UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

DESENVOLVENDO FRAMEWORK WEB BASEADO EM TAGLIBS PARA GERENCIAMENTO DE CONTEÚDO

DANIEL NASCHENWENG

BLUMENAU
2006

2006/1-05

DANIEL NASCHENWENG

DESENVOLVENDO FRAMEWORK WEB BASEADO EM TAGLIBS PARA GERENCIAMENTO DE CONTEÚDO

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 – Orientador

BLUMENAU
2006

2006/1-05

DESENVOLVENDO FRAMEWORK WEB BASEADO EM TAGLIBS PARA GERENCIAMENTO DE CONTEÚDO

Por

DANIEL NASCHENWENG

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, FURB
Membro:	Prof. Mauricio Capobianco Lopes, Mestre, FURB
Membro:	Prof. Ricardo Alencar de Azambuja, Mestre, FURB

Blumenau, 07 de junho de 2006

Dedico este trabalho a minha esposa Mariane que sempre esteve ao meu lado durante este trabalho. O nosso amor me impulsiona sempre a novas realizações.

AGRADECIMENTOS

À Deus por sempre iluminar os meus passos.

À Totall.com S.A. pelos dias que me dispensou para que pudesse realizar este trabalho.

Ao meu orientador, Alexander Roberto Valdameri, pelo apoio dados para que pudesse seguir em frente.

Supera-te e segue adiante.

Ivan de Albuquerque

RESUMO

Este trabalho apresenta o desenvolvimento de um *framework* baseado em bibliotecas de *tags* que visa facilitar a futura construção de web *sites* de conteúdo. Ele apresenta a especificação e implementação deste *framework*, bem como de um gerenciador de conteúdos. Este *framework* foi projetado para ser utilizados diretamente por web *designers*, que podem definir a estrutura do conteúdo, bem como a diagramação visual do *site*, sem a necessidade de conhecimento de programação.

Palavras-chave: Monografia. Framework. Taglibs. Gerenciamento. Conteúdo.

ABSTRACT

This monograph is development of a framework based on libary of tags and means to help on the construction of content web sites. It shows the especification and implemantation this framework, as well a content manager. This framework was projected to be used directly by web desingners. That allows to definy the content structure, as well the visual diagram of the site, whitout the needding knowledge of computer programming.

Key-words: Monograph. Framework. Taglibs. Manager. Content.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

CMS – *Content Management Systems*

DAO – *Data Access Object*

HTML – *HiperText Markup Language*

J2EE – Java 2 *Enterprise Edition*

JSP – JavaServer *Pages*

JSTL – JavaServer *Pages Standard Tag Library*

MVC – Modelo-Visão-Controlador

PHP – PHP: *Hipertext Preprocessor*

SQL – *Structured Query Language*

TLD – *Tag Library Descriptor*

XML – *Extensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Este capítulo apresenta a contextualização, os objetivos, a motivação e a estrutura do trabalho desenvolvido.

1.1 CONTEXTUALIZAÇÃO

Atualmente, para o desenvolvimento de web *sites* são necessários, no mínimo, dois tipos de profissionais definidos pelo Ministério do Trabalho e Emprego (2002) com a designação de:

a) desenhista de páginas da internet ou web *designer*;

b) programador de Internet.

Os web *designers* geralmente estão envolvidos com a criação de *layouts* para novos *sites* e reformulação de *sites* antigos. Estão constantemente preocupados com o uso de imagens e cores, com o posicionamento de objetos na tela, navegação e disposição do conteúdo necessário a cada página e principalmente com tamanho e peso das imagens para *web* (ABRAWEB, 2004).

O programador de internet e o web *designer* necessitam trabalhar de forma conjunta, pois dependem um do outro para exercer sua função.

A plataforma J2EE, edição do Java com extensões voltadas ao desenvolvimento corporativo (FIELDS; KOLB, 2000, p. 11), buscou simplificar a conexão entre estes dois profissionais, disponibilizando a tecnologia JavaServer Pages (JSP) que permite encapsular funcionalidades em *tags* personalizadas que são extensões da linguagem JSP. As *tags* personalizadas são normalmente distribuídas em forma de bibliotecas de *tags* ou *taglibs*. As bibliotecas de *tags* JSP são criadas por programadores Java e são usadas por web *designers*

que podem se preocupar exclusivamente com questões de apresentação (BODOFF, 2002, p. 254).

O uso de *taglibs* estimula a divisão de trabalho entre os programadores e os usuários da biblioteca que podem trabalhar de forma mais independente. Elas também aumentam a produtividade, encapsulando tarefas repetitivas, para serem utilizadas em mais de uma aplicação (BODOFF, 2002, p. 254).

Frameworks como Struts (STRUTS, 2005), Jakarta Taglibs (JAKARTA TAGLIBS, 2005) e o Makumba (MAKUMBA, 2005), trazem a proposta de separar os papéis do programador e do web *designer*, auxiliando no processo de criação do *site*. Porém ainda é exigido do web *designer* uma série de conceitos que não são próprios da área de criação.

Com base no exposto acima, este trabalho apresenta a implementação de um *framework* web, que permita ao web *designer*, o desenvolvimento de *sites* dinâmicos de conteúdos, de forma que o mesmo tenha liberdade de exercer sua função sem se tornar dependente do programador. O *framework* web é construído através de uma biblioteca de *tags* definidas para o uso diretamente pelo web *designer*. A estrutura do conteúdo do *site* também é definida pelo web *designer* de acordo com sua necessidade, através de um gerenciador de conteúdos, garantindo ao web *designer* uma flexibilidade maior para a criação do *site*.

Estes são alguns exemplos de *tags* que são disponibilizadas por este *framework* para o desenvolvimento do *site* pelo web *designer*:

a) *tags* de menu que permitem a navegabilidade no web *site*;
b) *tags* de conteúdo responsáveis pela listagem e exibição de conteúdos cadastrados no gerenciador de conteúdos, tais como textos, imagens e documentos;
c) *tags* de pesquisa que facilitam a localização de conteúdos.

1.2 OBJETIVOS DO TRABALHO

O objetivo deste trabalho consiste em implementar um *framework* web visando facilitar a futura construção de web *sites* de conteúdo com estrutura de dados e diagramação visual flexíveis.

Os objetivos específicos do trabalho são:

a) desenvolver uma biblioteca de *tags* para a construção de *sites* que obedeça ao padrão de projeto Modelo, Visualização e Controle (MVC);

b) produzir uma ferramenta para gerenciamento de conteúdos.

1.3 MOTIVAÇÃO

A dificuldade de identificar um *framework* voltado para o web *designer* originou o desenvolvimento deste trabalho. Os *frameworks* disponíveis no mercado exigem do profissional, conhecimento de programação para sua utilização, o que dificulta o processo de desenvolvimento atual.

Este trabalho tem por objetivo, definir e implementar um *framework* web que permita ao web *designer* o desenvolvimento de *sites* dinâmicos de conteúdos, de forma que o mesmo tenha liberdade de exercer sua função sem se tornar dependente do programador.

1.4 ORGANIZAÇÃO DO TRABALHO

No segundo capítulo é feita uma revisão bibliográfica para o entendimento do trabalho incluindo temas como desenvolvimento baseado em *tags*, tratadores de *tags*, *framework* modelo-visão-controlador, gerência de conteúdos e trabalhos correlatos.

O terceiro capítulo apresenta o processo de desenvolvimento do *framework*, exibindo seus principais requisitos, a especificação, implementação, operacionalidade e resultados alcançados.

O quarto capítulo apresenta as conclusões e sugestões de extensão deste trabalho.

2 FUNDAMENTAÇÃO BIBLIOGRÁFICA

Neste capítulo são apresentados alguns aspectos teóricos relacionados ao trabalho, tais como: desenvolvimento web baseado em bibliotecas de *tags*, *framework* Modelo-Visão-Controlador (MVC), gerenciador de conteúdos e trabalhos correlatos.

2.1 DESENVOLVIMENTO WEB BASEADO EM BIBLIOTECAS DE *TAGS*

O desenvolvimento web, hoje em dia, exige do web *designer* conhecimentos que não são próprios de sua área de atuação. O uso de bibliotecas de *tags* personalizadas pode simplificar este processo fazendo com que se escreva as páginas da interface sem utilizar código de programação. A utilização destas bibliotecas está se tornando cada vez mais difundida no desenvolvimento de páginas JSP. Com o uso delas é possível alterar o *layout* da página sem modificar a lógica e alterar esta última sem influir no *layout* (LEME, 2004, p. 37).

As *tags* personalizadas são extensões da linguagem JSP que simplificam o desenvolvimento e manutenção da página JSP. Elas são normalmente distribuídas na forma de bibliotecas de *tags* ou *taglibs* (BODOFF, 2002, p. 254).

> As bibliotecas de *tags* estendem mais ainda a filosofia das JSPs, para que seja possível a você escrever a maioria de suas páginas da Web sem usar código Java. [...] Os programadores Java estão acostumados a desenvolver a lógica de negócios em *tags* personalizadas, enquanto os desenvolvedores de HTML/JSP podem focalizar o desenvolvimento da lógica de apresentação e da aparência e comportamento das páginas web. (BOND, 2003, p. 552)

2.1.1 ELEMENTOS DE UMA TAG PERSONALIZADA

Tags personalizadas podem possuir até três partes (BOND, 2003, p. 555):

a) a *tag* inicial, como <demo:hello>;

b) a *tag* final, como </demo:hello>;

c) o corpo da *tag* que é todo o conteúdo entre a *tag* inicial e a final, podendo conter elementos de *script*, textos HTML e até mesmo outras *tags*.

Quando uma *tag* não possue corpo ou atributos é denominada *tag* simples, neste caso possui o seguinte formato: <demo:hello />.

Uma *tag* personalizada pode possuir atributos que ficam localizados na *tag* de início, e tem a sintaxe atributo="valor", sendo que o valor pode ser uma constante *String* ou uma variável alterada em tempo de execução. Os atributos de uma *tag* são semelhantes aos parâmetros de uma função, servindo para personalizar o seu comportamento.

Outro poderoso recurso é a cooperação entre *tags*, realizada através de objetos compartilhados. Estes objetos podem ser passados via parâmetro conforme o quadro 1 onde a tag1 cria o obj1 que posteriormente é reutilizado pela tag2.

```
<tt:tag1 attr1="obj1" value1="value"/>
<tt:tag2 attr1="obj1"/>
```

Fonte: Bodoff (2002, p. 263).

Quadro 1 – Cooperação entre *tags* via atributos

No quadro 2 apresenta-se um exemplo com duas *tags* aninhadas. Neste caso os objetos criados pela *tag* outerTag poderão ser reutilizados pelas *tags* internas sem que seja necessário nomear os objetos. Isto torna menor o risco de conflitos entre os nomes dos objetos (BODOFF, 2002, p. 261-262).

```
<tt:outerTag>
<tt:innerTag />
</tt:outerTag>
```

Fonte: Bodoff (2002, p. 263).

Quadro 2 – Cooperação entre *tags* aninhadas

O desenvolvimento baseado em *tags* é um recurso muito poderoso, sendo que as *tags* personalizadas JSP podem realizar diversas ações, tais como: inserir textos na página; implementar fluxo de controle; possuir atributos; ter um corpo (conteúdo entre as *tags* inicial e final) e interagir umas com as outras, permitindo inclusive a criação de hierarquias de *tags* (FIELDS; KOLB, 2000, p. 406).

2.1.2 DESCRITORES DE BIBLIOTECAS DE TAGS

Para ser executada uma *tag* personalizada necessita de um descritor que mapeie o nome da *tag* usada na página JSP e a classe Java que implementa a *tag*. Este descritor é denominado *Tag Library Descriptor* (TLD), sendo escrito em *Extensible Markup Language* (XML). No quadro 3 é apresentado um exemplo de TLD para a *tag* "*Hello World*" (BOND, 2003, p. 552-555).

```
<?XML version="1.0" encoding="ISSO-8859-1" ?>
<!DOCTYPE taglib PUBLIC
      "-//SUN Microsystems, Inc. //DTD JSP Tag Library 1.1//EN"
      "http://java.sun.com/dtd/web-jsptaglibrary_1_1.dtd">
<taglib>
   <tlib-version>1.0</tlib-version>
   <jsp-version>1.1</jsp-version>
   <short-name>mut</short-name>
   <info>Utility Tags for JSP.</info>
   <uri>http://www.taglib.com/wdjsp/tlds/mut_1_0.tld</uri>
   <tag>
     <name>forProperty</name>
     <tag-class>com.taglib.wdjsp.mut.ForPropertyTag</tag-class>
     <tei-class>com.taglib.wdjsp.mut.ForPropertyTEI</tei-class>
     <body-content>JSP</body-content>
     <attribute>
        <name>attr1</name>
        <required>true</required>
        <rtexprvalue>true</rtexprvalue>
        <type>java.lang.String</type>
     </attribute>
   </tag>
</taglib>
```

Fonte: adaptado de Fields e Kolb (2000, p. 410-413).

Quadro 3 – Exemplo de arquivo TLD

O arquivo TLD começa sempre com um prólogo de documento XML que define a versão do XML e a versão de uma especificação a qual o TLD obedece. Isto está representado pelas quatro primeiras linhas do figura 3. Estas linhas sempre devem ser incluídas no início do arquivo, sendo utilizadas por ferramentas que processam o documento para validar o XML (BOND, 2003, p. 552-555).

A definição da biblioteca de *tags* é feita pelo elemento <taglib>, sendo que no quadro 4 é especificado cada um de seus sub-elementos e sua respectiva função.

Elemento	Descrição
tlib-*version*	A versão da biblioteca de *tags*
jsp-*version*	A versão da especificação JSP que a biblioteca de *tags* requer
short-name	Nome opcional que poderá ser usado por uma ferramenta de criação de páginas JSP para criar nomes com um valor mnenônico
uri	Um URI que identifica exclusivamente a biblioteca de *tags*
info	Informações opcionais específicas para a *tag*
tag	Consulte Elemento *tag* mais adiante

Fonte: adaptado de Bodoff (2002, p. 265).

Quadro 4 – Subelementos *taglib*

O elemento <tag> descreve cada *tag* personalizada da biblioteca, definindo seu nome, a classe que fará o seu tratamento e os seus atributos. Os subelementos do elemento <tag> e suas descrições são exibidos no quadro 5.

Elemento	Descrição
name	O nome exclusivo da *tag*
tag-class	O nome totalmente qualificado da classe do tratador de *tags*
tei-*class*	Especifica o nome de uma classe auxiliar que pode ser utilizada no processamento da Tag. Subclasse de javax.servlet.jsp.tagext.TagExtraInfo.
body-context	Tipo de conteúdo do corpo. Os valores possíveis para o elemento <body-context> são: empty (corpo de *tag* vazio), JSP (a *tag* contém dados JSP) e tagdependent (o texto entre a *tag* inicial e final será tratado pela classe Java)
info	Informações opcionais específicas para a *tag*
variable	Informação opcional de variável de script.
attribute	Informação do atributo da *tag*

Fonte: adaptado de Bodoff (2002, p. 265).

Quadro 5 – Subelementos *taglib*

Cada elemento <tag> pode ter vários atributos que são definidos pelo elemento <attribute>. Seus subelementos são descritos no quadro 6. (BODOFF, 2002, p. 265-268).

Elemento	Descrição
name	O nome exclusivo do atributo
required	Especifica se o atributo é obrigatório. Seus valores podem ser: *true*, *false*, *yes* ou *no*.
rtexprvalue	Define se o valor pode ser definido por uma expresão. Seus valores podem ser: *true*, *false*, *yes* ou *no*.
type	Tipo de retorno esperado de qualquer expressão especificada.

Fonte: adaptado de Bodoff (2002, p. 268).

Quadro 6 – Subelementos *attribute*

2.2 TRATADORES DE *TAGS*

Quando o *container* web executa uma página JSP que possui uma *tag* personalizada JSP, são executadas operações por um objeto denominado de tratador de *tags* (BODOFF,

2002, p. 255). Este tratador de *tags* deve implementar a interface Tag ou BodyTag que estão contidas no pacote java.servlet.jsp.tagext do Java 2 *Enterprise Edition* (J2EE). Um conjunto de classes do tratador de *tags* é geralmente empacotado em um arquivo JAR (BODOFF, 2002, p. 263-264).

2.2.1 TRATADORES PARA *TAGS* SIMPLES

O tratador de *tag* simples deve implementar os métodos doStartTag e doEndTag da interface Tag. O quadro 7 exemplifica uma *tag* simples e seu tratador. Neste exemplo será impresso "Hello!" quando a *tag* for encontrada no JSP.

```
Tag Simples:
<tt:simple />

Tratator de Tag:
public SimpleTag extends TagSupport {
    public int doStartTag() trows JspException {
      try{
         pageContext.getOut().print("Hello.");
      } catch {
        thow new JspTagException("SimpleTag: "+
           ex.getMessage());
      }
      return SKIP_BODY;
    }
    public int doEndTag() {
        return EVAL_PAGE;
    }
}
```

Fonte: adaptado de Bodoff (2002, p. 267).

Quadro 7 – Exemplo de *tag* simples com seu tratador

O método do StartTag é invocado quando a *tag* de início é localizada. Este método está retornando SKIP_BODY, indicando que o corpo da *tag* deve ser ignorado. Já o método doEndTag é acionado pela *tag* de fim, que neste caso é a mesma de início. Este método pode retornar EVAL_PAGE, se o restante da página deve ser avaliado ou SKIP_PAGE, caso contrário (BODOFF, 2002, p. 266-267).

2.2.2 TRATADORES PARA *TAGS* COM ATRIBUTOS

Para suportar atributos uma *tag* deve seguir o padrão JavaBean, sendo que para cada atributo devem ser criados uma propriedade e seus respectivos métodos *get* e *set*. O método de configuração de cada atributo é chamado antes do método doStartTag(), deixando assim os valores das propriedades disponíveis aos demais métodos do ciclo de vida da *tag*. Assim a *tag* poderá utilizar os atributos passados para determinar o seu comportamento (BOND, 2003, p. 563-564).

