UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

GERADOR DE CÓDIGO JSP BASEADO EM PROJETO DE BANCO DE DADOS MYSQL

JULIANE MENIN

BLUMENAU
2005

2005/1-31

JULIANE MENIN

GERADOR DE CÓDIGO JSP BASEADO EM PROJETO DE BANCO DE DADOS MYSQL

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciências da Computação — Bacharelado.

Prof. Alexander Roberto Valdameri, Mestre, Orientador

BLUMENAU
2005

2005/1-31

GERADOR DE CÓDIGO JSP BASEADO EM PROJETO DE BANCO DE DADOS MYSQL

Por

JULIANE MENIN

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Alexander Roberto Valdameri, Mestre, Orientador
Membro:	Prof. Marcel Hugo, Mestre – FURB
Membro:	Prof. Joyce Martins, Mestre – FURB

Blumenau, 04 de julho de 2005

Dedico este trabalho a todos os amigos, especialmente aqueles que me ajudaram diretamente na realização deste.

AGRADECIMENTOS

A Deus, que tem me dado sabedoria, saúde e muita força de vontade para superar quaisquer desafios encontrados em minha vida.

Ao meu orientador, Alexander, por ter acreditado e me acompanhado nesta etapa importante de final de curso.

A toda minha família pelo apoio e incentivo, em especial aos meus pais Mario e Iraci Menin, que mesmo longe sempre estiveram presentes.

A todos os meus amigos e colegas de classe, em especial ao Fernando dos Santos pelo seu apoio no desenvolvimento deste trabalho.

“Nunca ande por caminho traçado pois ele conduz somente até onde os outros já foram.”

Alexandre Graham Bell

RESUMO

O processo de desenvolvimento de software tem se especializado dia-a-dia. Neste contexto, o uso de ferramentas para auxiliar a automatização de rotinas tidas como triviais, torna-se cada vez maior. Este trabalho apresenta uma ferramenta para geração automática de código para a tecnologia JSP através da leitura da estrutura armazenada em um banco de dados MySQL. As páginas geradas permitem fazer inclusões, exclusões, alterações e consultas no banco de dados.

Palavras-chave: banco de dados MySQL. JSP. Geração de código.

ABSTRACT

The process development software has specialized day-by-day. This context, the use of tools to assist the automatization of trivial routines, becomes each bigger time. This work present a tool for automatic generation code for technology JSP through the reading of the structure stored in a database MySQL. The generated pages allow to make inclusions, exclusions, updates and queries in the database.

Key-Words: data base MySQL. JSP. Code generation.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

ADO – *ActiveX Data Objects*

API – *Application Programing Interface*

ASP – *Active Server Page*

CSS – *Cascading Style Sheet*

GPL – *General Public License*

HTML – *Hiper Text Markup Language*

HTTP – *HyperText Transport Protocol*

IDE – *Integrated Development Environment*

JDBC – *Java Database Connectivity*

JDK – *Java Developers Kit*

JET – *Joint Engine Tecnology*

JSDK – *Java Software Development Kit*

JSP – *JavaServer Page*

JVM – *Java Virtual Machine*

MIME – *Multipurpose Internet Mail Extensions*

ODBC – *Open DataBase Connectivity*

PHP – *Personal Home Page*

RAD – *Rapid Aplication Development*

SGBD – Sistema Gerenciador de Banco de Dados

SQL – *Structured Query Language*

W3C – *World Wide Web Consortium*

XML – *Extensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Apesar da internet ter sido criada através de uma iniciativa não-comercial, isto é, exclusivamente para fins de segurança, educação e pesquisa, são cada vez maiores a demanda e o interesse por aplicativos comerciais, seja para uso pessoal ou corporativo.

Através da internet as empresas estão podendo melhorar suas operações e oferecer serviços novos e melhores para os seus clientes, como consultas de preços, ofertas, pedido de compra, orçamentos, entre outros, sem o deslocamento do cliente à empresa.

A maioria das empresas está migrando os seus sistemas para o ambiente web, e os desenvolvedores, para suprir esta demanda, passaram a utilizar ferramentas que auxiliam na criação e manutenção de aplicações para internet, obtendo rapidez, qualidade e facilidades na construção de seus aplicativos.

Até alguns anos segundo Abileboul, Peter e Sucice (2000, p. 2), a publicação de dados eletrônicos estava limitada a algumas poucas áreas científicas e técnicas. Agora isto está se tornando universal. A maioria das pessoas vê estes dados como documentos web, mas estes documentos, em vez de serem manualmente compostos, são cada vez mais gerados de forma automática, através de páginas codificadas dinamicamente com acesso a banco de dados.

Para construir um simples *site* dinâmico ou um complexo sistema *Business-to-Business* é necessária a utilização de ferramentas que possibilitem consultas ao banco de dados, integração com sistemas corporativos, entre outras inúmeras funcionalidades. Dentre as diversas tecnologias disponíveis atualmente para o desenvolvimento dessas aplicações, destaca-se *JavaServer Pages* – (JSP).

A utilização de páginas JSP, que conforme Fields e Kolb (2000, p. 2), é uma tecnologia baseada em Java que simplifica o processo de desenvolvimento de *sites* web dinâmicos, oferece diversas vantagens em relação ao uso de outras tecnologias como *Personal*

Home Page (PHP), *Active Server Page* (ASP), Microsoft.NET, Python. As principais vantagens são herdadas da própria linguagem Java, como a portabilidade, facilidade de programação, flexibilidade, entre outras.

Tendo em vista a necessidade dos desenvolvedores web possuírem ferramentas que facilitem a geração de código para suprir as demandas por serviços web, o presente trabalho apresenta uma ferramenta que auxilia os desenvolvedores na geração de código fonte para a tecnologia JSP. A ferramenta agrega a parte funcional de manutenção de dados, sem a necessidade de conhecer a linguagem, atendo-se especialmente às estruturas de armazenamento, sendo que estas são obtidas a partir do dicionário de dados encontrado no banco de dados.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho foi desenvolver uma ferramenta para a geração de código para auxiliar analistas e programadores na construção de seus aplicativos na linguagem JSP, a partir das definições encontradas em uma base de dados MySQL.

Os objetivos específicos do trabalho são:

a) permitir ao usuário a configuração das páginas a serem criadas, informando quais serão as tabelas e atributos de cada página;

b) gerar automaticamente o código para relatórios e cadastros com as opções de alteração, exclusão e consulta.

1.2 MOTIVAÇÃO

No desenvolvimento de aplicativos web os programadores deparam-se diariamente

com rotinas básicas, como por exemplo, o desenvolvimento de cadastros. Ao construir uma interface para cadastros, conseqüentemente o usuário deverá ter a opção de fazer alterações e exclusões do mesmo, como também ter uma relação de todos os registros cadastrados.

Em todas as aplicações, os programadores necessitam fazer essas repetições dispendiosas gastando o seu limitado tempo disponível, enquanto poderiam enfatizar a parte mais complexa da aplicação.

Neste trabalho é disponibilizada uma ferramenta para geração automática de rotinas básicas a partir de uma base de dados MySQL para a linguagem JSP, assim, aumentando a produtividade no desenvolvimento de aplicativos web e minimizando a necessidade de programação.

1.3 ESTRUTURA DO TRABALHO

O segundo capítulo apresenta alguns conceitos gerais sobre JSP, *Servlets*, *Applets*, Driver JDBC, Banco de Dados MySQL e conceitos sobre geração de código citando algumas ferramentas, para que o leitor possa entender melhor sobre o funcionamento da ferramenta e o código gerado.

O terceiro capítulo apresenta a especificação e o desenvolvimento do trabalho, mostrando os diagramas de classes, casos de uso e de atividades, uma explicação de como a ferramenta foi desenvolvida e um estudo de caso para explicar o código gerado.

O quarto capítulo apresenta as conclusões, limitações e sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados alguns aspectos teóricos relacionados ao trabalho.

2.1 A TECNOLOGIA JSP

As aplicações atuais da web envolvem normalmente a manipulação de dados armazenados em um banco de dados. Assim, para que a apresentação de uma página possa ser construída dinamicamente, sem a necessidade de reescrever manualmente uma única linha HTML, são utilizadas tecnologias como JSP, PHP e ASP. Essa tecnologia tem o objetivo de conectar páginas a dados, enviando ao navegador como resposta uma página HTML que reflete o estado de uma base.

Segundo Fields e Kolbs (2000, p. 2), JSP é uma tecnologia baseada em Java que simplifica o processamento e desenvolvimento de *sites* web dinâmicos. Com JSP, os *designers* da web e programadores podem rapidamente incorporar elementos dinâmicos em páginas web usando Java embutido e algumas *tags* de marcação simples. Essas *tags* fornecem ao *designer* de HTML um meio de acessar dados e lógica de negócios armazenados em objetos Java sem ter que dominar as complexidades do desenvolvimento de aplicações.

Os arquivos JSPs são arquivos de texto armazenados no servidor, normalmente com a extensão .jsp que substituem as páginas HTML tradicionais. Os arquivos JSP contêm HTML tradicional com código embutido para permitir que aplicações Java sejam executadas no servidor. Quando uma página JSP é solicitada, o código é processado no servidor e o conteúdo dinâmico é obtido juntamente com o conteúdo HTML que é enviado ao cliente que solicitou a página. A primeira vez que uma página JSP é acessada, ela é automaticamente transformada em um *servlet* pelo servidor, através do mecanismo JSP, sendo que este *servlet* é

executado nos próximos acessos à página.

Para executar o JSP é preciso uma máquina virtual Java (JVM) e um s*ervlet container*. Existem vários *servlets container* gratuitos, dentre eles, os mais populares são o Apache Tomcat , JBoss e o Jetty.

2.2 VANTAGENS DO JSP

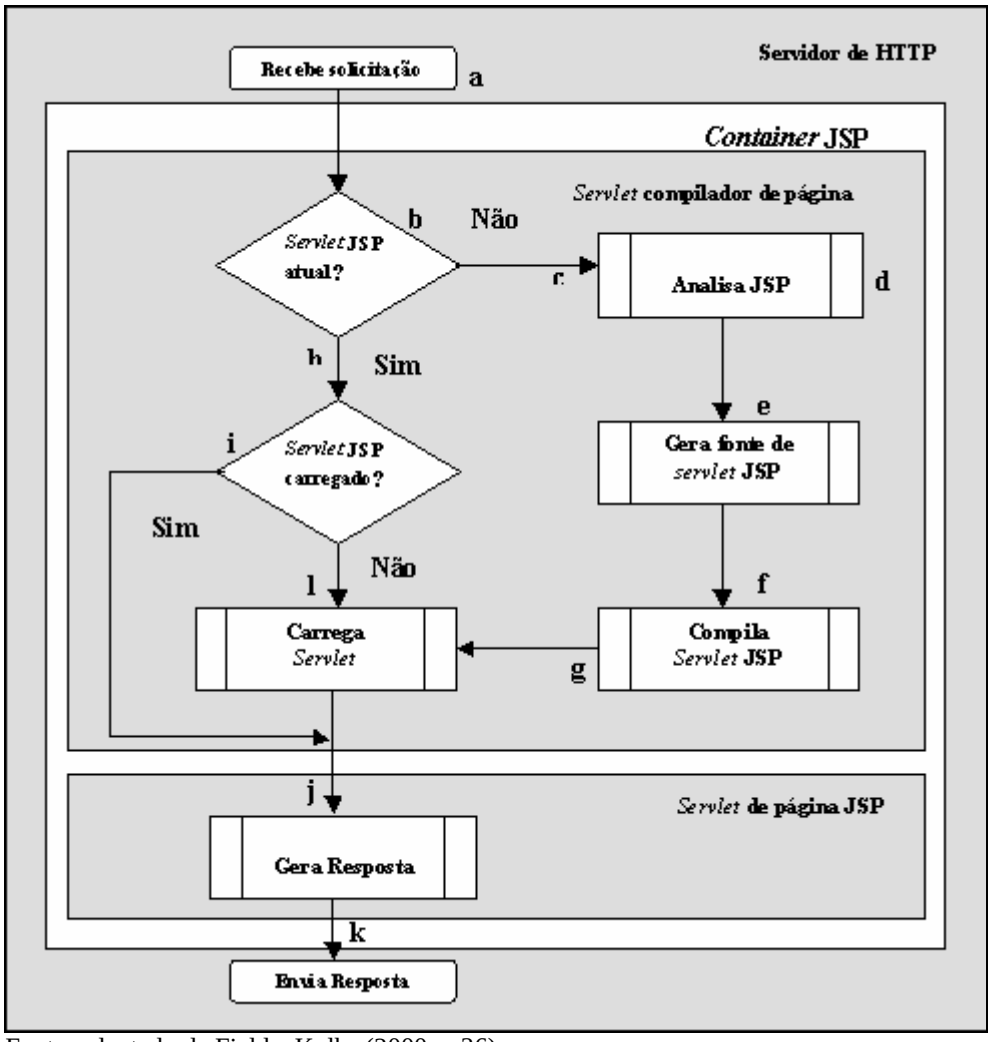
A necessidade de criar conteúdo dinâmico para a internet fez surgir uma grande variedade de tecnologias e interfaces para conectar programas e servidores web. Segundo Coelho (2000, p. 6), os arquivos JSP, assim como os *servlets*, são escritos em Java, os quais possuem as seguintes vantagens:

a) portabilidade: os programas em Java executam em qualquer plataforma e sistemas operacionais que tenham suporte sem a necessidade de recompilação ou ajustes;
b) reutilização: usando a orientação a objetos pode-se facilmente criar componentes e classes reutilizáveis diminuindo bastante o re-trabalho no desenvolvimento de sistemas;
c) padrão de mercado: a maioria das APIs lançadas pelos principais fabricantes de software do mercado oferece suporte para Java, entre elas a Oracle, IBM, Netscape. Além disso, existem APIs de extensão do Java que oferecem uma infinidade de recursos adicionais que podem ser usados em qualquer programa Java, seja ele um *servlet*, JSP ou aplicação;
d) separação da programação lógica e visual: JSP permite separar a programação da lógica (parte dinâmica – código Java) da programação visual (parte estática – código HTML), facilitando o desenvolvimento de aplicações mais robustas, onde programador e *designer* podem trabalhar no mesmo projeto, de forma

independente.

2.3 COMO FUNCIONA JSP

Segundo Field e Kolbs (2000, p.34), o componente principal de uma implementação baseada em *servlets* JSP é um *servlet* especial freqüentemente chamado de compilador de página. O processo do servidor para execução de *servlet* JSP é representado na Figura 1 e explicado a seguir:



Fonte: adaptado de Field e Kolbs (2000, p.36).

Figura 1 - Processo do servidor para a criação e execução de uma página JSP.

O *container* é configurado para chamar o *servlet* para todas as solicitações com URLs que combinem com a extensão JSP, e é a presença do *servlet* e suas classes associadas, que

transformam um *container servlet* em um *container* JSP. Cada página JSP é compilada em um *servlet* específico de página cujo propósito é gerar o conteúdo dinâmico específico através do documento JSP original.

A seguir são apresentados os detalhes do processo ilustrado na Figura 1:

a) o servidor HTTP recebe uma solicitação por URL correspondente a uma página JSP. Esta solicitação é enviada para o *container* JSP, que invoca o *servlet* compilador de página para tratar a solicitação;

b) o compilador de página, ao receber a solicitação de página JSP, verifica a hora da procura pelo arquivo JSP correspondente à URL solicitada, para determinar quando aquele arquivo foi modificado;

c) se nenhum *servlet* compilado for encontrado, ou se a hora do arquivo JSP for mais recente do que a hora do *servlet* de página compilado, então um novo *servlet* deve ser gerado, ou seja, o arquivo JSP deve ser analisado e traduzido em código fonte e este novo código fonte deve ser compilado;

d) para compilar a página, o compilador de página JSP faz uma análise de seu conteúdo, procurando por *tags* JSP;

e) enquanto analisa o arquivo, o *container* JSP traduz seu conteúdo no código fonte Java equivalente que, quando executado, irá gerar a saída indicada pelo conteúdo do arquivo original. O HTML estático é traduzido em cadeias Java que serão gravadas sem modificação e na sua seqüência original em um fluxo de saída;

f) uma vez que todo código *servlet* tenha sido construído, o *servlet* compilador de página chama o compilador Java resultante ao diretório apropriado no caminho de classe do *container* JSP;

g) o *servlet* recentemente compilado é carregado no *servlet* compilador de página;

h) se o *servlet* da página for o atual, então o *container* JSP precisa se certificar que o

servlet esteja atualmente carregado;

i) caso o *servlet* esteja carregado o controle é transferido do *servlet* compilador de página para o *servlet* de página JSP, que então lida com a solicitação;

j) a resposta é gerada pelo *servlet* da página JSP;

k) com a resposta gerada a página JSP é devolvida ao servidor HTTP para retornar ao navegador web;

l) se o *servlet* não estiver carregado, então o compilador de página carrega o mesmo.

Na primeira vez que uma página JSP é acessada, ela apresentará um *overhead* para ser exibida, devido ao processo que precisa ocorrer para criar os arquivos *.class* e então instanciar o *servlet* correspondente, mas todas as solicitações subseqüentes vão diretamente para o *servlet* compilado para geração de resposta.

