

JTal - FRAMEWORK PARA DESENVOLVIMENTO DE APLICAÇÕES WEB EM JAVA

Saulo Mendes Martins, Flávio Luiz Schiavoni (Orientador)

Departamento de Informática (DIN) – Universidade Estadual de Maringá (UEM)

saulo@uses.com.br, fls@ime.usp.br

Resumo: *Muitos são os frameworks disponíveis atualmente para o desenvolvimento de aplicação Web. Este trabalho propõe, a partir da observação de alguns desses frameworks, apresentar um novo projeto que visa a facilitação do uso, pois não exige do desenvolvedor conhecimentos avançados para a criação de uma página web, já que as marcações de HTML, JavaScript, estilos CSS e transições entre páginas permanecerão organizadas internamente na estrutura do framework. É válido observar que a utilização desse novo framework, para desenvolvimento de sistemas, limita-se somente a criação do layout de uma aplicação web.*

Palavras-chaves: *Java, framework, WEB, Orientação a Objeto.*

Abstract: *There are many frameworks available nowadays for developing web application. This paper proposes, based on the observation of some of these frameworks, introduce a new project that aims the facilitation of the use, whereas it not requires from the developer an advanced knowledge to create a web page, since the markings of HTML, JavaScript, CSS styles and transitions among page remain internally organized in the structure of the framework. It is worth to observe that the use of this new framework, for systems development, is limited only to the creation of a web application layout.*

Key words: *Java, Framework, Web, Object Oriented Programming.*

1 - Introdução

Impulsionadas pela grande evolução dos meios de comunicação e do processamento de informações, as relações existentes em nossa sociedade comercial vêm se desenvolvendo nos últimos anos em um ritmo totalmente frenético e, atualmente, alcançaram um nível em que tudo deve ser feito de maneira urgente e perfeita. A utilização da *Web* se tornou imprescindível para qualquer empresa e, da mesma forma, as aplicações *Web*. Exige-se que o desenvolvimento customizado de uma aplicação garanta urgência na entrega, um retorno ágil e satisfatório ao cliente, além de um bom funcionamento da aplicação, o que só é possível, se o desenvolvedor tiver o conhecimento necessário sobre a linguagem de programação utilizada.

A escolha de uma linguagem para se criar um novo projeto é um momento decisivo para o desenvolvedor, não só em relação à facilidade de desenvolvimento, mas

também ao custo que a linguagem utilizada exige para seu aprendizado. Quanto menor o custo, maior o retorno financeiro no desenvolvimento de um sistema. Muitas dessas ferramentas já chegam ao mercado a custo zero, facilitando o desenvolvimento de aplicações.

Para este trabalho foi escolhida a linguagem de programação *Java*. A escolha da linguagem *Java* é pertinente por se tratar de uma opção otimizada para se desenvolver aplicações dentro do prazo esperado, conforme podemos notar em [1]:

“Os desenvolvedores de hoje cada vez mais reconhecem a necessidade de aplicações distribuídas, transacionais e portáteis que alavancam a velocidade, segurança e confiabilidade da tecnologia de lado do servidor. No mundo da tecnologia da informação, aplicações empresariais devem ser projetadas, construídas e produzidas por menos dinheiro, com maior velocidade e com menos recursos.”

[2] afirma que *Java API* é “(...) *Java applications programming interface – interface de programas aplicativos em Java*”. Esta *API* da linguagem *Java* permite, por exemplo, o desenvolvimento de aplicações tanto no lado do cliente quanto no servidor. Entre as diversas maneiras de desenvolver uma aplicação seguindo esta arquitetura, o desenvolvimento para a *web* tem se tornado uma alternativa viável para soluções empresariais.

A Figura 1 abaixo apresenta o funcionamento básico de uma aplicação web entre cliente (Navegador) e servidor. O Cliente Web (WebBrowser) faz uma solicitação (Request) via HTTP ao Servidor Web (Web Container). Este, por sua vez, faz o processamento das informações (Process Request) e, posteriormente, retorna ao navegador a resposta da solicitação, encerrando o processo (Response).

