UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

GERADOR DE CÓDIGO HTML BASEADO EM DICIONÁRIO DE DADOS UTILIZANDO BANCO DE DADOS

LUIS FERNANDO COELHO

BLUMENAU
2006

2006/1-28

LUIS FERNANDO COELHO

GERADOR DE CÓDIGO HTML BASEADO EM DICIONÁRIO DE DADOS UTILIZANDO BANCO DE DADOS

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. Jomi Fred Hübner, Doutor - Orientador

BLUMENAU
2006

2006/1-28

GERADOR DE CÓDIGO HTML BASEADO EM DICIONÁRIO DE DADOS UTILIZANDO BANCO DE DADOS

Por

LUIS FERNANDO COELHO

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

Presidente:	Prof. Jomi Fred Hübner, Doutor – Orientador, FURB
Membro:	Prof. Marcel Hugo, Mestre – FURB
Membro:	Prof. Mauro Marcelo Mattos, Doutor – FURB

Blumenau, dia de mês de ano

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

AGRADECIMENTOS

A Deus, que deu força e saúde para que eu pudesse enfrentar e superar todos os desafios nesta jornada de formação profissional.

À minha família, que mesmo estando distante foi onde sempre encontrei uma fonte de amor, carinho e incentivo para me dar uma chance de um futuro melhor.

A empresa Wheb Sistemas que me deu condições e apoio no desenvolvimento deste trabalho.

Ao meu orientador, Jomi, por ter acreditado e me auxiliado no desenvolvimento e conclusão deste trabalho.

Os bons livros fazem “sacar” para fora o que a pessoa tem de melhor dentro dela.

Lina Sotis Francesco Moratti

RESUMO

Este trabalho apresenta uma ferramenta, que pode ser utilizada junto com *frameworks*, para a automatização e geração de código para aplicações web. A ferramenta pode gerar desde scripts SQL para a manipulação dos dados no banco até o HTML que irá compor a página. A geração das páginas é realizada de forma dinâmica, em tempo de execução, e são utilizadas as informações cadastradas em um dicionário de dados para a geração. As páginas geradas possibilitam a inclusão, alteração e exclusão dos registros do banco de dados.

Palavras-chave: HTML. Geração de código. Dicionário de dados.

ABSTRACT

This work presents a tool, which can be used with frameworks, for the automatization and generation of code for web applications. The tool can generate SQL scripts for the data manipulation in database and the html to compose the page. The page is generated in execution time, and use the information stored in the data dictionary. The generated pages make possible inclusion, alteration and exclusion of the registers of the data base.

Key-words: HTML. Code generation. Dictionary of data.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1 INTRODUÇÃO

Para quem conhece a Internet há menos de cinco anos, sua história poderá parecer surpreendente. Segundo RNP (2005), a necessidade dos computadores poderem comunicar-se remotamente surgiu por volta dos anos 60 durante a guerra fria. Era essencial proteger os dados militares de tal forma que, mesmo com um ataque inimigo, os dados fossem preservados. A solução proposta foi uma rede eletrônica de dados, na qual os dados deveriam estar distribuídos entre diversos computadores e poderiam ser atualizados no menor espaço de tempo possível. Para atender essa necessidade foi criada a rede ARPA-NET pelo grupo de pesquisa *Advanced Research Projects Agency* (ARPA), interligando três computadores no final de 1969. Três anos mais tarde já eram 40. Mas logo se percebeu que possuir computadores interligados apenas para fins militares não seria interessante. Foi quando o meio científico se interessou pela ARPA-NET como um meio de obter resultados de pesquisas realizadas em outras instituições. Com isso, o número de computadores interligados aumentou muito ocasionando a separação entre a parte militar e a parte civil. Nos anos 80 a *National Science Foundation* (NSF) interligou os mais importantes centros científicos a redes menores de universidades, fazendo com que diversas redes fossem unidas. Assim, o que era chamado de ARPA-NET foi batizado de Internet.

Hoje em todas as áreas podem-se encontrar documentos web, que vão desde uma simples página pessoal até grandes portais de vendas de mercadorias e transações eletrônicas. Uma das melhores formas de perceber este crescimento é a possibilidade de realizar compras em um supermercado sem precisar ir à loja, apenas utilizando portais na web, o que seria difícil de imaginar a alguns anos.

Nesse contexto, é necessário ter agilidade na geração de documentos eletrônicos. Ao invés de serem manualmente construídos, vem-se utilizando geradores de código para

automatizar e agilizar esse processo, diminuindo o tempo e ganhando em qualidade. Isso fez com que os desenvolvedores recorressem a ferramentas que auxiliassem na criação e na manutenção de aplicativos web.

Existem algumas tecnologias que podem ser usadas para construção de aplicações web, entre elas *Personal Home Page* (PHP), *Active Server Page* (ASP), Microsoft.NET e Java. Nesse cenário, a linguagem Java apresenta um potencial muito grande, pois possui diversas formas de utilização. Pode ser através de *servlet*, *Java Server Pages* (JSP) ou ainda através de *templates* desenvolvidos na própria linguagem com o *Velocity*, sendo todos executados e processados em um servidor e apenas a página produzida é visualizado no cliente.

Tendo em vista a grande procura por aplicações web e o tempo perdido com tarefas rotineiras como a criação de cadastros básicos e a dificuldade de integração entre programadores e *web-designers*, este trabalho descreve a construção de um gerador de código que deverá ser empacotado juntamente com a aplicação Java, para permitir a geração de código *HiperText Markup Language* (HTML) em tempo de execução. O gerador também gera comandos *Structured Query Language* (SQL) para consulta, alteração, inserção e exclusão de novos registros, baseando-se em informações armazenadas em um dicionário de dados encontrado no banco de dados. Essas informações que devem ser previamente cadastradas irão determinar como as páginas do sistema serão apresentadas. Nesta base, para cada atributo das tabelas poderá ser informado: as páginas onde o mesmo deverá aparecer, a descrição e a formatação, entre outras configurações. Como todas as páginas serão geradas dinamicamente, para realizar alterações em uma delas, será necessário apenas uma atualização no dicionário de dados, que na próxima requisição da página, a mesma já irá conter as novas alterações.

Como todas as tabelas do sistema estarão cadastradas no dicionário de dados e este pode estar em qualquer estrutura de banco de dados, basta referenciarem a tabela no gerador

que será gerado dinamicamente a página correspondente.

O gerador pode ser utilizado para gerar qualquer página do sistema não se limitando a páginas de cadastros como na maioria das ferramentas atuais. Como em uma página será possível utilizar diversas vezes o gerador, pode-se montar uma página com diversas tabelas como, por exemplo, de uma nota fiscal, onde necessariamente têm-se itens, tributos, totais, entre outras informações, as quais estão em tabelas diferentes. Caso seja necessário criar uma página que não possua uma tabela, pode-se utilizar apenas HTML, como é feito na maioria dos sistemas.

Para demonstrar o funcionamento do gerador foi desenvolvido um Prontuário Eletrônico do Paciente (PEP), no qual o médico tem acesso a todas as informações sobre um paciente internado em um hospital. Nele o médico pode acompanhar, por exemplo, os sinais vitais, ganhos e perdas, anamnese, evolução, entre outras informações. Foi desenvolvida no padrão *Model View Controller* (MVC).

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é criar um gerador de código HTML e *scripts* SQL para auxiliar analistas e programadores no desenvolvimento de aplicativos web utilizando a linguagem Java, a partir das definições cadastradas em um dicionário de dados.

Os objetivos específicos do trabalho são:

a) permitir a configuração das páginas a serem geradas, a partir dos atributos de cada tabela;

b) gerar automaticamente o código em tempo de execução para cadastros com as opções de alterações, exclusões, inclusões e consultas;

c) usar o *framework WebWork* para integrar as páginas geradas e as regras de negócio desenvolvidas em Java;

a) utilizar o *Velocity* para montar *templates* que irão receber o código gerado.

1.2 ESTRUTURA DO TRABALHO

O segundo capítulo apresenta conceitos sobre MVC, *Velocity*, *WebWork*, Dicionário de Dados, Banco de Dados Oracle, conceitos sobre geração de código e algumas ferramentas para o leitor possa acompanhar melhor como foi desenvolvido/funciona o gerador.

O terceiro capítulo apresenta a especificação e o desenvolvimento do gerador, mostrando os diagramas de casos de uso, atividades e a modelagem do banco de dados juntamente com explicações de como foi desenvolvido o gerador.

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

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentadas as tecnologias utilizadas no trabalho, sendo elas o padrão MVC, *Velocity, framework WebWork,* dicionário de dados, geração de código e ferramentas de geração de código.

2.1 MODEL VIEW CONTROLLER

Segundo Lozano (2004, pág. 34-39), a sigla MVC denota os três papéis no qual todo componente da aplicação deve ser classificado: *Model*, *View* e *Controller*, ou em português: Modelo, Visualização e Controlador.

Componentes de modelo são responsáveis pelo armazenamento dos dados durante a sessão do usuário logado para a execução de regras de negócios. Os componentes de visualização se encarregam da exibição final e de fornecer meios do usuário indicar as operações a serem realizadas pela aplicação. Componentes do controlador são responsáveis em fazer a ponte entre os outros dois tipos de componentes, interpretando os eventos gerados pelos componentes de visualização e disparando a execução do método correspondente ao Modelo.

A utilização de MVC para aplicações web faz com que se desenvolva uma aplicação bem estruturada e com módulos bem distintos. Sendo que os *web-designers*, que são os profissionais responsáveis pela parte visual da aplicação, não precisem entender rotinas de programação que são desenvolvidas por programadores propriamente ditos. Com o MVC é possível armazenar objetos na sessão do usuário o que auxilia as regras de negócios e torna a aplicação mais segura, pois não se utiliza *cookies* nas máquinas do cliente e evita-se o emprego de passagem de parâmetros através de *links*. Está última técnica é desaconselhável,

pois qualquer pessoa pode alterar o parâmetro de um link, e com isso ter acesso a áreas denominadas restritas.

O MVC consiste em trabalhar utilizando-se três camadas. A camada de visualização (*view)* foi utilizada a parte do gerador responsável por gerar código HTML, o qual é acoplado com as páginas web desenvolvidas com a linguagem *Velocity*. Na camada de controle (*control)* onde estão as regras de negócio será utilizando o *framework webwork* que será responsável pela integração entre a visualização e a modelagem. Foi utilizada para realizar o controle a parte do gerador que é responsável pela criação dos script SQL dinamicamente. Já na terceira camada, a de modelagem (*model),* foi utilizado o banco de dados Oracle.

2.2 VELOCITY

Segundo Steil (2002), *Velocity* é um *template-engine* feito em Java. 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. Apesar de o seu maior uso ser para aplicações web, pode ser usado para: formatação de mensagens com base em um *template*; criação de documentos *Rich Text Format* (RTF); geração de *scripts* SQL para a definição/manipulação de dados em um banco de dados relacional e transformação de arquivos *eXtensible Markup Language* (XML) em *templates*.

Para o desenvolvimento web, o *Velocity* implementa a camada de visualização do MVC. Dessa forma, os *designers* não precisam preocupar-se em entender os complicados códigos Java e os programadores não precisam ficar adaptando *loops* e variáveis no meio do HTML. O quadro 1 apresenta um exemplo de um *template Velocity.*

```
<!-- imprimir nome e idade cadastrados -->
<th>$nome - $idade anos</th> <br>

<th>Cidade :<th>
<select>
    <!-- listar cada elemento de $cidades -->
    #foreach($city in $cidades)
        <!-- selecionar a cidade cadastrada -->
        <option #if($city.equals($cidade)) selected #end> $city </option>
    #end
</select> <br>

<!-- listar cada elemento de $sexos -->
#foreach($sex in $sexos)
    <!-- selecionar o sexo cadastrado -->
    <input type="checkbox" #if($sex.startsWith($sexo)) checked #end> $sex
    </input> <br>
#end
```

Quadro 1 – Template VTL: Cadastro.vm

No quadro 1 pode-se verificar a *tag* # (sustenido), que é utilizada nas rotinas de comparação e repetição (#if e #foreach), e a *tag* $ (cifrão) que identifica o acesso a variáveis das ações. Com a criação dos métodos gets nas variávies em classes java, que são os métodos reponsáveis pelo acesso ao conteúdo e funcinalidade de uma variavel no conceito de orientação a objeto, com isso pode-se ter o acesso ao conteúdo e as suas funcionalidades desta variável no *Velocity* com a utilização do nome da variável precedidade da *tag* #. Desta forma o acesso de um uma váriavel no *Velocity* fica claro e semelhante ao acesso de uma variável entre obejtos Java.