2.3.1 Interação JSP e HTML

A maior parte do código de uma página JSP consiste em um *template text*. O *template text* é similar ao HTML. Para construir uma aplicação usando JSP, é escrito o texto no arquivo JSP entre as *tags* JSP que começam com "<%" e terminam com "%>". No Quadro 1 é mostrado um simples exemplo de página JSP.

```
<%@ page import = "java.util.*"%>
<HTML><HEAD>
<TITLE> Simples Teste de JSP </TITLE>
</HEAD>
<BODY>
      <CENTER>
            <H1><B>Olá, Hoje é dia: <%= new Date().toString()%></B></H1>
      </CENTER>
</BODY>
</HTML>
```

Quadro 1 - Exemplo de código JSP

Na Figura 2 é representada a execução do código apresentado no Quadro 1.

Figura 2 - Execução do código JSP

2.4 OBJETOS IMPLÍCITOS JSP

Uma característica do JSP é o *container* JSP deixar objetos implícitos que podem ser usados nas páginas JSP. Toda a aplicação JSP pode acessar objetos Java que são criados automaticamente. São nove objetos descritos a seguir:

a) *page*: referência à instância da classe que implementa o processamento JSP;

b) *config*: armazena dados de configurações de *servlet* na forma de parâmetros de inicialização;

c) *request*: objeto que o *container* usa para fornecer ao JSP informações sobre as requisições do cliente, incluindo nome e valores de parâmetros, atributos o fluxo de entrada;

d) *response*: representa a resposta que será enviada de volta para o usuário como resultado do processamento da página JSP;

e) *out*: representa o fluxo de saída para a página, cujo conteúdo será enviado para o navegador como o corpo da resposta;

f) *session*: identifica um usuário durante várias requisições ou visitas a um *site* web. A sessão de um usuário persiste durante um período que normalmente é

configurado no servidor, e o dono da sessão é identificado por meio de um identificador que é mantido usando *cookies* ou regravando URLs;

g) *application*: implementa a comunicação entre os *servlets* ou JSPs e o *container*. Permite obter os tipos MIME suportado pelo servidor, localizar outros *servlets* executados no servidor ou gravar no *log* do servidor;

h) *exception*: possui a exceção que a página de erro invocada tem de tratar. Este objeto é usado na página de erro (página JSP com o parâmetro *isErroPage* configurada com *true* na *diretive page*);

i) *pageContext*: fornece várias facilidades como gerenciamento de sessões, atributos, páginas de erro, inclusões e encaminhamento de requisições e fluxo de resposta.

2.5 APPLETS

Applets são pequenos programas construídos em Java e compilados em *bytecode*, formato reconhecido pela JVM de qualquer sistema operacional. Podem ser executados dentro de um navegador no cliente quando a página é acessada. Estes pequenos programas podem executar tarefas de controle de acessos, efeitos gráficos e segurança, como por exemplo, a criptografia.

Para rodar uma *applet* a partir de um documento HTML, usa-se a diretiva <*applet*>, a qual deve apresentar pelo menos três parâmetros: *code*, *width* e *height*, onde *code* define o nome da classe principal Java e *width* e *height* a largura e altura em *pixel* para a apresentação da *applet*, conforme mostrado no Quadro 2.

```
<html>
 <head><title>Exemplo de Applet</title></head>
 <body>
   <h1> Applet 1 </h1>
   <applet applet code="Applet1.class" width=500 height=500>
   </applet>
   <a href="Applet1.java"> Veja o Exemplo</a>
 </body>
</html>
```

Quadro 2 - Exemplo de chamada de applet

Como qualquer programa Java, uma *applet* deve conter ao menos uma classe pública, e essa classe pública deve estender a classe *applet*. A classe *applet* faz parte do *package applet* que integra o JDK e define o comportamento básico que uma *applet* deve ter. Esta permite a codificação de uma *applet* mínima em pouquíssimas linhas, como mostrado no Quadro 3.

```
import java.awt.Graphics;
import java.applet.*;

public class Applet1 extends Applet{
  public void paint(Graphics g){
   g.drawString("Olá ",5,50);
 }
}
```

Quadro 3 - Exemplo simples de *applet*

2.6 SERVLETS

Servlet é um componente que pode estender a funcionalidade de servidores. De acordo com Coelho (2000, p.31), os *servlets* por serem escritos em Java são independentes de plataforma podendo ser criados, compilados e executados em plataformas diferentes.

2.6.1 Estrutura de um *servlet*

Conforme Colla (1998), para criar um *servlet* utiliza-se a API `javax.servlet` que define a interface *servlet*. Todo *servlet* deve ser uma implementação desta interface, ou ser uma subclasse de uma classe que a tenha implementado. Nela estão definidos todos os métodos para estabelecer a comunicação com clientes.

Um exemplo de classe que implementa a interface *servlet* é a *HttpServlet*, utilizada para desenvolver *servlets* que são instalados em servidores web e que utilizam o protocolo HTTP para enviar e receber informações.

A comunicação entre um *servlet* e um cliente se dá por meio de dois objetos, um da classe `ServletRequest`, que encapsula as funções de comunicação do cliente para o servidor, permitindo que o *servlet* receba dados como o conteúdo de um formulário HTML, e outro da classe `ServletResponse` que encapsula as funções de comunicação do servidor para o cliente e as subclasses `HttpServletRequest` e `HttpServletResponse` implementam métodos para lidar com informações transmitidos via HTTP.

Os *servlets* possuem três métodos da interface *servlet*: `init`, `service` e `destroy`. Os métodos `init` e `destroy` são usados para alocar e liberar recursos que tem de estar disponíveis durante a vida do `servlet` como conexão com o banco de dados e outros objetos. O método `service` trata dos processamentos dos pedidos, ou seja, recebe as informações do cliente, realiza o processamento e envia uma resposta ao cliente.

Segundo Horstmann (2004, p.989), ao escrever um s*ervlet* é fornecida uma classe que estende a classe `HttpServlet` que implementa os métodos POST e GET. No .Quadro 4 é mostrado um exemplo simples de *servlet*. Neste exemplo o *servlet* obterá a hora atual, criará um documento HTML e o enviará de volta ao navegador.

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class Hello extendes HttpServlet{
      public void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException{

      response.setContentType (“text/html”);
      PrintWriter out = response.getWriter();
      Out.println(“<html><body>”);
      Out.println(“Agora são ”+new Date().toString());
      Out.println(“</body></html>”);
      Out.close();
      }
}
```

.Quadro 4 - Exemplo de *servlet*

O parâmetro `response` permite especificar o que enviar de volta ao servidor. No exemplo foi usado o `response.setContentType(“text/html”)` para configurar o tipo do conteúdo para HTML, e retornar ao navegador um documento HTML. O `PrintWriter` foi usado para coletar o documento que deve ser devolvido ao navegador.

2.7 JSP X SERVLETS X APPLETS

As tecnologias JSP e *servlets* são muito semelhantes. Todo JSP é compilado na forma de um *servlet*. Ambos são processados no lado do servidor e retornam para o cliente apenas conteúdo HTML. JSPs são feitos especialmente para gerar HTML, enquanto que *servlets* são mais adequados para tarefas de controle da aplicação. Porém nada impede que o programador utilize-os com os papéis invertidos. A principal diferença entre eles é que com o JSP o programador escreve código Java no HTML, e com *servlet* se escreve HTML dentro do código Java. Na Figura 3 estão representada graficamente a diferença entre *servlets* e JSP.

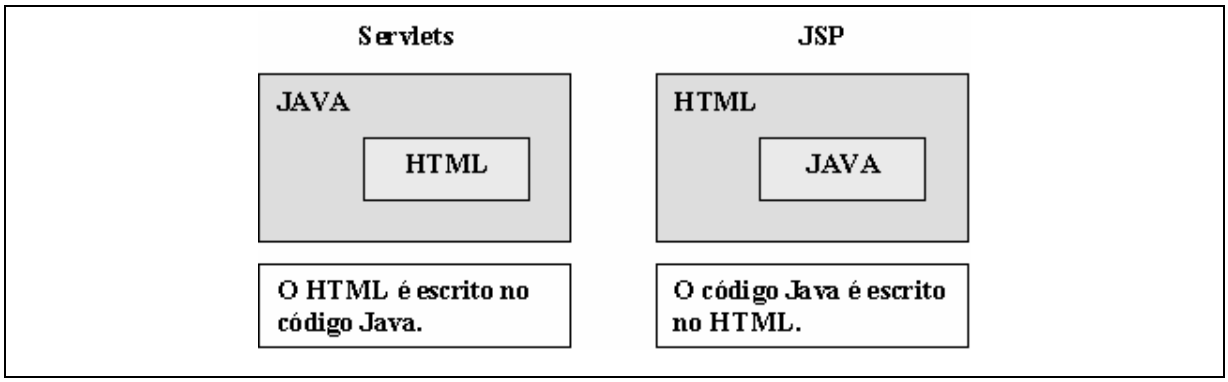
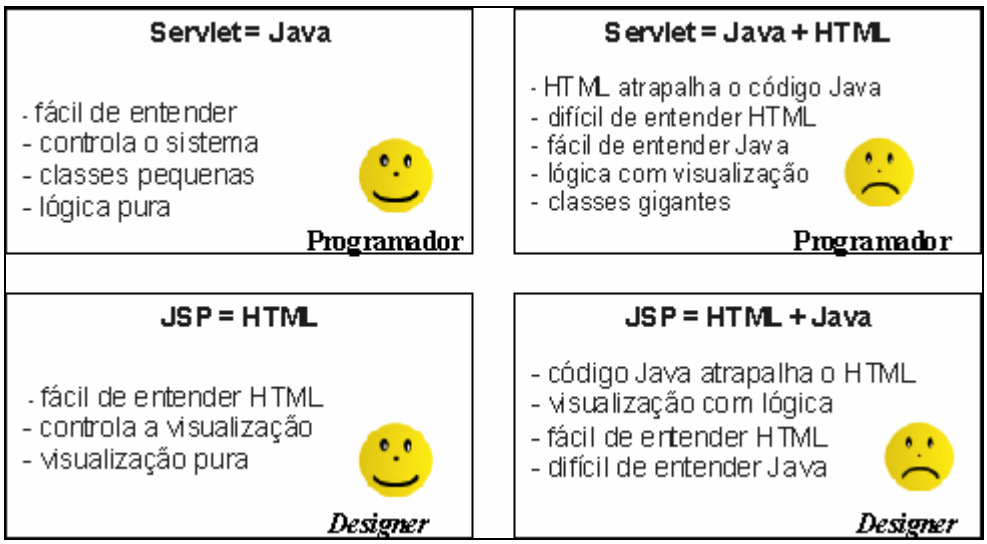


Figura 3 - Diferença entre JSP e *servlets*

A vantagem do JSP é a facilidade de se alterar o HTML, separando o HTML da parte funcional podendo ter várias pessoas realizando tarefas diferentes no momento do desenvolvimento, como por exemplo, o web *designer* pode fazer o HTML e os programadores Java a parte funcional. Enquanto que o JSP facilita a alteração do código HTML pelo *designer* os *servlets* facilitam a programação orientada a objetos pelos programadores. Na Figura 4 estão exemplificadas as facilidades e dificuldades de JSP e *servlets* em relação aos programadores e *designers*.



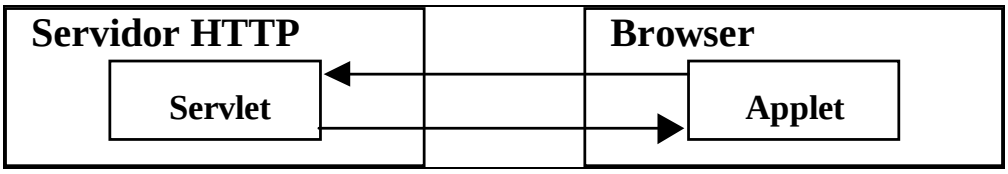
Fonte: adaptado de Caelum (2004, p.39).

Figura 4 - Facilidades e dificuldades dos programadores e *designer*

As *applets* rodam no lado do cliente, ou seja, todo o processamento é feito na máquina de quem está utilizando a *applet*. Elas não usam HTML como interface gráfica, ao invés disto usam as APIs de interface gráfica do Java, como o AWT e *Swing*. Isto requer que o usuário faça o *download* de toda a aplicação, o que gera uma espera muitas vezes inconveniente. A

grande vantagem é a possibilidade de colocar dentro do navegador, uma aplicação rica em recursos e de resposta rápida, como um editor de textos, por exemplo.

As *applets* são para os *browsers* o que os *servlets* são para os servidores, ou seja, da mesma forma que o *servlet* necessita do servidor Java, a *applet* necessita do *browser* para ser executada. Como pode ser observado na Figura 1 o processo de comunicação entre uma *applet* e um *servlet* é realizado em locais diferentes, uma vez que o *servlet* está instalado em um servidor remoto na internet e a *applet* pode estar rodando em um *browser* .



Fonte: adaptado de Furgeri (2003, p. 330)

Figura 5 - Comunicação entre o *servlet* e *applet*

2.8 DRIVER JDBC

O acesso a banco de dados é fundamental em aplicações web. Szolkowski e Todd (2003, p.361), afirmam que os bancos de dados são comumente usados para armazenar informações sobre produtos para um *site* comercial, informações sobre usuários, estatísticas de uso, entre outras.

Quando uma aplicação JSP precisa se comunicar com um banco de dados, ela o faz através do *driver* JDBC que permite programas Java interagirem com bancos de dados. Segundo Fields e Kolb (2000, p.182), o acesso ao banco de dados dentro de um *servlet* é isolado, mantendo os detalhes ocultos dos aspectos de apresentação da página JSP.

Para estabelecer a conexão com o banco de dados, utiliza-se a classe `DriverManager` que é registrada ao carregar o *driver*. A interface `Connection` designa um objeto para receber a conexão estabelecida. Um exemplo de conexão com o banco

MySQL é mostrado no Quadro 5. Pode-se usar a interface `Statement` que representa uma consulta de SQL ou a instrução `PreparedStatement` para executar comandos fundamentais do SQL (*Select*, *Insert*, *Update* ou *Delete*) no banco de dados. A instrução `PreparedStatement` é uma instrução parametrizada, ou seja, pré-compilada e os parâmetros são passados em tempo de execução, sendo ideal para aplicações web em que os valores desses parâmetros poderiam entrar a partir de formulários HTML.

```
      try{  //A captura de exceções é obrigatória em Java para usar JDBC
            //Registrar o Driver
            Class.forName("com.mysql.jdbc.Driver").newInstance();
            //conecta através da interface
            Connection conn = DriverManager.getConnection
("jdbc:mysql://127.0.0.1/base?user=root&password=''");
          //Criar o Statement
          Statement   stmt = conn.createStatement();
        }catch(SQLException e) {
         //se houve algum erro é gerada uma exceção
        System.out.println("Erro na Conexão "+e.getErrorCode() );}
```

Quadro 5- Conexão com o banco de dados MySQL

A interface `ResultSet` permite trazer os resultados obtidos na execução da consulta SQL no banco de dados. No Quadro 6 é mostrada a execução de uma SQL que busca registros no banco de dados.