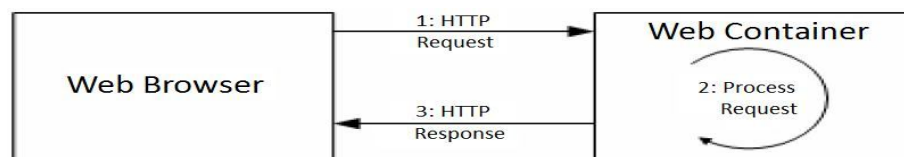


Figura 1 – Funcionamento Básico entre cliente e Servidor [3]

O funcionamento de uma aplicação *web* segue o modelo de requisição e resposta independentemente da linguagem de programação. Porém, para o desenvolvimento em *Java*, é necessário manipular e definir arquivos de configurações, conhecer conceitos básicos da linguagem para que se consiga atingir o término de uma aplicação, o que é complicado para desenvolvedores iniciantes em *Java*. Mais complicado ainda é aplicar na prática esses

conceitos de uma maneira coesa e com poucos acoplamentos. Segundo [4], “*Muitas pessoas podem compreender conceitos como objetos, interfaces, classes e herança. O desafio reside em aplicá-los à construção de software flexível e reutilizável (...)*”.

Há também a necessidade de se conhecer alguns padrões para o desenvolvimento. Um bastante conceituado é o padrão Model View Controller que, segundo [4], “*MVC é composto por três tipos de objetos. O Modelo é o objeto de aplicação, a Visão é a apresentação na tela e o Controle define a maneira como a interface do usuário reage às entradas do mesmo.*”.

2 - Objetivo

O objetivo desse artigo é desenvolver um *framework* para a criação de aplicações *Java* para a *web* de maneira que o desenvolvedor tenha uma menor necessidade de configuração na geração da visão da aplicação, conforme o modelo *MVC* citado. O desenvolvimento baseando-se neste *framework* teria um funcionamento similar ao mencionada por [2], utilizando o conceito de orientação a objetos para facilitar o desenvolvimento da aplicação, não interferindo no controle e modelo, que permanecem independentes.

Como objetivo secundário este trabalho serviu de estudo da *API Java*, *HTML DOM*, *Javascript* e Orientação a Objetos.

3 - Cenário Atual

Há atualmente vários *frameworks* que facilitam o desenvolvimento de aplicações. De acordo com o [5]:

“Frameworks são projetos reutilizáveis de todo ou parte de um sistema de software descrito por um conjunto de classes abstratas e a forma como as instâncias dessas classes colaboram. Um bom framework pode reduzir o custo de desenvolvimento de uma aplicação por uma ordem de grandeza, porque permite reutilizar tanto o design e código. Elas não exigem uma nova tecnologia, porque eles podem ser implementados com as existentes linguagens de programação orientada a objeto.”

Existe, atualmente no mercado, uma série de *frameworks* para desenvolvimento de aplicação *WEB Java*. Um estudo consideravelmente importante feito por [6], no qual relata e compara as estruturas e características importantes de alguns *frameworks* bastante populares, listados a seguir:

- JavaServer Faces, ou *JSF*, que é:

“... um padrão popular para server-side web GUI frameworks baseada em Java. Seu objetivo principal é, de acordo com suas especificações, a simples criação de aplicações web baseada em componentes reutilizáveis ...”, sobre o framework JSF afirma que “... contém os componentes individuais sob a forma de tags especiais, cuja funcionalidade está localizado na Tag Libraries. O controlador é representado como um servlet central que recebe todas as solicitações e inicia o processamento adequado ...” [6]

- Echo2, também baseado na tecnologia *Java*:

“... A completa funcionalidade necessária para a representação visual dos componentes e para o intercâmbio de dados com o cliente é encapsulado em um módulo auto-sustentável chamado Web Rendering Engine ...”, e para atualização utiliza um gerenciador de atualizações, que “...controla as mudanças no modelo de interface do usuário do componente e os apresenta para o processamento da Web Engine ...” [6].

O *Echo2* utiliza *Ajax* para a transmissão de dados entre cliente e servidor, utilizando também a *XML* para o envio e recebimento dessas informações, e faz as atualizações necessárias, no entanto esses *XML* *“... contém todas as instruções necessárias para o cliente para a atualização periódica da web GUI via DOM.” [6].*

- Google Web Toolkit (GWT), outro *framework* que:

Utiliza, na maior parte do seu funcionamento, a tecnologia *Ajax*, porém, suas principais características são diferenciadas, pois *“...o GWT compila todos os componentes visuais escritos em Java para código JavaScript ...”*. No entanto *“...o desenvolvedor tem que construir seus componentes com base no Google JRE Emulation Library, um subconjunto de 1,4 Java API. A aplicação completada é então executado em um pequeno, genérico motor de cliente.”*. Após o carregamento da aplicação no browser do cliente, somente são feitas pequenas atualizações de dados e, para isso, são utilizadas comunicações assíncronas, *“... Remote Procedure Calls (RPC) e mecanismos de serialização que permitem a pedido do server-side Services (sem Web Services) na forma de objetos Java.” [6]*