2.2.3 TRATADORES PARA *TAGS* COM CORPO

Um tratador de *tags* com corpo é implementado de forma diferente dependendo de sua necessidade de interagir ou não com o corpo, ou seja, se a *tag* lê ou modifica o conteúdo do corpo.

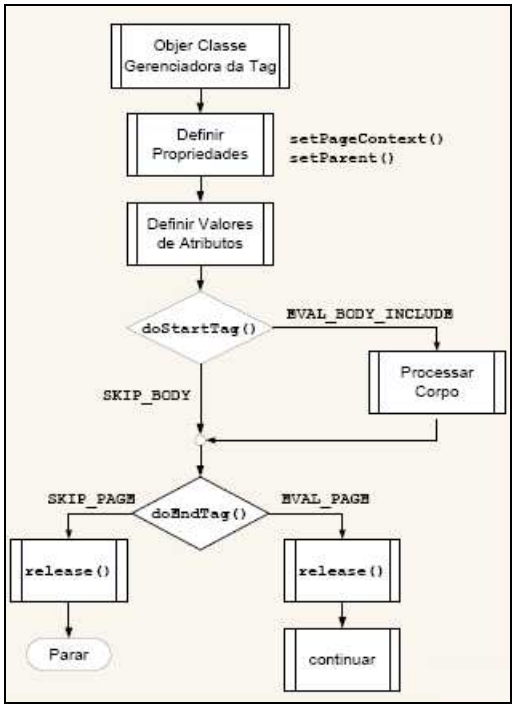
Se o tratador não necessita interagir com o corpo ele deve implementar a interface Tag ou ser derivado da classe TagSuport. Se o corpo precisa ser avaliado a *tag* deve retornar EVAL_BODY_INCLUDE, caso contrário deve retornar SKIP_BODY.

Caso o tratador necessite interagir com o conteúdo, ele deve implementar BodyTag ou ser derivado de BodyTagSuport. Esse tipo de tratador geralmente implementa os métodos doInitBody() e doAfterBody(), com os quais o tratador interage com o conteúdo (BODOFF, 2002, p. 270-271). Se o corpo da *tag* precisa ser avaliado, o método doStartTag retorna EVAL_BODY_BUFFERED, caso contrário, ele deve retornar SKIP_BODY (BOND, 2003, p. 270).

> O método doInitBody é chamado depois que o conteúdo do corpo é definido, mas antes de ele ser avaliado. Você geralmente usa este método para executar alguma inicialização que dependa do conteúdo do corpo. [...]
> O método doAfter Body é chamado depois que o conteúdo do corpo é avaliado. Como o método doStartTag, o doAfterBody deve retornar uma indicação

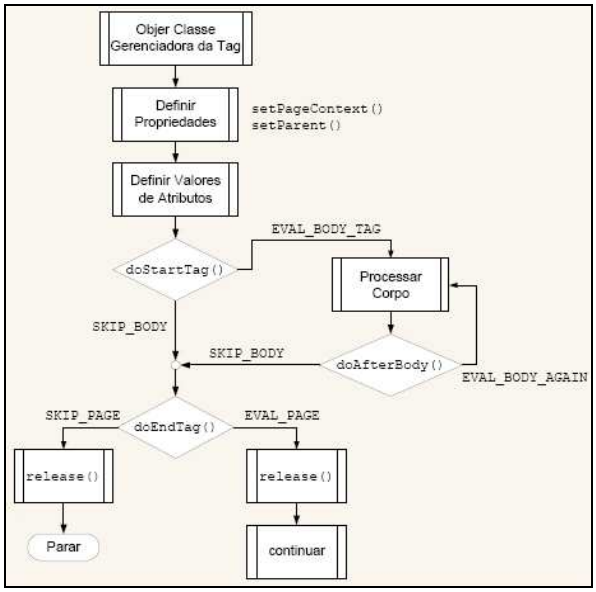
> informando se deve ou não continuar avaliando o corpo. Portanto, se o corpo precisa ser avaliado novamente, como seria o caso se você estivesse implementando uma tag de interação, doAfterBody deveria retornar EVAL_BODY_BUFFERED, caso contrário, doAfterBody deveria retornar SKIP_BODY. (BOND, 2003, p. 270-271).

A figura 1 demonstra o ciclo de vida da interface *Tag*, onde inicialmente a classe tratadora é acessada, após isto são definidas suas propriedades e os valores para seus atributos. Um atributo importante que é instanciado é o pageContext que guarda o contexto da página de onde podem ser recuperadas as requisições e objetos compartilhados entre *tags*. O evento doStartTag() é acionado e processa o corpo da *tag*. Após o processamento o evento doEndTag é acionado (FIELDS; KOLB, 2000, p. 415).



Fonte: adaptado de Fields e Kolb (2000, p. 415).
Figura 1 – Ciclo de vida da interface Tag

O ciclo da interface BodyTag é apresentado na figura 2, onde além dos eventos da interface Tag têm-se também o evento doAfterBody que permite a iteração com o corpo da *tag* (FIELDS; KOLB, 2000, p. 417).

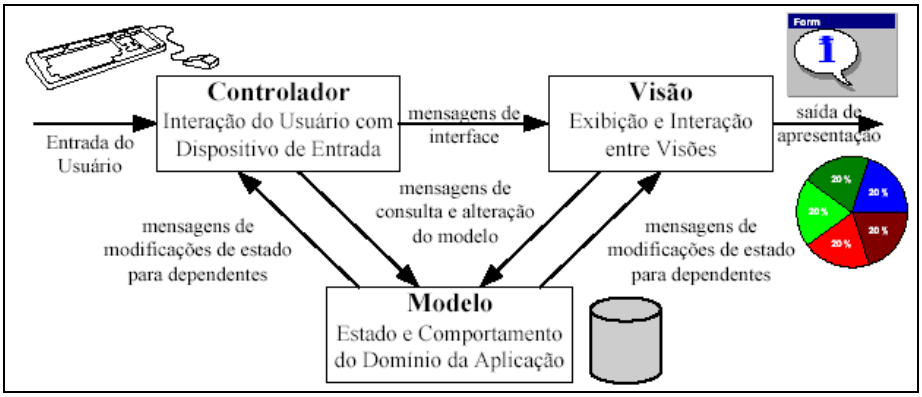


Fonte: adaptado de Fields e Kolb (2000, p. 417).

Figura 2 – Ciclo de vida da interface BodyTag

2.3 *FRAMEWORK* MODELO-VISÃO-CONTROLADOR (MVC)

O *framework* para interfaces gráficas MVC surgiu na linguagem Smalltalk e divide uma aplicação em três partes ou papéis: modelo (dados e classes de negócios), visão (apresentação) e controle (KRASNER; POPE, 1988). A figura 3 mostra a arquitetura inicial do MVC.



Fonte: Gerber (1999, p. 20).

Figura 3 – Cenário de interação do *framework* MVC

A figura 4 mostra as alterações sofridas neste modelo para a utilização em aplicações web. No modelo inicial a camada visual recebe notificações quando acontecem alterações na

camada modelo. Nota-se que todas as camadas comunicam-se entre si. Isto não ocorre na web por causa do desacoplamento entre cliente e servidor (SOUZA, 2004, p. 4). Na web toda a comunicação entre a camada visual e a camada modelo tem de passar pela camada controle, tornando as camadas modelo e visão ainda mais independentes.

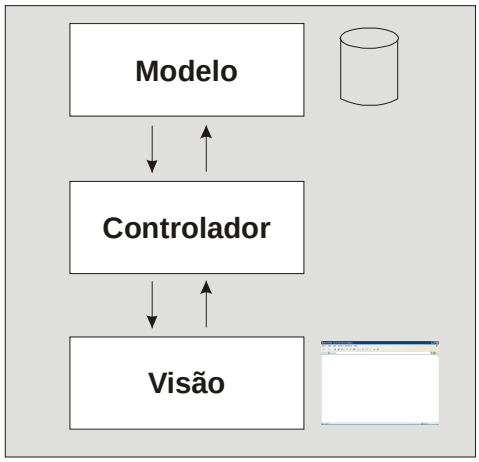


Figura 4 – Cenário de interação do *framework* MVC na web

A grande vantagem do padrão MVC é simplificar a manutenção e evolução de aplicações web, bem como o reuso de seus componentes (FREITAS; GALLINDO; OLIVEIRA, 2003, p. 40).

O JSP implementa parcialmente esta arquitetura desde a versão 0.91, em um padrão que ficou conhecido como Modelo1. Neste padrão os papéis não são bem definidos, tendo dois componentes (JSP e JavaBeans) para desempenhar os três papéis do MVC. O controle é descentralizado e tanto os desenvolvedores como os *designers* precisam editar páginas JSP (FREITAS; GALLINDO; OLIVEIRA, 2003, p. 41).

Para minimizar este problema foi desenvolvido o Modelo2 que adiciona o controlador. Implementado por um *servlet*, o controlador acessa a camada de negócios e coordena a geração do conteúdo (FREITAS; GALLINDO; OLIVEIRA, 2003, p. 41). Um *framework* que implementa este modelo de forma madura e completa é o Struts (STRUTS, 2005).

2.4 GERENCIADOR DE CONTEÚDOS

O Sistema de Gerenciamento de Conteúdos ou *Content Management Systems* (CMS) trata de um sistema onde conteúdos passam por um ciclo de vida finito, onde são criados, gerenciados, publicados e arquivados. O CMS providencia a infraestrutura necessária para que múltiplas pessoas possam contribuir com este conteúdo e colaborar com este ciclo de vida (HANNON HILL CORPORATION, 2006a, p. 3).

No conceito de CMS, o conteúdo representa informações puras ainda não formatadas para a exibição em *sites*. Este conteúdo pode ser, por exemplo, arquivos-texto, imagens, planilhas, entre outros.

O CMS visa facilitar a criação colaborativa de documentos, definindo suas regras e processos. Por exemplo, uma notícia pode ser escrita por várias pessoas, em seguida verificada por um revisor especializado e aprovada por um editor de publicação (VELLOSO, 2005, p. 53).

2.4.1 VANTAGENS DO USO DE UM GERENCIADOR DE CONTEÚDOS

> Gerenciadores de Conteúdo proporcionam uma série de vantagens sobre os métodos tradicionais de editar informações. Muitos dos benefícios são conquistados quando equipes distribuídas de pessoas são responsáveis por coordenar e contribuir com diferentes repositórios de conteúdo (HANNON HILL CORPORATION, 2005, p. 4, tradução nossa).

O quadro 8 mostra uma comparação entre o controle de informação on-line com e sem um CMS.

	Com um CMS	Sem um CMS
Criação de Nova Página	Uma nova página é criada baseada em uma pré-definida.Todos os *links* de navegação são automaticamente atualizados.	Uma nova página é criada com uma cópia de uma existente. O mapa do *site* e os *links* de navegação precisam ser atualizados manualmente.
Consistência de Conteúdo	O molde é separado do conteúdo da página, mantendo a consistência no *site* como um todo. A consistência no visual é forçada pelo CMS.	Conteúdo e o molde ficam amarrados, tornando difícil fazer alterações no *site*. A consistência visual é determinada pelos desenvolvedores.
Processos de Fluxo de Trabalho	Os fluxos de trabalhos são designados de acordo com os processos de negócio. A máquina de fluxo de trabalho do CMS registra uma auditoria descrevendo cada um dos passos. Ao final da aprovação o conteúdo é automaticamente publicado on-line.	O fluxo de trabalho é feito tipicamente via e-mail. E-mails são enviados para diferentes pessoas da organização para a sua aprovação e subseqüente publicação manual no *site*.
Tempo de Publicação	O conteúdo é publicado imediatamente após a aprovação necessária ser feita.	O conteúdo é publicado quando o responsável pelo web *site* tiver tempo disponível. Isto pode demorar dias e incorrer em erros de configuração.
Controle de Alterações	O controle de alterações é garantido pelo CMS com o registro de todas as alterações e publicações de conteúdo.	O controle de alterações deve ser garantido pelos membros da equipe. Alterações precisam ser copiadas manualmente para um registro de alterações.

Fonte: Hannon Hill Corporation (2006a, p. 7, tradução nossa).

Quadro 8 – Controle de informação com e sem CMS

2.5 TRABALHOS CORRELATOS

O *framework* Struts (STRUTS, 2005) foi criado por Craig R. McClanahan e doado à Apache Software Foundation em 2000. Ele visa facilitar aos desenvolvedores a criação de aplicações web baseadas em tecnologias Java *servlets* e JSP. Com a utilização do Struts o desenvolvedor preocupa-se mais com as regras de negócios do que com a infraestrutura do *site* (CAVANESS, 2003, p. 1). Ele provê um vasto conjunto de componentes, entre eles uma biblioteca de *tags* que são responsáveis por uma interação íntima com o resto do *framework.*

As *tags* do *framework* Struts estão agrupadas em (CAVANESS, 2003, p. 198-216):

a) HTML, utilizada para criar interfaces HTML;

b) *bean*, para acessar Java Beans;

c) *logic*, serve para gerar saída de texto de acordo com condições especificadas ou realizar iterações em coleções de objetos;

d) *template*, utilizada para criar padrões dinâmicos para páginas que possuem um mesmo formato.

Um repositório de *tags* muito conhecido é o Jakarta TagLibs (JAKARTA TAGLIBS, 2005) que é um projeto *open source* onde se encontra a implementação de referência (RI) da JavaServer *Pages Standard Tag Library* (JSTL), conjunto padrão de *taglibs* definido pela JSR-52 (LEME, 2004, p. 31). Entre as várias *tags* disponibilizadas estão *tags* de formatação, acesso à banco de dados, manipulação de XML e funções para uso diverso.

O *framework* Makumba (MAKUMBA, 2005) implementado em Java oferece uma *Application Program Interface* (API) e uma biblioteca de *tags* para o desenvolvimento rápido de aplicações web. A estrutura de dados é descrita em um arquivo texto de acordo com uma linguagem específica. O mesmo gera os *scripts Structured Query Language* (SQL) a partir desta definição. Através de *tags* próprias é possível consultar, inserir, alterar e excluir dados, bem como definir regras de negócios. Este projeto é *open source* e é mantido por uma comunidade de voluntários.

O Calandra KBX (CALANDRA, 2006) é uma ferramenta de gerenciamento de conteúdos baseada em *browser*. Suas principais funcionalidades são: edição de conteúdos; busca; definição do conteúdo a ser exibido e sua ordem; banco de arquivos; administração do conteúdo; controle de mensagens; fóruns de discussão; gerador de formulários; gerador de enquetes; personalização de conteúdos.

O produto TeamSite (INTERWOOVEN, 2006) é um gerenciador de conteúdos que permite criar, publicar e integrar conteúdos para a gerência de empresas. Sendo utilizado por indústrias ou empresas para portais de conteúdo internos e externos. Entre as empresas que o

utilizam estão inclusas GE, Siemens, e Cisco Systems.

Em Moratelli (2002) é realizado um estudo sobre sistemas de gerenciamento de conteúdos para ambiente web, visando facilitar o gerenciamento de conteúdo de um *site*. Também apresenta a especificação e implementação deste sistema utilizando a linguagem PHP: *Hipertext Preprocessor* (PHP). Esta implementação utiliza-se de *templates* para separar a identidade visual da programação. Segundo Moratelli (2002, p. 7) “[...] Os *templates* são um conjunto de regras, estilos e formato gráfico de apresentação de um *site*.”

3 DESENVOLVIMENTO

Nesta seção é abordado todo o processo de desenvolvimento do *framework* e do gerenciador de conteúdos, sendo detalhado os seus a especificação, a implementação e os resultados do processo de desenvolvimento.

3.1 ESPECIFICAÇÃO DO *FRAMEWORK*

Para realizar a especificação do *framework* web, utilizou-se a ferramenta Enterprise Architect, através da UML. A especificação das Tags foi feita através de um descritor de *tags* no formato DTD.

A seguir são apresentados os diagramas da implementação do MVC, diagrama de caso de uso, diagramas de classes, diagramas de seqüência e descrição das *tags*.

3.1.1 REQUISITOS PRINCIPAIS DO *FRAMEWORK*

a) fornecer *tags* para que o web *designer* possa criar o *site*, sendo que cada *tag* irá desempenhar uma função específica (RF);
b) possibilitar a construção de menus dinâmicos através de uma *tag* de menu, conforme a disposição desejada pelo web *designer* (RF);
c) disponibilizar *tags* para exibir conteúdos com estruturas diversas e conforme a disposição desejada pelo web *designer* (RF);
d) permitir a localização dos conteúdos cadastrados no web *site* através da *tag* de pesquisa (RF);
e) ser implementado utilizando a plataforma Java 2 Enterprise Edition (J2EE)

(requisito não funcional – RNF);

f) as *tags* serão implementadas utilizando especificação para *tags* personalizadas do JSP (RNF);

g) ser desenvolvido para ambiente web (RNF);

h) garantir compatibilidade com os *browsers* Internet Explorer 5.0 e Mozila 4 ou superiores (RNF).

3.1.2 DIAGRAMA DA IMPLEMENTAÇÃO MVC DO *FRAMEWORK*

O *framework* desenvolvido implementou a arquitetura MVC, conforme é apresentado na a figura 5. Sendo que a camada de modelo é especificada nas classes de persistência DAO, a camada de controle pelas classes tratadoras de *tags* e a camada de visão pelos arquivos JSP que utilizarão as *tags* especificadas.