```
//consulta a ser executada
String sql = "Select * from Clientes";
//executar a consulta no banco de dados
ResultSet rs = stm.executeQuery(sql);
//o método next () informa se houve resultados
//Como é esperado várias linhas é preciso um laço para recuperar os dados
 while (rs.next()){
        String nome   =   rs.getString("nome_cliente");
        String cidade =  rs.getString("cidade_cliente");
 }
```

Quadro 6 - Exemplo de execução de uma consulta SQL

Após o término do acesso, é importante liberar os recursos alocados pelo banco de dados, fechando o `Statement`, para liberar os recursos associados à execução das consultas SQL, deixando a conexão aberta para a execução da próxima consulta ou fechando diretamente a conexão com o método `close` que encerra a comunicação com o banco de dados.

2.9 BANCO DE DADOS MYSQL

Banco de dados é o local onde são guardados os dados pertencentes a uma aplicação. Segundo Lima (2003, p. 5), existe hoje no mercado vários SGBDs, sendo que grande parte deles oferece robustez, confiabilidade, rapidez, portabilidade, segurança, escalabilidade e outros benefícios.

Entre tantos gerenciadores de banco de dados disponíveis, encontra-se o MySQL, que consome poucos recursos do sistema operacional, possui extrema velocidade e compatibilidade, é fácil de usar, confiável e extremamente rápido além de possuir um baixo custo e ser *open source*.

O MySQL suporta tabelas de diferentes tipos como ISAM, HEAP, MyISAM, MERGE, BDB e InnoDB. Segundo Duarte (2004), o MySQL possui uma característica um pouco diferente dos SGBDs, uma vez que no MySQL é possível escolher o tipo da tabela no momento da criação da mesma. O formato de armazenamento dos dados, bem como alguns recursos do banco de dados é dependente do tipo de tabela escolhido.

A vantagem da utilização de tipos de tabela é que se pode optar por utilizar um determinado tipo ou outro dependendo dos requisitos exigidos pela aplicação. Por exemplo, se a aplicação é apenas de consulta pode-se optar pelo tipo MyISAM em vez do InnoDB, com isto evita-se o *overhead* da transação, obtendo um maior desempenho. É importante lembrar que se pode utilizar vários tipos de tabelas em um mesmo banco de dados. Neste caso, deve-se tomar cuidado com a utilização de determinados recursos, por exemplo, se tiver uma transação envolvendo tabelas InnoDB e MyISAM e for submetido um comando ROLLBACK, apenas os comandos executados no InnoDB serão desfeitos, enquanto que os comandos executados no MyISAM persistirão. A utilização do tipo de tabela correto é um fator chave para determinar o desempenho da aplicação.

No Quadro 7 são apresentados os tipos de tabelas existente no MySQL e uma breve explicação sobre cada uma delas.

Tabela	Características
HEAP	São armazenadas em memória; São rápidas; Conteúdo volátil; Ideal para tabelas que são consultadas com muita freqüência;
MERGE	Divisão de tabelas grandes em tabelas pequenas; Coleção de tabelas MyISAM idênticas;
BDB	Fácil de manipular o controle de transações; Recuperação automática de dados;
InnoDB	Possui integridade referencial; Tipo de tabela transacional; Armazenamentos de dados em *tablespace*; Requer mais espaço em memória;
MyISAM	Tabela padrão do MySQL; Melhor desempenho para leitura, devido aos índices serem armazenados em arvores binárias balanceadas; Não possui integridade referencial; Permite o controle de transações; Há limite de tamanhos; São tabelas rápidas;

Quadro 7 - Tipos de tabelas MySQL

2.10 GERAÇÃO DE CÓDIGO

Geradores de código são ferramentas que produzem o código sem erro de sintaxe a partir de projetos, gráficos e especificações de alto nível. De acordo com Santos (2002), no desenvolvimento de uma aplicação, os programas geradores de código são extremamente práticos quando se tem pouco tempo ou uma equipe pequena. Esta afirmação é verdadeira quando se imagina um aplicativo complexo quanto ao uso, que gere o código completo sem a necessidade de personalizações e fornece uma série de opções para a escolha da linguagem de programação ou do banco de dados que se deseja utilizar.

Segundo Herrington (2003, p.93), para o desenvolvimento de um gerador de código, o ideal seria seguir as seguintes etapas:

a) construir um código de teste: o primeiro passo é construir manualmente um protótipo de como será a saída do código gerado;

b) projetar o gerador: uma vez que possui a saída, deverá determinar como será construído o gerador que criará o código de teste como saída. O mais importante é a especificação dos dados requeridos. Tendo esses dados especificados, deverá ser definido um arquivo de entrada;

c) desenvolver um *parser* de entrada: o primeiro passo para a implementação é definir um arquivo *parser*;

d) desenvolver *template* para código de teste: com os arquivos de entrada já definidos, poderão ser criados *templates* para gerar os arquivos de saída;

e) desenvolver o código de saída: o último passo é escrever o código que através dos arquivos de entrada definidos no *template* e geram os arquivos de saída. No final do desenvolvimento pode ser testado com o código teste definido no início do processo.

Essas etapas são os pontos iniciais para o desenvolvimento de um gerador de código e estão representadas na Figura 6.

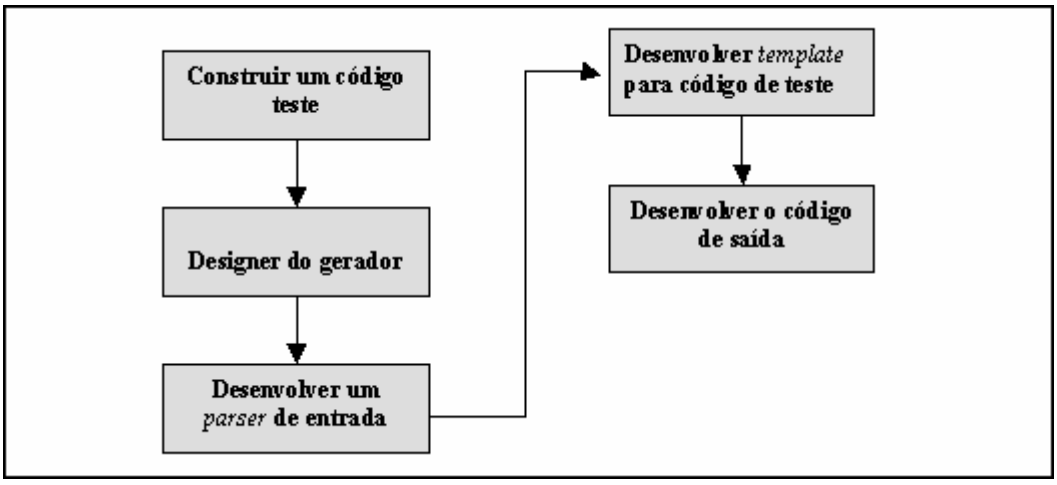


Figura 6- Passos para o desenvolvimento de um gerador

Cabe ressaltar que este trabalho não utiliza *templates*.

2.11 FERRAMENTAS PARA GERAÇÃO DE CÓDIGO

Existem no mercado ferramentas para geração de código, dos mais variados tipos, entre elas pode-se citar o CodeCharge, DBDesigner, ASPSYS e Genexus. A seguir são apresentadas algumas características das mesmas.

2.11.1 CodeCharge

Conforme Santos (2002), o CodeCharge é uma 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. É possível criar *sites* web nas linguagens ASP, JSP, PHP, Perl, ColdFusion, entre outras.

O CodeCharge suporta o uso de código SQL customizado, além do próprio código gerado pela aplicação, através do uso de formulários de tabelas, de registros, *login*, pesquisas e menus. O desenvolvedor pode especificar através desses formulários, a ação baseada em funções SQL e o que mostrar quando o banco de dados for acessado, além de determinar os níveis de autenticação de usuário na aplicação.

Além disso, possui funcionalidades que auxiliam no trabalho de *design*, como uma interface para a geração de arquivos CSS ou pode optar por *templates*.

O CodeCharge permite a conexão com vários bancos de dados relacionais, incluindo Oracle, Sybase, MySQL, Microsoft SQL Server e Microsoft Access, que sejam passíveis de conexão via JET, ODBC, JDBC, ADO e PHPLib.

Os códigos gerados são portáveis em qualquer versão do Windows, Unix e Linux.

2.11.2 DBDesigner

É um editor visual para criação de projetos de bancos de dados MySQL que integra criação, modelagem, desenvolvimento e manutenção dos bancos. O DBDesigner é o*pen source* distribuído sobre a licença GPL.

De acordo com Fabulous Force Database Tools (2003), no DBDesigner o foco é um modelo de dados. Um modelo de dados é uma visualização da meta-informação armazenada em uma base de dados (por exemplo, tabelas e índices, relacionamentos). Embora seja possível armazenar dados iniciais para cada tabela diretamente no modelo, é representada somente a meta-informação e não os próprios dados.

Um objeto pode ser uma tabela da base de dados com colunas e índices ou uma relação entre duas tabelas. Os novos modelos podem ser construídos colocando estes objetos no modelo ou podem ser recuperados das bases de dados existentes usando uma função da engenharia reversa. Pode ser criada uma base de dados exportando um modelo através de uma sentença SQL. O DBDesigner possui uma função para sincronizar as alterações na base de dados quando o modelo for alterado.

2.11.3 ASPSYS

De acordo com Rizo (2004), a ferramenta ASPSYS auxilia o desenvolvimento de aplicações para a web que utilizem banco de dados. Esta ferramenta disponibiliza aos seus usuários as funcionalidades básicas de uma ferramenta RAD e possibilita a geração de código fonte em rotinas básicas para o sistema pretendido pelos usuários.

A base de funcionamento da ferramenta é a criação de projetos separados para cada

sistema que se pretende construir.

Segundo Rizo (2004), um bom diferencial da ferramenta ASPSYS é o fato de ela ter sido implementada utilizando uma linguagem de programação para web (ASP), o que possibilita que fique “instalada” em um único local e possa ser utilizada por mais de uma pessoa ao mesmo tempo, lembrando que um usuário não interfere no trabalho de outro, pois o servidor web cria sessões diferentes de trabalho para cada pessoa que está acessando-o. Este fato possibilita boa produtividade e independência de criação e trabalho. No entanto, vale ressaltar que a única forma de um usuário interferir no trabalho de outro é se ambos estiverem utilizando o mesmo projeto dentro do ASPSYS.