- JQuery, é uma biblioteca *JavaScript*:

Mundialmente utilizada, é um software livre, com código-fonte aberto. De acordo com [7] *“JavaScript é uma técnica de programação que funciona percorrendo e buscando seus alvos (elementos de marcação) na árvore do documento ou no DOM, ou seja, um script que só consegue executar sua ação se todo o documento já tiver sido carregado.” [7].* Sendo assim, o *JQuery* simplifica o documento *HTML*, tanto para manipulação de eventos quanto na animação de interface e interação do *Ajax* para um desenvolvimento de páginas *Web* rápido,

pois facilita muito a maneira de trabalhar com javascript, tornando a aplicação *web cross-browser* (compatível com vários navegadores). Conforme [7] “*JQuery destina-se a adicionar interatividade e dinamismo às páginas web, proporcionando ao desenvolvedor funcionalidades necessárias à criação de scripts que visem a incrementar, de forma progressiva e não obstrutiva, a usabilidade, a acessibilidade e o design, enriquecendo a experiência do usuário*”. O *javascript* é executado no navegador (cliente) e, devido as características do *JQuery*, é preparado para funcionar nos diversos *browser* atualmente utilizados. Esse *framework* vem sendo bastante utilizado, inclusive por web site populares, como o *Google, Dell, Mozilla*, entre outros.

Apesar destes *frameworks* existentes no mercado, este trabalho propõe a construção de um *framework* que possa ser integrado com soluções como *Jquery*, mas que trabalhe exclusivamente com Objetos Java no desenvolvimento de aplicações.

4 - Metodologia

Para se construir o *framework* JTal foi necessário analisar os principais elementos *HTML*, utilizados para criação de um formulário. Foi criado representações de construções de elementos *HTML*, como por exemplo, janela, campo, botão, etc. Essa estrutura *HTML* para criação de formulários é o meio no qual foi utilizado para envio/recebimento e exibição.

A partir dos elementos *HTML* estudados, houve um mapeamento para classes Java dos principais elementos. De maneira que estes objetos fossem utilizados no desenvolvimento e a renderização do *HTML* fosse feita a partir de tais classes.

Assim, essa estrutura *HTML* para criação de formulários foi o meio utilizado pelo *framework* para envio, recebimento e exibição de informações no *browser*.

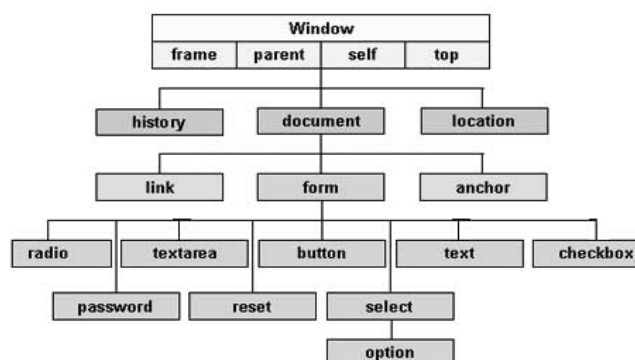


Figura 2 - Estrutura *DOM HTML* [8]

Sendo assim, a partir desses estudos, foi identificada a necessidade da definição de eventos e estilos de um elemento *HTML*. Todos os elementos *HTML*, da estrutura do *framework*, têm atributos utilizados para armazenar tais estilos e eventos. Seguindo a estrutura

do *DOM HTML*, cada *INPUT* visível no navegador pode ter estilos e também vários eventos. Os eventos são acionados conforme o usuário interage com uma página *Web*. Segundo [12] “(..) eventos são declarados através da atribuição de uma referência a uma instância da função de propriedades dos elementos *DOM*. Essas propriedades são definidas para lidar com um tipo de evento específico, por exemplo, um evento de clique é tratado por atribuir uma função à propriedade *onclick*, e um evento de *mouseover*, atribuindo uma função para a propriedade *onmouseover* de elementos que suportam estes tipos de eventos.”.

5 - Estrutura do framework.