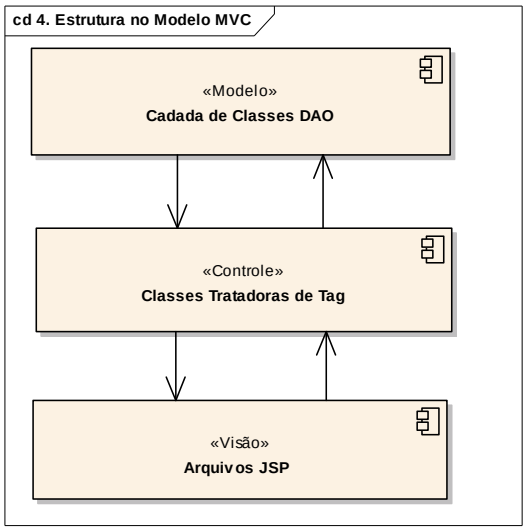


Figura 5 – Implementação da arquitetura MVC pelo *framework*

3.1.3 DIAGRAMA DE CASOS DE USO DO *FRAMEWORK*

A seguir na figura 6 é apresentado o diagramas de caso de uso do *framework*

implementado. Nele pode-se ver a interação de um visitante no web *site* construído com o *framework.*

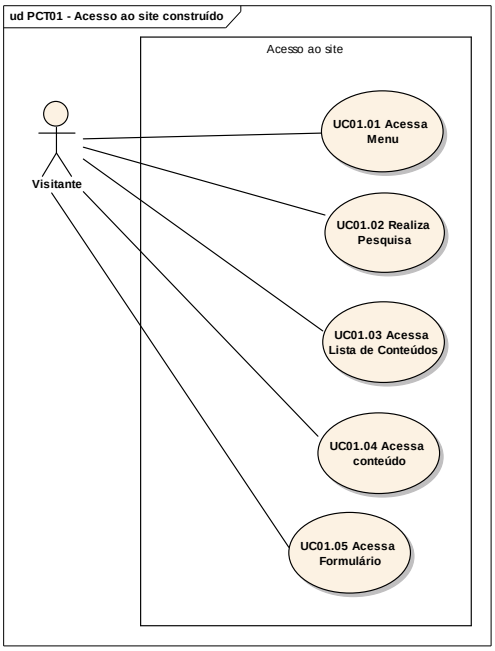


Figura 6 –Diagramas de caso de uso do acesso ao *site*

A seguir são descritos brevemente os casos de uso deste módulo:

a) acessa o menu: permite a visualização do menu e seus sub-menus pelo visitante do portal;

b) realiza pesquisa: permite que o visitante realize pesquisas no conteúdo do web *site* através de palavras chaves;

c) acessa lista de conteúdos: permite que o visitante visualize uma lista de conteúdos de um menu;

d) acessa conteúdo: permite e visualização de um conteúdo específico selecionado pelo visitante do *site*;

e) acessa formulário: permite a manutenção de conteúdo diretamente pelo visitante do *site* através de formulários.

3.1.4 DIAGRAMA DE CLASSES

Os diagramas de classes do *framework* foram divididos em 6 pacotes, conforme demonstrado na figura 7: br.com.lego.negocio; br.com.lego.taglibs, possui as classes que implementam as *tags* do *framework*; br.com.lego.dao; br.com.lego.exception; br.com.lego.servlets e br.com.lego.util.

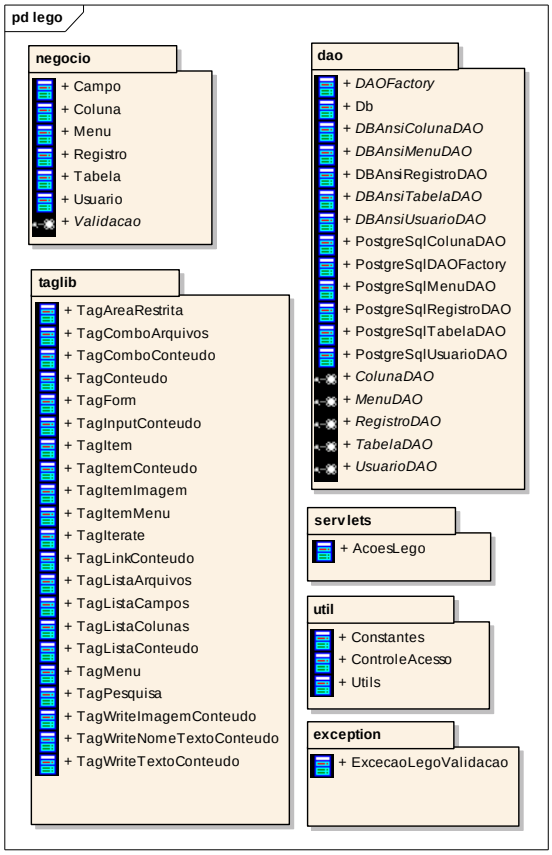


Figura 7 – Pacotes do diagrama de classes

O pacote br.com.lego.negocio que é demonstrado na figura 8 é responsável pelas classes de negócio. Este pacote é composto pelas seguintes classes:

a) classe Menu: representa um menu que dá acesso a um conteúdo no sistema. Esta classe permite que a navegação no portal seja configurável;

b) classe Tabela: representa uma estrutura de conteúdos. Esta classe simboliza uma tabela no banco de dados;

c) classe Coluna: define cada tipo de dado que uma tabela possui. Equivalente a uma coluna na tabela de banco de dados;

d) classe Registro: representa uma linha de conteúdo de uma tabela;

e) classe Campo: representa uma unidade de informação;

f) classe Usuário: representa os dados para *login* de um usuário da área de gerenciamento de conteúdos;

g) interface Validação: esta interface define um método responsável pela validação dos campos de um objeto que necessitam ser preenchidos. Caso algum dos campos esteja vazio ela gera um erro do tipo ExcecaoLegoValidacao.

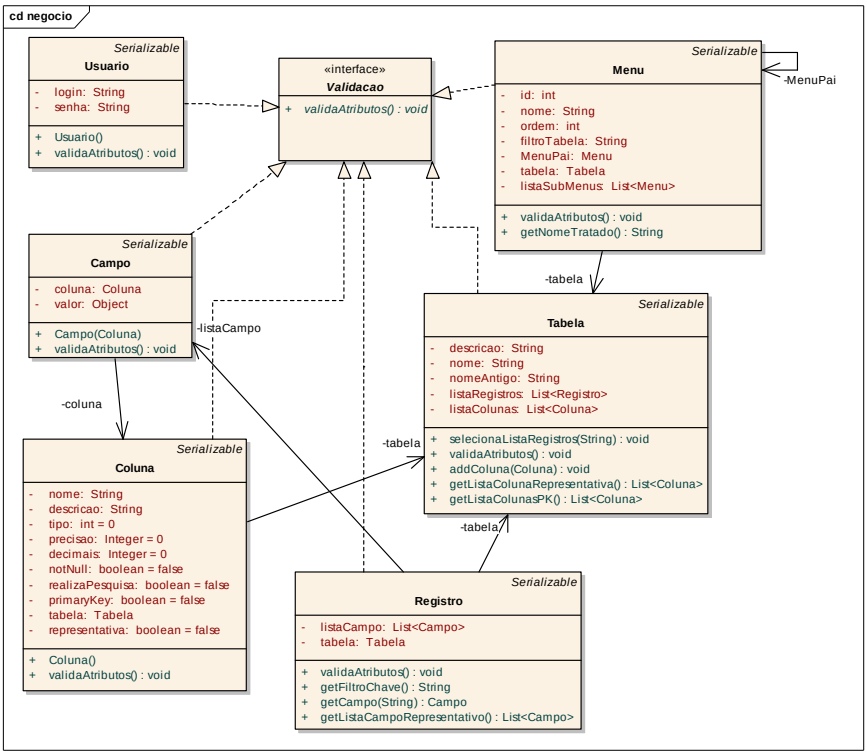


Figura 8 – Diagrama de classes de negócio

A figura 9 apresenta o pacote br.com.lego.taglib que é responsável pelas classes que tratam as *tags* do *framework*. Cada *tag* especificada terá uma classe neste pacote responsável por sua execução.

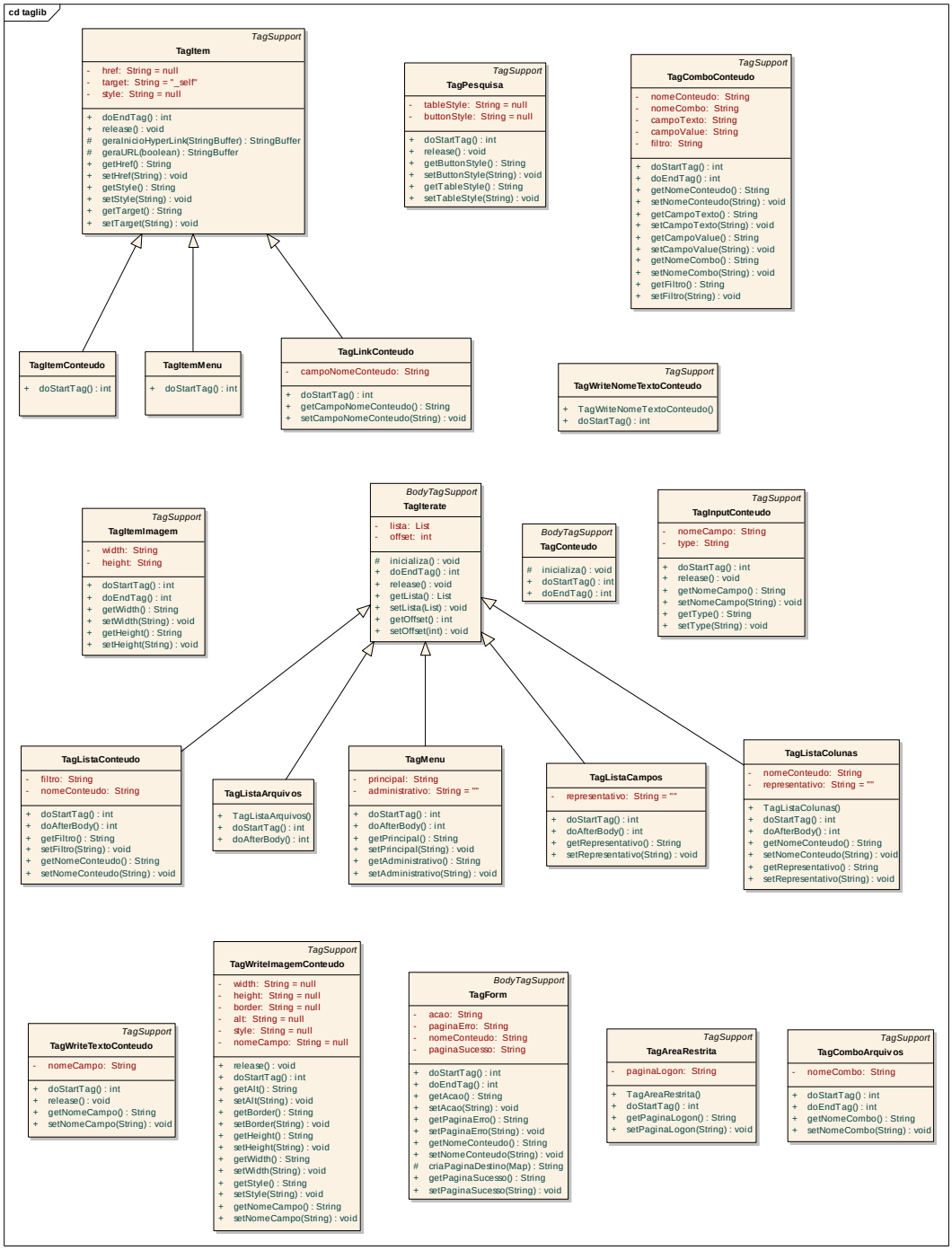


Figura 9 – Diagrama de classes da TagLib

O pacote br.com.lego.taglib é composto pelas seguintes classes:

a) classe TagItem: possui propriedades e métodos comuns a *tags* de itemMenu, itemConteudo e linkConteudo;

b) classe TagItemConteudo: escreve na página um link para um item de conteúdo;

c) classe TagItemMenu: escreve na página um link passando o menu selecionado;

d) classe TagLinkConteudo: cria um link de para um determinado conteúdo;

e) classe TagPesquisa: disponibiliza um formulário para entrada de palavras chaves para pesquisa de um conteúdo;

f) classe TagComboConteudo: cria um *combo* com uma lista de objetos Registro;

g) classe WriteNomeTextoConteudo: escreve na página o nome do campo de um registro na forma de texto;

h) classe TagItemImagem: exibe um campo do tipo imagem de uma determinada lista de imagens;

i) classe TagIterate: possui métodos e propriedades comuns a *tags* que realizam iterações em uma Lista. Ela define uma lista e a posição da lista que está sendo tratada no momento. A cada iteração ela deixa disponível a TagItem o próximo objeto da lista;

j) classe TagListaConteudo: responsável por realizar a iteração em uma lista de conteúdos;

k) classe TagListaArquivos: cria uma iteração com os arquivos disponíveis para o site;

l) classe TagMenu: responsável por realizar a iteração em uma lista de menus;

m) classe TagListaCampos: responsável por criar uma iteração em uma lista com os Campos de um determinado objeto Registro;

n) classe TagListaColunas: responsável por criar uma iteração em uma lista com os objetos Coluna de um objeto Tabela;

o) classe TagConteudo: disponibiliza um objeto Conteudo de acordo com o identificador passado no link da TagItemConteudo;

p) classe TagInputConteudo: escreve na página um campo para entrada de dados através de um formulário. Mostrando o valor do conteúdo disponível no momento;

q) classe WriteTextoConteudo: escreve na página o valor do campo de um registro na forma de texto;

r) classe WriteWriteImagemConteudo: escreve na página o valor do campo de um registro na forma de uma *tag* de imagem HTML;

s) classe TagForm: escreve na página um formulário HTML com ação para alterar um conteúdo;

t) classe TagAreaRestrita: valida a autenticação do usuário redirecionando para a página de *login* caso não esteja autenticado;

u) classe TagComboArquivos: cria um *combo* com os arquivos disponíveis no site.

A figura 10 apresenta o pacote br.com.lego.dao responsável pela persistência das informações no banco de dados. Para realizar a persistência utilizou-se o padrão de projeto *Data Access Object* DAO que visa tornar a camada de negócios mais independente da camada de persistência e do local onde os dados são persistidos.

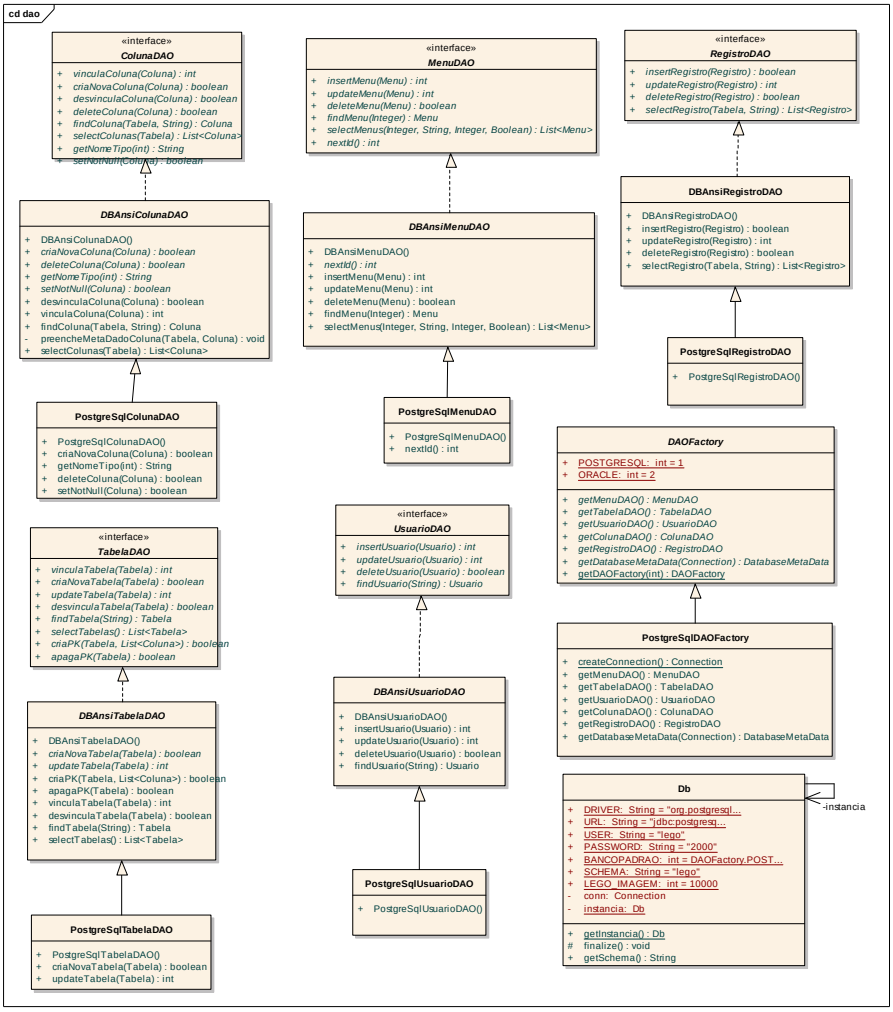


Figura 10 – Diagrama de classes dao

O pacote br.com.lego.dao é composto pelas seguintes classes:

a) interface ColunaDAO: define os métodos para realizar a persistência de objetos da classe Coluna;

b) classe DBAnsiColunaDAO: classe abstrata que implementa os métodos da interface ColunaDAO que são comuns a todos os bancos que atendam a definição ANSI para linguagem SQL;

c) classe PostgreSqlColunaDAO: extende a classe DBAnsiColunaDAO, implementando os métodos da interface ColunaDAO que esta não implementa;

d) interface MenuDAO: define os métodos para realizar a persistência de objetos da classe Menu;

e) classe DBAnsiMenuDAO: classe abstrata que implementa os métodos da interface

MenuDAO que são comuns a todos os bancos que atendam a definição ANSI para linguagem SQL;

f) classe PostgreSqlMenuDAO: extende a classe DBAnsiMenuDAO, implementando os métodos da interface MenuDAO que esta não implementa;

g) interface RegistroDAO: define os métodos para realizar a persistência de objetos da classe Coluna;

h) classe DBAnsiRegistroDAO: classe abstrata que implementa os métodos da interface RegistroDAO que são comuns a todos os bancos que atendam a definição ANSI para linguagem SQL;

i) classe PostgreSqlRegistroDAO: extende a classe DBAnsiRegistroDAO, implementando os métodos da interface RegistroDAO que esta não implementa;

j) interface TabelaDAO: define os métodos para realizar a persistência de objetos da classe Tabela;

k) classe DBAnsiTabelaDAO: classe abstrata que implementa os métodos da interface TabelaDAO que são comuns a todos os bancos que atendam a definição ANSI para linguagem SQL;

l) classe PostgreSqlTabelaDAO: extende a classe DBAnsiTabelaDAO, implementando os métodos da interface TabelaDAO que esta não implementa;

m) interface UsuarioDAO, define os métodos para realizar a persistência de objetos da classe Usuario;

n) classe DBAnsiUsuarioDAO: classe abstrata que implementa os métodos da interface UsuarioDAO que são comuns a todos os bancos que atendam a definição ANSI para linguagem SQL;

o) classe PostgreSqlUsuarioDAO: extende a classe DBAnsiUsuarioDAO, implementando os métodos da interface UsuarioDAO que esta não implementa;

p) classe DAOFactory: fábrica de objetos DAO. É uma classe abstrata devendo ser implementada para cada tipo de persistência;

q) classe PostgreSqlDAOFactory: implementação de uma fábrica DAO para o banco de dados PostgreSQL;

r) classe Db: responsável por disponibilizar uma conexão com o banco de dados.