2.11.4 Genexus

Genexus é uma ferramenta de desenvolvimento que faz a geração automática de aplicativos. Segundo Dias (2002, p. 55) Genexus usa objetos de usuários como um começo de análise, capturando o conhecimento desses objetos e sistematizando-os em uma base de conhecimento. Os tipos de objetos utilizados pela ferramenta Genexus são:

a) transações: são visões do usuário que possuem um diálogo associado e que podem modificar o conteúdo da base de dados;
b) relatórios: são objetos que permitem realizar consultas, geradas pela aplicação;
c) procedimentos: são objetos que permitem criar ou modificar as informações na base de dados;
d) painéis de trabalhos: são consultas interativas que permitem aos usuários obterem informações de forma dinâmica, orientando a pesquisa em tempo de execução;
e) web *objects*: possuem as mesmas características dos painéis de trabalhos, porém via internet.

O desenvolvimento de uma aplicação implica tarefas de análise, desenvolvimento e implementação. A ferramenta de desenvolvimento de aplicações Genexus objetiva liberar as pessoas das tarefas automatizadas, permitindo que estas se dediquem mais as tarefas de análise.

3 DESENVOLVIMENTO DO TRABALHO

Neste capítulo é apresentada a ferramenta e a implementação juntamente com os resultados obtidos.

3.1 VISÃO GERAL DA FERRAMENTA

A ferramenta desenvolvida facilita a construção de aplicativos para web na linguagem JSP. Na Figura 7 está representada uma visão geral sobre o funcionamento da ferramenta.

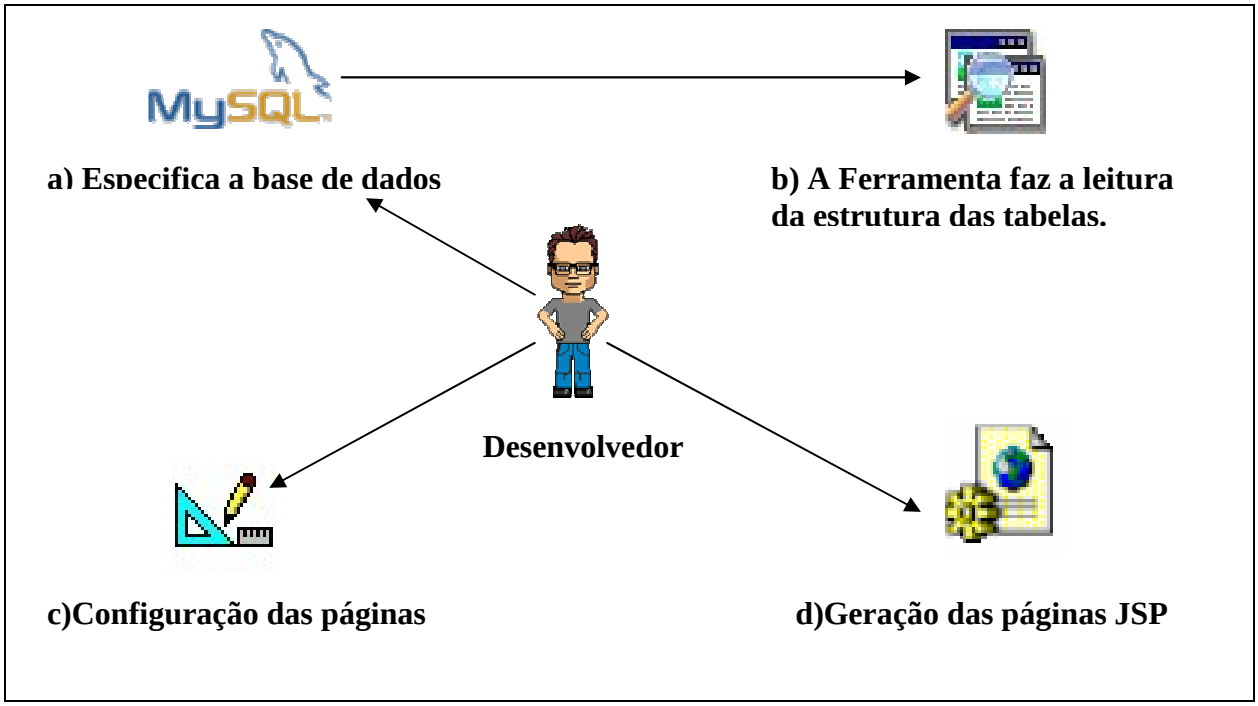


Figura 7 - Visão geral da ferramenta

Para usar a ferramenta primeiramente os desenvolvedores precisam ter definido a estrutura do dicionário de dados no banco de dados MySQL. Feita a especificação do modelo de dados, o desenvolvedor faz a conexão com o banco de dados e a ferramenta faz a leitura da estrutura das tabelas, campos e tipos. Toda a estrutura lida será armazenada na memória em uma estrutura de lista que será utilizada pelo desenvolvedor para configurar suas páginas e a

geração das mesmas.

3.2 LEVANTAMENTO DOS REQUISITOS

A atividade de levantamento de requisitos corresponde à etapa de compreensão do problema aplicada ao desenvolvimento do software. O principal objetivo do levantamento de requisitos é de que usuários e desenvolvedores tenham a mesma visão do problema a ser resolvido.

Os requisitos funcionais determinam as funcionalidades do sistema. Esta ferramenta possui os seguintes requisitos funcionais:

a) permitir a leitura da base de dados do projeto, verificando os relacionamentos entre as tabelas e a estrutura dos campos;
b) permitir que o usuário personalize sua página, selecionando as tabelas e atributos que serão visualizadas na página;
c) permitir que o usuário faça alterações no código fonte a ser gerado;
d) permitir a geração de código na linguagem JSP para relatórios e formulários de cadastros, alterações e exclusões;
e) permitir o cadastro de projetos.

Os requisitos não funcionais declaram as características de qualidade que o sistema deve possuir e que estão relacionadas as suas funcionalidades. Esta ferramenta tem os seguintes requisitos não funcionais:

a) permitir a conexão com um banco de dados MySQL para o acesso à base de dados do projeto;
b) o código gerado deverá ser compatível com o padrão HTML definido pela W3C.

3.3 ESPECIFICAÇÃO

A especificação da ferramenta foi feita através do Enterprise Architect para o desenvolvimento dos diagramas de caso de uso, atividades e classes.

3.3.1 Diagrama de caso de uso

Um caso de uso é a especificação de uma seqüência de interações entre um sistema e os agentes externos. Para o funcionamento da ferramenta foram identificados os seguintes casos de usos: criar projeto, selecionar base de dados, configurar páginas, visualizar páginas e gerar código que estão representados na Figura 8.

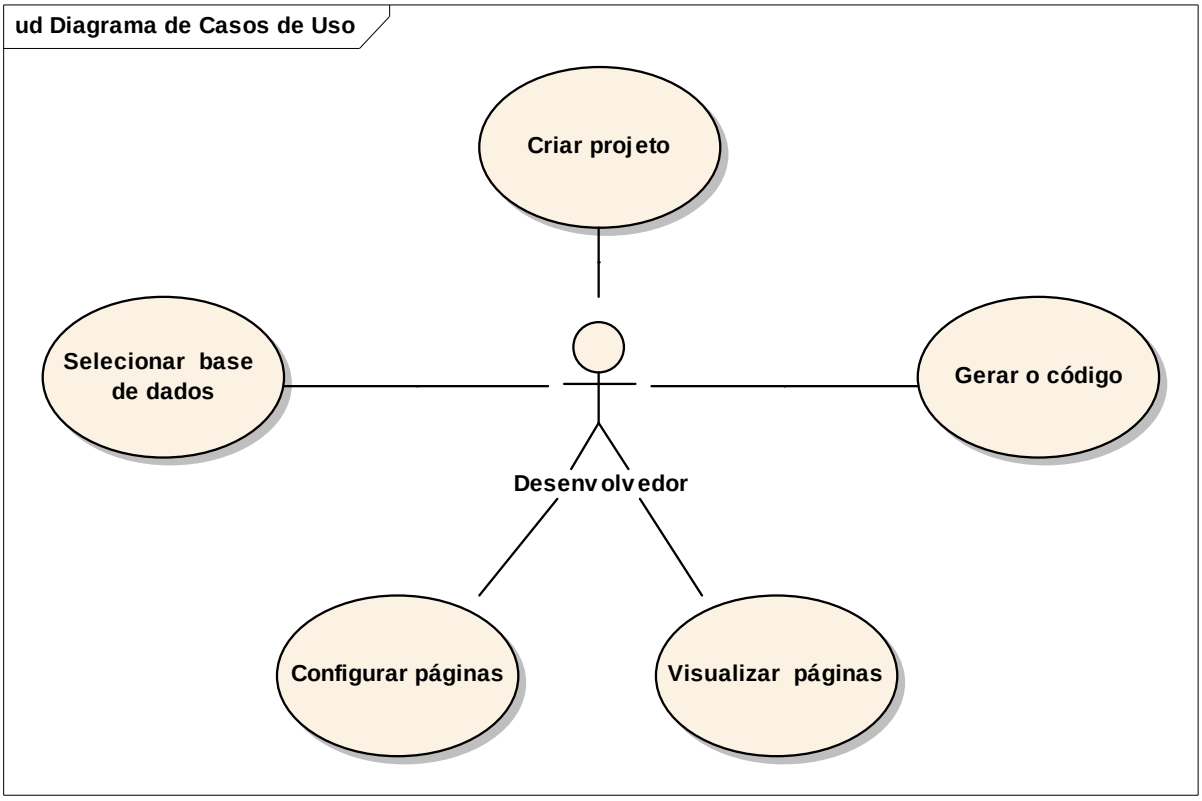


Figura 8 - Diagrama de caso de usos

Segue a descrição dos casos de usos identificados:

a) criar projeto: o desenvolvedor cria um projeto onde será cadastrada a informação geral do projeto;
b) selecionar base de dados: o desenvolver precisa selecionar a base de dados já especificada para o início de um novo projeto e para a leitura das configurações da base de dados;
c) configurar páginas: o desenvolvedor cria as visões do projeto, os menus e configura as opções de relatório e formulários das páginas que serão geradas;
d) visualizar páginas: o desenvolvedor faz uma pré-visualização das páginas de relatório que serão geradas;
e) gerar o código: o desenvolvedor solicita o início da geração e o sistema gera as páginas de menu, relatórios, consultas, inclusões, exclusões e alterações.

3.3.2 Diagrama de atividades

De acordo com Bezzera (2002, p.228), um diagrama de atividade é um tipo especial de diagrama de estados de uma atividade. Na Figura 9 é representado o diagrama de atividades da ferramenta.

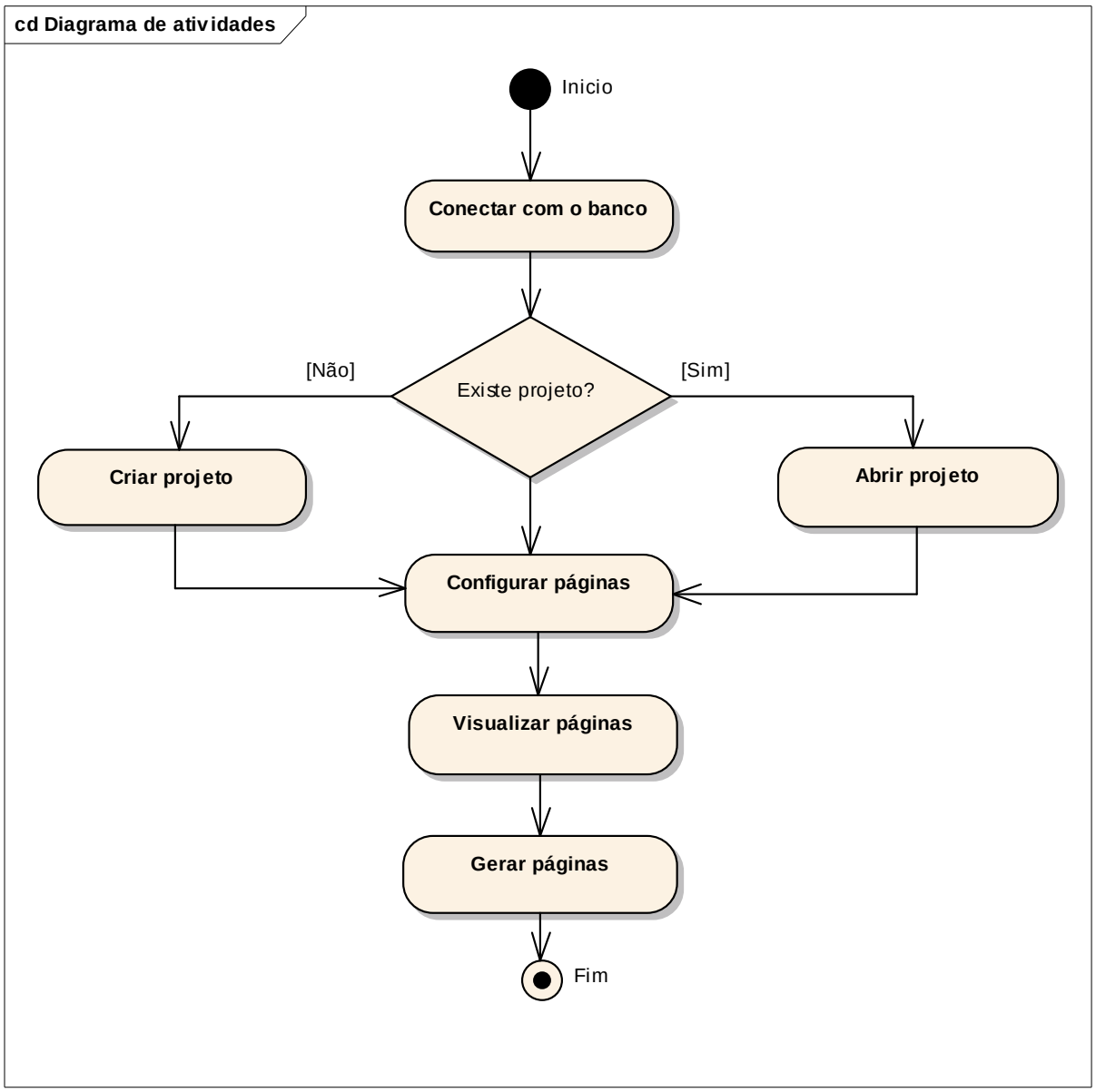


Figura 9 - Diagrama de atividades

3.3.3 Diagrama de classes

O diagrama de classes da ferramenta está representado na Figura 10.

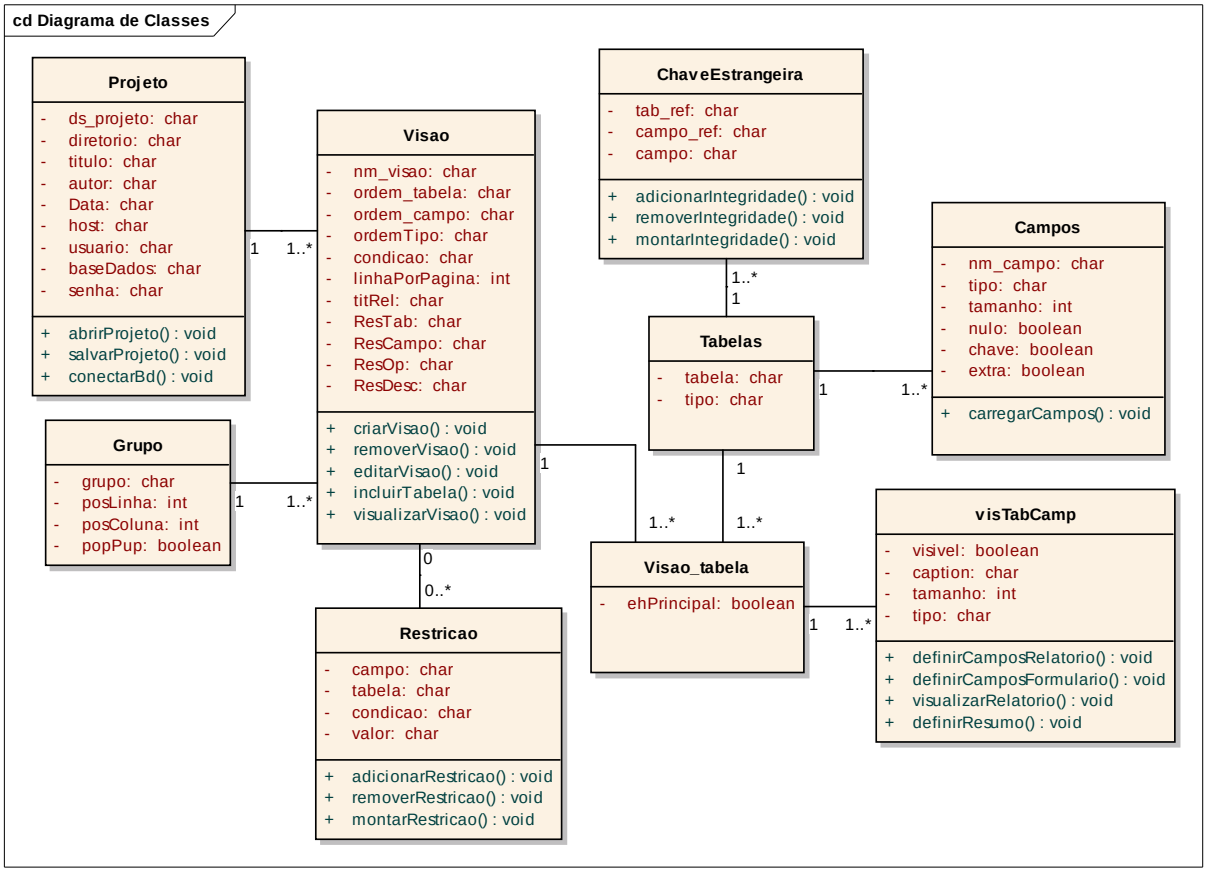


Figura 10 - Diagrama de classes

Segue uma breve descrição das classes utilizadas para o desenvolvimento da ferramenta:

a) projeto: esta classe controla as configurações gerais do sistema como a descrição do projeto, o título da página web a ser gerada e o caminho em que serão gravadas as páginas geradas;

b) visão: esta classe mantém as informações da visão a ser criada, como o campo e a tabela que será utilizado para fazer a ordenação, as opções gerais de configurações dos relatórios e formulários;

c) restrição: esta classe guarda as restrições da visão;

d) tabelas: controla as tabelas encontradas no banco de dados;

e) chaveEstrangeira: esta classe controla a integridade referencial entre as tabelas;

f) campos: armazena toda a estrutura de um determinado campo;

g) visTabCamp: armazena a forma em que cada campo será visualizado nos formulários e relatórios;
h) grupo: armazena as informações de um determinado grupo, que é a associação de várias visões.

3.4 IMPLEMENTAÇÃO

Esta ferramenta foi desenvolvida na linguagem Java JDSK 1.5 através do ambiente de programação Eclipse 3.0 e com as definições encontradas no banco de dados MySQL para a geração das páginas.

3.4.1 Técnicas e ferramentas utilizadas

O ambiente de programação Eclipse (ECLIPSE, 2004) é um software livre criado pela IBM, que visa fornecer infra-estrutura para construção de softwares, baseada em arquitetura de *plug-ins* independente de sistemas operacionais.

Para testar o código gerado pela ferramenta, foi utilizado o servidor de aplicações JBoss que é um projeto *open source* destinado a servir aplicações Java. Em uma arquitetura web de três camadas, o servidor JBoss ocupa a segunda camada onde controla a inteligência da aplicação controlando o trafego de dados entre o servidor web e o banco de dados.

3.4.2 Arquitetura

A Figura 11 apresenta a navegabilidade da ferramenta desenvolvida.

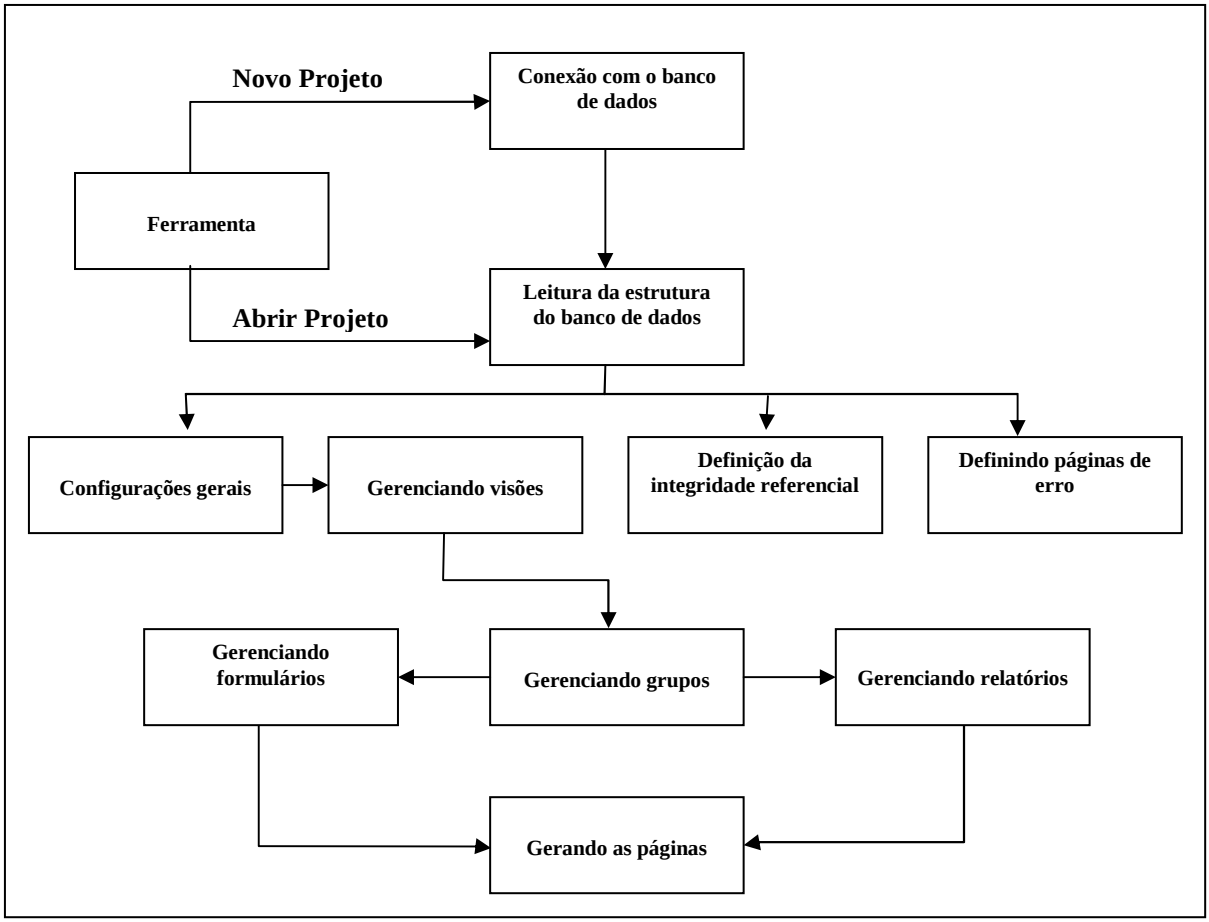


Figura 11 – Navegabilidade da ferramenta

A seguir são apresentadas as telas de cada elemento apresentado na arquitetura da ferramenta com uma breve explicação de sua funcionalidade.

Na tela inicial da ferramenta tem-se a opção de criar um novo projeto ou selecionar um projeto existente conforme é mostrado na Figura 12. Um projeto é a junção de várias páginas. O conjunto destas páginas pode formar um determinado sistema.

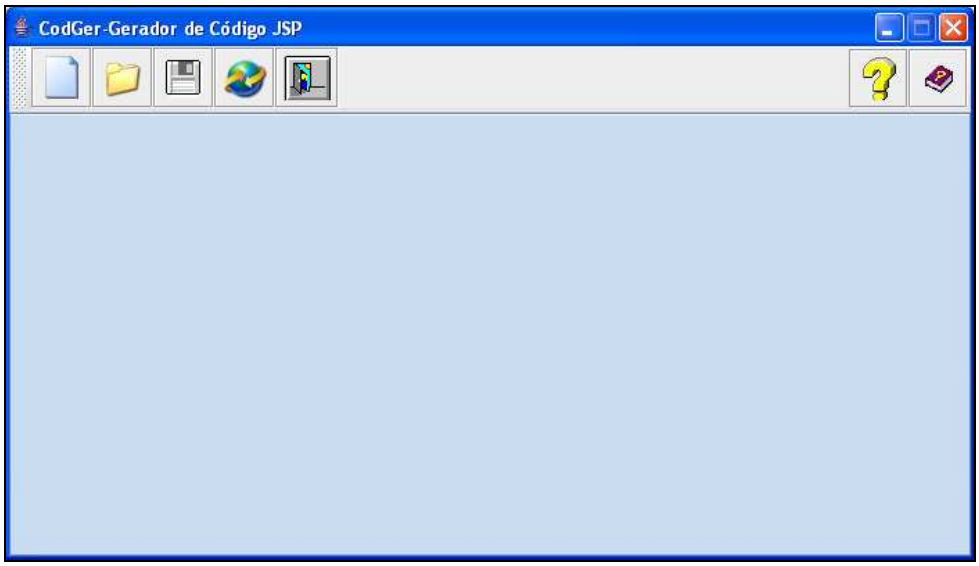


Figura 12 - Tela de abertura da ferramenta

Para o início das configurações das páginas do projeto, é preciso fazer a conexão com o banco de dados. Quando um projeto é aberto, a conexão é automática, pois junto com o projeto são salvos os dados de conexão. Ao criar um novo projeto o desenvolvedor deverá informar os dados de conexão como é mostrado na Figura 13.

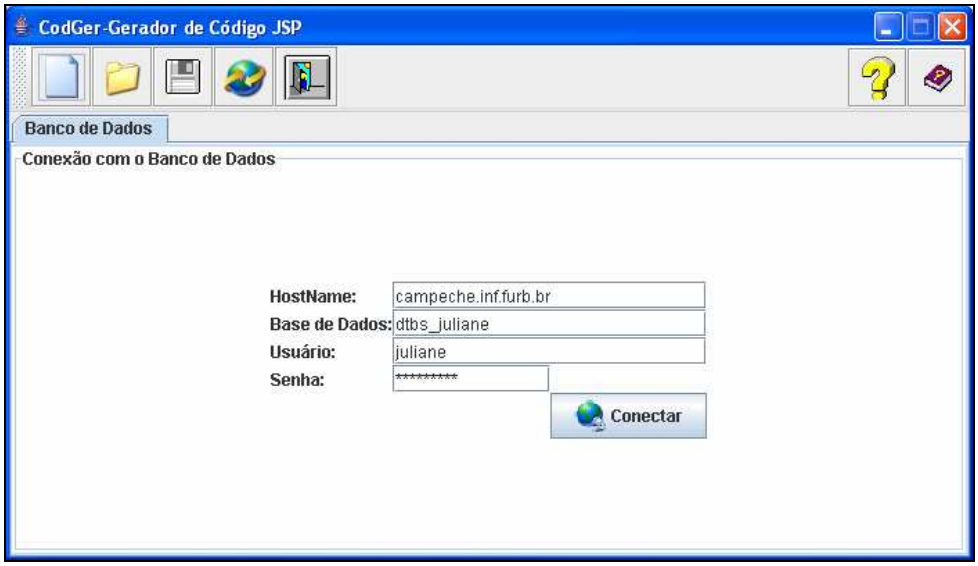


Figura 13 - Tela de conexão com o banco de dados

Feita a conexão, inicia-se a leitura da estrutura do banco de dados armazenando-a na memória em uma estrutura de lista para oferecer ao desenvolvedor uma maior rapidez na configuração de suas páginas e evitar acessos sucessivos ao banco de dados. No Quadro 8 está o código utilizado para fazer a leitura da estrutura do banco de dados.