No *framework JTal*, existe alguns objetos *html*, necessários para a criação de um cadastro, como por exemplo: *Window, Form, Input, Head, Table*, etc. Esses objetos são representados por classes finais, pois “*A classe final, não pode ser uma super classe, ou seja, não pode ser base em uma estrutura de herança. Se definirmos uma classe como final pelo operador FINAL, ela não poderá ser mais especializada...*” [9] . Essas classes citadas não podem ser redefinidas pelo desenvolvedor que utiliza o *framework*, limitando ao uso da estrutura inicial, mas existira a possibilidade de alterar as propriedades e estilos dos objetos através de métodos públicos disponíveis. Existe somente a opção de utilizar os objetos *html* na estrutura principal do *framework*. A intenção dessa limitação é: impedir a possibilidade de utilização incorreta, na construção da estrutura de uma página *web*.

O *framework JTal* está estruturado da seguinte forma: existe uma classe que representa uma página, onde é possível adicionar os objetos que representam os elementos *html*: *Head, Body, Form, Window, input, select*, etc. Uma vez que, analisando os principais elementos *HTML*, a criação de um formulário em uma página *Web* exige a existência de ao menos um formulário, campos e botões.

Utilizando esse *framework*, o desenvolvedor consegue criar vários formulários para um sistema, exigindo menos do desenvolvedor quanto à responsabilidade de definir configurações ao desenvolver.

A Figura 3 mostra qual objeto pode ser adicionado em outro objeto. Por exemplo: no objeto *Página*, pode-se ter somente um *head* e um *body*; no objeto *head*, objetos do tipo *scripts*; no objeto *body*, objetos do tipo *script* e *form*; dentro do *form*, pode-se ter somente objetos do tipo *window*, que é o principal, no qual é permitido adicionar objetos necessários para a construção de um formulário *web*, inclusive outras *windows*, mantendo, desta forma, sempre a hierarquia de classes para todos.

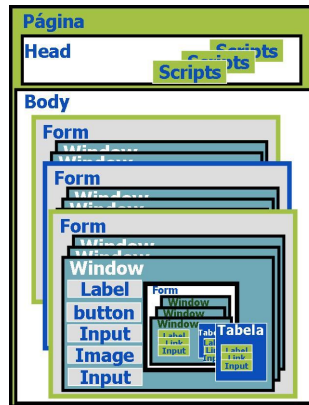


Figura 3 – Representação da hierarquia dos objetos do framework

5.1 - Controle

O funcionamento básico de uma página *web* resume-se em eventos acionados pelo usuário. Esses eventos ocorrem a cada momento que o usuário pressiona uma tecla, executa o evento clique do mouse, ou até mesmo em um simples movimento do cursor. Sendo assim, existem momentos em que uma nova página é solicitada, por meio de um evento, que pode ser um clique em um botão ou em um link. [7] detalha o funcionamento dos eventos de uma página, afirmando que:

“Na linguagem JavaScript, a entrada em funcionamento de um script depende de um evento. É lícito afirmar que se não houvesse eventos, simplesmente não haveria JavaScript. Evento, também é chamado de gatilho, é uma ação do usuário que desencadeia o início do script ou faz rodá-lo.”

Sendo assim, no *framework* JTal, toda solicitação feita ao servidor é atendida através de um mesmo *servlet* que, por sua vez, contém a coleção de páginas visitadas e faz o controle dessas páginas para que, quando uma página já acessada for solicitada novamente, possa encontrá-la em sua coleção e exibir a visão da página em seu último estado. Esse *servlet* é o núcleo principal do funcionamento do *framework*, pois ele é responsável por instanciar uma classe ou resgatar um objeto já instanciado do tipo página e exibir as informações da visão. Assim, caso haja a solicitação de uma página não existente no lado servidor, ele também é responsável por alertar a solicitação de página inexistente.

Compara-se, então, o funcionamento com o *JSF*, que possui seu funcionamento interno descrito a seguir:

“(…) O aplicativo então adquire referências necessárias para o objeto de visão e solicita FacesContext.renderResponse, o que obriga a prestação imediata de visão saltando para a fase de resposta de processamento do

ciclo de vida, como é mostrado pelas setas rotuladas Render Response no diagrama.” [10]

Esses exemplos e comparações são para compreender o funcionamento interno do *framework*, pois, como a própria *API JSF* diz:

“Os detalhes do ciclo de vida explicadas nesta secção destinam-se principalmente para os desenvolvedores que precisam saber informações como quando as validações, conversões e eventos geralmente são manipulados e que eles podem fazer para mudar a forma como e quando elas são tratadas. Os autores da página não precisam necessariamente de conhecer os detalhes do ciclo de vida.” [10]

6 - Implementação do Framework

As classes aqui criadas e detalhadas têm como objetivo gerar alguns elementos *Html*, são fundamentais para a criação de uma página *web*, e com *CSS*, o desenvolvedor consegue manipular o *layout* das páginas. O resultado da utilização dessas classes gera especificamente a visão das telas, ou seja, é utilizado para criar a interface gráfica.