O quadro 2 apresenta uma classe java contendo métodos gets correspondentes as variáveis que estão sendo acessadas através do *Velocity* no quadro 1.

No quadro 2 pode-se verificar que a classe cadastro herda carecteristísticas de uma super classe chamada SystemAction, a qual possui diversos métodos comuns a todas as classes do sistema, entre eles os mais importantes são os métodos para acesso e manutenção da sessão do usuário, os métodos insert, update e delete. Está classe está representanda pelo quadro 3.

```
package tcc;
import br.com.wheb.action.SystemAction;
import java.util.ArrayList;
import java.util.Arrays;

/**
 * @author Luis Coelho
 **/


public class Cadastro extends SystemAction{
    ArrayList cidades = new ArrayList(Arrays.asList(new String[]
                                                    {"Blumenau",
                                                     "Florianópolis",
                                                      "Lages"}));
    ArrayList sexos = new ArrayList(Arrays.asList(new String[]
                                                  {"Masculino",
                                                   "Feminino"}));
    String nome = null, cidade = null, sexo = null;
    int idade;

    public String mostrarCadastro(){
        nome   = "Luis Fernando Coelho";
        cidade = "Lages";
        idade  = 21;
        sexo   = "M";
        return SUCCESS;
    }
    public String    execute() throws Exception { return SUCCESS; }
    public Object    getModel()                 { return null; }
    public ArrayList getCidades()               { return this.cidades; }
    public ArrayList getSexos()                 { return this.sexos; }
    public String    getNome()                  { return this.nome;}
```

Quadro 2 – Classe Cadastro.Java

O quadro 3 abaixo pode-ser verificar que a classe possui uma super classe chamada de ActionSupport, assim como a interface ModelDrive , que estão empacotadas nas bibliotecas do *framework Xwork*, o qual está empacotado nas bilbiotecas do *framework WebWork.*

Para a utilização do Xwork todas as classes de ações do sistema devem ser estendida da classe ActionSupport para que está classe faça a comunicação com o Xwork e mantenha a sessão e os objetos na sessão enquanto o usuário fique autenticado no sistema. Desta forma o programador fica livre da responsabilidade de verificar se o usuário está autenticado no sistema a cada vez que uma ação é solicitada ou página acessada.

```
package br.com.wheb.tcc.action;
import com.opensymphony.xwork.ActionSupport;
import com.opensymphony.xwork.ModelDriven;
//Demais os imports

/**
 * @author Luis Coelho
 *
 * Classe base para todas as Action do sistema.
 * - Exige que todas implementem o ModelDriven
 * - Adiciona suporte a identificação de usuário
 *
 */
public abstract class SystemAction extends ActionSupport implements ModelDriven,
        UserSessionAware {

    /** Objeto armazenado na sessão com info do usuário */
    protected UserSession userSession;
    public void setUserSession(UserSession usr)      {…}
    public UserSession getUserSession()              {…}
    public HttpSession getSessao()                   {…}
    public String update()throws Exception           {…}
    public String delete()throws Exception           {…}
    public HashMap insert(HashMap campos,String nrFrame)throws Exception{…}
    public Usuario getUsuario(){…}
```

Quadro 3 – Classe SystemAction.java

2.3 FRAMEWORK WEBWORK

Segundo Mota (2004), *WebWork* é o *framework* MVC propriamente dito. É construído totalmente baseado no *Xwork*, uma implementação genérica do *Command Pattern* que é desacoplada do ambiente web, facilitando os testes de suas ações. São ações possíveis do webwork: converter tipos; verificar se todos os campos de um formulário foram preenchidos corretamente; verificar se o usuário está autenticado no sistema; verificar se o sistema provê acesso a determinadas páginas; redirecionar para uma página específica, por exemplo, a página de *login*, caso o usuário esteja tentando acessar indevidamente alguma página; etc.

O *WebWork* é um *framework*, assim como Struts, JBanana, e o Maverick. Todos implementam o modelo MVC, porém o *WebWork*, através de ações e resultados, provê ao desenvolvedor interceptadores, os quais são executados após o usuário invocar uma ação em uma página e antes de á classe correspondente a ação ser executada. Com isso os interceptadores são responsáveis por validar os campos de um formulário evitando um

possível erro por um campo não ter sido informado pelo usuário. O *WebWork* também possui *tags* JSP e macros *Velocity*. É um utilitário de validação, conversão automática de dados e internacionalização. Com o *WebWork* é possível separar tudo que é específico para a web do que é genérico. O *WebWork* foi utilizado na camada de controle sendo responsável pela integração entre a camada de visualização e a camada de modelagem. Sendo a camada de modelagem o banco de dados e será acessado via JDBC.

No quadro 3 tem-se a ação `montarCadastro.action` que faz o mapeamento entre a classe `Cadastro.java` (quadro 2) e o *template* `Cadastro.vm` (quadro 1). Quando a ação é executada pelo navegador, é invocado o método `mostrarCadastro` da classe `Cadastro.java`. Se o resultado for *SUCESS*, ou seja, se não ocorrer nenhum erro, será direcionado para o *template* `Cadastro.vm`, que será processado e irá gerar como resultado a página mostrada na figura 1.

```
<action name="montarCadastro" class="tcc.Cadastro" method="mostrarCadastro">
    <result name="success" type="velocity">Cadastro.vm</result>
</action>
```

Quadro 3 – Mapeamento - arquivo Xwork.xml

No quadro 2 pode-se ver a definição e inicialização das variáveis que irão ser acessadas no *Velocity* através do GET correspondente. Sempre que se for acessar uma variável no *Velocity* a mesma deverá ter associada um método *GET*, e o caminho inverso também é válido. Se possuir um campo no *Velocity*, para ter acesso ao valor deste campo, deve-se criar o método *SET* correspondente ao nome do campo definido na página

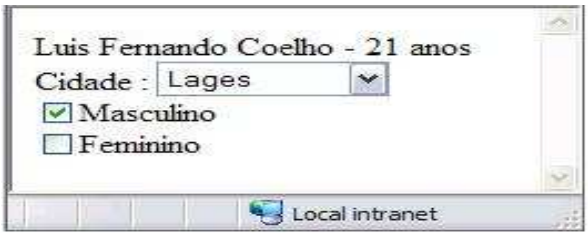


Figura 1 – Resultado em HTML

2.4 DICIONÁRIO DE DADOS

Segundo (Reyes,Amaro,Patrício, 2000), um dicionário de dados é uma descrição de características e atributos relevantes para um projeto ou trabalho em particular. É usado para o armazenamento das informações no levantamento do projeto, também para o controle da coleta de objetos e suas características e atributos. É formado por uma lista de características a serem coletadas. Cada característica possui uma lista de atributos que a descrevem. O dicionário organiza a coleta de dados a ser feita durante o levantamento.

O gerador utiliza as informações do dicionário de dados para a geração das páginas. No dicionário de dados está cadastrado a modelagem das tabelas, sendo o cadastro em nível de coluna da tabela, onde é possível definir todas as características de cada uma. Estas informações são utilizadas pelo gerador para fazer a construção da página dinamicamente sendo acoplado nos *templates Velocity*.

No dicionário deverão ser cadastradas todas as tabelas do sistema, bem como suas integridades com outras tabelas. Para cada coluna de uma tabela será definido o tamanho que o campo deverá aparecer na página, se terá uma descrição antes do campo, a ordem que o campo irá aparecer na página, se o campo é o tipo somente leitura onde não é possível alterar o seu conteúdo. Se for um campo numérico ou data é possível cadastrar uma máscara para o campo que será automaticamente formatado na geração da página.

2.5 GERAÇÃO DE CÓDIGO

Conforme Shimabukuro Junior (2005, p. 3), o termo gerador de código pode assumir diferentes significados, tais como: compiladores, pré-processadores, meta-funções que geram classes e procedimentos, ou *wizards*. Os compiladores são geradores de código, uma vez que recebem uma informação em certo nível de abstração, como programas escritos na linguagem Java ou C++, e transformam essa informação em uma linguagem de mais baixo nível, como código objeto, *bytecode* ou código de máquina. Um *wizard* é um programa gráfico que recebe uma especificação em alto nível de abstração e a transforma em artefatos de software. Algumas ferramentas *Computer Aided Software Engineering* (CASE) geram uma parte do código de uma aplicação, utilizando como entrada diagramas e especificações que são inseridos na ferramenta pelo engenheiro de aplicação.

Segundo Santos (2002, p. 1), os geradores de código devem ser utilizados quando se tem uma pequena equipe de trabalho e quando o aplicativo a ser desenvolvido não requer personalizações, como a linguagem e o banco de dados a serem utilizados. São ferramentas capazes de gerar código sem erros, utilizando modelos de alto nível como os diagramas de ações, de eventos, de seqüência, entre outros.

Conforme Herrington (2003, p. 90), para a criação de um gerador deve-se seguir algumas etapas. A figura 2 representa o fluxo de atividades para a criação de um gerador:

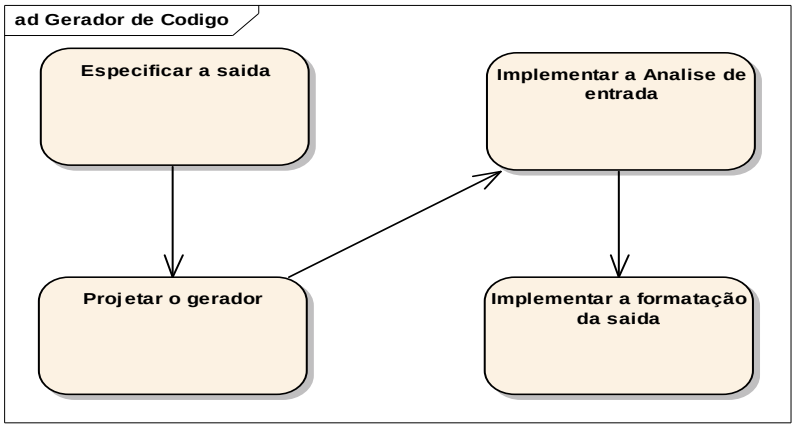


Figura 2 – Gerador de Código

A seguir são descritos as etapas de construção de um gerador de código com detalhes:

a) especificar a saída: consiste em determinar e implementar manualmente como deverá ser a saída;

b) projetar o gerador: consiste em determinar como será a entrada do gerador, como a entrada será analisada, como será o código a ser gerado (*templates*), como a saída será construída;

c) implementar a análise da entrada: consiste em programar o processamento da entrada para extrair/armazenar as informações necessárias para gerar a saída;

d) implementar a formatação da saída: consiste em implementar a geração dos arquivos apropriados (*templates*).

2.6 FERRAMENTAS DE GERAÇÃO DE CÓDIGO

Com o crescimento das aplicações para web surgiram diversas ferramentas para auxiliar na geração de código, destacando-se: DBDesigner, Geração automática para ASP, Ferramenta CASE para geração de páginas ASP, GerCod.

Segundo Fabulous Force Database Tools (2003), o DBDesigner é uma ferramenta de

modelagem visual livre para quem utiliza o banco de dados MySQL. Através dele pode-se fazer a modelagem das tabelas, bem como seus relacionamentos, representando-os de forma gráfica. Depois de feita essa modelagem é possível conectar-se ao MySQL para realizar a sincronização, ou seja criar as tabelas e os relacionamentos. É possível também trabalhar com os dados das tabelas.

Em Silveira (2003) é descrito o desenvolvimento de uma ferramenta em Delphi com a capacidade de criar um sistema de cadastro e consultas usando a linguagem ASP, baseando-se na leitura da estrutura relacional do banco de dados Access, através de *Open Database Connectivity* (ODBC). As páginas geradas pela ferramenta permitem a total manipulação (alteração, exclusão, inserção e consulta) dos registros no banco de dados.