```
private void carregaTabs(){ //lê todas as tabelas da base
      ResultSet rsTab;
       try {
            rsTab = this.cn.SQLexecuteQuery ("show table status");
            if (rsTab != null){
               while (rsTab.next()){
                   Tabela tab  =  criaTabelas(rsTab);
                      tabelas.add(tab);
               }
             }
        }
        catch (Exception e){System.err.println (e.getMessage());
}
private void carregaEstrutura(){ //lê a estrutura dos campos da tabela
   ResultSet rsCamp;
   try {
      Iterator i = tabelas.iterator();
      while ( i.hasNext()) {
        Tabela tp = (Tabela) i.next();
        rsCamp = this.cn.SQLexecuteQuery ("Desc "+tp.getTabela());
        if (rsCamp != null)
         while (rsCamp.next()){
            Campos camp =  criaCampos(rsCamp,tp.getTabela().toString());
            campos.add(camp);
        }
     }
  }
  catch (SQLException e) {  System.err.println (e.getMessage());}


private  Campos criaCampos(ResultSet rsCamp, String tabela){ //instancia os
campos {
  try{
        Campos camp = new Campos();

        camp.setTabela(tabela);
        camp.setCampo(rsCamp.getString(1));
        String tipo = rsCamp.getString(2);
        //verificifa o tipo do campo
        camp.setTipo(verificaTipo(tipo));
        //verifica o tamanho
        camp.setTamanho(verificaTam(tipo));
        //verifica se o campo é nulo
        camp.setNulo(verificaNulo(rsCamp.getString(3)));
        //verifica se é chave
        camp.setChave(verificaChave(rsCamp.getString(4)));
             //verifica se é auto_incremento
        camp.setIncremento(verificaIncremento(rsCamp.getString(6)));

      return camp;
}catch(Exception e){System.out.print(e.getMessage()); return null;}
}
```

Quadro 8 - Leitura da estrutura do banco de dados

Após feita a conexão com o banco de dados, é preciso que sejam informadas as configurações gerais do projeto conforme a Figura 14.

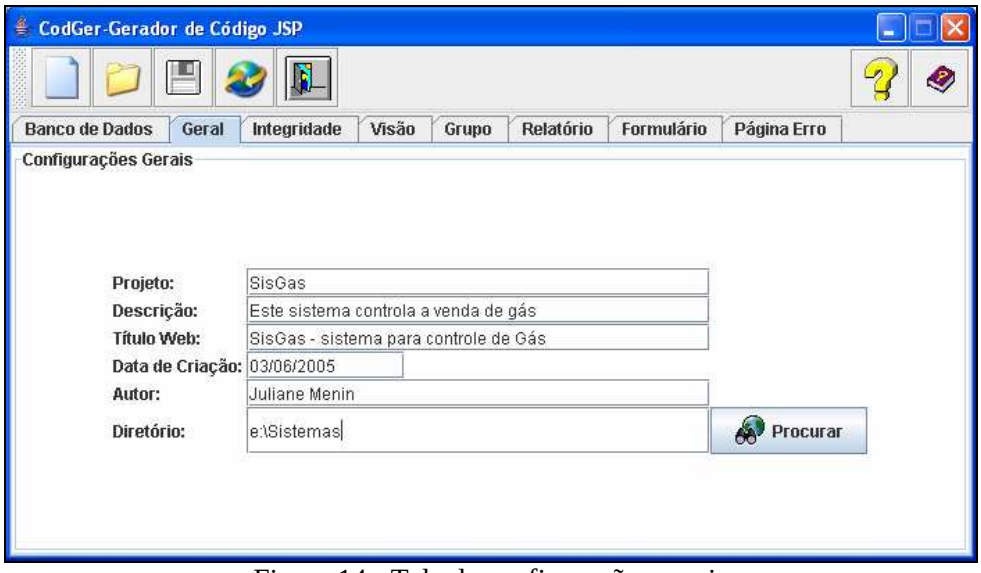


Figura 14 - Tela de configurações gerais

Estas configurações gerarão um cabeçalho de autoria padrão para todos os arquivos gerados pela ferramenta. Na Figura 15 está representado um exemplo de módulo de autoria.