A classe *Html* é utilizada para manter uma estrutura padrão entre os objetos. Tem a definição dos atributos principais das classes, utilizada na construção dos objetos *html*, ou seja, é uma classe abstrata que mantém a principal estrutura de quase todos os objetos desse *framework*, pois, conforme [9], classes abstratas são:

"(...) classes que nunca serão instanciadas na forma de objetos, somente suas filhas serão. Nestes casos, é interessante marcar essas classes como sendo classes abstratas, de modo que cada classe abstrata é tratada diferentemente pela linguagem de programação, a qual irá automaticamente impedir que se instanciem objetos a partir dela."


```

public Pagina() { //construtor da Pagina
    this.body = new Body("bodyPrincipal"); //Atributo body
    this.documentReady = new HashMap<String, Script>();
    //Atributo Coleção de Scripts do documentRead para
    //execução de códigos javascript
    this.head = new Head(); // instanciando objeto Head no atributo head
}
}

```

Um exemplo da orientação a objetos para esta classe é a herança da mesma na criação de uma página inicial conforme apresentado o no quadro 2.

Quadro 2

```

public class PaginaInicial extends Pagina {
    public PaginaInicial() { //Criando método construtor da classe PaginaInicial
        super(); // Chamando método super do construtor da classe herdada Pagina
        //... Espaço para outras definições no construtor
        //da classe PaginaInicial
        //this.head.addMeta(); this.head.addCSS();
        //this.head.addJavaScript(); this.head.setDocType();
    }
}

```

6.1.1 - Objeto *Head*

Head contém outros três importantes atributos: *javascript*, *style* e *meta*. Para o preenchimento desses valores existem os métodos *addScript*, *addCss*, *addMeta*, passando para esses métodos dois parâmetros: nome e valor, que é o *script*, *css* ou *meta*. Conforme apresentado no quadro 3.

Quadro 3

```

Head head = new Head();
head.addCss(...); //Adicionar estilos na página, ou arquivo de estilos
head.addJavaScript(...); //Ad. código javascript ou arquivos de código
arquivo.js
head.addMeta(...); //Adicionar metas ao head da página

```

6.1.2 - Objeto *Script*

Utilizada para armazenar códigos *JavaScript* ou estilos *CSS*, os atributos da classe *script* são apenas dois: um é o nome, do tipo texto, que armazena o nome do *script* adicionado pelo desenvolvedor e o segundo é o *script*, também do tipo texto, que armazena o código-fonte *javascript* que posteriormente é utilizado para executar no navegador.

6.1.3 - Atributo *documentReady*

É uma coleção de objetos que agrega objetos do tipo *script*, porém, esses objetos permanecem no atributo *documentReady* e são executados no momento em que a página *html* for totalmente carregada no navegador. O atributo título da página, do tipo texto, armazena o

texto recebido por parâmetro pelo desenvolvedor, utilizando o método “setTitulo(texto)”, fazendo com que o texto recebido via parâmetro apareça na barra de título do navegador.

6.1.4 - Métodos

Existem outros métodos importantes referentes aos atributos citados acima, são eles: *getDocType*, *getScript*, *getCss*, *getMeta*, *getTitulo* e o mais importante método, o “*show*”. O método *show* funciona da seguinte forma: ele cria uma variável do tipo texto e é responsável por fazer a chamada de todos esses métodos citados. No final desse método “*show*”, é chamado o método “*show*” do atributo *body*, que utiliza o resultado dos métodos anteriores para gerar, então, o *HTML* da página, pronto para ser enviado ao navegador.

O desenvolvedor precisa simplesmente estender essa classe e deve ter o método “*montaInterface*”, que é executado no construtor da classe *Página*, pois é ele quem gera os objetos adicionados à página. É no método “*montaInterface*” que o desenvolvedor adiciona formulário, janelas, campos e botões que forma uma página *HTML*.

6.2 - Objeto *Body*

Body tem uma coleção de objetos do tipo *form*, agrupando, então, vários objetos do tipo *form*. Essa classe estende a classe *HTML* citada anteriormente e herda suas propriedades e métodos. O ideal é que sejam utilizados, no mínimo, dois formulários em uma tela de cadastro. Por exemplo, nessa tela poderia ter uma parte para cadastrar certa informação e uma segunda parte para fazer pesquisa das informações cadastradas, utilizando dois *forms*. É possível simplesmente utilizar um botão *reset*, que serve para limpar os dados do formulário, sem interferir em outro. Essa é uma das vantagens de se ter mais de um formulário em uma página *html*.

