TAB_CLIENTESDao.java  
TAB_UFDao.java  
TAB_PROFISSAODao.java
```

*Template: ViewList*  
*Arquivo Template: JspList.wce*  
*Tipo: TABELA*  
*Arquivos Gerados:*  
TAB\_CLIENTESList.jsp  
TAB\_UFList.jsp  
TAB\_PROFISSAOList.jsp

*Template: Servlet*  
*Arquivo Template: Servlet.wce*  
*Tipo: TABELA*  
*Arquivos Gerados:*  
srvTAB\_CLIENTES.java  
srvTAB\_UF.java  
srvTAB\_PROFISSAO.java

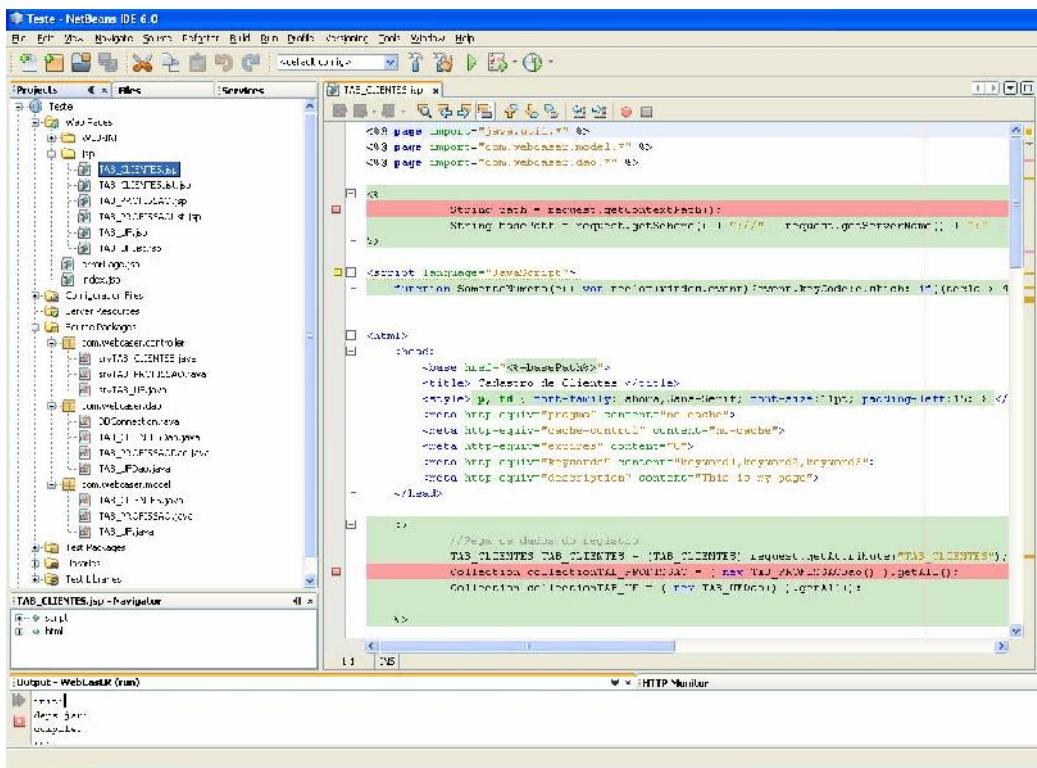
*Template: Model*  
*Arquivo Template: Model.wce*  
*Tipo: TABELA*  
*Arquivos Gerados:*  
TAB\_CLIENTES.java  
TAB\_UF.java  
TAB\_PROFISSAO.java

*Template: web*  
*Arquivo Template: Web.wce*  
*Tipo: UNICO*  
*Arquivos Gerados:*  
web.xml

*Template: DBConnection*  
*Arquivo Template: DBConnection.wce*  
*Tipo: UNICO*  
*Arquivos Gerados:*  
DBConnection.java

*Template: errorPage*  
*Arquivo Template: errorPage.wce*  
*Tipo: UNICO*  
*Arquivos Gerados:*  
errorPage.jsp

O último passo é externo a ferramenta, na qual neste ponto é feito a utilização dos arquivos gerados. Para isso é iniciado uma nova aplicação web no Netbeans (foi escolhido o NetBeans como IDE, mas pode-se utilizar qualquer outra), e distribuídos os arquivos em pastas/packages conforme configurados os templates e exibido na figura 6.