A figura 11 apresenta o pacote br.com.lego.exception que é responsável pela exceções específicas do *framework* lego. Ele é composto pela classe ExcecaoLegoValidacao que é gerada quando um objeto tem um campo obrigatório não preenchido durante o processo de validação.

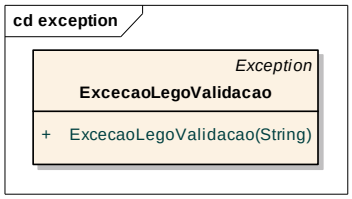


Figura 11 – Diagrama de classes exception

O pacote br.com.lego.servlets apresentado na figura 12 possui a classe AcoesLego que é responsável pelo tratamento das ações geradas por um formulário do *framework* Lego.

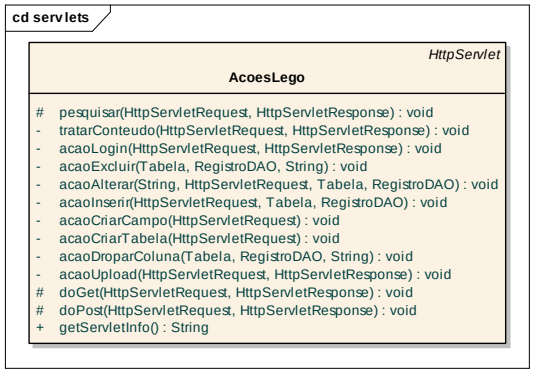


Figura 12 – Diagrama de classes servlets

O pacote br.com.lego.util representado na figura 13 possui classes utilitárias do *framework* Lego. As classes são:

a) classe Utils: possui métodos de formatação e tratamentos de Strings;

b) classe Constante: nela está definida as constantes utilizadas no *framework*;

c) classe ControleAcesso: possui um método que determina se foi iniciada uma

sessão com login para o acesso de uma página na área restrita do *site*.

Figura 13 – Diagrama de classes util

3.1.5 DIAGRAMA DE SEQÜÊNCIA

Nesta seção são especificados, através de diagramas de seqüência, os fluxos de mensagens trocados entre as classes, na execução das *tags* especificadas.

A figura 14 demonstra o diagrama de seqüência no acesso de uma página contendo uma *tag* de menu. Ao ser identificado na página JSP a *tag* inicial de menu é acionado o evento doStartTag(). Este evento inicializa a lista de menus, requisita uma conexão com o banco de dados, requisita ao PageContext o id do menu pai e solicita a FabricaDAOMenu a lista de objetos Menu. Para cada objeto Menu na lista, o evento doAfterBody() disponibiliza o menu atual para o PageContext. Ao final o evento doEndTag() imprime na página o resultado da *tag* através da classe JspWriter.

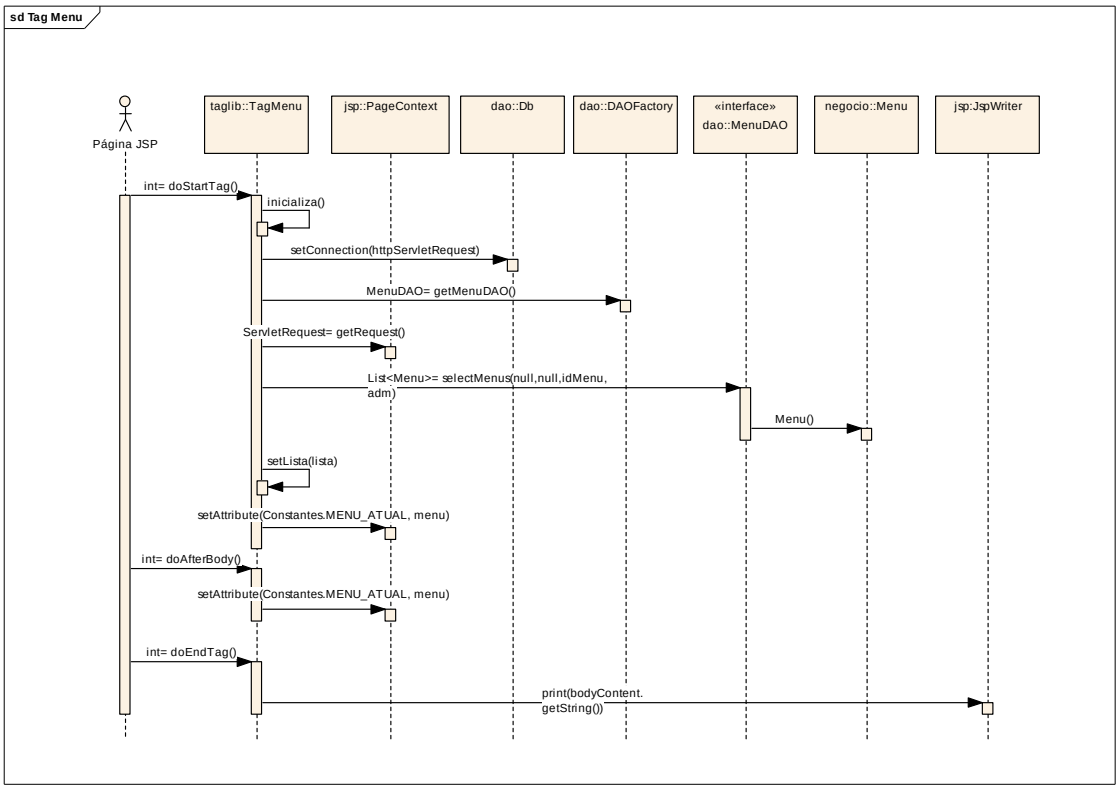


Figura 14 – Diagrama de seqüência TagMenu

O diagrama de seqüência da figura 15 mostra o objeto TagItemMenu, no evento doStartTag(), requisitando do PageContext o menu atual, que foi disponibilizado pela iteração do objeto TagMenu e escrevendo o link de menu na página JSP através do objeto JspWriter.

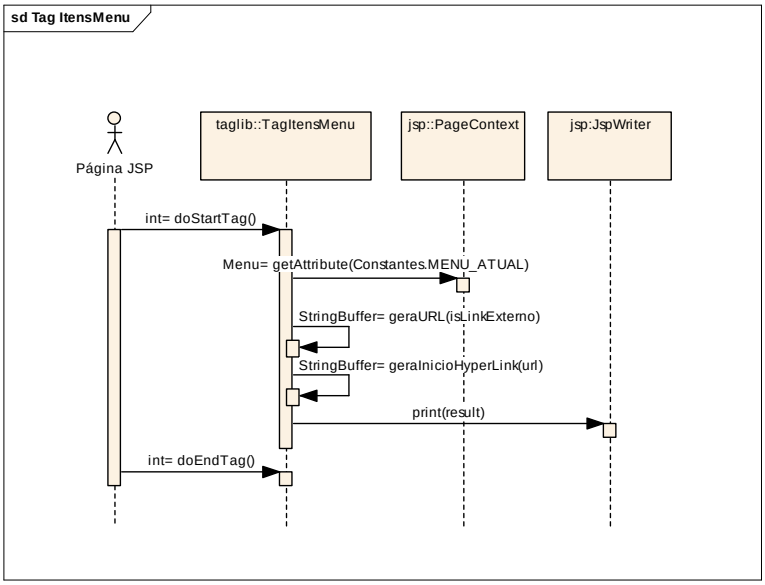


Figura 15 – Diagrama de seqüência TagItemMenu

No diagrama de seqüência da *tag* de ListadeMenus, que está disponível da figura 16. O evento doStartTag() inicializa a lista de menus, requisita uma conexão com o banco de dados,

solicita ao PageContext os dados de requisição e ao FabricaDAOMenu o objeto Menu que corresponde ao id recebido na requisição. Após isto o método doStartTag() solicita ao Menu o objeto Tabela que corresponde ao menu selecionado. Utilizando o filtro passado por parâmetro ele requisita a Tabela a lista de objetos Registro correspondente. A Lista de registros é armazenada para ser utilizada na iteração.

Para cada objeto Registro na lista, o evento doAfterBody() disponibiliza o menu atual para o PageContext. Ao final o evento doEndTag() imprime na página o resultado da *tag* através da classe JspWriter.

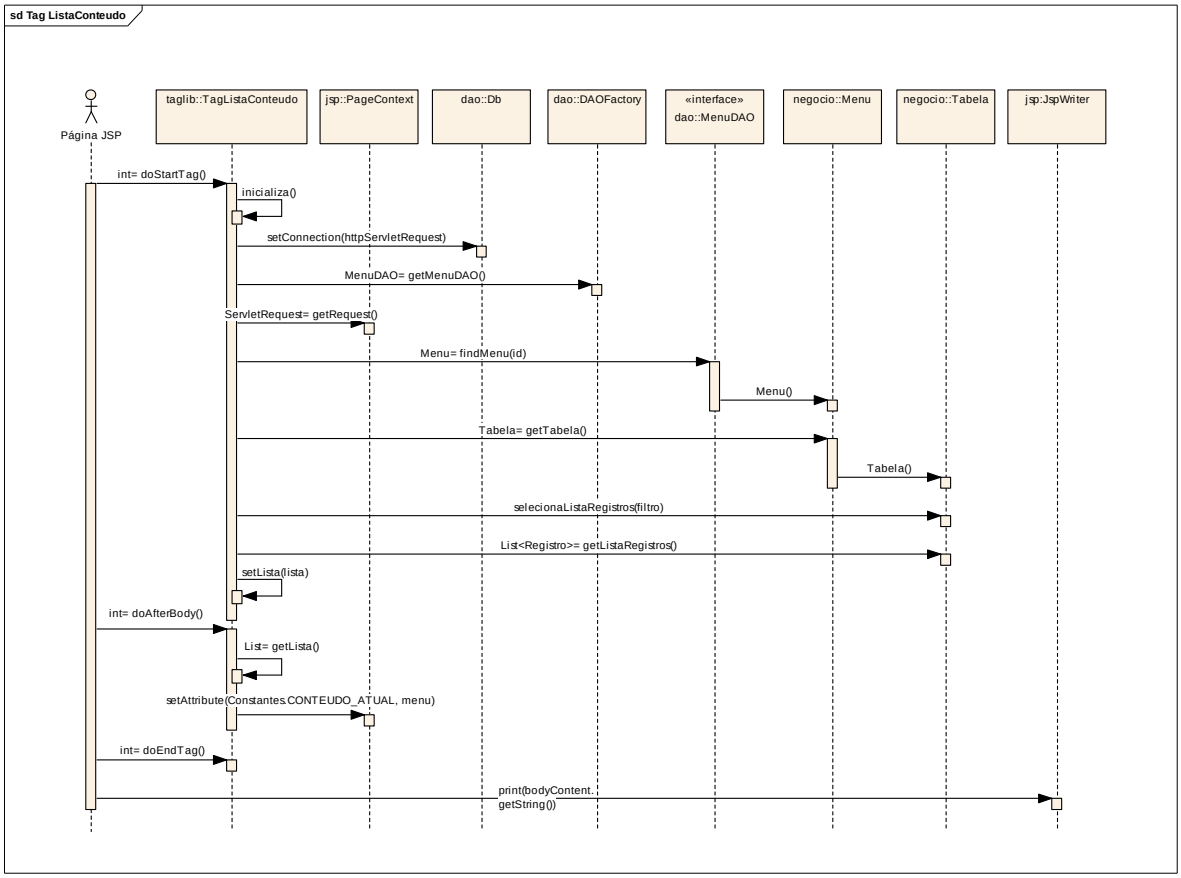


Figura 16 – Diagrama de seqüência ListaConteudo

O diagrama de seqüência da figura 17 apresenta o objeto TagItensConteudo, no evento doStartTag(), requisitando do PageContext o conteúdo atual, que foi disponibilizado pela iteração do objeto TagListaConteudo e escrevendo o link de menu na página JSP através do objeto JspWriter.

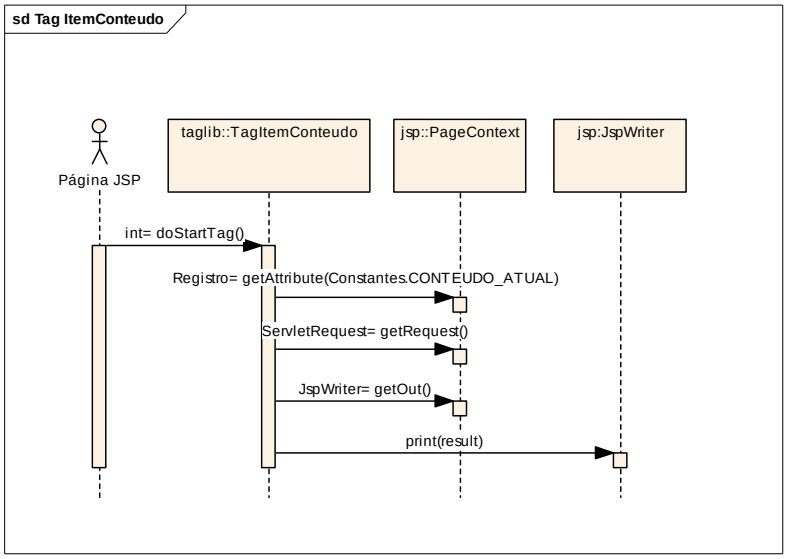


Figura 17 – Diagrama de seqüência TagItemConteudo

A figura 18 apresenta o objeto TagConteudo no evento doStartTag() pegando o objeto TabelaDAO da FabricaDAO. Após isto ele requisita ao PageContext o nome do conteúdo a ser disponibilizado e o filtro para selecionar o registro. Com o nome do conteúdo, é solicitado o objeto Tabela ao TabelaDAO e o Registro que corresponde ao filtro é disponibilizado no contexto da página através do objeto PageContext. Ao final o evento doEndTag imprime na página o corpo da *tag.*

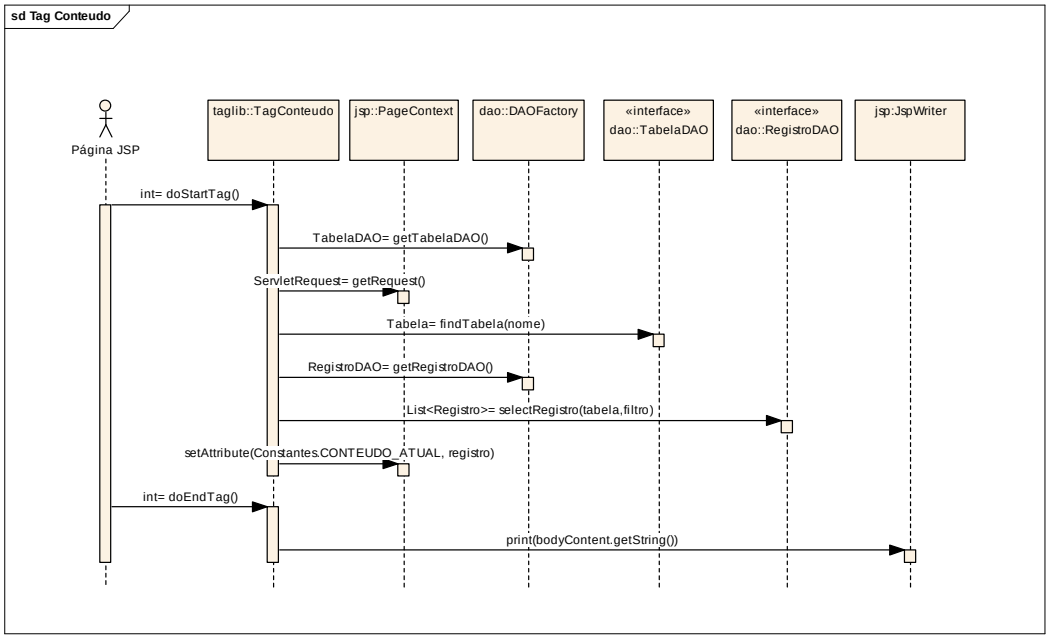


Figura 18 – Diagrama de seqüência TagConteudo

O diagrama de seqüência TagForm representado na figura 19 é apresentado como o

framework processa uma *tag* de formulário. O evento doStartTag() solicita ao PageContext os dados da requisição e escreve na página o formulário com a ação informada. O evento doEndTag apenas imprime na página o corpo da *tag.*

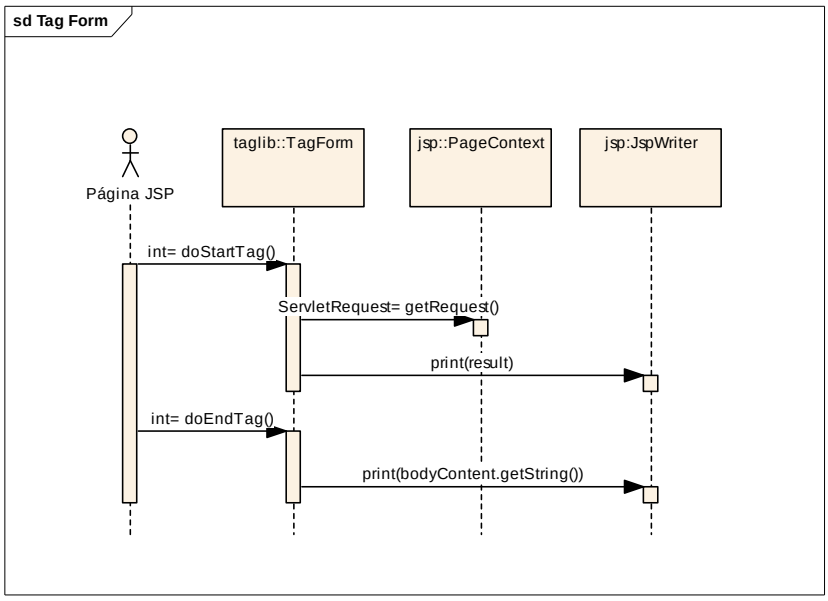


Figura 19 – Diagrama de seqüência TagForm

A figura 20 apresenta o processamento da *tag* de *input* de formulário. Ao evento doStartTag() é solicitado ao PageContect o objeto Registro correspondente ao conteúdo atual. Através do objeto Registro é solicitado o objeto Campo correspondente à propriedade nomeCampo da TagInputConteudo. O valor do campo é impresso dentro de um campo HTML *input* através do Objeto JspWriter.