```
/**************************************************************************
* Este arquivo foi gerado através da ferramenta GerCod - Gerador de Código para JSP,
* desenvolvido por Juliane Menin como trabalho de conclusão de curso sob orientação do
*professor Alexander Roberto Valdameri  da Universidade Regional de Blumenau – FURB.
*
* Projeto:SisGas
* Descrição:Este sistema controla a venda de gás
* Data de Criação:03/06/2005
* Gerado por: Juliane Menin
*
**************************************************************************/
```

Figura 15 - Módulo de autoria impresso em todos os arquivos gerados

Devido a uma limitação do próprio MySQL, no qual não é possível identificar a integridade referencial, o desenvolvedor definirá manualmente, informando para cada tabela que possui referência o campo referência, assim como a tabela e o campo referenciado.

Na Figura 16 é mostrada a tela de definição de integridade referencial.

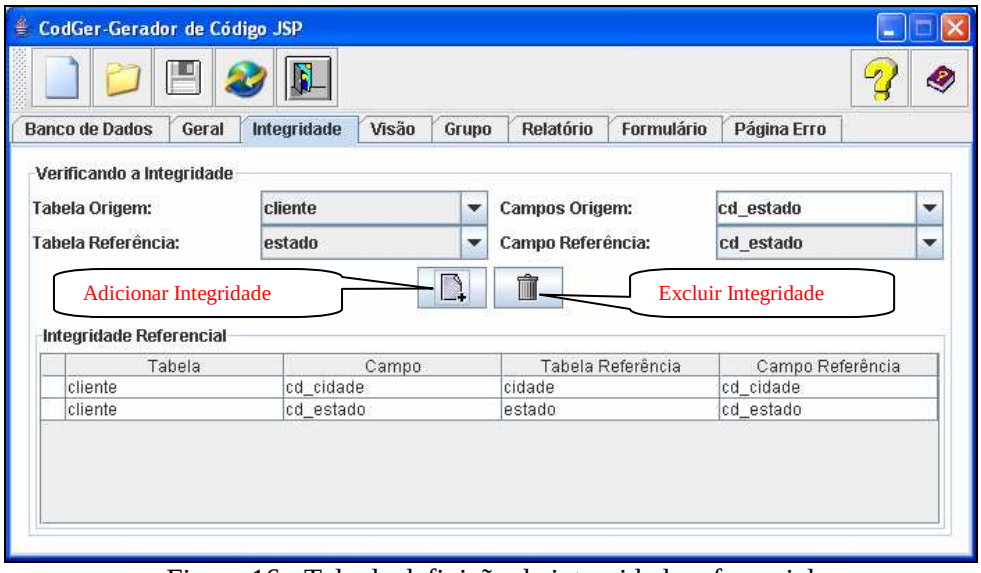


Figura 16 - Tela de definição da integridade referencial

Através da tela da Figura 16, poderá ser definida a sua integridade referencial, sendo que a ferramenta não permite que sejam referenciados campos com tipos diferentes, ou seja, quando for selecionada a tabela com o campo que possui referência somente serão relacionadas as tabelas que possuem o mesmo tipo.

A ferramenta permite a configuração das visões que é aonde serão definidas as tabelas a serem visualizadas. Na Figura 17 é apresentada a tela onde o desenvolvedor faz o gerenciamento das visões.

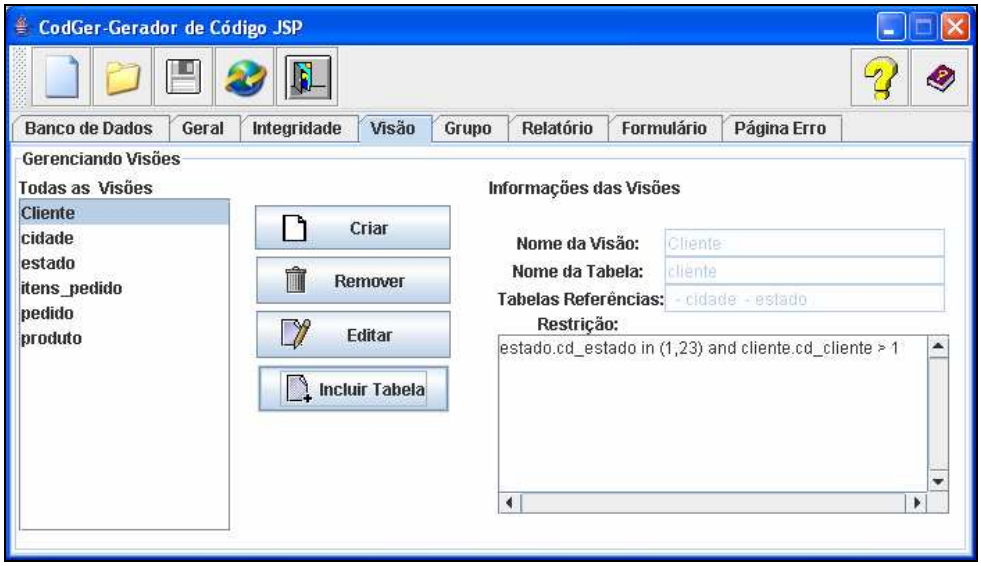


Figura 17 - Tela de gerenciamento das visões

O desenvolvedor pode gerenciar suas visões, criando, editando, removendo, incluindo tabelas com uma visão ou visualizando as configurações, como as tabelas referenciadas, tabela principal e suas restrições. Na Figura 18 é mostrada a tela de criação de uma visão.

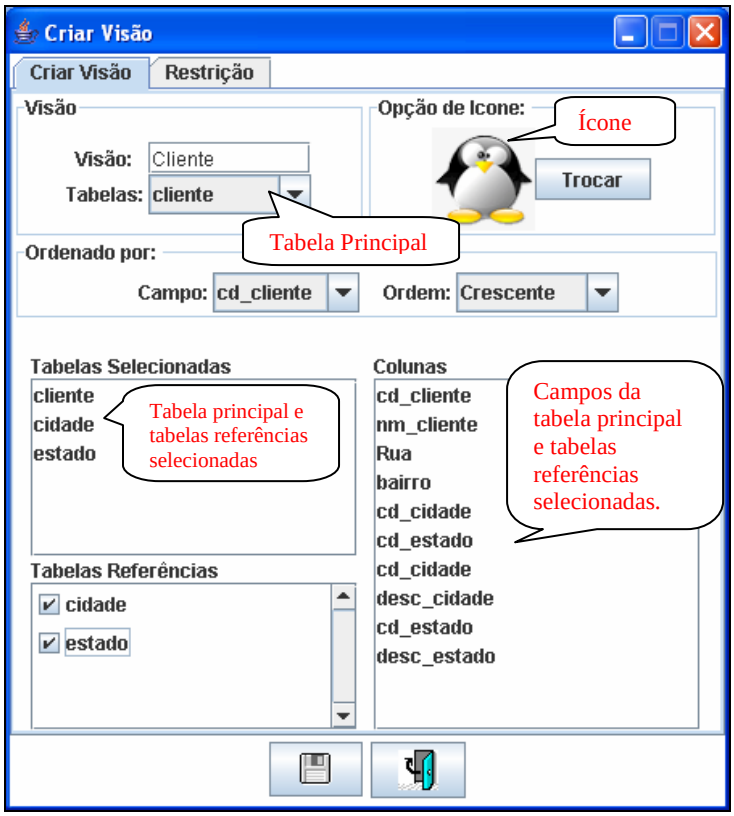


Figura 18 - Tela de criação da visão

Em uma visão, pode-se definir quais tabelas referenciadas serão usadas para fazer as configurações de relatórios e formulários, escolher um ícone para a visão que será o *link* de acesso às páginas no projeto gerado, o critério para a ordenação dos campos na apresentação dos relatórios, e as restrições conforme são mostradas na tela da Figura 19.

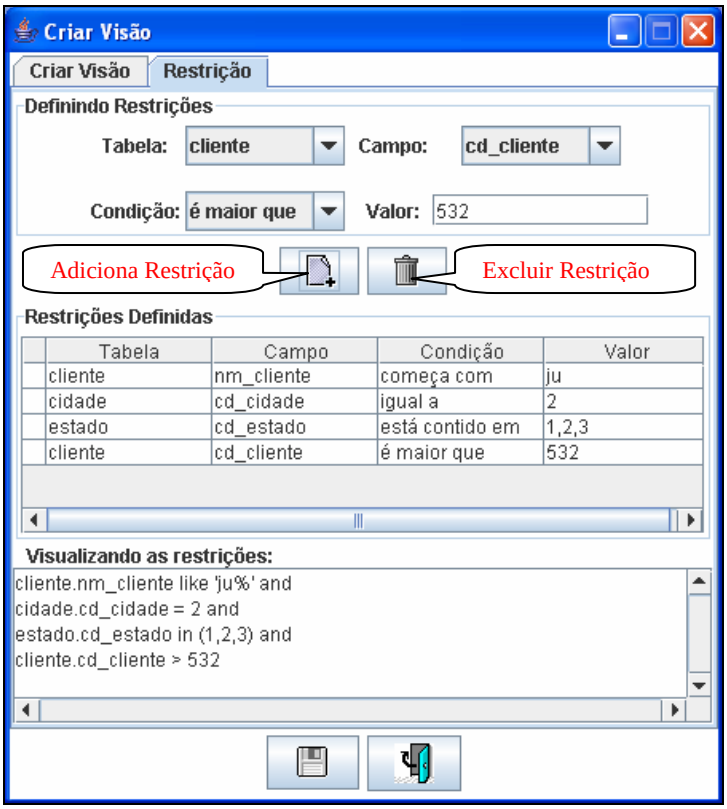


Figura 19 - Tela de definição das restrições

Pode ser definida uma ou mais restrições para cada visão, selecionando a tabela, o campo, a condição e digitando um valor, sendo que ele pode optar pelas seguintes condições: **igual a**, **é menor que**, **é maior que**, **começa com**, **termina com** e **está contido em**. Conforme são feitas as definições, a ferramenta mostra a relação das restrições criadas e monta a cláusula *where* em SQL para a visualização.

A ferramenta permite a criação de grupos, ou seja, agrupa várias visões conforme o segmento da aplicação para a apresentação dos *links* na página inicial do projeto, como está representado na Figura 20.

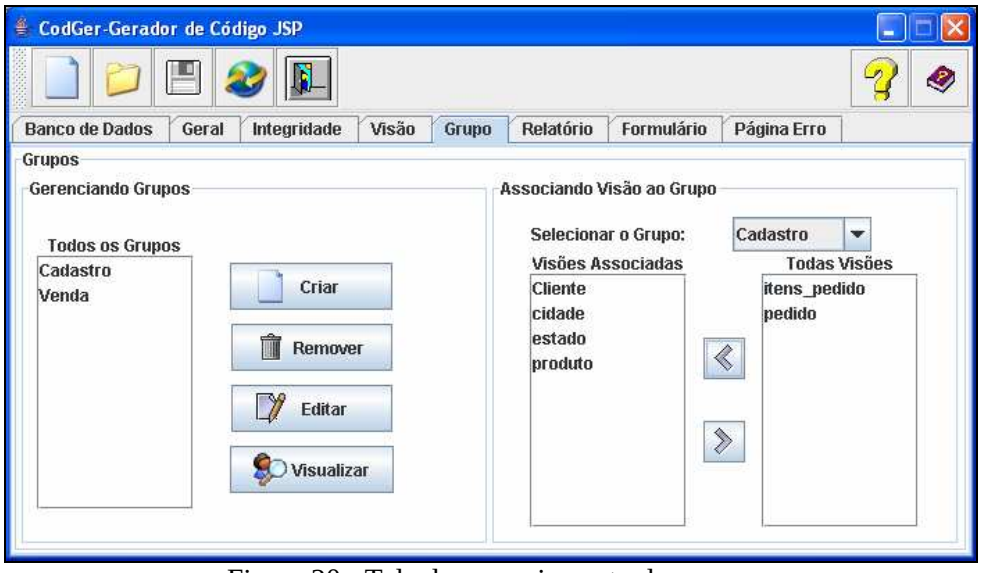


Figura 20 - Tela de gerenciamento de grupos

A ferramenta permite o gerenciamento dos grupos, criando, removendo, editando, excluindo, pré-visualizando e fazendo as associações com as visões, sendo que um grupo poderá ter várias visões associadas.

Na Figura 21 mostrada a tela para a criação de um grupo.

Figura 21 - Tela de criação de grupo

Ao criar um grupo, é definido o posicionamento que ocupará na página principal, ou seja, a linha e a coluna que serão apresentados os *links* das visões associadas ao grupo, assim como a forma que a página será apresenta, ou seja, em uma janela normal ou *pop-up*.

Pode-se gerenciar a forma que serão apresentadas as páginas dos relatórios, definindo

quais campos serão visíveis, quantos registros serão visíveis por página, um título para a identificação do relatório e campos, conforme a tela da Figura 22.

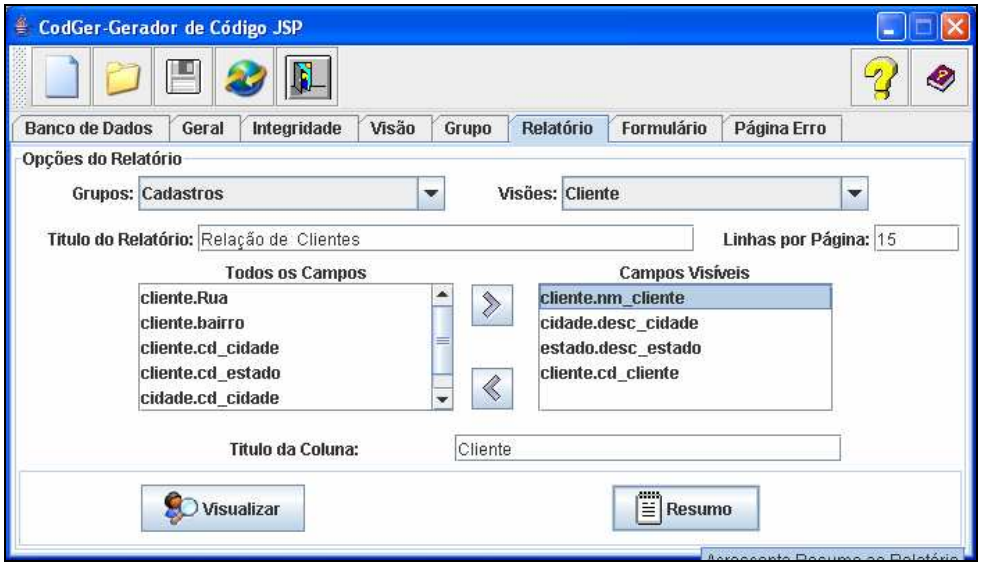


Figura 22 - Tela de configuração de relatórios

No gerenciamento de relatórios, tem-se a opção de fazer uma pré-visualização dos dados que serão apresentados na página de relatório, conforme é mostrado na Figura 23.

Código	Cliente	Cidade	Estado
8	Alexander Valdameri	Palmitos	Santa Catarina
57	Anderson Benetti	Guaraciaba	Santa Catarina
12	Ariberto Montibeller	Brusque	Santa Catarina
14	Carolina Alexandra	Guarujá do Sul	Santa Catarina
24	Cristiano Menin	São José do Cedro	Santa Catarina
6	Diogo Correia	Porto Alegre	Rio Grande do Sul
52	Fernando dos Santos	Blumenau	Santa Catarina
56	Gabriel Matias	Blumenau	Santa Catarina
55	Joice Martins	Blumenau	Santa Catarina
19	Joni Cereja	Blumenau	Santa Catarina
53	Joyce Martins	Florianópolis	Santa Catarina
54	Karly Vargas	Porto Alegre	Rio Grande do Sul
50	Leo Fath	Blumenau	Santa Catarina
4	Marilia Benetti	São José do Cedro	Santa Catarina
10	Roberta Davila	Blumenau	Santa Catarina
15	Robson Luiz Reguim	Bauru	São Paulo
45	Ruan Carlos	Chapeco	Santa Catarina
11	Vilmar Orsi	Blumenau	Santa Catarina
7	Willian Roberto	Cascavel	Paraná

Figura 23 - Visualização dos dados a serem apresentados na ferramenta

A ferramenta permite também ao desenvolvedor configurar um resumo, que é um totalizador de um campo na página de relatório, baseado nas principais funções de grupo, que

são as seguintes: **soma**, **contar**, **valor máximo**, **valor mínimo** e **média**, conforme é ilustrado na Figura 24.

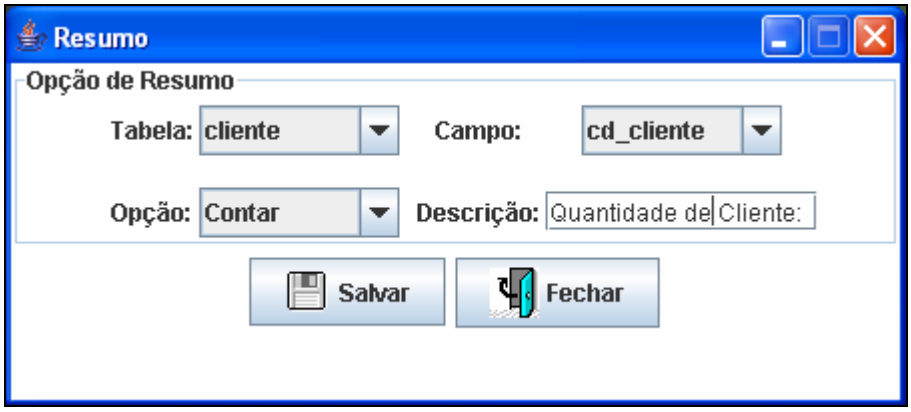


Figura 24 - Tela de configuração de resumo

No resumo podem ser configuradas a tabela, campo, a opção de resumo e uma breve descrição.

No gerenciamento de formulário pode-se definir quais campos e o tamanho que serão visíveis nos formulários de inclusão e alteração, conforme é mostrado na Figura 25.

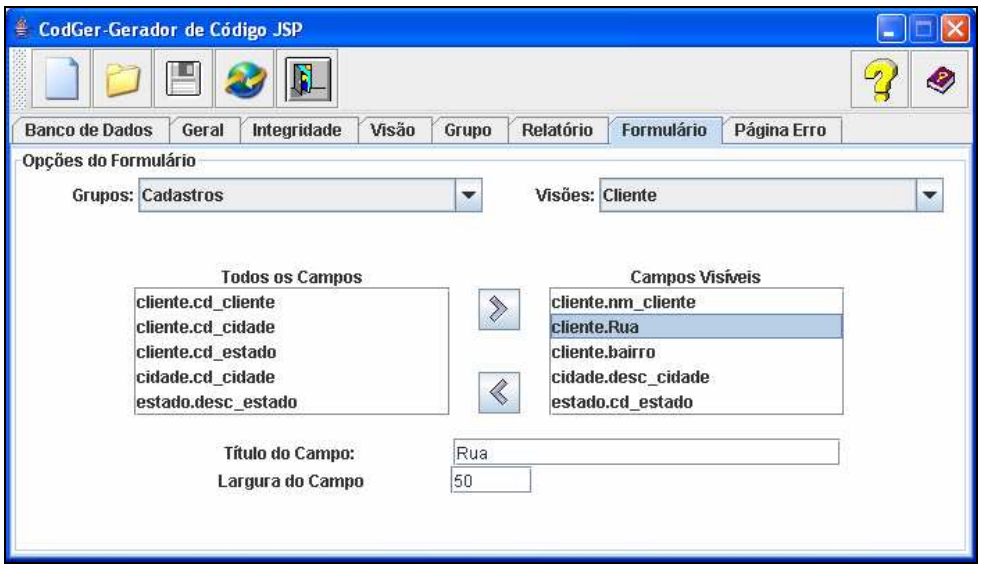


Figura 25 - Tela de gerenciamento de formulários

Para a geração de formulários foram seguidas as seguintes regras:

a) se o campo for incrementável então será apresentado desabilitado;

b) se o campo chave-primária não estiver visível então será apresentado como oculto;

c) se o tamanho do campo for menor que 80 então será apresentado em uma caixa de texto, do contrário será apresentado em um *textArea*;

d) se o campo possuir chave estrangeira então é apresentado o campo selecionado como visível da tabela referência em um *selected*, e no *value* do *selected* terá o campo chave estrangeira.

As mensagens nas páginas de erro que serão redirecionadas quando o usuário solicitar uma página, que não existe no servidor ou quando ocorrer algum erro na busca da próxima página, podem ser personalizadas através da ferramenta. Na Figura 26 está apresentada a tela de configuração das páginas de erro, com uma mensagem *default* para ambas situações, mas podem ser alteradas pelo desenvolvedor.

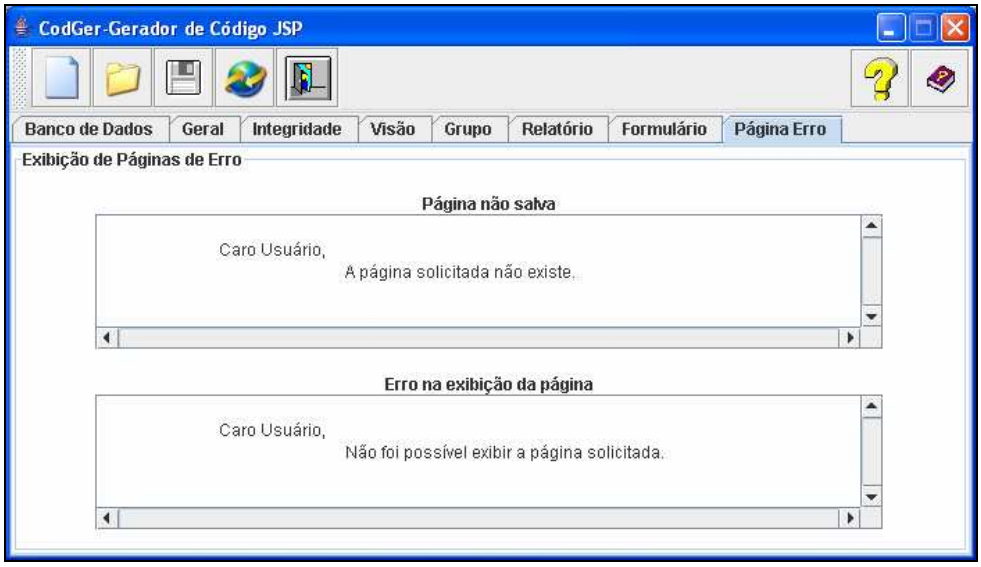


Figura 26 - Tela de configuração das páginas de erro

Após as configurações das páginas, podem ser visualizados os arquivos gerados para o seu projeto, que incluem o nome da página, o seu tipo e o caminho em que será gravada, como é mostrado na Figura 27.

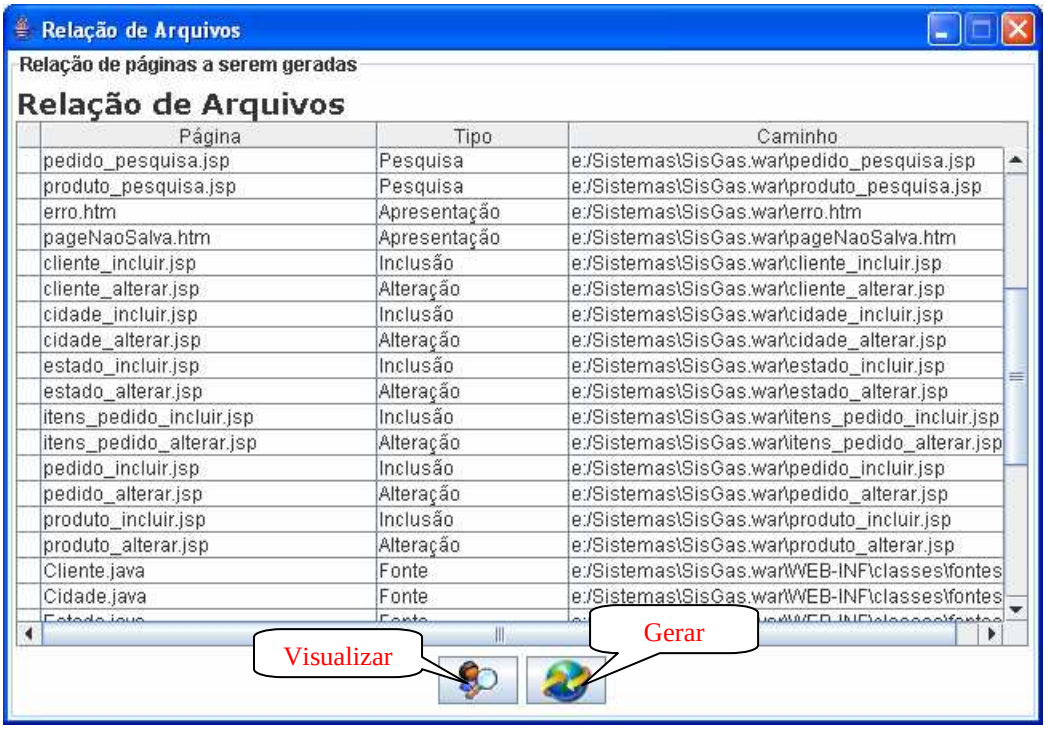


Figura 27 - Relação de arquivos gerados

Pode-se fazer uma pré-visualização do código antes de fazer a geração do arquivo. Ao visualizar o código de um arquivo, tem a opção de alterar o código gerado pela ferramenta como é mostrado na Figura 28.

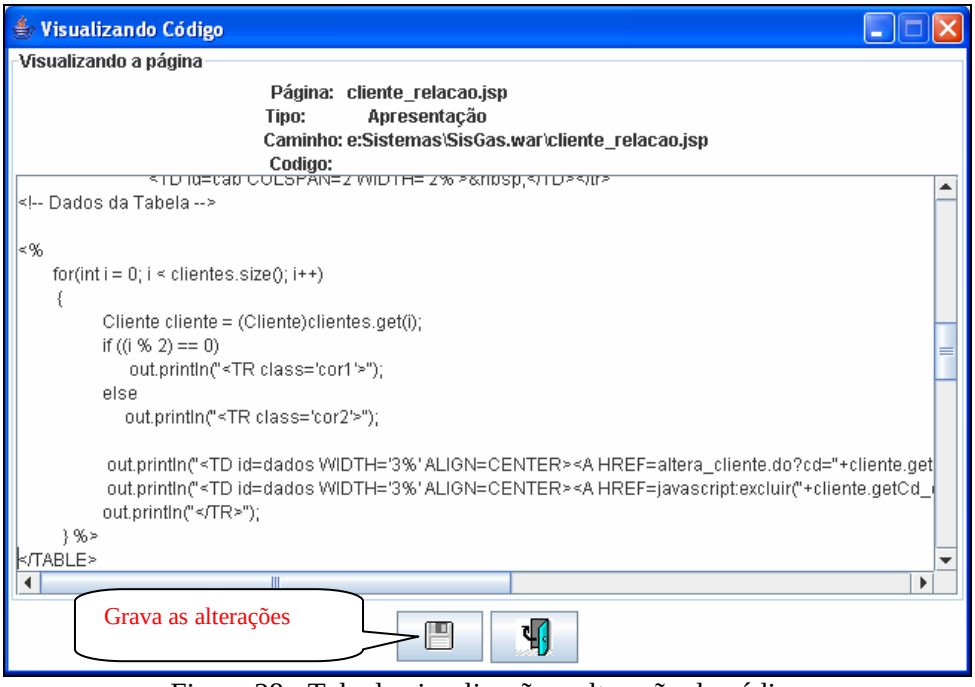


Figura 28 - Tela de visualização e alteração do código

Ao solicitar a geração do código, os arquivos serão gravados no local definido pelo desenvolvedor, seguindo a estrutura de diretórios mostrados na Figura 29, e os arquivos fontes

com a extensão .Java serão compilados gerando os *.class*, conforme é mostrado no Quadro 9.