Existem, desta forma, somente alguns métodos, que são utilizados a partir dessa classe. O “*addForm(Nome)*”, método que recebe como parâmetro somente o nome do formulário, automaticamente é adicionado a coleção de objetos do atributo *forms* do *Body*. Outros dois métodos privados e importantes dessa classe são: *getOpen*, método que gera o *HTML* necessário para iniciar uma *tag* da classe *body*, carregando também suas propriedades e estilos; *getClose*, que simplesmente finaliza a *tag* do elemento *body*. Conforme apresentado no quadro 4.

Quadro 4

```
//Instanciando um objeto Body disponibilizará métodos para
Body body = new Body();//adicionar formulários
//Adicionar formulário ao body, recebe um Objeto já Instanciado
body.addForm(...); //do tipo Form
```

6.3 - Objeto *Form*

O *Form* também estende a classe *HTML* e contém somente um atributo, porém, esse atributo é uma coleção de objetos do tipo *Window*, que é detalhado posteriormente. Sendo assim, a classe *form* também contém os métodos privados *getOpen* e *getClose*, com a mesma funcionalidade da classe *body*, que é resgatar propriedades e estilos, porém gerando as *tags* do elemento *form* da documentação *HTML*, conforme ilustra quadro 5.

| |
|---|
| Quadro 5 |
| <pre>//Instanciando um objeto Form disponibilizará métodos para Form form = new Form();//adicionar janelas dentro do form //Adicionar janelas ao form, recebe um Objeto já Instanciado do //tipo Window. form.addWindow(...);</pre> |

6.4 - Objeto *Div*

O *Div* é de grande importância para a classe *Window*. Este objeto estende a classe *Html*. Esta é a camada que permite utilizar a propriedade “*position*” com o valor “*absolute*”, que é um dos principais motivos para se utilizar o *div* como camada dos elementos. Isso faz com que os elementos *HTML* possam ter posições absolutas, ou seja, posições fixas ao serem exibidos no navegador. As coordenadas de um elemento é referente ao *div* no qual o mesmo foi adicionado, possibilitando ao desenvolvedor posicionar esses elementos facilmente em seu *layout* final.

6.5 - Objeto *Window*

A *Window* estende a classe *Div*, tem alguns atributos importantes, que são coleções de objetos agregando outros tipos de objetos, por exemplo, o próprio tipo *window*, *input*, *table*, *label*, *image*, etc. Na utilização do objeto *Window* é possível definir suas propriedades e estilos, herdados da classe *HTML*. Sendo assim, após instanciar um novo objeto *window*, são disponibilizados ao desenvolvedor alguns métodos públicos, que definem a estrutura final para a exibição da *window*. Os métodos herdados da classe *HTML* são: *setPropriedade*, *getPropriedade*, *setEstilo* e *getEstilo*.

O *setPropriedade* recebe como primeiro parâmetro o nome da propriedade, sendo o segundo parâmetro o valor da propriedade. Uma propriedade muito utilizada na formação de um elemento *input* no *html* é o “*width*”, que, como o próprio nome já diz, define a largura do campo. Então, na prática, a utilização do método *setPropriedade* ficaria desta forma: *objeto.setPropriedade(“width”, “100”)*. Para facilitar ao desenvolvedor, alguns métodos básicos estão disponíveis a ele, como o *objeto.setTamanho(width, height)*, que define as

propriedades *width* e *height* do elemento *html* no qual foi acionado o método. Conforme apresentado no quadro 6.

Quadro 6

```
//Instanciando um objeto Window disponibilizará métodos para
//adicionar objetos para criação de telas de cadastros e //movimentações
Window win1 = new Window();
```

6.5.1 - Objetos disponíveis para se adicionar em um objeto Window

Vários objetos podem ser adicionados a um objeto *Window*: *Label* (legenda), *Input* (campo), *Image* (imagem); *Button* (botão), *Table* (Tabela) e *Window* (Janela). Para cada um é possível definir estilos e propriedades, como posicionamento, valor a ser exibido, tamanho, fonte, etc. Utilizando métodos de *addPropriedade* e *addEstilos*. Observam-se nos quadro 7 a 12 exemplos no qual é criado e adicionado objetos na *Window*:

Quadro 7 – Adicionando um objeto *Label*:

Quadro 7

```
Label lab1 = new Label();//Ad. legenda p/ ser posicionada na janela, recebe Obj.
Label.
win1.addLabel(lab1);
```