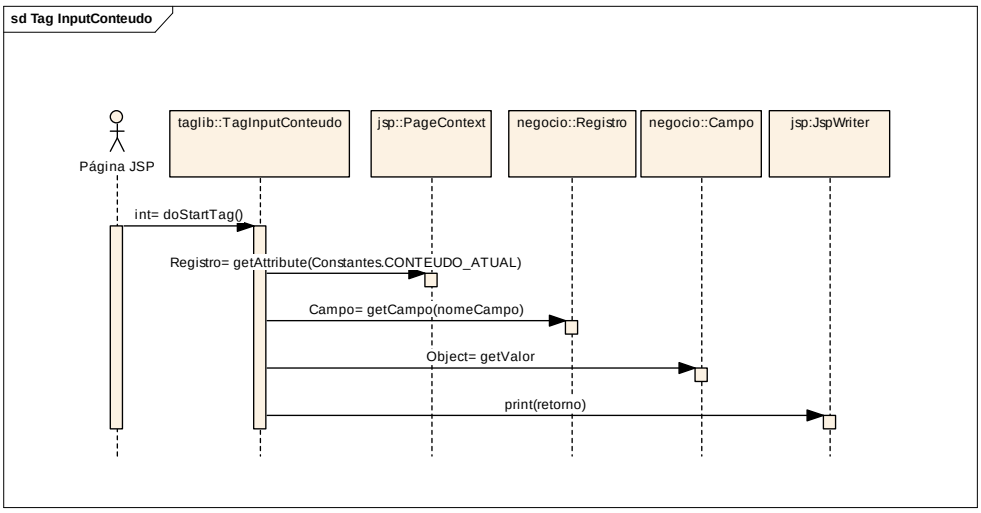


Figura 20 – Diagrama de seqüência InputConteudo

O diagrama de seqüência TagPesquisa, conforme figura 21, exibe a *tag* de pesquisa ao evento doStartTag(), requisitando os dados necessários e escrevendo na página JSP através do objeto JspWriter o form de pesquisa no qual o visitante poderá colocar as palavras a serem pesquisadas.

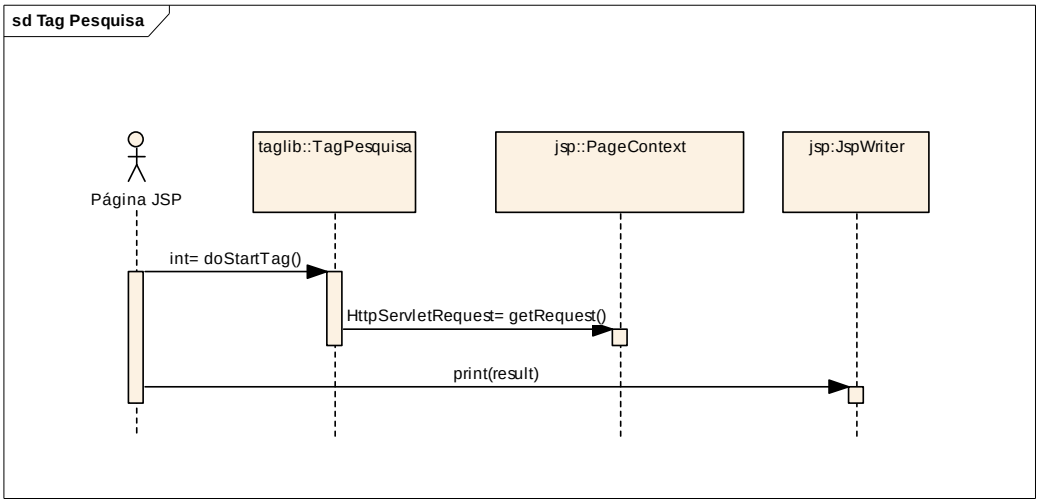


Figura 21 – Diagrama de seqüência TagPesquisa

3.1.6 DESCRIÇÃO DAS TAGS

No quadro 9 é apresentado o descritor das *tags* do *framework*, no qual pode-se visualizar as definições da biblioteca de *tags*. A especificação de cada *tag* em específico foi omitida deste quadro para facilitar a visualização.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE taglib PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag
Library       1.1//EN"       "http://java.sun.com/j2ee/dtds/web-
jsptaglibrary_1_1.dtd">
<taglib>
    <tlibversion>1.0</tlibversion>
    <jspversion>1.0</jspversion>
    <shortname>lego</shortname>
    <info>
        Biblioteca de tag's do projeto Lego
    </info>
    <tag>
        <name>menu</name>
        (...)
    </tag>
    (...)
    <tag>
        <name>pesquisa</name>
        (...)
    </tag>
</taglib>
```

Quadro 9 – Descritor das *tags* no formato DTD

Este descritor define o nome da biblioteca de *tags* e informa que a versão da biblioteca.

Abaixo o quadro 10 disponibiliza a descrição da *tag* menu.

```
<tag>
        <name>menu</name>
        <tagclass>br.com.lego.taglib.TagMenu</tagclass>
        <bodycontent>JSP</bodycontent>
        <info>
            O atributo principal é obrigatório e define se um
menu é o menu principal ou se é um sub-menu.
           O atributo administrativo somente é utilizado pela
área de gerenciamento do próprio framework. Para o
desenvolvimento de um web site este atributo pode ser ignorado.
        </info>
        <attribute>
            <name>principal</name>
            <required>true</required>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
        <attribute>
            <name>administrativo</name>
            <required>false</required>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
</tag>
```

Quadro 10 – Descritor da *tag* menu

A especificação completa através de arquivo TLD da biblioteca de *tags* está disponível no apêndice A.

3.2 ESPECIFICAÇÃO DO GERENCIADOR DE CONTEÚDOS

Para realizar a especificação do gerenciador de conteúdos, utilizou-se a ferramenta Enterprise Architect, através da UML. Como a área de gerência de conteúdos foi desenvolvida pelo próprio *framework*. Não houve necessidade de digramas de classes e nem seqüência para esta área.

3.2.1 REQUISITOS PRINCIPAIS DO GERENCIADOR DE CONTEÚDOS

O gerenciador de conteúdos deverá deverá:

d) fornecer uma área de gerência de conteúdos (requisito funcional - RF);
e) gerenciar um dicionário de dados permitindo definir as estruturas com as quais os conteúdos serão armazenados (RF);
f) permitir o cadastramento de conteúdos (RF);
g) permitir o cadastramento de menus (RF);
h) permitir cadastrar usuários que irão administrar o web *site* (RF);
i) Possibilitar que um menu tenha sub-menus (RF);
j) fornecer uma área de gerência de arquivos que serão disponibilizados na web (RF);
k) armazenar seus dados no banco de dados PostgreSQL (RNF);
l) garantir a autenticação de usuários na área de gerência (RNF).

3.2.2 DIAGRAMA DE CASOS DE USO DO GERENCIADOR DE CONTEÚDOS

Na figura 22 é apresentado o diagrama de caso de uso do gerenciador de conteúdos que

está disponível ao portal.

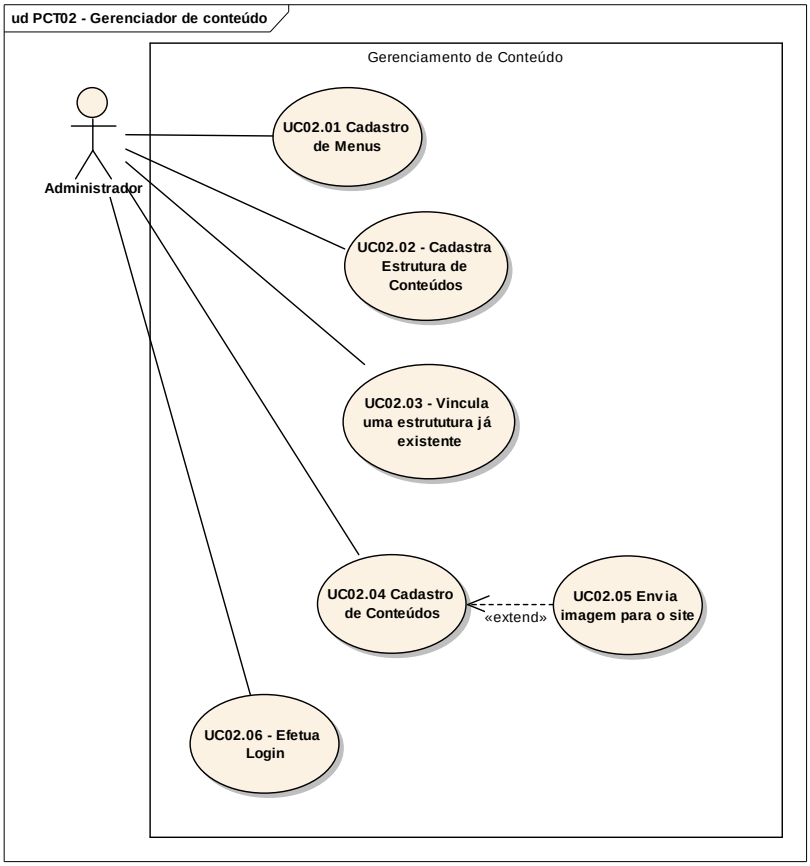


Figura 22 –Diagramas de caso de uso da área de gerência de conteúdos

A seguir são descritos brevemente os casos de uso deste módulo:

a) cadastra menus: permite o cadastro, pelo administrador, de menus no web *site*. O administrador poderá inserir, alterar ou excluir menus e definir hierarquia entre os menus;

b) cadastra estrutura de Conteúdos: permite a alteração da estrutura de um conteúdo por parte do administrador. Ele poderá criar novas estruturas de conteúdo ou alterar estruturas já existentes;

c) vincula uma estrutura já existente: permite o vinculo de tabelas existentes no banco de dados ao sistema. Esta opção permite que o administrador possa selecionar uma tabela existente no banco de dados e definí-la como uma estrutura a ser utilizada pelo portal;

d) cadastro de conteúdos: permite o cadastro pelo administrador de conteúdos no

web *site*. O administrador seleciona uma estrutura existente e insere, altera ou exclui conteúdos nesta estrutura;

e) envia imagem para o *site*: permite ao administrador o envio de imagem para o web *site*. O usuário seleciona a imagem a ser enviada ao *site* e pressiona o botão de envio. O sistema irá guardar esta imagem para que possa ser utilizada no *site*;

f) efetua *login*: permite ao administrador efetuar o *login* na área de administração do web *site*. O sistema solicita ao administrador que informe seu usuário e senha. O sistema valida o usuário e senha e caso estejam corretos libera o acesso à área de administração de conteúdos.

3.3 IMPLEMENTAÇÃO

Esta seção detalha o processo de implementação do *framework*. Ela aborda as técnicas e ferramentas utilizadas para a implementação.

3.3.1 TÉCNICAS E FERRAMENTAS UTILIZADAS

Para o desenvolvimento deste trabalho utilizou-se a plataforma NetBeans, por ser uma plataforma gratuita com diversos recursos para o desenvolvimento web em Java. A utilização desta plataforma facilitou em muito a implementação do trabalho.

O processo de implementação iniciou-se pela camada de negócios. As classes de negócio basicamente foram implementadas com os seus atributos, sendo que o NetBeans gerou automaticamente os métodos de get() e set() para cada atributo. Todas as classes de negócio implementam a interface Validacao que permite ao objeto validar seus atributos antes de persistir os mesmos.

Outra etapa foi a implementação da camada responsável pela persistência dos objetos de negócio no banco de dados. O *framework* foi especificado para persistir seus dados no banco de dados PostgreSQL, que foi escolhido por ser um banco de dados robusto e gratuito.

Havia, porém, uma preocupação em que futuramente fosse possível a persistência utilizando outros bancos de dados. Para atender a isto foi escolhido desenvolver a camada de persistência utilizando o padrão de projeto DAO. Visando facilitar ainda mais a futura implementação de outros bancos de dados, foram implementadas classes abstratas com os métodos que tem sintaxe comum a todos os bancos que atendem ao padrão ANSI. Construiu-se então uma fábrica de objetos implementada para o banco de dados PostgreSQL, sendo que para implementar a persistência utilizando outro banco de dados, somente será necessária a alteração na camada DAO. No apêndice B exemplifica-se a persistência realizada para a classe Tabela, mostrando a implementação da classe TabelaDAO, DBAnsiTabelaDAO e PostgreSqlTabelaDAO.

A persistência das classes Tabela, Coluna, Campo e Registro foi feita em tabelas que são criadas dinamicamente pelo gerenciador de conteúdos. Sendo assim um objeto da Classe Tabela corresponde a uma tabela no banco de dados. Um objeto Registro a um registro de uma tabela e assim por diante. Foram criadas apenas tabelas de controle onde ficam armazenados dados que não são encontrados no próprio metadados do banco.

Para testar a camada DAO, foram criados testes unitários utilizando o *framework* JUnit. Com isto pode-se corrigir problemas com persistência dos dados antes mesmo de ter as *tags* implementadas e a cada alteração necessária na camada de negócio foram realizados novamente os testes com a persistência dos objetos.

Com as classes de negócio e a sua persistência implementada, passou-se a implementação das *tags* do *framework*. Para isto a primeira etapa foi incluir o arquivo lego-*tags*.tld, que possui a especificação das *tags*, no projeto. Para isto, além de copiar o arquivo

para a pasta WEB-INF, foi necessário alterar o arquivo de configuração web.xml. Conforme verifica-se no quadro 11. Na seção <jsp-config> definiu-se a taglib, informando a <taglib-uri> como lego-*tags* e a <taglib-location> com a localização física do arquivo no projeto.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app   version="2.4"   xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
    <servlet>
        (…)
    </servlet>

    <servlet-mapping>
        (…)
    </servlet-mapping>

    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
        </welcome-file-list>

    <jsp-config>
        <!-- LegoDesign Tag Library Descriptors -->
        <taglib>
            <taglib-uri>lego-tags</taglib-uri>
            <taglib-location>/WEB-INF/lego-tags.tld</taglib-
location>
        </taglib>
    </jsp-config>
</web-app>
```

Quadro 11 – Arquivo web.xml com a definição da biblioteca de *tags*

Para o funcionamento das *tags* faltava então a implementação das classes que fariam o seu tratamento. Para isto cada *tag* definida tem uma classe que é sua correspondente e responsável pela sua execução, com exceção das classes TagIterate e TagItem que foram criadas para reaproveitamento de código e não possuem uma *tag* específica a qual representem.

A classe TagIterate, exibida no quadro 12, implementou os métodos comuns às *tags* de iteração do *framework*. Ela basicamente define uma lista com a qual será feita a iteração e a posição que a mesma se encontra no momento. A lista é representada pela variável lista do tipo java.util.List e a posição pela variável offset do tipo int. Ela repete o seu corpo o número de itens que possua em sua lista e a cada iteração disponibiliza um dos objetos da lista no contexto da página. Através deste contexto, que é representado pelo objeto PageContext da

Jsp as *tags* conseguem trocar objetos. Um exemplo disto é a classe TagMenu que possui uma lista de objetos Menu que devem der tratados. A cada iteração ela disponibiliza um dos menus. Este objeto Menu fica no contexto da página podendo ser utilizado por qualquer *tag* que esteja no corpo da tagMenu, como por exemplo a TagItemMenu que imprime o link do menu específico.

```
public class TagIterate extends BodyTagSupport {

      private List lista;

      private int offset;

      protected void inicializa() {
            this.setBodyContent(null); // Limpa o corpo da tag
            this.setLista(null);
            this.setOffset(0);
                Db.getInstancia().setConnection((HttpServletRequest)
pageContext.getRequest());
      }

    /**
     * Método para tratamento do fim de tag de iteração.
     * @return Retorna EVAL_PAGE para que continue a ser tratado o
restante da página.
     * @throws javax.servlet.jsp.JspException Exceção JSP
     */
      public int doEndTag() throws JspException {
            JspWriter writer = pageContext.getOut();
            try {
                  writer.print(this.getBodyContent()    !=    null    ?
this.getBodyContent()
                              .getString() : "");
            } catch (IOException IOe) {
                  throw new JspException(IOe.getMessage());
            }
            return EVAL_PAGE;
      }

    /**
     * Libera as variáveis
     */
      public void release() {
            super.release();
            this.setLista(null);
            this.setOffset(0);
      }

    /**
     * Retorna a lista de objetos
     * @return Retorna um objeto List
     */
      public List getLista() {
            return lista;
      }
```

```
    /**
     * Seta a lista de objetos
     * @param lista Objeto List
     */
      public void setLista(List lista) {
            this.lista = lista;
      }

    /**
     * Retorna a posição da lista que está sendo tratada.
     * @return int com a posição atual.
     */
      public int getOffset() {
            return offset;
      }

    /**
     * Seta a posição da lista que está sendo tratada.
     * @param offset int com a posição atual.
     */
      public void setOffset(int offset) {
            this.offset = offset;
      }
}
```

Quadro 12 – Implementação da TagIterate

A classe TagItem implementa as funcionalidades comuns às classes TagItemConteudo e TagItemMenu que é tratar as propriedades href, target e style e criar um *hiperlink* apontando para o item atual. A classe TagItemMenu implementa um link de menu e a TagItemConteúdo um link de conteúdo.