Castilhos (2004) descreve uma ferramenta CASE para criação, definição, documentação e geração de páginas em ASP. Através da linguagem ASP e com a estrutura do projeto definida no banco de dados SQL Server é possível a geração de qualquer sistema de formulários com inclusão, alteração, exclusão, consulta, localização e a paginação dos registros.

A ferramenta CodGer desenvolvida por Menin (2005) consiste em uma ferramenta desenvolvida em Java que auxilia programadores e analistas na geração e edição de código JSP. A geração é feita utilizando as definições encontradas em uma base de dados MySQL. A ferramenta permite ao usuário a configuração das páginas a serem criadas. Para realizar esta configuração devem-se informar quais serão as tabelas e atributos de cada página. As páginas geradas permitem fazer inclusões, exclusões, alterações e consultas no banco de dados.

3 ESPECIFICAÇÃO E DESENVOLVIMENTO DO TRABALHO

Neste capítulo é apresentado o gerador desenvolvido bem como uma aplicação de exemplo demonstrando os resultados obtidos.

3.1 VISÃO GERAL DO GERADOR

O gerador desenvolvido agiliza o processo de criação de aplicativos para a web, podendo ser utilizado com as mais diversas tecnologias Java para a internet. A figura 3 demonstra os passos executados deste a solicitação de uma página até receber a página gerada:

a) o usuário utiliza um *browser* para realizar uma requisição de uma página através de uma URL;

b) a requisição é a solicitação de uma página feita pelo *browser* para o servidor de aplicação;

c) o servidor de aplicação irá decodificar a ação e invocar o *webwork*, que irá encaminhar para o método correspondente que irá atender a esta ação com o acesso ao gerador.

d) o gerador irá realizar a leitura do dicionário de dados, que fica armazenado no banco de dados, e irá fazer a geração da página e encaminhar para o servidor web acoplar ao *template Velocity*, e após processá-lo irá encaminhar a página para visualização pelo usuário.

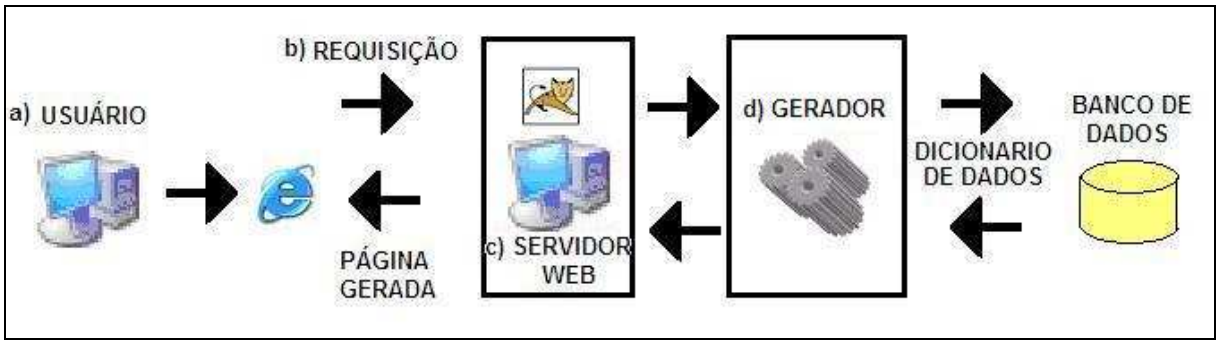


Figura 3 – Processos

Para desenvolver uma aplicação para web, o usuário deve utilizar a interface web da ferramenta para cadastro das tabelas bem como os relacionamentos entre elas. Para cada coluna de uma tabela é possível determinar o tipo de componente que será gerado, uma máscara, identificar se os dados do componente podem ser alterados entre outras características. Com esse dicionário cadastrado é possível a geração das páginas dinamicamente.

3.2 LEVANTAMENTO DOS REQUISITOS

Com o levantamento dos requisitos é possível que usuário e desenvolvedores tenham a mesma visão do problema a ser resolvido. Essa etapa é muito importante para saber a dimensão do problema e aplicar no desenvolvimento do gerador.

A seguir são relacionados os Requisitos Funcionais (RF) e os Requisitos Não Funcionais (RNF) da ferramenta:

a) permitir a conexão com qualquer banco de dados para acesso ao dicionário de dados de uma aplicação (RF);
b) permitir a leitura do dicionário de dados, verificando os atributos de cada campo de uma tabela (RF);

c) permitir que o usuário personalize sua página, cadastrando as informações de cada campo de uma tabela no dicionário de dados, através de uma interface web (RF);
d) permitir a geração de código HTML em tempo de execução (RF);
e) permitir a geração de comandos SQL em tempo de execução (RF);
f) deverá ser acoplada em aplicações Java para web (RNF);
g) ser desenvolvida em Java, utilizando a ferramenta NetBeans versão 4.1 (RNF).

3.3 ESPECIFICAÇÃO

Para a realização da especificação foi utilizado a ferramenta Sybase PowerDesigner para a modelagem do dicionário de dados e o Enterprise Architect para o diagrama de caso de uso ,atividades e classes.

3.3.1 Modelagem do dicionário de dados

A modelagem do dicionário de dados apresenta a estrutura do banco de dados com todos os relacionamentos entre as tabelas. Com isso é possível ter uma visão de como o dicionário está estruturado. A figura 4 apresenta a modelagem.

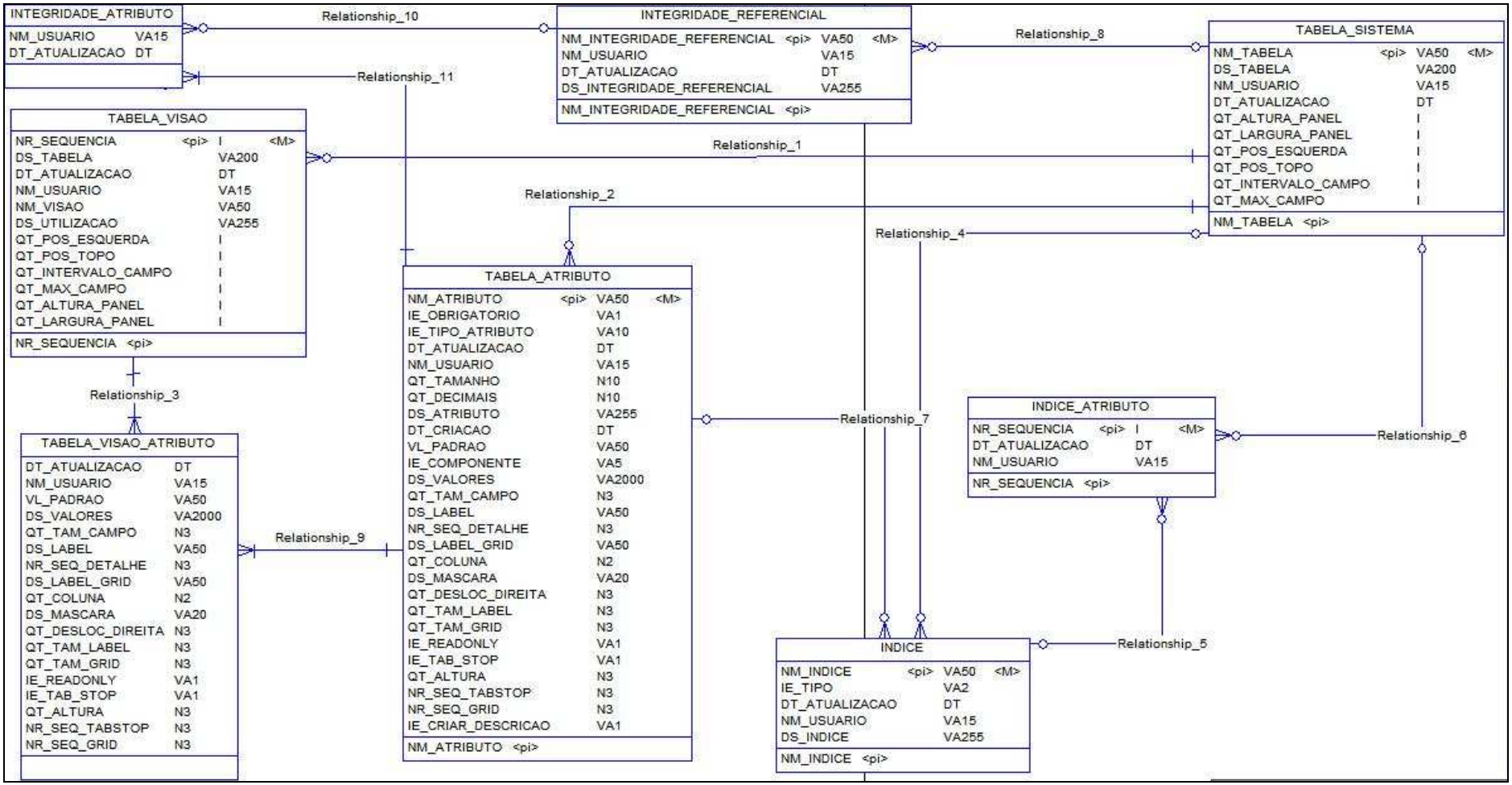


Figura 4 – Modelagem do Dicionário de Dados

Na figura 4 tem-se uma tabela chamada **tabela_sistema**, onde são cadastradas todas as tabelas do sistema. Nesta tabela, há atributos como **qt_largura_panel**, **qt_altura_panel**, **qt_pos_esquerda**, **qt_pos_topo** que identificam o tamanho e posição que página html correspondente a tabela deverá ser gerada. Já o atributo **qt_intervalo_campo** identifica o espaço que deve haver entre os campos em cada linha e **qt_max_campo** identifica o tamanho máximo que um campo pode ocupar na página.

Tem-se o relacionamento entre a **tabela_sistema** e a **tabela_atributo**, indicando que uma **tabela_sistema** poderá conter diversos registros na **tabela_atributo**. Na tabela **tabela_atributo** é feito o cadastro de cada coluna da tabela. Nela é possível verificar os atributos que caracterizam como o campo será gerado. Entre eles o tamanho do campo na página (**qt_tamanho**) se o campo será um *edit*, *lookup*, *checkbox*, *combox* ou *text_area*, se o campo irá ter uma descrição antes (**ds_label_detalhe**). Caso seja uma campo numérico e for necessário formatar, basta cadastrar uma máscara no campo **ds_mascara**. A figura 5 demonstra a utilização destes atributos.

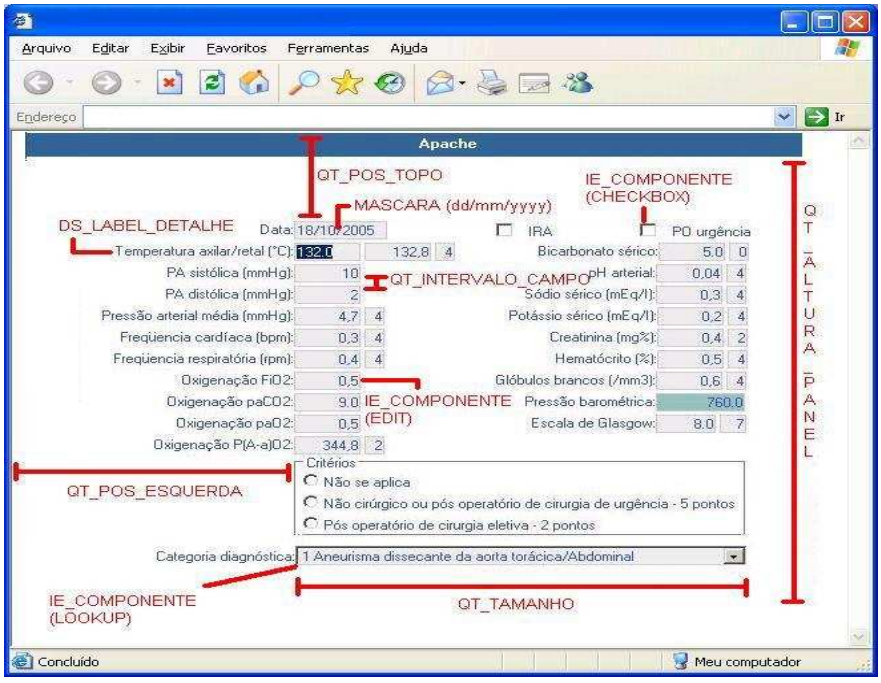


Figura 5 – Detalhes e configurações

Na figura 5 pode-se verificar a utilização do atributo **qt_pos_topo,** indicando onde deve iniciar a geração dos campos em relação a parte superior da página, e o atributo **qt_pos_esquerda**, que indica a posição em relação a margem esquerda. Tem-se a utilização do atributo **ie_tipo_componente**, na criação de um *checkbox*, de um *edit* e de um *lookup*. O atributo **ds_label_detalhe** refere-se a uma breve descrição do campo gerado. No campo data, tem-se associado ao atributo **ds_mascara** o valor **dd/mm/yyyy** indicando que o campo deverá ser formatado como uma data e que somente poderá ser informado uma data como valor no campo pelo usuário. O campo **qt_intervalo_campo** indica o espaço em branco que deve conter entre as linhas.