Quadro 8 – Adicionando um objeto *Input*:

Quadro 8

```
Input camp1 = new Input();//Criando novo campo
camp1.addEstilo("color","blue");//Adicionando Estilo
camp1.addPropriedade("onclick","alert('');");//Adicionando Propriedade
win1.addInput(camp1);
```

Quadro 9 – Adicionando objeto *Image*:

Quadro 9

```
Image imagem1 = new Image();//Criando novo campo
imagem1.addEstilo("color","blue");//Adicionando Estilo
imagem1.addPropriedade("onclick","alert('');");//Adicionando Propriedade
//Adicionar imagens para ser posicionadas dentro da janela,
//recebe um Objeto já Instanciado do tipo Input.
win1.addImage(imagem1);
```

Quadro 10 – Adicionando objeto *Button*:

Quadro 10

```
Button bota1 = new Button();//Criando novo botão
bota1.addEstilo("color","blue");//Adicionando Estilo
bota1.addPropriedade("onclick","alert('');");//Adicionando Estilo
//Adicionar botões para ser posicionadas dentro da janela, //recebe um Objeto já
Instanciado do tipo Input.
win1.addButton(bota1);
```

Quadro 11 – Adicionando objeto *Table*:

Quadro 11

```
Table tab1 = new Table();//Criando nova tabela
Label legendal = new Label();//Criando um novo label
legendal.setValor("teste");//Definindo valor
tab1.addLabel(legendal,1,1);//Adicionando label na linha 1 coluna 1
tab1.addEstilo("color","blue",1,1);//Adicionando Estilo na linha 1 coluna 1
tab1.addPropriedade("onclick","alert('');//Adicionando Estilo na linha 1 coluna
tab1.addInput(...); tab1.addButton(...); tab1.addImage(...);
//Adicionar tabelas para ser posicionadas dentro da janela, //recebe um Objeto já
Instanciado do tipo Input.
win1.addTable(tab1);
```

Quadro 12 – Adicionando objeto *Window*:

Quadro 12

```
win1.setTamanho(500,500);
Window win2 = new Window();
win2.setTamanho(200,200);
win1.addLabel(win2);
```

7 - Comparações com outras tecnologias

Percebe-se que, além da estrutura dessa nova ferramenta detalhada anteriormente, o principal controle interno que trabalha para manter um bom funcionamento em relação à navegação entre as páginas segue uma sequência de passos. As páginas geradas com base no *framework* JTal são solicitadas via requisição *http* e renderizadas em formato *html* para finalmente serem enviadas ao navegador, como o próprio funcionamento do *JSP (JavaServer Pages)* e *JSF (JavaServer Faces)*.

Um componente principal da API Java que foi utilizado no *framework* JTal é o *Servlet*. “(...) *Servlets* são classes da linguagem de programação Java que dinamicamente processa solicitações e constrói respostas. (...)” [11].

Quanto ao ciclo de vida do *JavaServer Faces*, conforme a API disponibilizada atualmente, uma comparação é feita entre *JSF* e *JSP* :

“... o ciclo de vida do *JavaServer Faces* é diferente do ciclo de vida *JSP* em que é dividido em várias fases, a fim de apoiar o modelo de componentes sofisticados de interface do usuário. Este modelo exige que os dados do componente ser convertidos e validados, eventos do componente ser manipulados, e dados do componente ser propagadas para o beans em uma forma ordenada.”. [10]

Ainda no *JSF*, durante a o funcionamento da solicitação de uma página *“JavaServer Faces deve construir o ponto de visão, considerando simultaneamente estado salvo de uma apresentação prévia da página.”* [10], no *framework* *JTaal*, algo semelhante a essa propriedade do *JSF* foi implementado. Após a geração de uma página no lado do servidor, essas informações permanecem na memória do servidor. Portanto, a interface é gerada somente uma vez até que o usuário feche a mesma.

Sendo assim, enquanto uma página já aberta não for fechada, o servidor mantém informações digitadas pelo usuário, permitindo que o usuário possa navegar entres páginas pré-preenchidas sem que perca essas importantes informações. Suponha-se que se incie um cadastro de produto. No preenchimento desse cadastro, o usuário se depara com o campo *“Marca”* e percebe que ainda não existe cadastrado no sistema a marca que ele precisa naquele momento. Com o recurso citado anteriormente, o usuário pode ir até o cadastro de marca, sem fechar o cadastro de produto. Em seguida, pode voltar para o cadastro de produto e as informações digitadas anteriormente permanecem na tela, permitindo ao usuário informar a marca e finalizar o cadastro de produto sem ter que redigitar novamente informações que já estavam preenchidas. É importante perceber que essas informações ainda não foram *“gravadas”* em nenhuma base de dados, estão somente em memória no lado do servidor.