As *tags* foram implementadas de forma que os filtros são realizados automaticamente ao se navegar pelo *site*. Por exemplo: ao clicar sobre o *link* gerado por uma *tag* ItemMenu, o código do menu selecionado é passado à próxima página que pode mostrar uma lista dos conteúdos daquele menu utilizando a *tag* de listaConteudo. A listaConteudo por sua vez possui *tags* itemConteudo que direcionam a outra página que exibe o conteúdo específico através da *tag* conteúdo.

Para completar a implementação foram desenvolvidas então as páginas JSP com a área de gerência de conteúdos. Esta área foi desenvolvida com o próprio *framework* e por isto não necessitou de classes específicas. Esta experiência foi muito interessante pois mostrou a qualidade do novo *framework* na prática e proporcionou que o *framework* fosse desenvolvido utilizando suas próprias *tags*. No quadro 13 pode-se visualizar um exemplo de *tag* de iteração

aplicada na área de gerência de conteúdos. Trata-se da *tag* menu que é uma extensão da classe TagIterate.

```
<table   width="800"   border="0"     align="center"   cellpadding="0"
cellspacing="0" bgcolor="#999999">
    <tr>
      <td>
<table   width="798"   border="0"     align="center"   cellpadding="0"
cellspacing="0"  bgcolor="#FFFFFF">
    <tr>
        <lego:menu principal="S" administrativo="sim">
            <td width="15px"> </td>
            <td width="" align="center">
                <lego:itemMenu href="/adm/*.jsp" />
            </td>
        </lego:menu>
    </tr>
</table>
            </td>
    </tr>
</table>
<table   width="798"   border="0"     align="center"   cellpadding="0"
cellspacing="0"  bgcolor="#999999">
    <tr><td height="2"></td>
    </tr>
</table>
```

Quadro 13 – Implementação da menu_admInc.jspf

No quadro 14 está demonstrado um exemplo mais completo. Esta é a página responsável pela listagem dos registros de um conteúdo que podem ser alterados na área de gerência.

```
<%@ include file="headerInc.jspf"%>
<lego:areaRestrita paginaLogon="index.jsp" />
<table border="0" cellspacing="0" cellpadding="0" width="" height=""
    align="">
    <table width="550" border="1" align="center" cellpadding="2"
        cellspacing="0" bordercolor="#666666" bgcolor="#CCCCCC">
        <tr>
            <td width="100%">
                <div align="left">
                <table width="100%" border="0" cellspacing="0"
cellpadding="0">
                    <tr>
                        <td width="50%"><span class="titulo">Escolha o
registro a editar:</span></td>
                        <td width="50%">
                            <div align="right"><lego:itemConteudo
href="/adm/incluir_conteudo.jsp" target="" style="">
                                <img src="../img/novo.GIF" alt="Novo"
border="0"/>
                            </lego:itemConteudo></div>
                        </td>
                    </tr>

                </table>
                <font face="Arial, Helvetica, sans-serif" size="3"><b>
</b></font></div>
            </td>
        </tr>
    </table>
(…)
    <lego:listaConteudo>
        <table width="550" border="1" align="center" cellpadding="2"
            cellspacing="0" bordercolor="#666666" bgcolor="#CCCCCC">
            <tr>
                <td>
                    <table width="100%" border="0" align="center"
cellpadding="1"
                        cellspacing="2">
                        <tr bgcolor="#CCCCCC">
                            <td bgcolor="" width="50%">
                                <lego:listaCampos representativo="sim">
                                    <lego:escreveNomeCampoConteudo />:
<lego:escreveCampoConteudo /><br>
                                </lego:listaCampos>
                            </td>
                            <lego:form acao="excluir">
                                <td bgcolor="" width="50%" align="right">
                                    <lego:itemConteudo
href="/adm/editar_conteudo.jsp" target="" style="">
                                        <img src="../img/editar.GIF"
alt="Editar" border="0" />
                                    </lego:itemConteudo>
                                    <input type="image" name="imageField"
src="../img/excluir.GIF">

                                </td>
                            </lego:form>
                        </tr>
                    </table>
                </td>
            </tr>
        </table>
    </lego:listaConteudo>
(…)
</table>

<%@ include file="footerInc.jspf"%>
```

Quadro 14 – Arquivo cad_listaConteudo.jsp

O resultado obtido através destas *tags* é exibido na próxima seção, a qual exibe a área de gerência de conteúdos que foi construída fazendo uso das *tags*.

3.4 OPERACIONALIDADE DA IMPLEMENTAÇÃO

Para demonstrar a operacionalidade do *framework*, escolheu-se a área de gerência, pois além de ser uma parte importante do mesmo, ela foi desenvolvida utilizando as *tags* do próprio *framework*. Portanto, é um exemplo completo, tanto do uso da biblioteca de *tags*, quanto da própria área de gerenciamento. A tela da área de gerenciamento, como pode ser vista na figura 23, é composta por:

a) um menu, o qual foi construído utilizando a *tag* menu. O código que gerou o menu foi exibido no quadro 13 que consta na página 55;

b) um formulário de autenticação, o qual foi construído utilizando-se a *tag form*.

Na figura 23 o administrador deverá digitar seu *login* e senha para utilizar a área restrita.

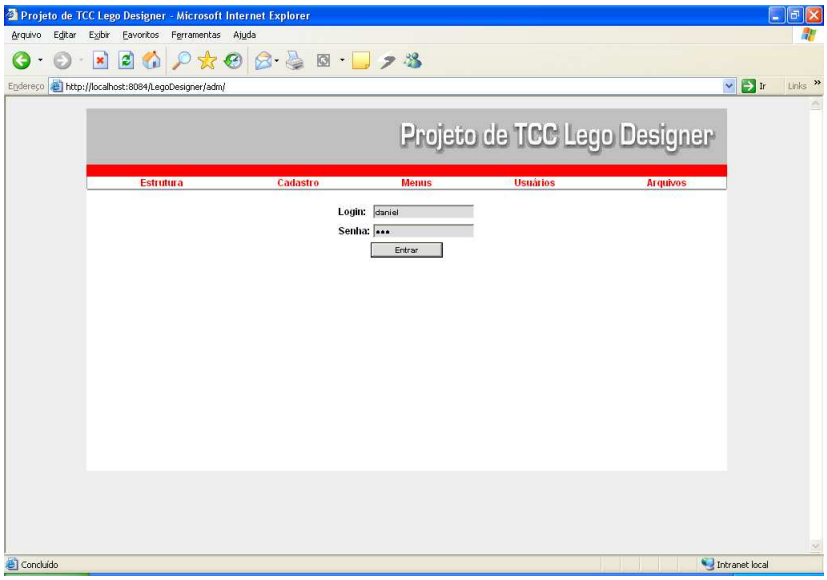


Figura 23 – Autenticação de usuário

Caso o *login* e senha sejam válidos o administrador será levado a tela de boas vindas, caso contrário será exibida mensagem informando que os dados estão incorretos. Este

processo é obrigatório para acessar qualquer página da área de gerência. Caso contrário a *tag* areaRestrita que consta em todas as páginas irá novamente redirecionar para a página de autenticação.

Outro recurso do sistema de gerência é a criação de novas estruturas de dados. Isto está acessível através do menu Estrutura. Na figura 24 pode-se visualizar a área de alteração de estruturas. Cada estrutura representa uma tabela criada no banco de dados. Através desta tela pode-se: criar uma nova estrutura, alterar uma estrutura existente e desvincular uma estrutura. Para maior segurança, ao desvincular uma estrutura, não é excluída a tabela fisicamente, mas apenas desvinculada do sistema.

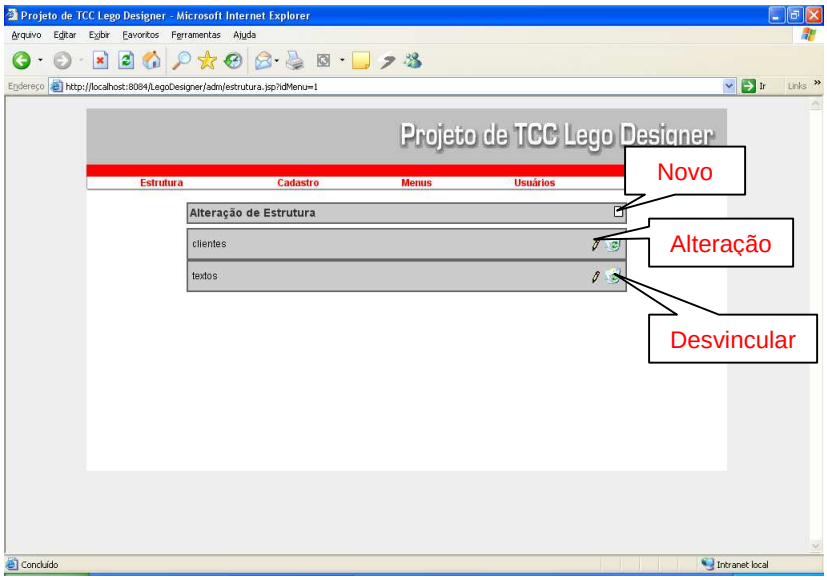


Figura 24 – Tela de alteração de estruturas

Selecionando o ícone com o lápis apresentado na figura 24, tem-se acesso à área de definição das colunas de uma estrutura. Esta página, demonstrada na figura 25, possui um ícone para criar um novo campo para a estrutura, um para editar uma coluna existente e outro para excluir uma coluna.

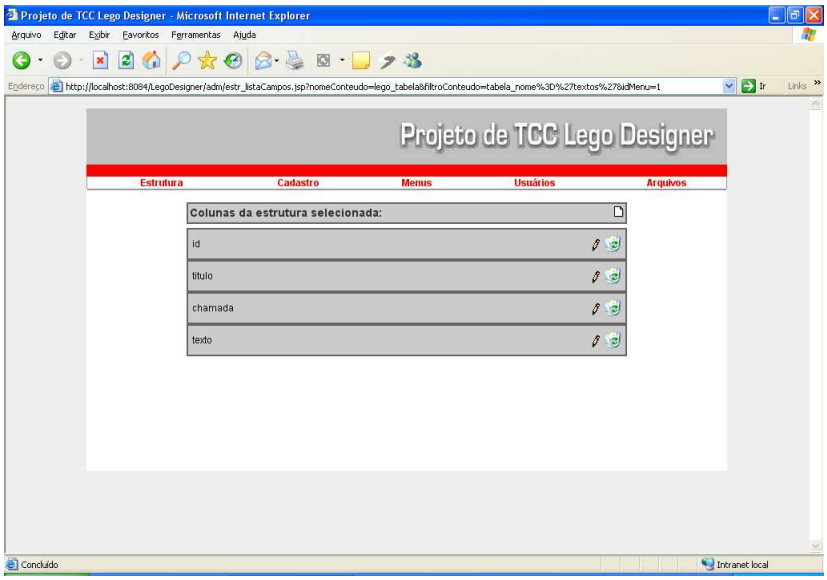


Figura 25 – Alteração de colunas

A página de inclusão de coluna é exibida na figura 26. Para sua confecção utilizou-se as seguintes *tags*:

a) areaRestrita, utilizada para garantir que o acesso seja autenticado;

b) form, para criar um formulário com ação de inclusão;

c) comboConteudo, para exibir um *combo* com lista de tabelas que estão disponíveis para a criação da coluna.

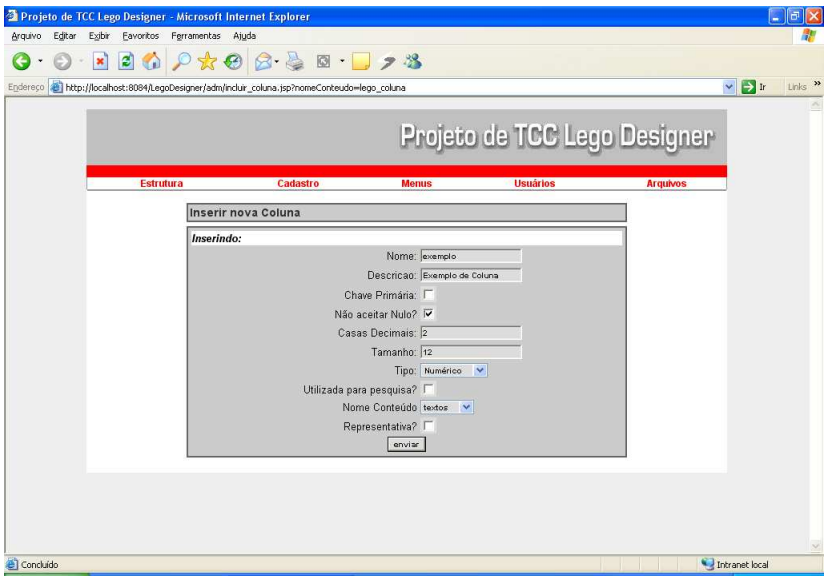


Figura 26 – Inclusão de nova coluna

A figura 27 apresenta a página cadastro de conteúdos no site, a partir dela é possível cadastrar um novo conteúdo, bem como editar e excluir um existente. O código fonte que

originou esta página é exibido no quadro 14 disponível na página 56.

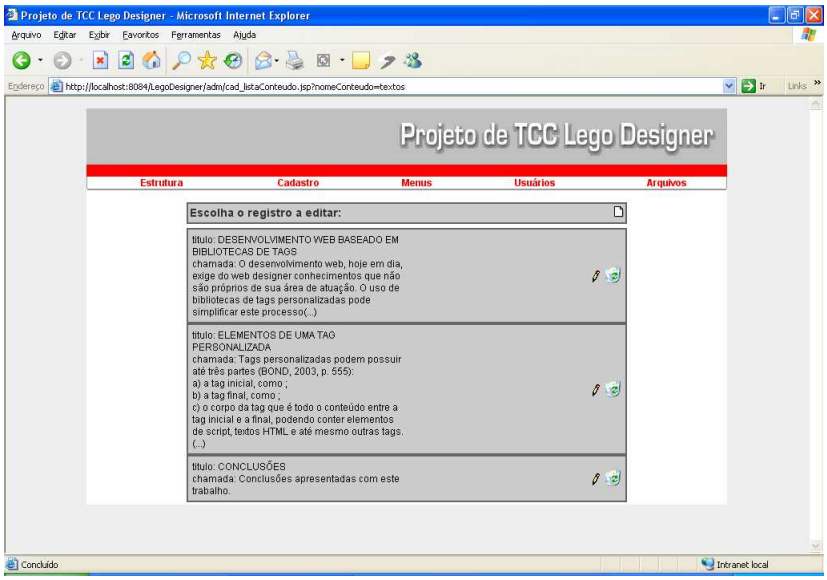


Figura 27 – Edição de registros

Para a confecção desta página utilizaram-se as seguintes *tags*:

a) areaRestrita, impede que a página seja exibida sem que se tenha efetuado o *login*;

b) listaConteudo, lista os registros do conteúdo;

c) listaCampos, cria uma iteração com os campos de um conteúdos permitindo a criação de formulários dinâmicos;

d) itemConteudo, que cria um link para um registro de conteúdo;

e) escreveNomeCampoConteudo, mostra o nome do campo.

Estes conteúdos serão disponibilizados no web *site* através de menus. A tela de cadastro de menus é apresentada na figura 28. Através dela é possível criar menus e vincula-los a conteúdos existentes.

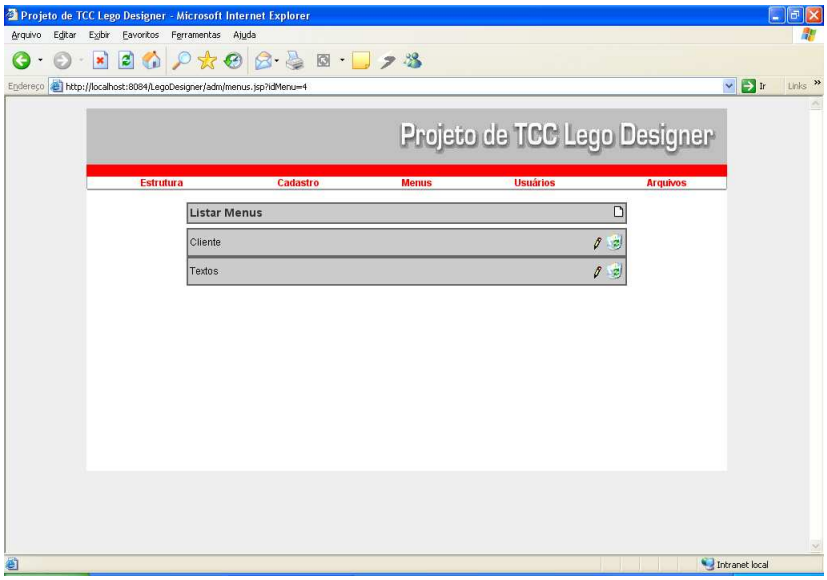


Figura 28 – Cadastro de menus

Concluído o cadastro de conteúdos na área de gerência, basta agora desenvolver a diagramação do site utilizando as *tags* do *framework*. A figura 29 exibe um *site* que exemplifica as *tags* sendo utilizadas na prática. Nesta página de abertura foram utilizadas as tags de menu e de pesquisa, o código que gerou esta página está disponível no Apêndice C.

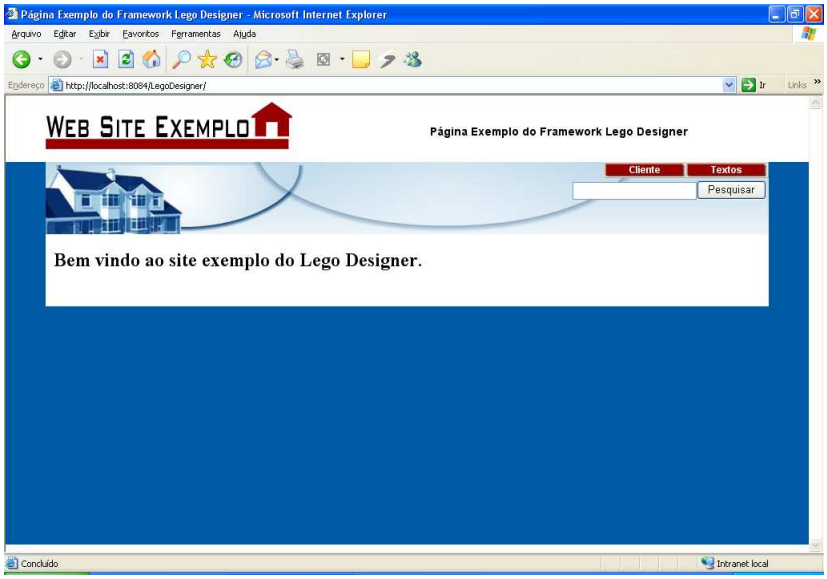


Figura 29 – Web site exemplo para aplicação do *framework*

Na figura 30 é exemplificada uma página com uma lista de conteúdos, esta lista é exibida através do menu ou realizando uma pesquisa por palavras no site. O código que gerou esta página está disponível no Apêndice D.