Na tabela índice é cadastrado as integridades entre as tabelas. Nela será definido se o atributo de uma tabela será uma *primary key*, ou *foreign key*.

A tabela **tabela_visao** e **tabela_visao_atributo** serão utilizadas quando se necessitar que uma tabela gere páginas diferentes, por exemplo em uma página de cadastro necessita-se que todos os campos de uma tabela **pessoa_fisica** apareçam para que o usuário realize o cadastro, porém em uma página de alteração de dados, por exemplo, pode-se definir que alguns campos o usuário não poderá alterar. Para que isso seja possível é necessário cadastrar uma visão na **tabela_visao** e somente os campos que o usuário terá acesso na **tabela_visao_atributo**. Com isso através de uma tabela é possível criar diversas páginas de acordo com a necessidade da aplicação.

A tabela **integridade_referencial** e **integridade_atributo** serão utilizadas para que um campo de uma tabela possa ser referenciado com outro tabela. Por exemplo: Ao criar uma tabela que irá representar um aluno podemos criar o campo **CD_CIDADE**. Se for necessário identificamos na tabela **integridade_refenrencial** que a tabela ALUNO faz referencia a tabela CIDADE e na tabela **integridade_atributo** indicamos qual atributo irá fazer referência

aquela tabela. Neste caso seria cadastrado o atributo **CD_CIDADE**.

Desta forma se for cadastrado que o atributo **CD_CIDADE** é um componente do tipo *lookup*, automaticamente será carregado uma lista de cidades neste atributo. Caso o componente for um *edit* será carregado o nome da cidade correspondente ao código armazenado no banco na tabela aluno.

3.3.2 Diagrama de caso de uso

O diagrama de caso de uso indica os passos necessários para o funcionamento do gerador. A figura 6 representa os casos de usos levantados.

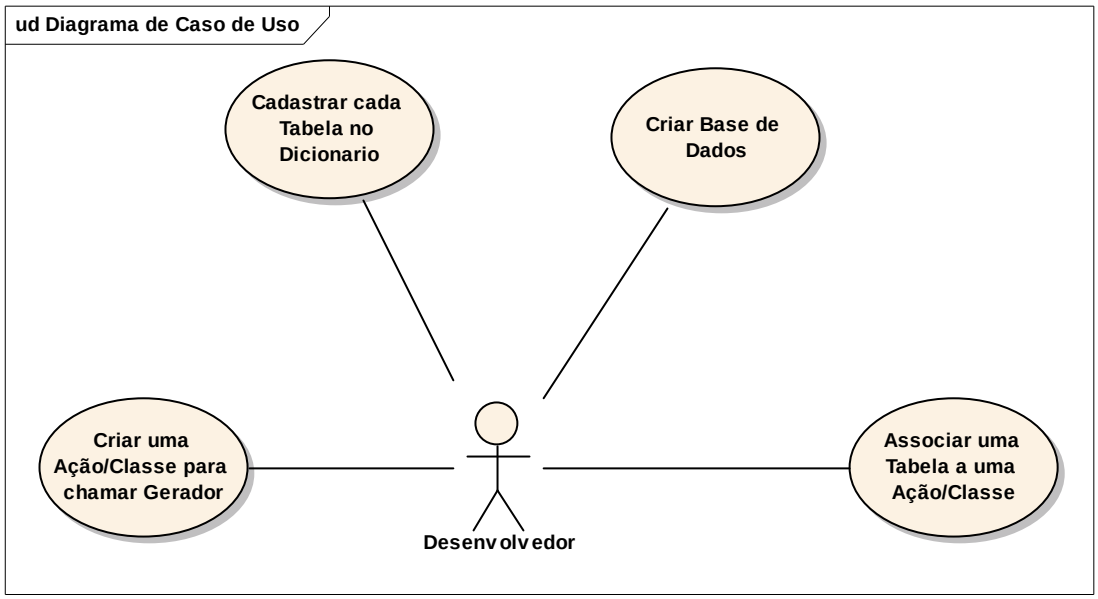


Figura 6 – Diagrama de casos de usos

Abaixo segue a descrição dos casos de uso:

a) criar base de dados: o desenvolvedor deve criar as tabelas do projeto que está desenvolvendo;

b) cadastrar cada tabela no dicionário: o desenvolver configura como cada tabela, bem como seus atributos, serão apresentados;

c) associar uma tabela a uma ação/classe: o desenvolver desenvolve as regras de

negócio referente a uma tabela;

d) criar uma ação/classe para chamar o gerador: o desenvolvedor associa a regra de negócio com o gerador.

3.3.3 Diagrama de classes.

As classes do gerador estão divididas nos seguintes pacotes e representadas na figura 7:

1. br.com.wheb.tcc.action
    a. Logoff – classe que remove todos os objetos da sessão e encera a sessão;
    b. SystemAction – superclasse para as ações, que estende e implementa métodos do *framework WebWork*;
    c. WhebAcion – superclasse para as classes que contém as regras de negócio de cada tabela;
    d. TabelaWebAction - classe genérica que faz a integração entre os eventos da visualização e invoca a classe TabelaAction.
2. br.com.wheb.tcc.dao
    a. WhebDAO - classe que realiza toda a interação com o banco de dados.
3. br.com.wheb.tcc.classes
    a. TabelaAction – classe genérica que implementa os métodos insert/update/delete para tabelas que não possuem regras de negócio específicos como cadastros.
4. br.com.wheb.tcc.components

a. UserSessionAware - Interface que representa o usuário;

b. UserSession – Representa o usuário logado no sistema.

5. br.com.wheb.tcc.funcoes

a. CamposFormulario – faz a geração dos *scripts* SQL para inclusão, exclusão e alteração. Além de fazer a validação dos campos, verificando o que o usuário alterou antes de montar os *scripts* de inclusão e alteração;

b. CamposFormularioWeb – armazena os campos do formulário HTML;

c. MontaPastas – faz a montagem de pastas na parte inferior da tela simulando abas para a navegação entre as páginas;

d. MontaMensagemErro – monta uma mensagem de erro personalizada e mostra o erro para usuário;

e. CarregarTabelaSistema – carrega todos os atributos da **tabela_sistema**, **tabela_atributo**, **integridade_referencia**, **integridade_atributo**;

f. DicionarioDados – armazena todas as informações das tabelas e campos do formulário;

g. Conversores – faz formatação de números e datas;

h. FuncoesWeb – possui funções *JavaScript* que serão executada quando o usuário estiver informando o formulário;

i. Funcoes – trata os dados que irão compor um os componentes *radio group* e *checkbox*;

j. MontaLinks – monta links para ser executado em *JavaScript.*

6. br.com.wheb.tcc.system

a. DisableSecurityAction – interface do *framework WebWork* que indica que para a execução de uma ação não precisa ter usuário autenticado;

   b. ApagaArquivosTemporarios – *servlet* que apaga os *scripts* SQL gerados durante a execução do gerador.
   c. Propriedades – armazena as propriedades do sistema que podem ser deficinidas em um arquivo texto.
   d. Initializar – *servlet* que faz a carga das propriedades do arquivo texto.
   e. ApplicationSecurityInterceptor – *servlet* que verificar se existe usuário autenticado no sistema antes de executar uma ação.

7. br.com.wheb.tcc.models
   a. User – classe que implementa os métodos necessários para a para validação do usuário na seção;
   b. Usuário – herda os métodos e atributos de user e representa o usuário que da aplicação de administração do dicionário de dados;
8. br.com.wheb.tcc.classes
   a. TabelaSistema – classe que representa a tabela tabela_sistema;
   b. TabelaAtributo – classe que representa a tabela tabela_atributo.
9. br.com.wheb.tcc.gerador.funcoes
   a. GeraDetalhe – faz a geração da página no modo detalhe;
   b. GeraGrid - faz a geração da página no modo *grid*.
10. br.com.wheb.tcc.io
   a. Arquivo – implementa os métodos para a manipulação de arquivos;
   b. GeraArquivo – monta os arquivos de erro, sql e logs do sistema personalizados.

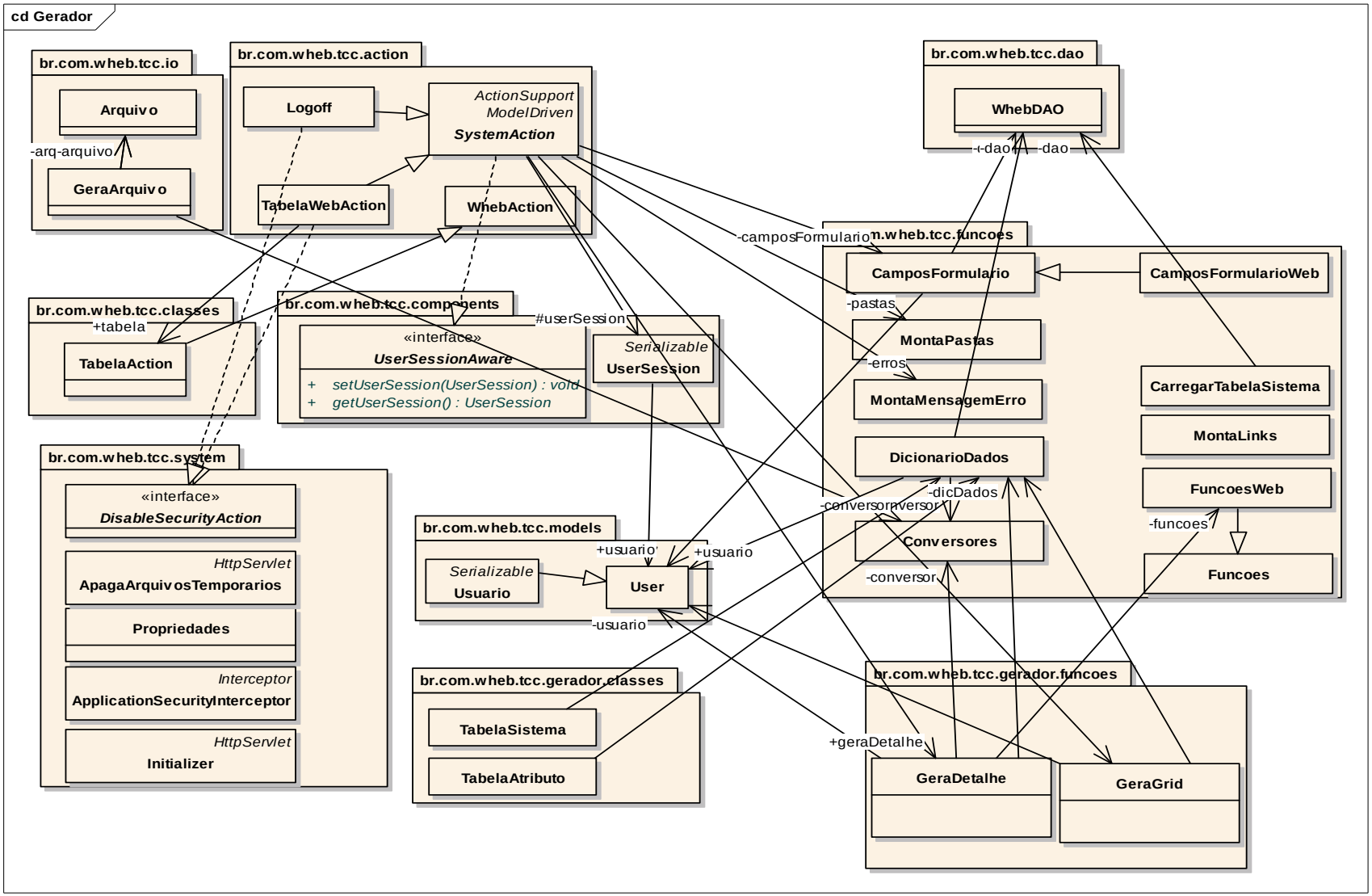


Figura 7 - Diagrama de classes

3.3.4 Diagrama de atividades

O diagrama de atividade representa o fluxo necessário para a realização de uma atividade.