```
Iterator k = ger.arquivos.iterator();
 while (k.hasNext()){
       PaginasGeradas pg = (PaginasGeradas)k.next();

       //grava o arquivo no local especificado
      File file = new File(pg.getCaminho());
      try {
      FileOutputStream to = new FileOutputStream(file);
      DataOutputStream dt = new DataOutputStream(to);
       //escreve o código no arquivo
      dt.writeBytes(ger.autoria()+pg.getConteudo());
      //busca a extensão do arquivo
           String extensao =
pg.getPagina().substring(pg.getPagina().indexOf("."),pg.getPagina().length(
));         if(extensao.equals(".java")){//gera os .class
                   compilarArquivos(pg.getCaminho());
       }
           }
      catch (IOException exc){System.err.println("Erro ao escrever no
arquivo "+pg.getPagina());}
}

private boolean compilarArquivos(String caminho){
   //Este método compila um arquivo java
   boolean compilou = false;
   try {
        //compilar o arquivo java
        Process p = Runtime.getRuntime().exec("javac "+caminho);
        //Aguardar a compilação
        try {
            int compila = p.waitFor();
            if (compila != 0)
                compilou =true;
            }
            catch (InterruptedException e1)
                {System.out.println("Erro na compilação");}
       }
       catch (IOException e)
              {System.out.println("Compilação "+e.getMessage());}

   return compilou;
}
```

Quadro 9 - Código que faz a compilação e gravação dos arquivos

O código gerado pela ferramenta organiza os arquivos em diretórios conforme o seu tipo. Na Figura 29 está representada a estrutura de diretórios gerada pela ferramenta e os tipos de arquivos contidos em cada diretório, sendo que “SisGas” é o nome do projeto.

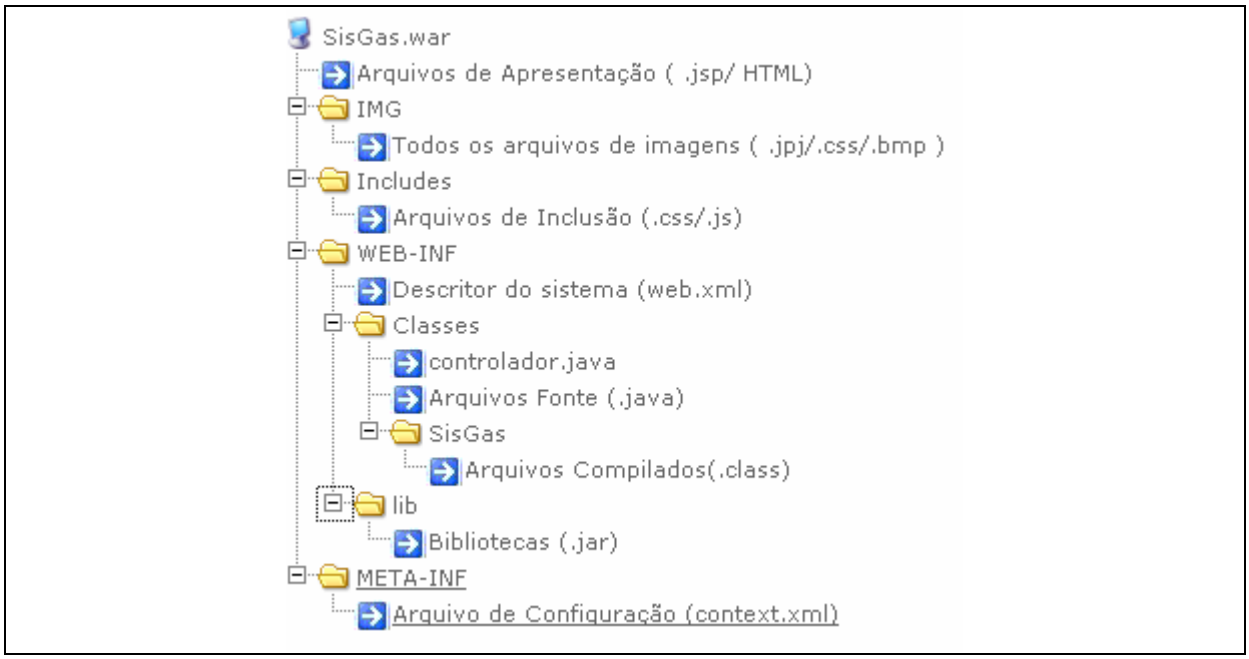


Figura 29 - Estrutura de diretórios gerado pela ferramenta

Segue abaixo a explicação de cada diretório e seus tipos de arquivos:

a) arquivos de apresentação: os arquivos de apresentação (.jsp e .html) são gravados no diretório principal. Esses arquivos possuem todo o conteúdo enviado para o navegador do usuário;

b) arquivos de imagens: todas as imagens são armazenadas no diretório IMG;

c) arquivos de inclusão: os arquivos com extensão .css, que definem o *layout* básico, e .js que definem algumas funcionalidades, são gerados no diretório *includes*;

d) arquivos fontes: o código fonte das classes é armazenado em um arquivo com a extensão .Java no diretório classes;

e) arquivos compilados: os arquivos compilados (.*class*) são gravados em um diretório com o mesmo nome do projeto dentro do diretórios classes;

f) arquivo descritor: a ferramenta gera o arquivo web.xml no diretório WEB-INF. Este arquivo descritor é que define as funcionalidades e características do projeto;

g) bibliotecas: as bibliotecas estão armazenadas no diretório lib;

h) arquivos de configuração: a ferramenta gera o context.xml para as configurações de contexto da aplicação no diretório META-INF.

3.5 OPERACIONALIDADE DA IMPLEMENTAÇÃO

Para demonstrar a funcionalidade da ferramenta, optou-se por gerar um sistema para uma distribuidora de gás, que deseja automatizar o controle de suas vendas. Para atender esta demanda, o sistema deverá permitir o cadastro dos clientes sendo que possui clientes em diferentes cidades e estados. É necessário que o sistema permita registrar os produtos e as vendas. O sistema também deverá trazer facilidades à secretária que precisa semanalmente enviar ao seu gerente o total de clientes da empresa.

Baseado nas necessidades do sistema, foi construído o modelo físico que está representado na Figura 30.

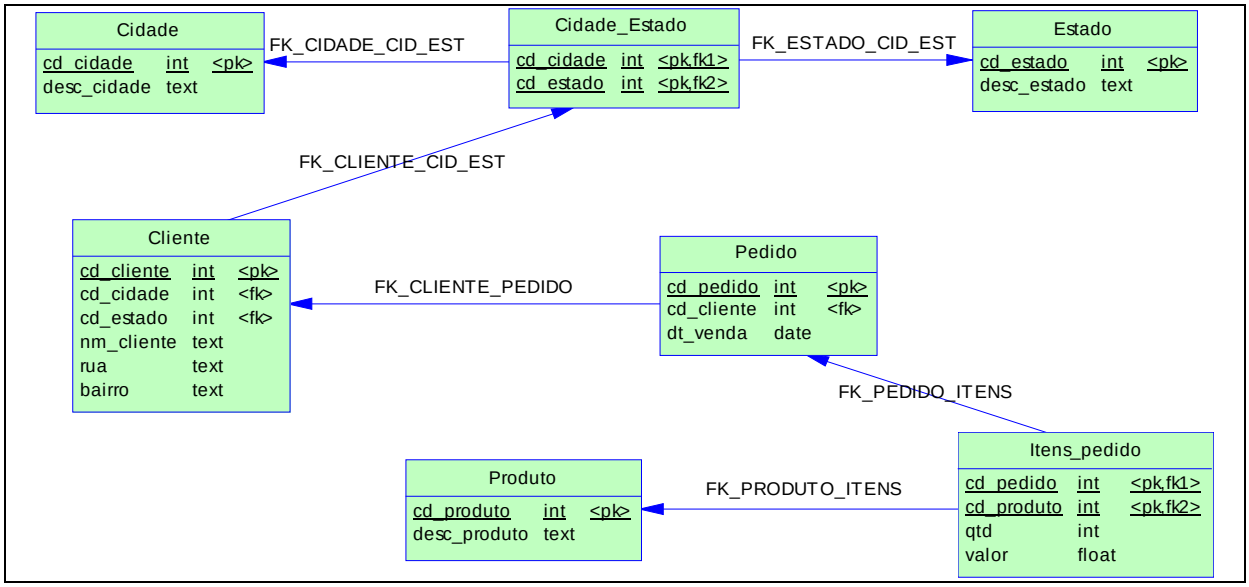


Figura 30 – Modelo físico do SisGas

Para um melhor entendimento das páginas geradas pela ferramenta, será mostrado na Figura 31, o processo de execução das páginas.

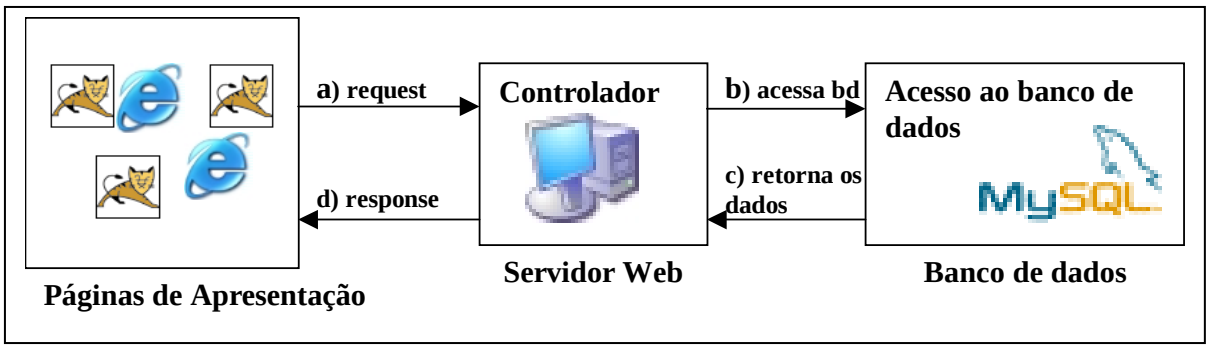


Figura 31 - Processo de execução de uma página

A seguir é explicado o processo de execução das páginas ilustrado na Figura 31:

a) as páginas de apresentação enviam uma ação ao controlador através da URL;

b) o controlador decodifica esta ação e encaminha para o método correspondente que irá atender esta ação, sendo que este método irá fazer o acesso ao banco se for necessário;

c) caso seja feito o acesso ao banco de dados, as informações obtidas do banco de dados serão enviadas ao controlador;

d) o controlador encaminha essas informações para a página de apresentação.

Conforme a estrutura das páginas geradas, quando o navegador web faz uma solicitação através das páginas de apresentação será acessado o *servlet* controlador, que fará o controle das requisições, retornando a resposta às páginas de apresentação. No Quadro 10 é apresentado um trecho do código do *servlet* controlador.

```
public class Controlador extends HttpServlet {

 protected void processRequest(HttpServletRequest request,
HttpServletResponse response)
       throws ServletException, java.io.IOException {

       //decodifica a ação que é para executar de acordo com URL
       String acao = decodificaAcao(request);

      // indica a página que sera mostrada após a execução da ação
       String proximaPagina = "";

       //controle para verificar se a página foi encontrada
       boolean achouPagina = false;

       if (acao.equals("abrir_Cliente")){
            proximaPagina = abrirCliente(request,response);
            achouPagina = true;
       }
      if (acao.equals("excluir_Cliente")){
            proximaPagina = excluirCliente(request,response);
            achouPagina = true;
       }
       if(!achouPagina)
            proximaPagina = "/pageNaoSalva.htm";
        ...
getServletContext().getRequestDispatcher(proximaPagina).forward(request,
response);
 }
       //Decodifica a ação selecionada
       protected String decodificaAcao(HttpServletRequest request) {
             String caminho = request.getRequestURI();
             String acao = caminho.substring(caminho.lastIndexOf('/')+1,
caminho.lastIndexOf(".do"));
       return acao; }

       //Necessário para extender o HttpServlet
       protected void doGet(HttpServletRequest request, HttpServletResponse
response)throws ServletException, java.io.IOException {
             processRequest(request, response); }

      protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, java.io.IOException {
processRequest(request, response);
}
//implementação dos métodos
      ...
```

Quadro 10 - Código que faz o controle das requisições

A ferramenta gera o arquivo web.xml no diretório WEB-INF, que declara o *servlet* controlador, mapeia as ações para o controlador e define a página inicial do sistema, conforme o Quadro 11.

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE web-app  PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application
2.3//EN" "http://java.sun.com/dtd/web-app_2_3.dtd">
 <web-app>

 <!-- declara o servlet controlador -->
<servlet>
      <servlet-name>Controlador</servlet-name>
       <display-name>Controlador</display-name>
       <servlet-class>Controlador</servlet-class>
       <load-on-startup>1</load-on-startup>
</servlet>

 <!-- mapeia urls de acoes para o servlet controlador -->
 <servlet-mapping>
       <servlet-name>Controlador</servlet-name>
       <url-pattern>*.do</url-pattern>
 </servlet-mapping>

 <session-config>
      <session-timeout>5</session-timeout>
 </session-config>

<!—define a página inicial -->
<welcome-file-list>
<welcome-file>
       index.jsp
 </welcome-file>
</welcome-file-list>
</web-app>
```

Quadro 11 - Código do arquivo web.xml

A página inicial do sistema definida pelo arquivo de configuração é a index.jsp, que possui um *layout* pré definido pela ferramenta e os *links* que poderão ser acessados através de ícones, que foram selecionados pelo desenvolvedor na criação da visão e estão agrupados conforme as visões associadas ao grupo. Na Figura 32 está ilustrada a página inicial do sistema.

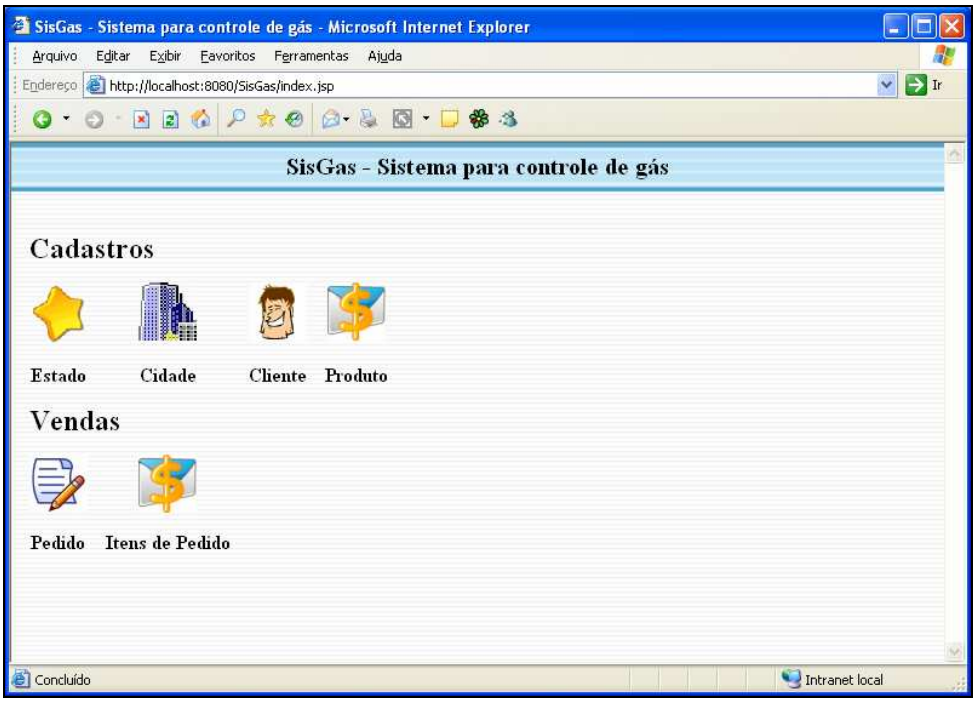


Figura 32- Página inicial do sistema

Ao clicar sobre uma das opções dos *links*, a ação será enviada ao controlador, que receberá a ação e redirecionará para o método que irá buscar no banco de dados todos os registros correspondentes àquela ação, guardando em uma estrutura de lista.

No Quadro 12 é apresentado o código do controlador que busca os registros no banco de dados. Caso não ocorram problemas, o controlador redirecionará para página de relatórios enviando a lista com os registros buscados no banco de dados. Caso contrário, redirecionará para página de erro que foi definida pelo desenvolvedor através da ferramenta.

```
protected String abrirCliente(HttpServletRequest request,ttpServletResponse
response)   throws ServletException, java.io.IOException{
 String paginaErro= "/erro.htm";
 String paginaSucesso= "/cliente_relacao.jsp";
 String proximaPagina = paginaErro;
 try {
      ...

      ArrayList clientes = Conn.getListaCliente(param, limitePorPagina);
      request.getSession().setAttribute("listaCliente",clientes);

      proximaPagina = paginaSucesso;
 }catch(Exception e){ System.out.print(e.toString());}
return proximaPagina; }
```

Quadro 12 – Codigo que controla as solicitações para a página de relatórios

No Quadro 13 está apresentado o método que busca os registros no banco de dados e armazena em uma lista.

```
public static ArrayList getListaCliente(int inicio, int qtd){
try{
      String sql = "Select CLIENTE.NM_CLIENTE as CLIENTE, CLIENTE.RUA,
CLIENTE.BAIRRO, CIDADE.DESC_CIDADE as CIDADE, ESTADO.DESC_ESTADO as ESTADO
from CLIENTE, CIDADE, ESTADO where  CLIENTE.CD_CIDADE = CIDADE.CD_CIDADE
and CLIENTE.CD_ESTADO = ESTADO.CD_ESTADO  order by CLIENTE.CD_CLIENTE asc
LIMIT "+inicio+","+qtd;
      connecta();
      rs = st.executeQuery(sql);
      ArrayList cliente = new ArrayList();
      while (rs.next()){
            Cliente umCliente =  criaCliente(rs);
            cliente.add(umCliente);
      }
 return cliente;
 }
 catch ( Exception e ){ System.out.println(e.getMessage()); return null; }
 }
```

Quadro 13 – Código que consulta o banco de dados

No Quadro 14 é apresentado o código que percorre a lista e apresenta os registros na página de relatórios.

```
...

<% ArrayList clientes = (ArrayList)session.getAttribute("listaCliente");
 request.getSession().setAttribute("cliente", null);
...

<!-- Dados da Tabela -->
<%
 for(int i = 0; i < clientes.size(); i++){
      Cliente cliente = (Cliente)clientes.get(i);
      if ((i % 2) == 0)
            out.println("<TR class='cor1'>");
      else
            out.println("<TR class='cor2'>");

      out.println("<TD id=dados>"+cliente.getNm_cliente()+"</TD>");
      out.println("<TD id=dados>"+cliente.getDesc_cidade()+"</TD>");
      out.println("<TD id=dados>"+cliente.getDesc_estado()+"</TD>");

      out.println("<TD id=dados WIDTH='3%' ALIGN=CENTER><A
HREF=altera_cliente.do?cd="+cliente.getCd_cliente()+"><IMG
SRC='img/editar.png' BORDER=0 ALT='Editar Cliente'></IMG></A></TD>");

      out.println("<TD id=dados WIDTH='3%' ALIGN=CENTER><A
HREF=javascript:excluir("+cliente.getCd_cliente()+")><IMG
SRC='img/excluir.png' BORDER=0 ALT='Excluir Cliente'></IMG></A></TD>");

      out.println("</TR>");
} %>
...
```

Quadro 14 - Código da página de Relatório

O resultado da execução dos códigos acima está apresentado na Figura 33.