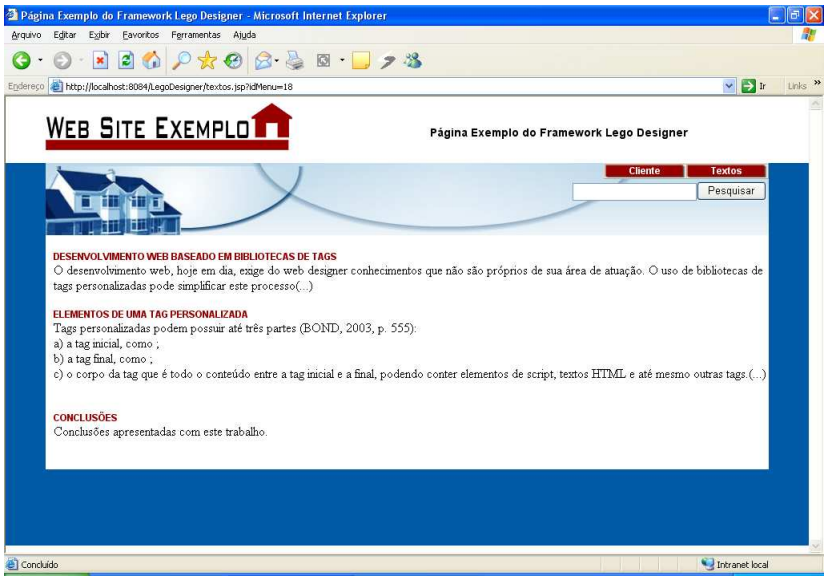


Figura 30 – Página com lista de conteudos

Para finalizar o exemplo de *site*, na figura 31 demonstramos uma página exibindo um conteúdo acessado ao selecionar o título da figura 30. O código que gerou esta página está disponível no Apêndice E.

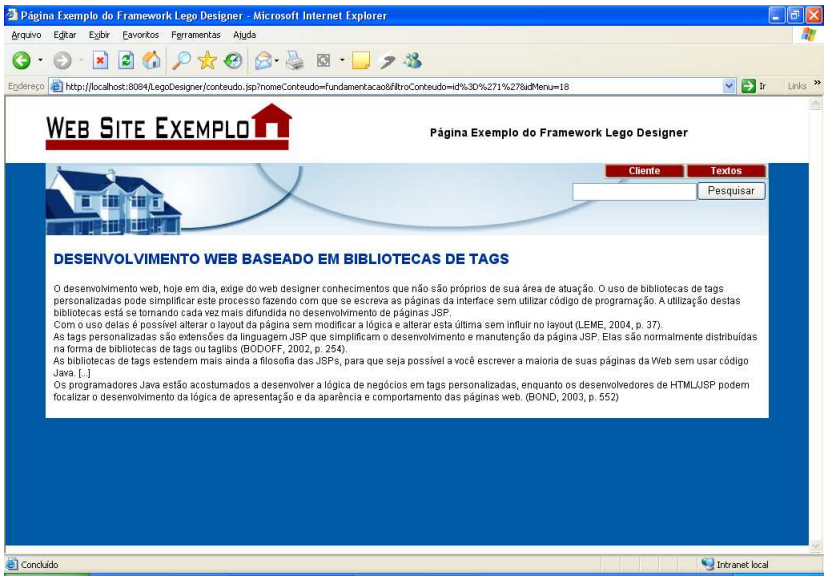


Figura 31 – Exibição de conteúdo selecionado

3.5 RESULTADOS E DISCUSSÃO

Com relação ao trabalho apresentado por Morateli (2002), a estrutura visual ficou muito mais flexível do que a utilização de *templates*, pois apenas utiliza *tags* não tendo

nenhum tipo de *script* embutido ao HTML. Além disto, as *tags* podem ser aplicadas de diversas formas, ficando isto a cargo do web *designer*, sem a necessidade de preocupação com a definição de modelos.

Os gerenciadores de conteúdo apresentados são ferramentas proprietárias e com um custo elevado, sendo direcionadas ao uso por grandes empresas. Além disto estão voltadas a função de gerência de conteúdo, enquanto o *framework* apresentados pode ser utilizados para desenvolvimento de sites em geral.

Em comparação aos outros *frameworks* apresentados nos trabalhos correlatos, os resultados obtidos com o *framework* foram positivos em vista que as *tags* desenvolvidas são de fácil entendimento e utilização. A sua implementação foi voltada ao web *designer*, sendo que não são necessários conhecimentos de programação para desenvolver um *site* utilizando o mesmo.

Sendo assim o *framework* se mostrou prático para o desenvolvimento de web *sites* de conteúdo.

4 CONCLUSÕES

O *framework* foi implementado utilizando o conceito de biblioteca de *tags*. Com isto o visual ficou independente da programação e a diagramação visual dos *sites* desenvolvidos tornou-se flexível.

A criação de uma ferramenta de gerência de conteúdos, permite que os dados sejam estruturados diretamente pelo administrador do web *site*. Desta forma, possibilitou a manutenção e estruturação do conteúdo de forma independente do programador.

O *framework* obedece ao padrão MVC, onde o modelo é gerenciado pela camada DAO, a visualização pelas *tags* disponibilizadas e o controle pelas classes controladoras das *tags*.

A área de gerência de conteúdos foi construída com o próprio *framework*.,,,,,,,,,,,,, O seu funcionamento na prática demonstra que a sua aplicação poderá ser feita até mesmo em *sites* mais complexos do que os *sites* de conteúdos previstos inicialmente. Isto também fez com que o *framework* fosse além do planejado inicialmente permitindo a utilização de formulários.

Através da implementação do *framework* e da área de gerência de conteúdos, este trabalho alcançou todos os seus objetivos. Possibilitando que a criação de *sites* possa ser realizada pelo web *designer*.

4.1 EXTENSÕES

Este *framework* tem potencial de crescimento, são várias as extensões possíveis ao projeto que o tornarão ainda mais amplo. Abaixo foram relacionadas algumas delas:

a) criação de *tags* para fins específicos tais como salas de bate papo, envio de e-

mails, fórum, entre outros;

b) criação das demais *tags* de formulário HTML: poderiam ser criadas as *tags* de formulário *radio button* e textarea;

c) implementar persistência para outros bancos de dados;

d) melhorar o processo de validação de campos. Atualmente o *framework* apenas valida campos obrigatórios, poderia também validar campos de data, numéricos e outras formatações definidas pelo usuário;

e) permitir a definição de papeis na área de gerência de conteúdos, com conceito de aprovação de conteúdos antes da publicação dos mesmos.

REFERÊNCIAS BIBLIOGRÁFICAS

ABRAWEB. **Associação Brasileira de Web Designers e Web Masters**. [S.l.], 2004. Disponível em: <http://www.abraweb.com.br/*site*/fan.php>. Acesso em: 21 nov. 2005.

BODOFF, StephanieK. **Tutorial J2EE**. Rio de Janeiro: Campus, 2002.

BOND, Martin. **Aprenda J2EE:** com EJB, JSP, Servlets, JNDI, JDBC e XML. São Paulo: Pearson Education, 2003.

CALANDRA. **Calandra KBX**. [S.l.], 2006. Disponível em: <http://www.calandra.com.br/>. Acesso em: 04 jul. 2006.

CAVANESS, Chuck. **Programming Jakarta Struts**. California, USA: O'Reilly Media, 2003.

FREITAS, Daniel; GALLINDO, Miguel A.; OLIVEIRA, Richardson. Struts: primeiros passos. **Java Magazine**, Grajaú, v. 1, n. 6, p. 41-44, [nov.] 2003.

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

GERBER, Luciano D. **Uma linguagem de padrões para o desenvolvimento de sistemas de apoio à decisão baseado em frameworks**. 1999. 145 f. Dissertação (Mestrado em Informática) - Faculdade de Informática, Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre.

HANNON HILL CORPORATION. **Content management white paper, part I – A CMS Primer**. [S.l.], 2006. Disponível em: <http://www.hannonhill.com/downloads/pdf/Hannon_Hill_Content_Management_White_Paper.pdf/>. Acesso em: 16 mar. 2006a.

_____. **Workflow in a content management system**. [S.l.], 2006. Disponível em: <http://www.hannonhill.com/downloads/pdf/Hannon_Hill_Workflow_White_Paper.pdf/>. Acesso em: 16 mar. 2006b.

INTERWOVEN. **Enterprise content management solutions for business**. [S.l.], 2006. Disponível em: <http://www.interwoven.com/products/content_management/teamsite/index.html>. Acesso em: 04 jul. 2006.

JAKARTA TAGLIBS. **Tha Jakarta taglibs project**. [S.l.], 2005. Disponível em: <http://jakarta.apache.org/taglibs/index.html/>. Acesso em: 21 nov. 2005.

KRASNER, Gleen E.; POPE, Stephen T. A cookbook for using the model-view-controller user interface paradigm in Smalltalk-80. **Journal of Object-Oriented Programming**, New York, v. 1, n. 3, p. 26-40, aug./sept. 1988.

LEME, Felipe. Jakarta taglibs. **Java Magazine**, Grajaú, v. 1, n. 15, p. 31-37, [ago.] 2004.

MAKUMBA. **Makumba helps you rapidly develop data driven web applications**. [S.l.], 2005. Disponível em: <http://www.makumba.org/index.html/>. Acesso em: 30 ago. 2005.

MINISTÉRIO DO TRABALHO E EMPREGO. **Classificação brasileira de ocupações.** [Brasília], 2002. Disponível em: <http://www.mtecbo.gov.br/busca/descricao.asp?codigo=2624-10>. Acesso em: 30 ago. 2005.

MORATELLI, Alexandre S. **Sistema de gerenciamento de conteúdo para ambiente web**. 2002. 59 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

SOUZA, Marcos V. B. **Estudo comparativo entre frameworks Java para construção de aplicações web**. 2004. 67 f. Trabalho de Graduação (Bacharelado em Ciências da Computação) - Universidade Federal de Santa Maria, Santa Maria.

STRUTS. **Apache Struts Project**. [S.l.], 2005. Disponível em: < http://struts.apache.org/>. Acesso em: 21 nov. 2005.

VELLOSO, Fábio L. Gerência de conteúdo em Java. **Java Magazine**, Grajaú, v. 1, n. 27, p. 54-58, ago. 2005.

APÊNDICE A – Arquivos TLD com definição da biblioteca de *tags* lego

O quadro 15 mostra a definição TLD da biblioteca de *tags* lego.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE taglib PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag
Library 1.1//EN" "http://java.sun.com/j2ee/dtds/web-
jsptaglibrary_1_1.dtd">

<taglib>
    <tlibversion>1.0</tlibversion>
    <jspversion>1.0</jspversion>
    <shortname>lego</shortname>
    <info>
        Biblioteca de tag's do projeto Lego
    </info>

    <tag>
        <name>menu</name>
        <tagclass>br.com.lego.taglib.TagMenu</tagclass>
        <bodycontent>JSP</bodycontent>
        <info>
            Tag menu.
        </info>
        <attribute>
            <name>principal</name>
            <required>true</required>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
        <attribute>
            <name>administrativo</name>
            <required>false</required>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
    </tag>

    <tag>
        <name>itemMenu</name>
        <tagclass>br.com.lego.taglib.TagItemMenu</tagclass>
        <bodycontent>empty</bodycontent>
        <info>
            Tag itens menu.
        </info>
        <attribute>
            <name>href</name>
            <required>true</required>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
        <attribute>
            <name>target</name>
            <required>false</required>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
        <attribute>
            <name>style</name>
            <required>false</required>
            <rtexprvalue>true</rtexprvalue>
```

```
        </attribute>
    </tag>

    <tag>
        <name>listaConteudo</name>
        <tagclass>br.com.lego.taglib.TagListaConteudo</tagclass>
        <bodycontent>JSP</bodycontent>
        <info>
            Tag lista conteúdos.
        </info>
        <attribute>
            <name>filtro</name>
            <required>false</required>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
        <attribute>
            <name>nomeConteudo</name>
            <required>false</required>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
    </tag>

    <tag>
        <name>itemConteudo</name>
        <tagclass>br.com.lego.taglib.TagItemConteudo</tagclass>
        <bodycontent>JSP</bodycontent>
        <info>
            Tag para item conteúdo.
        </info>
        <attribute>
            <name>href</name>
            <required>true</required>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
        <attribute>
            <name>target</name>
            <required>false</required>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
        <attribute>
            <name>style</name>
            <required>false</required>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
    </tag>

    <tag>
        <name>linkConteudo</name>
        <tagclass>br.com.lego.taglib.TagLinkConteudo</tagclass>
        <bodycontent>JSP</bodycontent>
        <info>
            Tag para link entre dois conteúdos.
        </info>
        <attribute>
            <name>campoNomeConteudo</name>
            <required>true</required>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
        <attribute>
            <name>href</name>
            <required>true</required>
```

```
            <rtexprvalue>true</rtexprvalue>
        </attribute>
        <attribute>
            <name>target</name>
            <required>false</required>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
        <attribute>
            <name>style</name>
            <required>false</required>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
    </tag>

    <tag>
        <name>conteudo</name>
        <tagclass>br.com.lego.taglib.TagConteudo</tagclass>
        <bodycontent>JSP</bodycontent>
        <info>
            Tag conteúdos.
        </info>
    </tag>

    <tag>
        <name>form</name>
        <tagclass>br.com.lego.taglib.TagForm</tagclass>
        <bodycontent>JSP</bodycontent>
        <info>
            Tag form.
        </info>
        <attribute>
            <name>acao</name>
            <required>true</required>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
        <attribute>
            <name>paginaErro</name>
            <required>false</required>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
        <attribute>
            <name>paginaSucesso</name>
            <required>false</required>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
        <attribute>
            <name>nomeConteudo</name>
            <required>false</required>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
    </tag>

    <tag>
        <name>escreveCampoConteudo</name>

<tagclass>br.com.lego.taglib.TagWriteTextoConteudo</tagclass>
        <bodycontent>empty</bodycontent>
        <info>
            Tag que imprime um campo de um determinado conteúdo.
        </info>
        <attribute>
```

```
            <name>nomeCampo</name>
            <required>false</required>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
    </tag>

    <tag>
        <name>escreveNomeCampoConteudo</name>

<tagclass>br.com.lego.taglib.TagWriteNomeTextoConteudo</tagclass>
        <bodycontent>empty</bodycontent>
        <info>
            Tag que imprime o nome de um campo de um determinado
conteúdo.
        </info>
    </tag>

    <tag>
        <name>input</name>
        <tagclass>br.com.lego.taglib.TagInputConteudo</tagclass>
        <bodycontent>empty</bodycontent>
        <info>
            Tag que imprime um campo input de um determinado
conteúdo.
        </info>
        <attribute>
            <name>nomeCampo</name>
            <required>false</required>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
        <attribute>
            <name>type</name>
            <required>false</required>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
    </tag>

    <tag>
        <name>imagemConteudo</name>

<tagclass>br.com.lego.taglib.TagWriteImagemConteudo</tagclass>
        <bodycontent>empty</bodycontent>
        <info>
            Tag que imprime a imagem do conteúdo.
        </info>
        <attribute>
            <name>nomeCampo</name>
            <required>true</required>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
        <attribute>
            <name>width</name>
            <required>false</required>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
        <attribute>
            <name>height</name>
            <required>false</required>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
        <attribute>
```

```
            <name>border</name>
            <required>false</required>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
        <attribute>
            <name>alt</name>
            <required>false</required>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
        <attribute>
            <name>style</name>
            <required>false</required>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
    </tag>

    <tag>
        <name>pesquisa</name>
        <tagclass>br.com.lego.taglib.TagPesquisa</tagclass>
        <bodycontent>empty</bodycontent>
        <info>
            Tag de pesquisa de conteúdos.
        </info>
        <attribute>
            <name>tableStyle</name>
            <required>false</required>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
        <attribute>
            <name>buttonStyle</name>
            <required>false</required>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
    </tag>
    <tag>
        <name>areaRestrita</name>
        <tagclass>br.com.lego.taglib.TagAreaRestrita</tagclass>
        <bodycontent>empty</bodycontent>
        <info>
            Tag para retringir acesso ao gerenciador de
conteúdos.
        </info>
        <attribute>
            <name>paginaLogon</name>
            <required>true</required>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
        <attribute>
            <name>buttonStyle</name>
            <required>false</required>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
    </tag>
    <tag>
        <name>listaCampos</name>
        <tagclass>br.com.lego.taglib.TagListaCampos</tagclass>
        <bodycontent>JSP</bodycontent>
        <info>
            Tag para listar os campos de um conteúdo.
        </info>
        <attribute>
```

```
            <name>representativo</name>
            <required>false</required>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
    </tag>
    <tag>
        <name>listaColunas</name>
        <tagclass>br.com.lego.taglib.TagListaColunas</tagclass>
        <bodycontent>JSP</bodycontent>
        <info>
            Tag para listar os campos de um conteúdo.
        </info>
        <attribute>
            <name>nomeCampo</name>
            <required>false</required>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
        <attribute>
            <name>representativo</name>
            <required>false</required>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
    </tag>
    <tag>
        <name>listaArquivos</name>
        <tagclass>br.com.lego.taglib.TagListaArquivos</tagclass>
        <bodycontent>JSP</bodycontent>
        <info>
            Tag para listar os arquivos de um conteúdo.
        </info>
    </tag>
    <tag>
        <name>itemImagem</name>
        <tagclass>br.com.lego.taglib.TagItemImagem</tagclass>
        <bodycontent>empty</bodycontent>
        <info>
            Tag para item imagem.
        </info>
        <attribute>
            <name>width</name>
            <required>false</required>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
        <attribute>
            <name>height</name>
            <required>false</required>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
    </tag>
    <tag>
        <name>comboConteudo</name>
        <tagclass>br.com.lego.taglib.TagComboConteudo</tagclass>
        <bodycontent>empty</bodycontent>
        <info>
            Tag para combo de conteúdo.
        </info>
        <attribute>
            <name>nomeConteudo</name>
            <required>true</required>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
```

```
        <attribute>
            <name>nomeCombo</name>
            <required>true</required>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
        <attribute>
            <name>campoTexto</name>
            <required>true</required>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
        <attribute>
            <name>campoValue</name>
            <required>true</required>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
        <attribute>
            <name>filtro</name>
            <required>false</required>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
    </tag>
    <tag>
        <name>comboArquivos</name>
        <tagclass>br.com.lego.taglib.TagComboArquivos</tagclass>
        <bodycontent>empty</bodycontent>
        <info>
            Tag para combo com arquivos de upload.
        </info>
        <attribute>
            <name>nomeCombo</name>
            <required>true</required>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
    </tag>
</taglib>
```

Quadro 15 – Arquivo lego.tld

APÊNDICE B – Persistência da classe Menu utilizando padrão DAO

Nos quadros 16, 17 e 18 exibe-se a interface TabelaDAO, a implementação desta interface para bancos de dados ANSI e por fim a implementação específica para banco de dados PostgreSQL.