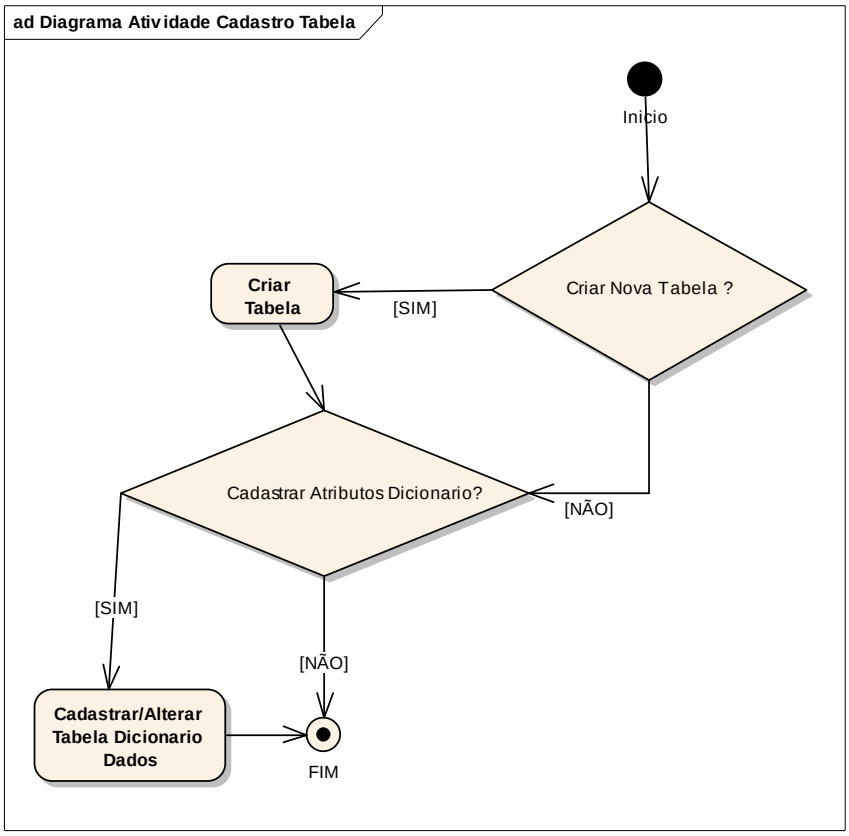


Figura 8 – Diagrama Atividade Cadastro Tabela

Na figura 8 representa o inicio do processo pedindo um novo registro na **tabela_sistema**, pois se pretende cadastrar uma nova tabela para o sistema. Uma vez cadastrada a tabela é possível cadastrar os atributos desta tabela na **tabela_atributo**. Todo esse processo de cadastramento de tabelas e atributos é feito através de uma interface web.

A figura 9 representa o fluxo de atividades necessárias para a geração de uma página bem como a verificação de alteração das informações e a atualização das mesmas no banco de dados.

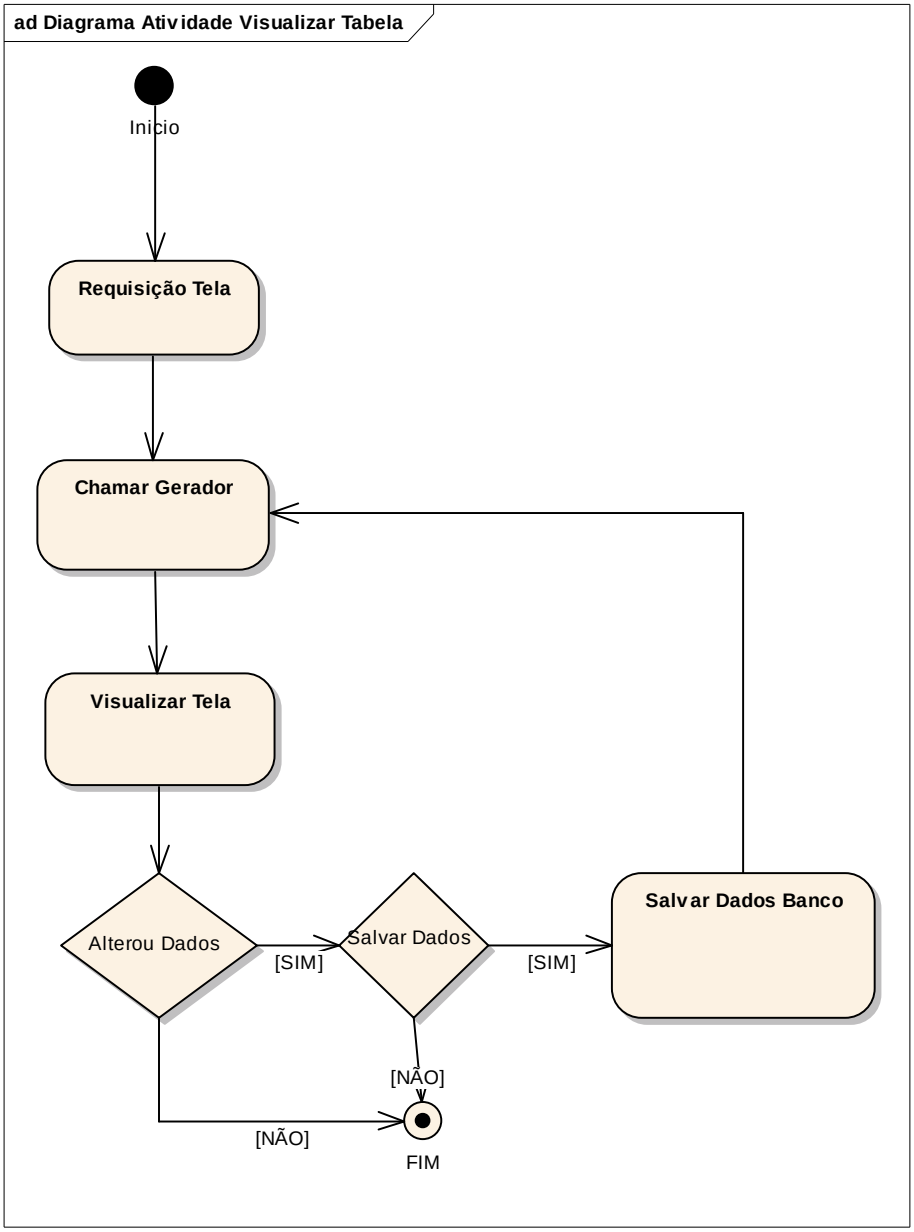


Figura 9 – Diagrama Atividade Visualizar Tabela

Na figura 9 se tem os passos que serão executados pelo servidor web e pelo gerador desde o momento em que usuário executa um *link* até a visualização da página. Após a visualização estar disponível para o usuário, ele poderá alterar as informações e optar por salvar as alterações. Uma vez o usuário pedindo para salvar as alterações o gerador irá fazer a verificação do que foi alterado e irá atualizar o banco de dados somente com o que foi atualizado.

3.4 IMPLEMENTAÇÃO

O gerador foi desenvolvido utilizando o ambiente de programação NetBeans 4.1 com a linguagem de programação Java 1.5. Para demonstrar as páginas geradas dinamicamente através da aplicação PEP de exemplo, foi utilizando o servidor de aplicação TomCat 5.5, o qual é o responsável em controlar o tráfego de informações entre o banco de dados e as regras de negócio da aplicação.

3.4.1 Arquitetura do Gerador

A figura 10 apresenta as etapas do gerador.

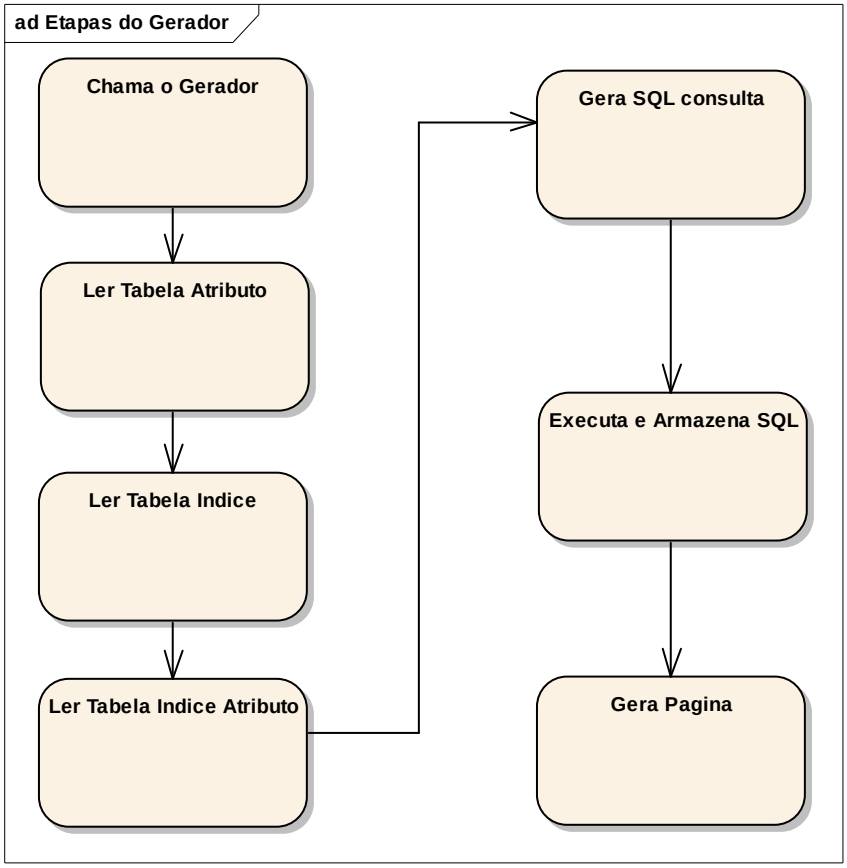


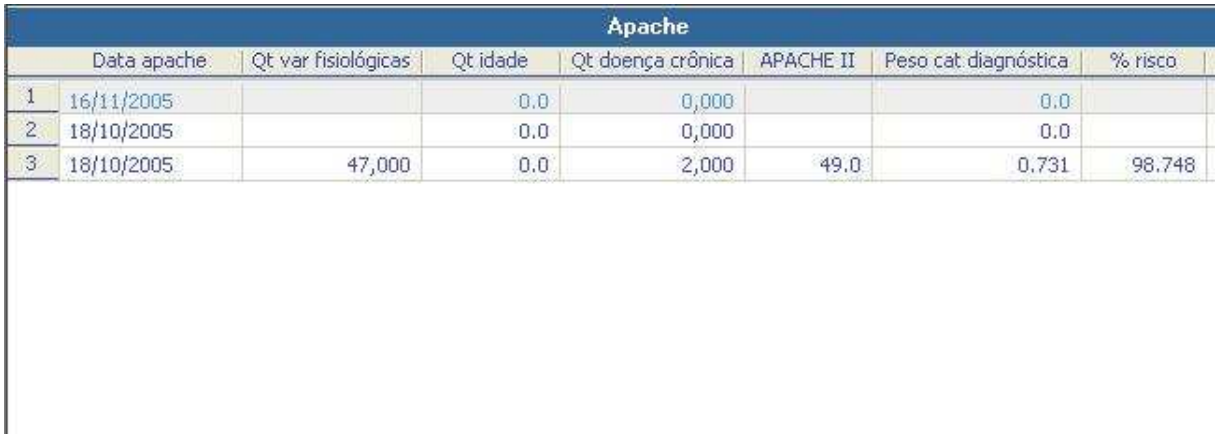
Figura 10 - Etapas do Gerador

Para iniciar a geração de uma página, deve-se através de um *link* invocar uma ação do

webwork que irá receber como parâmetro o nome da tabela que se deseja gerar uma página. Com o nome da tabela, a ação irá invocar o gerador passando como parâmetro o nome da tabela. Ao invocar o gerador, o mesmo irá fazer a leitura da tabela **tabela_sistema** para identificar o tamanho que a página será gerada. Feito isso, o próximo passo é a leitura da tabela **tabela_atributo**, onde estarão especificados quais campos irão compor a página a ser gerada; da tabela **tabela_indice** para que o gerador tenha a informação de qual índice será a *primary key* da tabela; com o índice é feita a leitura da tabela **índice_atributo** obtendo-se o nome da coluna correspondente, para quando for solicitada a visualização de apenas um registro ser feita a restrição por esse campo. Ainda é feita a leitura da tabela **integridade_referencial** para verificar se está tabela possui referências para alguma outra tabela do sistema, e caso tenha uma referência é feita a leitura da tabela **integridade_atributo** para armazenar qual atributo está sendo referenciado.