**Figura 6. Utilização dos códigos gerados**

A figura 7 apresenta a execução da aplicação criada a partir dos arquivos gerados pela ferramenta e utilizando os templates descritos anteriormente. Pode-se visualizar a forma como foi criado o relacionamento entre as tabelas de clientes, UF e profissão, na qual é exibido um combobox para seleção das tabelas relacionadas e utilizado como detalhe os nomes de UF e Profissão, que foram informados na aba “Tabelas”.

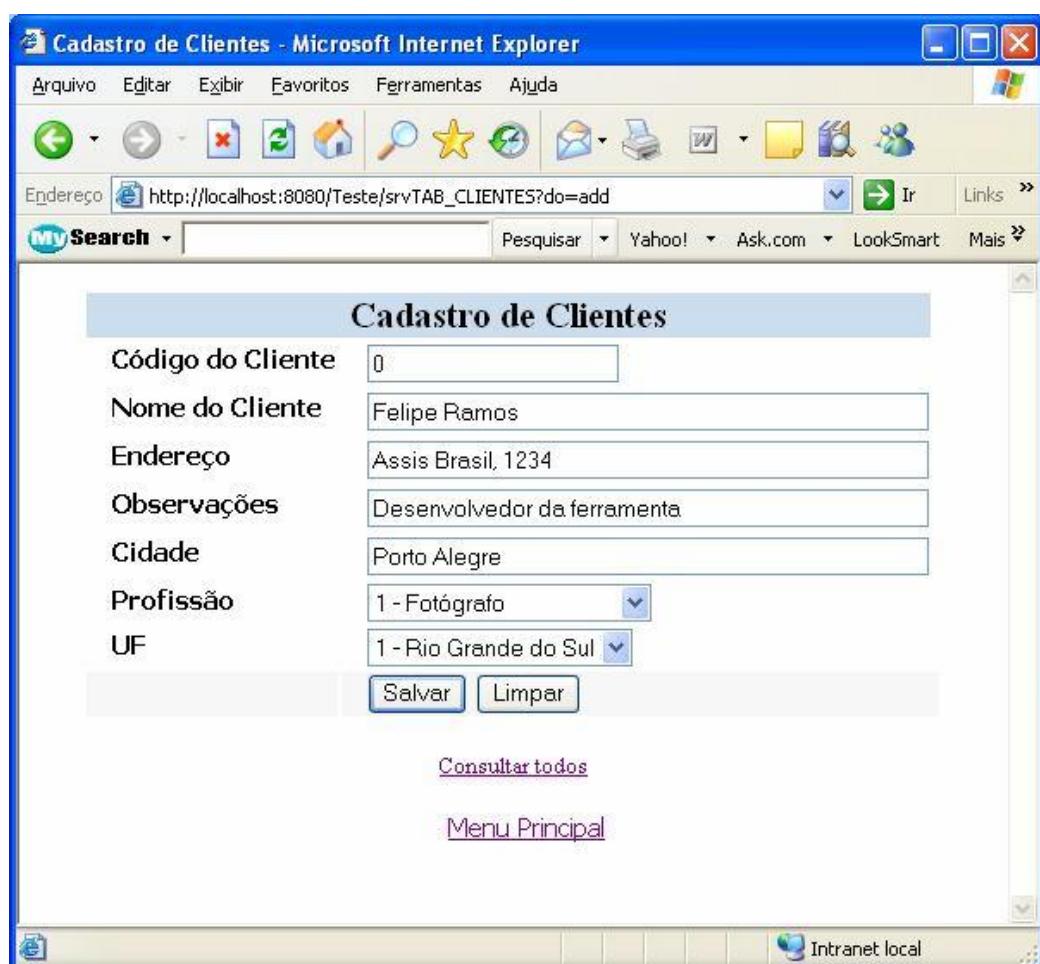


Figura 7. Tela de cadastro de clientes