```
public interface MenuDAO {

    /**
     * Método responsável pela inserção de novos de objetos Menu.
     * @param menu Objeto menu a ser persistido.
     * @return id do Menu inserido no banco.
     * @throws br.com.lego.exception.ExcecaoLegoValidacao Exceção caso o
objeto tenha algum campo obrigatório não preenchido.
     * @throws java.sql.SQLException Exceção causado por erro SQL no
banco de dados.
     */
    public int insertMenu(Menu menu) throws ExcecaoLegoValidacao,
SQLException;

    /**
     * Método responsável pela alteração de objetos Menu.
     * @param menu Objeto menu a ser persistido.
     * @return número de linha afetadas pelo update.
     * @throws br.com.lego.exception.ExcecaoLegoValidacao Exceção caso o
objeto tenha algum campo obrigatório não preenchido.
     * @throws java.sql.SQLException Exceção causado por erro SQL no
banco de dados.
     */
    public int updateMenu(Menu menu) throws ExcecaoLegoValidacao,
SQLException;

    /**
     * Método responsável pela exclusão de objetos Menu.
     * @param menu Objeto menu a ser persistido.
     * @return boolean  - true para sucesso, false para falha
     * @throws java.sql.SQLException Exceção causado por erro SQL no
banco de dados.
     */
    public boolean deleteMenu(Menu menu) throws SQLException;

    /**
     * Método responsável pela localização de um objeto Menu pelo seu
id.
     * @param id Identificado de Menu
     * @return Menu - Objeto Menu localizado
     * @throws java.sql.SQLException Exceção causado por erro SQL no
banco de dados.
     */
    public Menu findMenu(Integer id) throws SQLException;

    /**
     * Método responsável pela criação de uma lista de objetos Menu de
acordo com os parâmetros passados.
     * Os parâmetro que forem passados como null não serão filtrados.
```

```
     * @param id Identificado de Menu
     * @param nome Nome do Menu
     * @param idPai Identificador
     * @param menuAdm exibir menus administrativos?
     *  @throws java.sql.SQLException Exceção causado por erro SQL no
banco de dados.
     * @return List<Menu> Lista de menus localizados
     */
    public List<Menu> selectMenus(Integer id, String nome, Integer
idPai, Boolean menuAdm) throws SQLException;

    /**
     * Retorna o próximo id a ser utilizado
     *  @throws java.sql.SQLException Exceção causado por erro SQL no
banco de dados.
     * @return próximo id
     */
    public int nextId() throws SQLException;

}
```

Quadro 16 – Arquivo MenuDAO.java

```
public abstract class DBAnsiMenuDAO implements MenuDAO {

    /** Creates a new instance of DBAnsiMenuDAO */
    public DBAnsiMenuDAO() {
    }

    /**
     *
     * @see br.com.lego.dao.MenuDAO
     */
    public abstract int nextId() throws SQLException;

    /**
     *
     * @see br.com.lego.dao.MenuDAO
     */
    public int insertMenu(Menu menu) throws ExcecaoLegoValidacao,
SQLException {
        int proxId;
        menu.validaAtributos();
        String sql = new String();
        sql = "INSERT INTO "+Db.getInstancia().getSchema()+".LEGO_MENU "
+
                "(    menu_id,    menu_pai,    menu_nome,    menu_ordem,
menu_filtro, tabela_nome) " +
                "VALUES " +
                "(?,?,?,?,?,?)";
        PreparedStatement                    stat                    =
Db.getInstancia().getConnection().prepareStatement(sql);
        proxId = nextId();
        stat.setInt(1,proxId);
        Menu menuPai = menu.getMenuPai();
        if (menuPai!=null){
            stat.setInt(2,menuPai.getId());
        } else {
            stat.setNull(2,java.sql.Types.INTEGER);
        }
        stat.setString(3,menu.getNome());
        stat.setInt(4,menu.getOrdem());
```

```
        stat.setString(5,menu.getFiltroTabela());
        Tabela tabela = menu.getTabela();
        if (tabela!=null){
            stat.setString(6,tabela.getNome());
        } else {
            stat.setNull(6,java.sql.Types.VARCHAR);
        }
        stat.executeUpdate();
        stat.close();
        return proxId;
    }

    /**
     *
     * @see br.com.lego.dao.MenuDAO
     */
    public int updateMenu(Menu menu) throws ExcecaoLegoValidacao,
SQLException {
        int linhas_afetadas;
        menu.validaAtributos();
        String sql = new String();
        sql = "UPDATE "+Db.getInstancia().getSchema()+".LEGO_MENU " +
                " SET MENU_NOME=?," +
                " MENU_ORDEM=?," +
                " MENU_PAI=?," +
                " TABELA_NOME=?" +
                " WHERE MENU_ID=?";
        PreparedStatement                                          stat
=Db.getInstancia().getConnection().prepareStatement(sql);
        stat.setString(1,menu.getNome());
        stat.setInt(2,menu.getOrdem());
        Menu menuPai = menu.getMenuPai();
        if (menuPai!=null){
            stat.setInt(3,menuPai.getId());
        } else {
            stat.setNull(3,java.sql.Types.INTEGER);
        }
        Tabela tabela = menu.getTabela();
        if (tabela!=null){
            stat.setString(4,tabela.getNome());
        } else {
            stat.setNull(4,java.sql.Types.VARCHAR);
        }
        stat.setInt(5,menu.getId());

        linhas_afetadas = stat.executeUpdate();

        stat.close();

        return linhas_afetadas;

    }

    /**
     *
     * @see br.com.lego.dao.MenuDAO
     */
    public boolean deleteMenu(Menu menu) throws SQLException {
        int linhas_afetadas;
        String sql = new String();
        sql  =  "DELETE  FROM  "+Db.getInstancia().getSchema()+".LEGO_MENU
```

```
WHERE MENU_ID=?";
        PreparedStatement                                                stat
=Db.getInstancia().getConnection().prepareStatement(sql);
        stat.setInt(1,menu.getId());
        linhas_afetadas = stat.executeUpdate();

        stat.close();

        return (linhas_afetadas==1);
    }

    /**
     *
     * @see br.com.lego.dao.MenuDAO
     */
    public Menu findMenu(Integer id) throws SQLException {
        List<Menu> menus = selectMenus(id, null, null, null);
        if (menus.size()>0){
            return menus.get(0);
        }
        return null;
    }

    /**
     *
     * @see br.com.lego.dao.MenuDAO
     */
    public  List<Menu>  selectMenus(Integer  id,  String  nome,  Integer
idPai, Boolean menuAdm) throws SQLException {
        Menu menu;
        List<Menu> menus  = new LinkedList<Menu>();
        String query = new String();

        query  =  "select  menu_id,  menu_pai,  menu_nome,  menu_ordem,
menu_filtro, tabela_nome " +
                "from "+Db.getInstancia().getSchema()+".lego_menu ";
        if (id != null) {
            query = query + " and menu_id = " + id;
        }
        if (nome != null && !nome.equals("")) {
            query = query + " and menu_nome = '" + nome + "'";
        }
        if (idPai != null) {
            if (idPai == 0) {
                query = query + " and menu_pai is null";
            }  else {
                query = query + " and menu_pai = " + idPai + "";
            }
        }
        if (menuAdm!=null){
            if (menuAdm) {
                query = query + " and menu_adm = 'T'";
            } else {
                query = query + " and menu_adm = 'F'";
            }
        }

        query = query.replaceFirst("and", "where");

        query += " order by menu_ordem, menu_nome";
```

```
        PreparedStatement stmt = null;
        ResultSet rs = null;
        try {
            stmt                                                    =
Db.getInstancia().getConnection().prepareStatement(query);
            rs = stmt.executeQuery();
            while (rs.next()) {
                menu = new Menu();
                menu.setId(rs.getInt("MENU_ID"));
                menu.setNome(rs.getString("MENU_NOME"));
                menu.setOrdem(new Integer(rs.getInt("MENU_ORDEM")));
                Menu menuPai = findMenu(rs.getInt("MENU_PAI"));
                menu.setMenuPai(menuPai);
                if (menuPai!=null){
                    menuPai.getListaSubMenus().add(menu);
                }
                menu.setFiltroTabela(rs.getString("MENU_FILTRO"));

                DAOFactory genericFactory =
                        DAOFactory.getDAOFactory(Db.bancoPadrao);

                TabelaDAO tabelaDAO = genericFactory.getTabelaDAO();
                Tabela                      tabela                  =
tabelaDAO.findTabela(rs.getString("TABELA_NOME"));
                menu.setTabela(tabela);
                menus.add(menu);
            }
        } catch (SQLException e) {
            throw e;
        }
        return menus;
    }
}
```

Quadro 17 – Arquivo DBMenuDAO.java

```
public class PostgreSqlMenuDAO extends DBAnsiMenuDAO {

    /** Cria uma nova instância de PostgreSqlMenuDAO */
    public PostgreSqlMenuDAO() {
    }

    /**
     * Ver interface MenuDAO.nextId()
     * @return Poximo id
     * @see br.com.lego.dao.MenuDAO
     *  @throws java.sql.SQLException Exceção causado por erro SQL no
banco de dados.
     */
    public int nextId() throws SQLException {
        String               query               =               "select
nextval('"+Db.getInstancia().getSchema()+".lego_menu_menu_id_seq')     as
id";

        PreparedStatement stmt;
        ResultSet rs;

        stmt = Db.getInstancia().getConnection().
                prepareStatement(query.toString());
        rs = stmt.executeQuery();
        while (rs.next()) {
            return rs.getInt("ID");
        }
        return 0;
    }
}
```

Quadro 18 – Arquivo PostgreSqlMenuDAO.java

APÊNDICE C – Página principal do site de exemplo

```
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<!-- Import das tags do lego -->
<%@ taglib uri="lego-tags" prefix="lego"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-
8" />
        <title>Página Exemplo do Framework Lego Designer</title>
        <link href="padrao.css" rel="stylesheet" type="text/css" />
    </head>
    <body bgcolor="#FFFFFF">

        <table  width="90%"   border="0"  align="center"  cellpadding="0"
cellspacing="0">
            <tr>
                <td     width="42%"     height="75"     valign="top"><img
src="img_sind/logo.gif"></td>
                <td  width="58%"  height="75"><div  align="center"><span
class="textog">Página       Exemplo       do       Framework       Lego
Designer</span></div></td>
            </tr>
        </table>
        <table   width="100%"   height="85%"   border="0"   cellpadding="0"
cellspacing="0" background="img_sind/fundo.gif">
            <tr>
                <td valign="top">
                    <table    width="90%"    border="0"    align="center"
cellpadding="0" cellspacing="0">
                        <tr>
                            <td  height="90"  valign="top"  align="right"
background="img_sind/fundocasa.jpg">
                                <table    height="10"         border="1"
cellpadding="0" cellspacing="0">
                                    <tr>
                                        <lego:menu principal="S">
                                            <td
background="img_sind/botao.jpg" width="100" height="18" align="center">
                                                <span class="textomenu">
                                                    <lego:itemMenu
href="/*.jsp" style="color:#FFFFFF;"/>
                                                </span>
                                            </td>
                                        </lego:menu>
                                    </tr>
                                </table>
                                <table     border="0"     cellspacing="0"
cellpadding="0">
                                    <tr>
                                        <td><lego:pesquisa
nomeConteudo="textos" paginaPesquisa="textos.jsp"/>
                                        </td>
                                    </tr>
                                </table>
                            </td>
```

```
                        </tr>
                        <tr>
                            <td          height="90"          valign="top"
bgcolor="#FFFFFF">
                                <div style="margin-left:10;">
                                <br>
                                <h2>Bem  vindo  ao  site  exemplo  do  Lego
Designer.</h2>
                                <br></div>
                            </td>
                        </tr>
                    </table>
                </td>
            </tr>
        </table>
    </body>
</html>
```

APÊNDICE D – Página com lista de conteúdos

```
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<!-- Import das tags do lego -->
<%@ taglib uri="lego-tags" prefix="lego"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-
8" />
        <title>Página Exemplo do Framework Lego Designer</title>
        <link href="padrao.css" rel="stylesheet" type="text/css" />
    </head>
    <body bgcolor="#FFFFFF">

        <table  width="90%"   border="0"  align="center"  cellpadding="0"
cellspacing="0">
            <tr>
                <td     width="42%"     height="75"     valign="top"><img
src="img_sind/logo.gif"></td>
                <td  width="58%"  height="75"><div  align="center"><span
class="textog">Página       Exemplo       do       Framework       Lego
Designer</span></div></td>
            </tr>
        </table>
        <table   width="100%"   height="85%"   border="0"   cellpadding="0"
cellspacing="0" background="img_sind/fundo.gif">
            <tr>
                <td valign="top">
                    <table    width="90%"    border="0"    align="center"
cellpadding="0" cellspacing="0">
                        <tr>
                            <td  height="90"  valign="top"  align="right"
background="img_sind/fundocasa.jpg">
                                <table    height="10"         border="1"
cellpadding="0" cellspacing="0">
                                    <tr>
                                        <lego:menu principal="S">
                                            <td
background="img_sind/botao.jpg" width="100" height="18" align="center">
                                                <span class="textomenu">
                                                    <lego:itemMenu
href="/*.jsp" style="color:#FFFFFF;"/>
                                                </span>
                                            </td>
                                        </lego:menu>
                                    </tr>
                                </table>
                                <table     border="0"     cellspacing="0"
cellpadding="0">
                                    <tr>
                                        <td><lego:pesquisa
nomeConteudo="clientes" paginaPesquisa="clientes.jsp"/>
                                        </td>
                                    </tr>
                                </table>
                            </td>
```

```
                        </tr>
                        <tr>
                            <td          height="90"          valign="top"
bgcolor="#FFFFFF">
                                <div style="margin-left:10;">
                                <br>
                                <lego:listaConteudo>
                                    <lego:itemConteudo
href="/conteudo.jsp" style="color: #990000;">
                                        <lego:escreveCampoConteudo
nomeCampo="titulo" />
                                    </lego:itemConteudo><br>
                                    <lego:escreveCampoConteudo
nomeCampo="chamada"/>
                                    <br><br>
                                </lego:listaConteudo>
                                <br></div>
                            </td>
                        </tr>
                    </table>
                </td>
            </tr>
        </table>
    </body>
</html>
```

APÊNDICE D – Página de apresentação do conteúdo selecionado

```
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<!-- Import das tags do lego -->
<%@ taglib uri="lego-tags" prefix="lego"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-
8" />
        <title>Página Exemplo do Framework Lego Designer</title>
        <link href="padrao.css" rel="stylesheet" type="text/css" />
    </head>
    <body bgcolor="#FFFFFF">

        <table  width="90%"   border="0"  align="center"  cellpadding="0"
cellspacing="0">
            <tr>
                <td     width="42%"     height="75"     valign="top"><img
src="img_sind/logo.gif"></td>
                <td  width="58%"  height="75"><div  align="center"><span
class="textog">Página       Exemplo       do       Framework       Lego
Designer</span></div></td>
            </tr>
        </table>
        <table   width="100%"   height="85%"   border="0"   cellpadding="0"
cellspacing="0" background="img_sind/fundo.gif">
            <tr>
                <td valign="top">
                    <table    width="90%"    border="0"    align="center"
cellpadding="0" cellspacing="0">
                        <tr>
                            <td  height="90"  valign="top"  align="right"
background="img_sind/fundocasa.jpg">
                                <table    height="10"         border="1"
cellpadding="0" cellspacing="0">
                                    <tr>
                                        <lego:menu principal="S">
                                            <td
background="img_sind/botao.jpg" width="100" height="18" align="center">
                                                <span class="textomenu">
                                                    <lego:itemMenu
href="/*.jsp" style="color:#FFFFFF;"/>
                                                </span>
                                            </td>
                                        </lego:menu>
                                    </tr>
                                </table>
                                <table     border="0"     cellspacing="0"
cellpadding="0">
                                    <tr>
                                        <td><lego:pesquisa
nomeConteudo="clientes" paginaPesquisa="clientes.jsp"/>
                                        </td>
                                    </tr>
                                </table>
                            </td>
```

```
                        </tr>
                        <tr>
                            <td          height="90"          valign="top"
bgcolor="#FFFFFF">
                                <div style="margin-left:10;">
                                <br>
                                <lego:conteudo>
                                    <p><span
class="titulo"><lego:escreveCampoConteudo nomeCampo="titulo"/>
                                    </span></p>
                                    <span
class="texto"><lego:escreveCampoConteudo nomeCampo="texto"/></span>
                                </lego:conteudo>
                                <br></div>
                            </td>
                        </tr>
                    </table>
                </td>
            </tr>
        </table>
    </body>
</html>
```