Feita a leitura dos campos, o gerador irá montar e executar um script SQL de consulta para a tabela. Com a execução do SQL na memória e a tabela atributo, será realizado a validação dos campos. Nesse momento será realizada a formatação de cada campo de acordo com o que foi cadastrado no dicionário de dados. Caso seja um campo numérico será formatado de acordo com a máscara informada, assim como para um campo do tipo data.

Após a validação dos registros o gerador inicia a geração da página. Serão percorridos todos os campos que compõem a página para a criação de um *grid* conforme a figura 11 que representa a tabela apache, que será descrita na seção 4.



Apache

	Data apache	Qt var fisiológicas	Qt idade	Qt doença crônica	APACHE II	Peso cat diagnóstica	% risco
1	16/11/2005		0.0	0,000		0.0	
2	18/10/2005		0.0	0,000		0.0	
3	18/10/2005	47,000	0.0	2,000	49.0	0.731	98.748

Figura 11 – Tabela APACHE em modo *grid*.

A seqüência de apresentação dos campos no *grid* é determinada pelo atributo **nr_seq_grid**, bem como seus *labels* definidos pelo campo **ds_label_grid**. Após a geração do *grid* o mesmo será acoplado ao *template Velocity* e enviado para visualização pelo usuário. O usuário poderá navegar pelo *grid* e optar por ver detalhes do item selecionado, solicitando uma nova requisição para o servidor de aplicação que irá chamar novamente o gerador indicando que deverá ser gerada uma nova página conforme a figura 12.

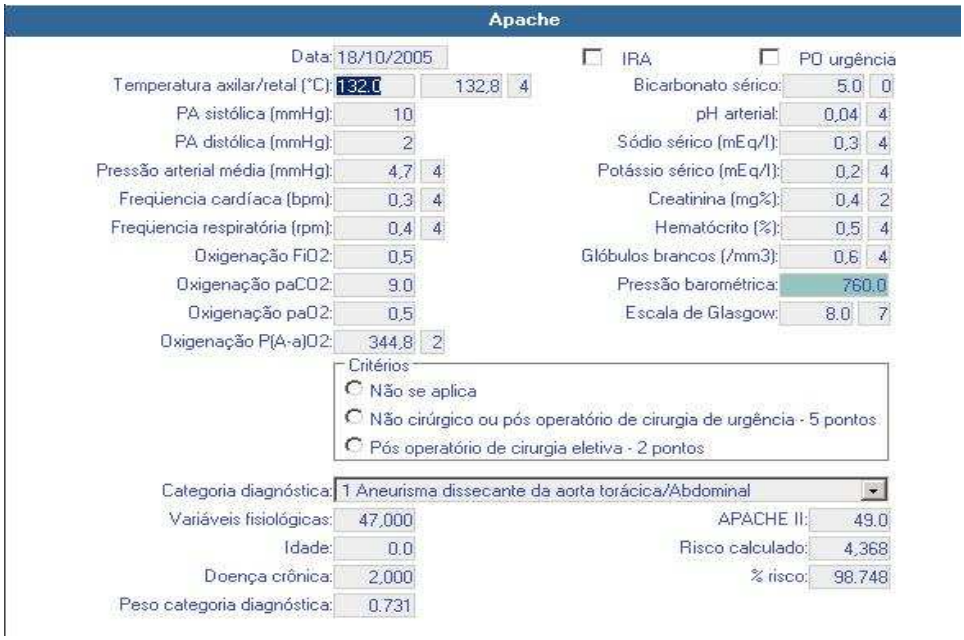


Figura 12 – Tabela APACHE em modo detalhe

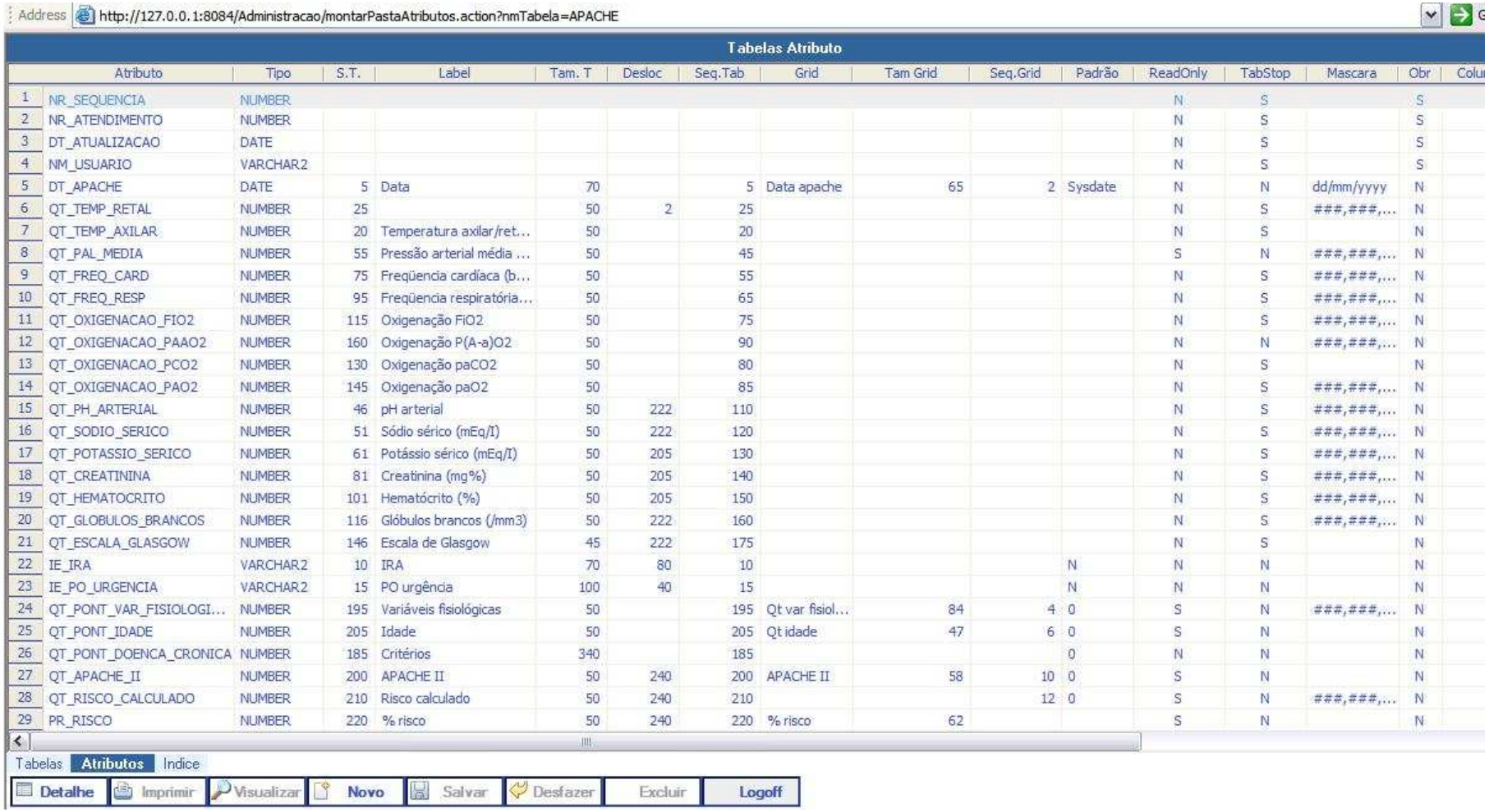
O gerador será invocado e como a definição do dicionário de dados já está na memória, conforme padrão MVC, será executado um script SQL para o item selecionado pelo usuário através do grid e será gerada uma nova página com os detalhes do item. A seqüência dos campos na página é determinada pelo campo **nr_seq_detalhe** criando *edits, lookups, textarea* de acordo com o tipo do atributo. A geração é feita em linhas e enquanto os campos possuírem o campo **qt_desloc_direta** informado, são criados na mesma linha. Quando próximo atributo não possuir este campo informado automaticamente será gerada uma nova linha e o processo se repete até o fim dos atributos da tabela. Os tipos dos campos estão definidos pelo atributo **ie_tipo_atributo**. Este campo irá definir se o campo é um *edit, lookup, checkbox, combox* ou *textarea,* o tamanho que o mesmo irá ocupar na página está definido no atributo **qt_tamanho** e **qt_altura**. Será gerado um *label* antes do campo caso tenha sido informado o atributo **ds_label_detalhe**. Após todo o processo de geração da página ser executada a mesma será acoplada ao *template Velocity* e enviada para visualização do usuário.

Com a página montada no modo detalhe, o usuário poderá alterar os registros da página e salvá-los. Se o usuário optar por salvar, será novamente chamado o gerador, que irá armazenar os campos da página e fazer a verificação dos campos montados primeiramente com os armazenados após intervenção do usuário, separando os campos que foram realizados alguma alteração. Assim o gerador irá montar um script SQL para atualizar o banco de dados com os campos alterados pelo usuário. O mesmo processo ocorre caso o usuário opte por criar um novo registro, pois mesmo um novo registro vem com valores padrões definidos pelo campo **vl_padrao**. E ao salvar um novo registro é verificado se existe algum campo, que foi cadastrado como obrigatório e caso não tenha sido informado, irá aparecer uma mensagem de aviso e será posicionado no campo que não foi informado.

4 OPERACIONALIDADE DA IMPLEMENTAÇÃO

Para a demonstração da funcionalidade do gerador, foi desenvolvida uma aplicação que será utilizada na área médica. Essa aplicação corresponde ao Prontuário Eletrônico do Paciente - PEP, onde é possível cadastrar e monitorar o que está ocorrendo com um paciente durante o período no qual está internado em um hospital. A seguir os passos necessários para a utilização do gerador na aplicação de exemplo.

O primeiro passo a ser feito é o cadastro das tabelas do sistema bem com os seus relacionamentos através da interface web. Deve-ser realizar o cadastro da tabela na **tabela_sistema** e os atributos de cada tabela bem como as características de cada um na **tabela_atributo**. Para cada atributo de uma tabela, é possível a configuração de suas características conforme representados na figura 13. O cadastro dessas informações será utilizado para a geração das páginas bem como para a atualização dos registros no banco. A figura 13 representa os atributos da tabela APACHE, os quais estão cadastrados no dicionário de dados e estão sendo mostrados em um *grid*. A tabela APACHE irá armazenar as informações do paciente, referente a avaliação do médico pelo método denominado apache. Segundo SOMERJ (2005), o método apache consiste em um sistema de severidade (quantitativa) e prognóstica (qualitativa) dos pacientes críticos que auxilia e agiliza a decisão da equipe médica quanto aos benefícios e cuidados a estes paciente. A figura 14 representa a modelagem de dados do PEP.