Figura 33 – Página de relatórios

As páginas de relatórios geradas pela ferramenta possuem as opções de incluir, editar, excluir, pesquisar registro, fazer a paginação e ordenar os registros.

No Quadro 15 é apresentado o código que faz a inserção de um registro no banco de dados.

```
public static boolean insereCliente(Cliente c){
try{
      connecta();
      String sql="insert into CLIENTE(CD_CIDADE, CD_ESTADO, NM_CLIENTE,
RUA, BAIRRO) values("+c.getCD_CIDADE()+" , "+c.getCD_ESTADO()+" ,
'"+c.getNM_CLIENTE()+"', '"+c.getRUA()+"', '"+c.getBAIRRO()+"')";
      st.executeUpdate(sql);

      return true;
}
catch ( Exception ed ){ System.out.println(ed.getMessage());
      return false; }
}
```

Quadro 15 Código que faz a inclusão de um registro no banco de dados

No Quadro 16 é apresentado o código que faz a alteração de um registro no banco de dados.

```
public static boolean alteraCliente(Cliente c){
try{
      connecta();
      String sql="update CLIENTE set CD_CIDADE ="+c.getCD_CIDADE()+",
CD_ESTADO ="+c.getCD_ESTADO()+",NM_CLIENTE='"+c.getNM_CLIENTE()+"', RUA=
'"+c.getRUA()+"', BAIRRO = '"+c.getBAIRRO()+"' where  CD_CLIENTE =
"+c.getCD_CIDADE();

st.executeUpdate(sql);

      return true;
}
catch ( Exception ed ){ System.out.println(ed.getMessage());
      return false; }
}
```

Quadro 16 - Código que faz a alteração de um registro no banco de dados

Na Figura 34 é apresentado o formulário para a inclusão e alteração de um registro no banco de dados.

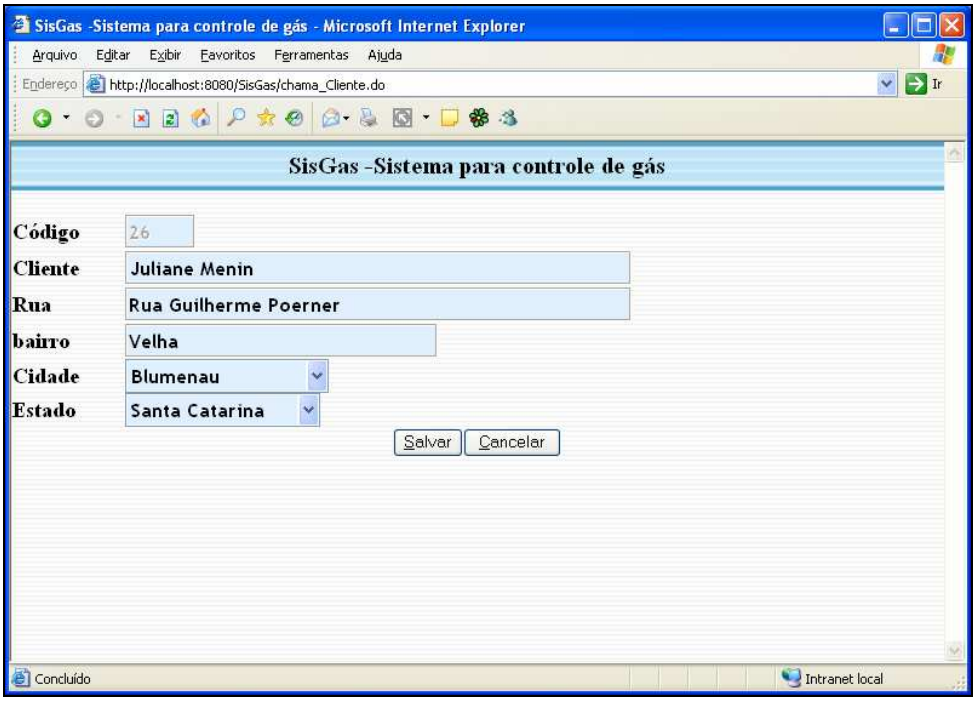


Figura 34- Formulário de inclusão de um novo registro

O código gerado pela ferramenta permite excluir registros no banco de dados, clicando no ícone da lixeira apresentado na Figura 33. Ao solicitar a exclusão aparecerá uma mensagem de confirmação para o usuário conforme é mostrado na Figura 35.

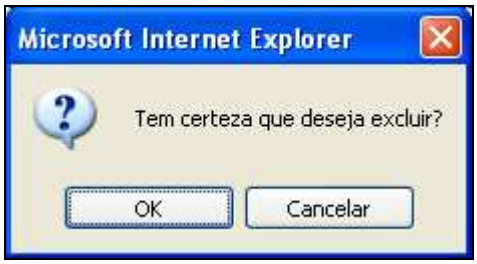


Figura 35 - Mensagem de confirmação de exclusão do registro

Quando o usuário confirmar a exclusão, a solicitação será enviada ao controlador que fará a exclusão conforme o Quadro 17 e retornará para página de relatórios.

```
public static boolean excluiCliente(int codigo){
      try{
            String sql = "delete from CLIENTE where CD_CLIENTE="+codigo;
            connecta();
            st.executeUpdate(sql);
      return true;
      }
       catch(SQLException e){System.out.print(e.toString()); return false;}
}
```

Quadro 17- Código que faz a exclusão de um registro no banco de dados

Ao clicar no ícone lupa apresentado na Figura 33, o usuário tem a opção de estabelecer critérios de filtro para recuperar registros no banco de dados conforme é ilustrada na Figura 36.

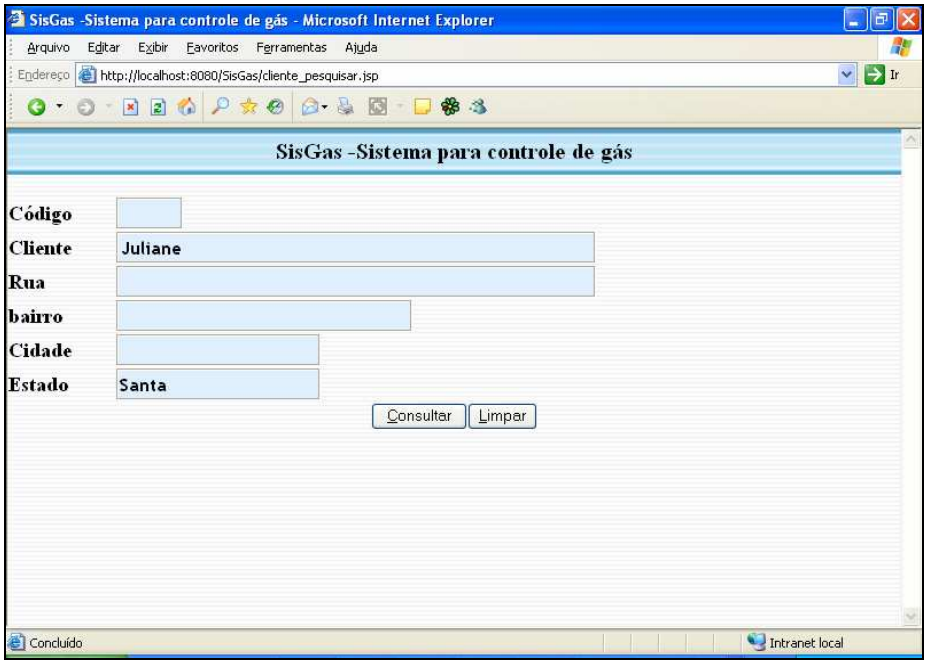


Figura 36 - Página de Consulta

Caso ocorra algum erro no recebimento ou envio de uma solicitação no controlador, ou execução de uma consulta no banco de dados, será apresentada uma página de erro com a

mensagem definida pelo desenvolvedor através da ferramenta, que é ilustrada na Figura 37.

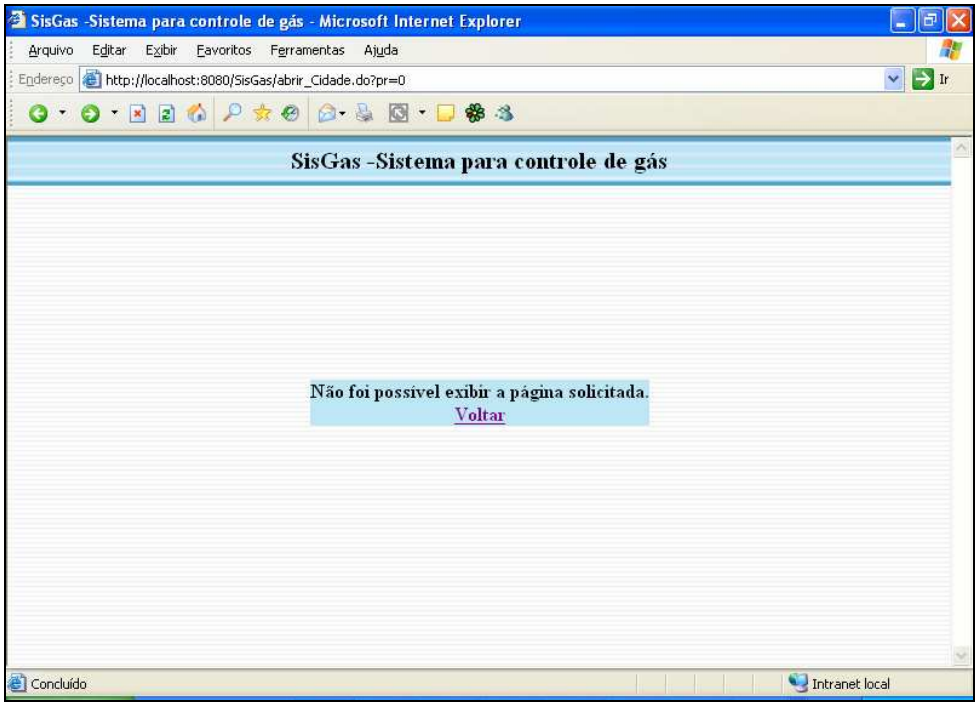


Figura 37 - Página de erro

Caso o usuário informe alguma página que não exista no servidor, será apresentada uma página informativa conforme é mostrada na Figura 38.

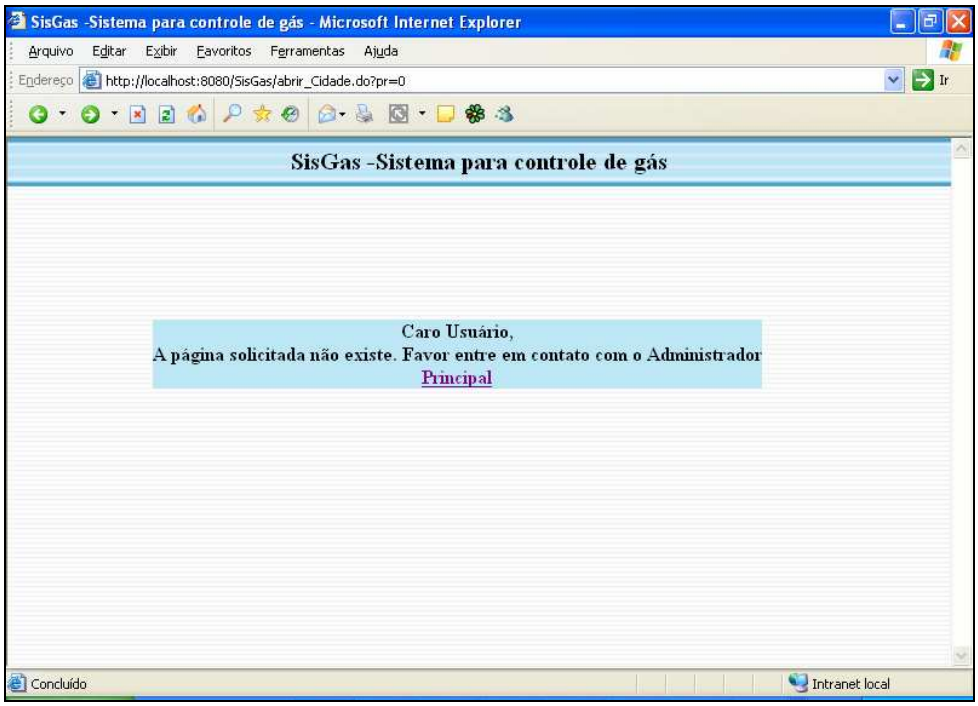


Figura 38 - Página inexistente

4 CONCLUSÕES

O presente trabalho mostrou uma ferramenta construída para auxiliar os desenvolvedores na construção de aplicativos web utilizando a tecnologia JSP com um banco de dados *open source* MySQL. A ferramenta permite a geração de código para as rotinas básicas de um sistema automaticamente como: inclusão, exclusão, alteração, consultas e relatórios. Nos relatórios é permitida a customização, como exemplo, definir atributos de ordenação e paginação de registros.

Um grande diferencial no trabalho apresentado é a geração de código para tecnologia JSP que é baseada em Java, sendo assim é independente de plataforma. Com o uso da ferramenta, os desenvolvedores podem reduzir o tempo necessário para a programação assim como o tempo gasto na construção de aplicativos podendo enfatizar a parte mais complexa da aplicação e usar a ferramenta para fazer a parte mais simples que são as rotinas de cadastros, mas que consome um tempo considerável.

A utilização da IDE Eclipse mostrou-se adequada à construção e depuração da ferramenta. A forma pela qual a ferramenta foi especificada e codificada oferece uma facilidade na adequação para a geração de código a partir do metadados de outros SGBDs.

Em relação a outras ferramentas similares, acredita-se que a mesma apresentada um diferencial importante que é a possibilidade de execução multiplataforma.

4.1 LIMITAÇÕES

A ferramenta implementada não reconhece automaticamente os relacionamentos entre as tabelas, devido a uma limitação do próprio MySQL, sendo que a mesma permite ao desenvolver configurar os relacionamentos manualmente.

4.2 EXTENSÕES

Como sugestão para trabalhos futuros, tem-se as seguintes:

a) permitir a utilização de outros bancos de dados como PostGreeSQL, SQLServer ou Oracle;

b) permitir a definição do próprio *layout* do aplicativo, através de folhas de estilos;

c) permitir a geração de relatórios mais complexos baseados nas funções de grupos.

REFERÊNCIAS BIBLIOGRÁFICAS

ABILEBOUL, Serge; PETER, Buneman; SUCICE, Dan. **Gerenciamento de dados na web**. Rio de Janeiro: Campus, 2000.

BEZERRA, Eduardo. **Princípios de análise e projeto de sistemas com UML**. Rio de Janeiro: Campus, 2002.

CAELUM. **Ensino e soluções em Java**. São Paulo, [2004?]. Disponível em: < http://www.caelum.com.br/download/fj21.pdf > . Acesso em: 24 mar. 2005.

COELHO, Ademir. **JavaServer pages**: guia de consulta rápida. São Paulo: Novatec, 2000.

COLLA, Ernesto Coutinho. **Servlets, Java do lado do servidor**. São Paulo, 1998. Disponível em: < http://www.insite.com.br/docs/develop_servlet95.htm>. Acesso em: 20 mar. 2005.

DIAS, Paulo R. **Sistema de informação baseado em regras de negócio utilizando a ferramenta Genexus estudo de caso no setor têxtil**. 2002. 103 f. Dissertação(Mestrado em Engenharia de Produção) - Curso de Pós-Graduação em Engenharia de Produção, Universidade Federal de Santa Catarina, Florianópolis.

DUARTE, Eber.**Trabalhando com vários tipos de tabelas MySQL**. [S.l.], 2004. Disponível em: < http://www.sqlmagazine.com.br/Colunistas/eber/10_Tabelas_MyISAM.asp >. Acesso em: 18 mar. 2005.

ECLIPSE.ORG. **Eclipse**. Version 3.0. [S.l.], 2004. Disponível em: <http://www.eclipse.org>. Acesso em: 13 mar. 2004.

FABULOUS FORCE DATABASE TOOLS. **DB Designer** 4. [S.l.], 2003. Disponível em: <http://www.fabforce.net>. Acesso em: 15 set. 2004.

FIELDS, Duane K.; KOLB, Mark A. **Desenvolvimento na web com JavaServer Pages**. Rio de Janeiro: Ciência Moderna, 2000.

FURGERI, Sérgio. **Java 2 ensino didático**: desenvolvendo e implementando aplicações. São Paulo: Érica, 2003.

HERRINGTON, Jack. **Code generation in action.** Greenwich: Manning, 2003.

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

LIMA, Adilson S. **MySQL Server**: open source 4.X: soluções para desenvolvedores e administradores de banco de dados. São Paulo: Érica, 2003.

RIZO, Eduardo H. **ASPSYS**: uma ferramenta de apoio ao desenvolvimento de sistemas para web com uso de banco de dados. 2004. 99 f. Trabalho de Conclusão de Curso (Especialização em Desenvolvimento para Web) – Departamento de Informática, Universidade Estadual de Maringá, Maringá.

SANTOS, Edgar H. CodeCharge: gerador de códigos para aplicações web. **Comunicado técnico**, Campinas, SP, n. 45, dez. 2002. Disponível em: <http://www.cnptia.embrapa.br/modules/tinycontent3/content/2002/comuntec45.pdf>. Acesso em: 02 set. 2004.

SZOLKOWSKI Mark; TODD, Nick. **JavaServer pages:** o guia do desenvolvedor. Tradução Edson Furmankiewicz. Rio de Janeiro: Elsevier, 2003.