	Atributo	Tipo	S.T.	Label	Tam. T	Desloc	Seq.Tab	Grid	Tam Grid	Seq.Grid	Padrão	ReadOnly	TabStop	Mascara	Obr	Colu
1	NR_SEQUENCIA	NUMBER										N	S		S	
2	NR_ATENDIMENTO	NUMBER										N	S		S	
3	DT_ATUALIZACAO	DATE										N	S		S	
4	NM_USUARIO	VARCHAR2										N	S		S	
5	DT_APACHE	DATE	5	Data	70		5	Data apache	65	2	Sysdate	N	N	dd/mm/yyyy	N	
6	QT_TEMP_RETAL	NUMBER	25		50	2	25					N	S	###,###,...	N	
7	QT_TEMP_AXILAR	NUMBER	20	Temperatura axilar/ret...	50		20					N	S		N	
8	QT_PAL_MEDIA	NUMBER	55	Pressão arterial média ...	50		45					S	N	###,###,...	N	
9	QT_FREQ_CARD	NUMBER	75	Freqüencia cardíaca (b...	50		55					N	S	###,###,...	N	
10	QT_FREQ_RESP	NUMBER	95	Freqüencia respiratória...	50		65					N	S	###,###,...	N	
11	QT_OXIGENACAO_FIO2	NUMBER	115	Oxigenação FiO2	50		75					N	S	###,###,...	N	
12	QT_OXIGENACAO_PAAO2	NUMBER	160	Oxigenação P(A-a)O2	50		90					N	N	###,###,...	N	
13	QT_OXIGENACAO_PCO2	NUMBER	130	Oxigenação paCO2	50		80					N	S		N	
14	QT_OXIGENACAO_PAO2	NUMBER	145	Oxigenação paO2	50		85					N	S	###,###,...	N	
15	QT_PH_ARTERIAL	NUMBER	46	pH arterial	50	222	110					N	S	###,###,...	N	
16	QT_SODIO_SERICO	NUMBER	51	Sódio sérico (mEq/l)	50	222	120					N	S	###,###,...	N	
17	QT_POTASSIO_SERICO	NUMBER	61	Potássio sérico (mEq/l)	50	205	130					N	S	###,###,...	N	
18	QT_CREATININA	NUMBER	81	Creatinina (mg%)	50	205	140					N	S	###,###,...	N	
19	QT_HEMATOCRITO	NUMBER	101	Hematócrito (%)	50	205	150					N	S	###,###,...	N	
20	QT_GLOBULOS_BRANCOS	NUMBER	116	Glóbulos brancos (/mm3)	50	222	160					N	S	###,###,...	N	
21	QT_ESCALA_GLASGOW	NUMBER	146	Escala de Glasgow	45	222	175					N	S		N	
22	IE_IRA	VARCHAR2	10	IRA	70	80	10				N	N	N		N	
23	IE_PO_URGENCIA	VARCHAR2	15	PO urgência	100	40	15				N	N	N		N	
24	QT_PONT_VAR_FISIOLOGI...	NUMBER	195	Variáveis fisiológicas	50		195	Qt var fisiol...	84	4	0	S	N	###,###,...	N	
25	QT_PONT_IDADE	NUMBER	205	Idade	50		205	Qt idade	47	6	0	S	N		N	
26	QT_PONT_DOENCA_CRONICA	NUMBER	185	Critérios	340		185				0	N	N		N	
27	QT_APACHE_II	NUMBER	200	APACHE II	50	240	200	APACHE II	58	10	0	S	N		N	
28	QT_RISCO_CALCULADO	NUMBER	210	Risco calculado	50	240	210			12	0	S	N	###,###,...	N	
29	PR_RISCO	NUMBER	220	% risco	50	240	220	% risco	62			S	N		N	

Figura 13 - Tabela Atributo para tabela APACHE

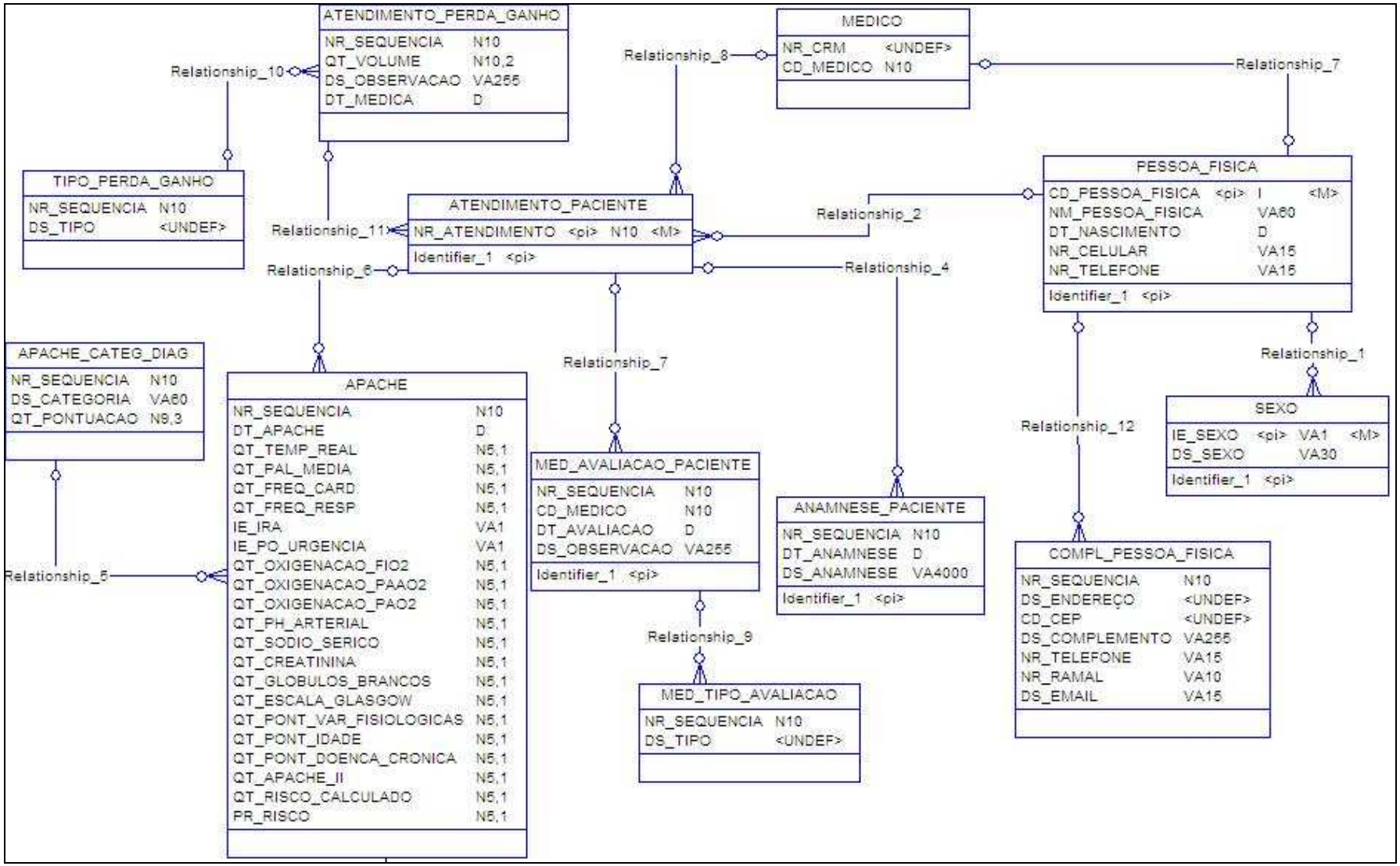


Figura 14 – Modelagem do PEP

O segundo passo é a criação de uma classe de ação quadro 6 , que irá possuir uma super classe WhebAction(quadro 4), que contém atributos que serão utilizados por todas as classes de ação.

```
package br.com.wheb.tcc.action;

import br.com.wheb.tcc.funcoes.MontaMensagemErro;
import java.util.ArrayList;
import java.util.Collection;

/**
 *
 * @author Luis Coelho
 */
public class WhebAction {
    private ArrayList         filtro            = new ArrayList(),
                              parametrosFiltro  = new ArrayList();
    private String            dsTabela          = "",
                              nomeTabela        = "";

    private Integer           nrSeqVisao        = null;
    private boolean     ehNovoRegistro = false;
    /**
     * Creates a new instance of WhebAction
     */
    public WhebAction(String dsTabela,String nomeTabela,Integer nrSeqVisao){
        this.dsTabela   = dsTabela;
        this.nomeTabela = nomeTabela;
        this.nrSeqVisao = nrSeqVisao;
    }

    public Integer    getNrSeqVisao(){return this.nrSeqVisao;}
    public ArrayList  getFiltro(){return this.filtro;}
    public void       addRestricaoFiltro(String
restricao){this.filtro.add(restricao);}
    public ArrayList  getParametrosFiltro(){return this.parametrosFiltro;}
    public void       addParametroFiltro(Object parametro){if(parametro !=
null)this.parametrosFiltro.add(parametro);}
    public String     getDsTabela(){return this.dsTabela;}
    public String     getNomeTabela(){return this.nomeTabela;}

    public void setEhNovoRegistro(){this.ehNovoRegistro = true;}
    public boolean getEhNovoRegistro(){return this.ehNovoRegistro;}
}
```

Quadro 4 – SuperClasse WhebAction

O quadro 5 representa a classe de ação da tabela APACHE a qual contém as regras de negócio, uma segunda classe (quadro 7) é necessária para realizar a integração entre as regras de negócio o gerador e as páginas web.

```
package br.com.wheb.pep.same.apacheII.apacheII;
import br.com.wheb.tcc.action.WhebAction;
/**
 * @author  Luis coelho
 */
public class ApacheIIAction extends WhebAction{
    public ApacheIIAction(Integer nrAtendimento) {
        super("Apache", "APACHE", null);
        addRestricaoFiltro("AND nr_atendimento = :nr_atendimento");
        addParametroFiltro(nrAtendimento);
    }
    public void setPk(Object nrSequencia){
        addRestricaoFiltro("AND nr_sequencia = :nr_sequencia");
        addParametroFiltro(nrSequencia);
    }
}
```

Quadro 5 – Classe de ação para a tabela APACHE

```
public class ApacheIIwebAction extends SystemAction{
    //.... atributos e construtor
    public String montarApachesII(){
        try{
       //Pega da sessão um objeto
        AtendimentoPacienteVO atendimentoPaciente = (AtendimentoPacienteVO)
       getSessao().getAttribute("atendimentoPaciente");
       //Instancia da tabela APACHE
                apacheIIAction = new
ApacheIIAction(atendimentoPaciente.getNrAtendimento());
            //Identifica se é modo Grid ou Detalhe
            if(ehLayer_1()){
                // Adiciona a sequencia do registro seleciona pelo usuário
através do grid
                apacheIIAction.setPk(nrSeqApacheII);
       // Instancia do Gerador
                geraDetalhe = new GeraDetalhe(apacheIIAction);
                // Geração da tela em modo detalhe
                setRetorno(geraDetalhe.geraDetalhe());
            }else{
       // Instancia do Gerador
                geraGrid = new GeraGrid(apacheIIAction);
       // Geração da tela em modo grid
                setRetorno(geraGrid.gerarGrid());
            }
         }catch(Exception e){ setErros(e,this); return ERROR;}
  }//.. set/gets dos atributos
}
```

Quadro 6 – Integração entre regra de negócio e gerador

No quadro 6 pode-se verificar duas linhas em destaque onde é feita a instância do gerador que é dividido em duas opções: a geração de um *grid* com todos os registros da tabela ou a geração de uma página com o detalhe correspondente a um registro.

Na declaração da classe pode-se verificar que a mesma possui uma superclasse chamada *SystemAction*, a qual possui métodos e atributos para a comunicação com o *framework WebWork.* A mapeamento da página gerada com a classe é feito através do arquivo *Xwork.xml*, como apresentado no quadro 7.

```
<!DOCTYPE xwork PUBLIC "-//OpenSymphony Group//XWork 1.0//EN"
"http://www.opensymphony.com/xwork/xwork-1.0.dtd">
<xwork>
      <!-- Importa as configurações default do webWork -->
    <include file="webwork-default.xml"/>

    <package name="default" extends="webwork-default">
     <!--ApacheII - 123-->
       <action name="montarApache"
       class="br.com.wheb.pep.same.apacheII.apacheII.ApacheIIwebAction"
       method="montarApachesII">
       <result name="success" type="velocity">grid.vm</result>
       </action>
</package>
</xwork>
```

Quadro 7 – Mapeamento no Xwork.xml

Através do mapeamento o *WebWork* faz a ligação entre a ação **montarApache** com o método **montarApachesII** da classe **ApacheIIWebAction**. Se o resultado da classe for ***success*** o mapeamento faz a ligação entre a classe e o *template* grid.vm desenvolvido na linguagem *Velocity*. O quadro 8 apresenta o *template.*

```
<html>
      <body>
      $!req.getSession(true).getAttribute('pág
ina_1')
      </body>
</html>
```

Quadro 8 – Template *Velocity*

Pode-se ver que o *template*, é uma página HTML com a linha de comando na linguagem *Velocity* que irá pegar a conteúdo gerado pelo gerador e acoplar ao *template* dinamicamente. As figuras 15 e 16 mostram respectivamente as páginas geradas pela instância do **GeraGrid** e **GeraDetalhe.**

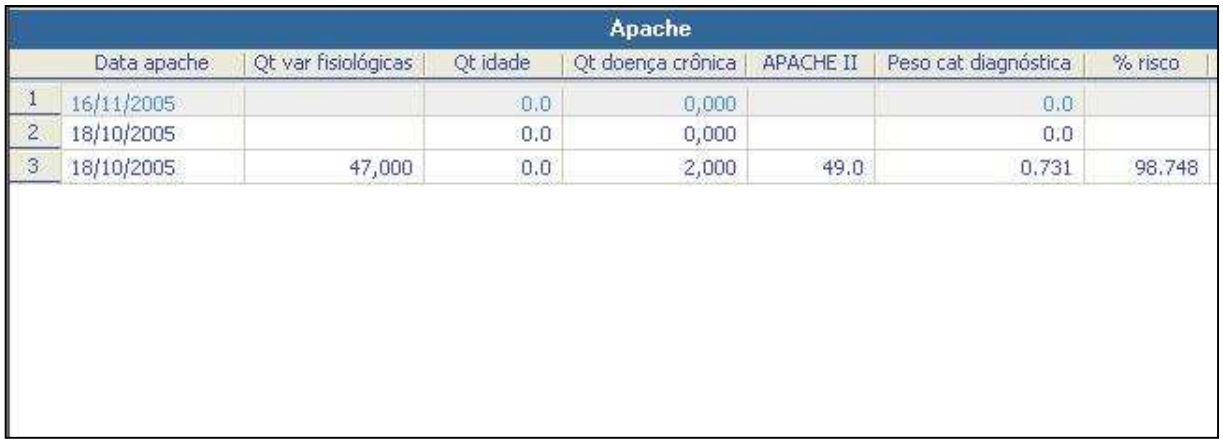


Figura 15 – GeraGrid

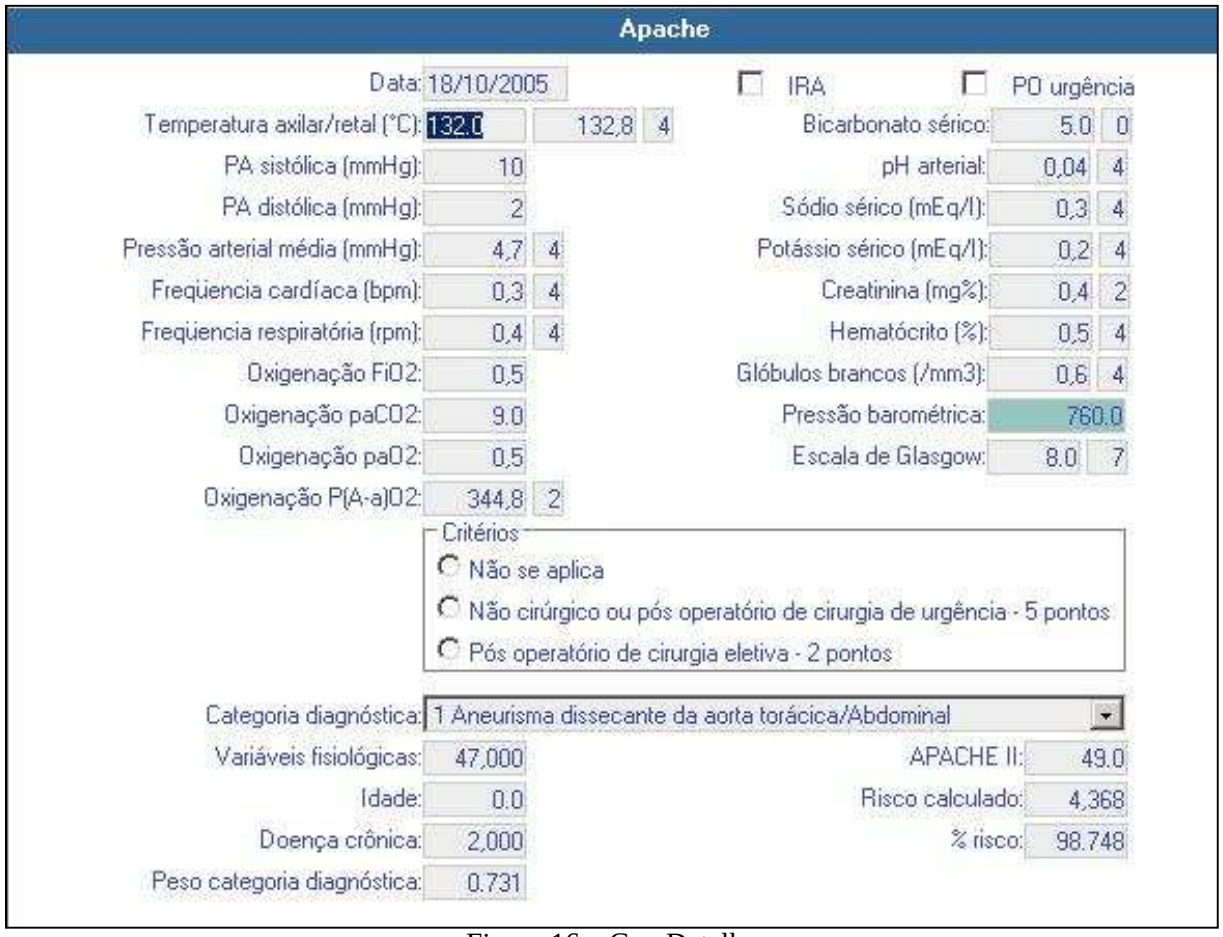


Figura 16 – GeraDetalhe

5 CONCLUSÕES

Foi apresentado um gerador de código HTML, scritps SQL e validador de dados de formulários HTML, baseado na modelagem de dados armazenada no dicionário de dados, juntamente com um módulo de administração do dicionário de dados que auxilia a programadores e desenvolvedor na construção de sistemas para a web. O módulo de administração permite o cadastro das tabelas e os campos das tabelas, bem como o cadastro de atributos referentes as características que um campo terá na visualização, para que a mesma possa ser gerada. Com o gerador é possível a geração das mais diversas páginas bem como a atualização, inserção e exclusão dos registros de forma automática.

O grande diferencial do trabalho deve-se ao fato que todas as páginas do sistema serão geradas em tempo de execução. Não sendo necessário uma prévia geração das páginas e fazendo com que se for necessário incluir um campo em uma página, tenha-se que gerar novamente as páginas. Como são todas geradas dinamicamente para a inclusão de um campo basta acessar o modulo de administração e localizar a tabela que corresponde à página que se deseja alterar e incluir uma nova coluna na tabela, bem como cadastrar os atributos no dicionário de dados, que na próxima requisição da página o campo será incluído automaticamente.

A *IDE* Netbeans, foi utilizada e mostrou eficiente e facilitou os passos de depuração de código no desenvolvimento do gerador. Já que possui integrado o servidor de aplicação TomCat foi possível a execução passo a passo do código a ser gerador mesmo sendo para web, o que normalmente é uma dificuldade encontrada no desenvolvimento de sistemas web.

Com relação aos trabalhos apresentados, acredita-se que o mesmo apresentou um diferencial importante, pois possibilita a geração de código dinamicamente e além de poder ser utilizando por diversas tecnologias Java para a web, como *JSP*, *Servlet, Velocity* entre

outras.

Foi necessária a criação de 39 classes contendo 4320 linhas para a criação do gerador, 17 classes e 1000 linhas para a aplicação de Administração do dicionário de dados e 31 classes e 1470 para a aplicação de exemplo PEP. Totalizando 87 classes e 6790 linhas em 480 horas de programação.

Fica como sugestão para próximos trabalhos a criação de classes que permita a criação das integridades interface gráfica bem como a geração automática das chaves primárias de uma tabela, quando a mesma possuir uma chave primária composta.

REFERÊNCIAS BIBLIOGRÁFICAS

APACHE SOFTWARE FOUNDATION. ***Velocity***. [S.l.], 2005. Disponível em: <http://jakarta.apache.org/velocity/index.HTML>. Acesso em: 12 out. 2005.

CASTILHOS, Cristiano. **Ferramenta CASE para geração de páginas ASP.** 2004. 74 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

FABULOUS FORCE DATABASE TOOLS. **DB Designer 4**. [S.l.], 2003. Disponível em: <http://www.fabforce.net>. Acesso em: 08 out. 2005.

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

LOZANO, Fernando. Desenvolvendo aplicações web com MVC**. Java Magazine,** São Paulo, v. 2, n. 20, p. 34-39, fev. 2004.

MACIEL, Cristiane P. **ASPSYS:** uma ferramenta de apoio ao desenvolvimento de sistemas para web com uso de banco de dados PostgreSQL. 2004. 138 f. Trabalho final do Curso (Especialização em Desenvolvimento para web) – Departamento de Informática, Universidade Estadual de Maringá, Maringá.

MENIN, Juliane. **Gerador de código JSP baseado em projeto de banco de dados MySQL.** 2005. 71 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

MOTA, Samuel. **Guia do *WebWork*2**: aprenda a tirar proveito desse framework MVC. [S.l.], 2004. Disponível em: <http://www.guj.com.br/Java.tutorial.artigo.1351.1.guj>. Acesso em: 12 set. 2005.

Rede Nacional de Ensino e Pesquisa - RNP. **A origem da internet.** Rio de Janeiro, 2005. Disponível em: <http://www.rnp.br/noticias/imprensa/2002/not-imp-020219.html>. Acesso em: 07 out. 2005.

Reyes, A.; Amaro, B.; Patrício, M.; **Manual de usuário para levantamentos com correção diferencial com pós-processamento**. São Paulo, 2000. Disponível em: <http://gps.ciagri.usp.br/apostila.pdf>. Acesso em: 25 abr. 2006.

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. 2005.

SHIMABUKURO JUNIOR, Edison Kicho. **Desenvolvimento de geradores de aplicações configuráveis por linguagens de padrões.** São Paulo, 2005. Disponível em: <http://www-di.inf.puc-rio.br/~julio/anais/EJunior.pdf>. Acesso em: 15 set. 2005.

SILVEIRA, Claudionor. **Geração automática de cadastros e consultas para linguagem ASP baseado em banco de dados**. 2003. 77 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

SOMERJ, **Socidade médica do estado do Rio de Janeiro.** Rio de Janeiro, 2005. Disponível em <http://www.somerj.com.br/revista/200502/2005_02_artigocientifico.htm>. Acesso em: 13 maio 2006.

STEIL, Rafael. ***Velocity template* language**: aprenda a usar essa linguagem de *templates* para o *Velocity*. [S.l.], 2002. Disponível em: <http://www.guj.com.br/Java.tutorial.artigo.26.1.guj>. Acesso em: 15 set. 2005.