Esse funcionamento é basicamente semelhante ao *JSF*, pois:

“Quando um usuário faz um pedido inicial de uma página, ele ou ela está solicitando a página pela primeira vez. Quando um usuário executa um postback, ele ou ela envia o formulário contido em uma página que foi previamente carregado no browser, como resultado da execução de um pedido inicial.” [10]

Esse *postback* ocorre quando o usuário envia o formulário para um futuro processamento no lado do servidor. Desse mesmo modo funciona no *framework* *JTaal* e, o mais importante, é que esse *postback* também existe no momento em que houver uma navegação entre outras páginas, ou seja, existe um *postback* *“mascarado”*, sem que o usuário perceba que os dados já estão indo para o servidor. Além disso, quando o usuário voltar para uma página já aberta, o *framework* funciona como na abertura inicial de uma página *JSP*, descrita a seguir:

“(...) o ciclo de vida lida com um pedido inicial, ele só executa a restaurar a visão e tornar as fases de resposta porque não há nenhuma entrada do usuário ou ações de processo. Inversamente, quando o ciclo de vida lida com uma nova postagem, ele executa todas as fases.” [10]

Como no *framework JSF*, a cada nova página solicitada pelo usuário, uma nova visão é criada, mas uma referência a essa visão é armazenada em uma coleção de páginas que existe internamente, mantendo assim as visões das novas páginas solicitadas por cada sessão de um dos usuários.

8 - Estudo de Caso

Para analisar as vantagens do framework JTal foi desenvolvida a aplicação exibida abaixo. No quadro 13 está uma parte do código-fonte utilizado na criação de uma tela. Foi criado um exemplo de cadastro de produto e o resultado desse exemplo é exibido na Figura 5.

Quadro 13 – Criando a visão da aplicação:

```
Quadro 13
Form formCad = this.body.addForm("formCad");
Window winCad = new Window("winCad", "Cadastro de Produto", 20, 20, 500,
500);
formCad.addWindow(winCad);
Text codigo = new Text("01_codigo");
codigo.setOculto();
codigo.setValor("teste");
winCad.addCampo(codigo);
...
Select unidadeMedida = new Select("04_unidadeMedida");
unidadeMedida.setLegenda("Unidade de Medida");
unidadeMedida.addOpcao("UN", "UN");
...
unidadeMedida.setLargura(100);
winCad.addCampo(unidadeMedida);
...
Button botSalvar = new ButtonAjax("1_botSalvar", "Salvar",
"html.CadastroProduto", "salvar");
winCad.addButton(botSalvar);
...
Button botNovo = new ButtonAjax("2_botNovo", "Inlcuir Novo",
"html.CadastroProduto", "novo");
botNovo.addEstilo("float", "left");
winCad.addButton(botNovo);
winCad.setResizable();
winCad.setMovel();
```

Resultado do código-fonte do quadro 12:



Figura 5 – Aplicação gerada com a utilização do código exibido no quadro 12

9 – Conclusão

Os resultados obtidos com o *framework* JTal foram positivos, uma vez que sua utilização facilita a criação da interface de uma aplicação *web*, já que mantém uma estrutura simples para a criação e manutenção de interfaces, não exigindo do desenvolvedor conhecimento avançado. Provavelmente, com alguns exemplos da utilização do *framework*, um desenvolvedor iniciante já pode começar a criar *layout* de páginas, sem muito esforço.

A utilização do *framework* JTal torna o desenvolvimento ágil, pois, comparando com outros *frameworks* existentes, como o JSP e JSF que trabalham com utilização da biblioteca de *tag's*, o JTal não utiliza *tag's*. A sintaxe do *framework* JTal foi criada usando o conceito de orientações a objetos. Dessa forma, não exige do desenvolvedor a abertura e fechamento de *tag's*.

Assim, por não exigir um conhecimento avançado do desenvolvedor, as empresas podem utilizar o *framework* JTal para desenvolvimento de *layout* de módulos especificamente de um sistema. Ele permite também uma empresa manter uma rotatividade de desenvolvedores iniciantes na equipe que atua no processo de desenvolvimento de *layout*, devido a simplicidade do *framework*. Uma vez comprovado o aprendizado individual de cada desenvolvedor, na utilização da ferramenta, este pode ser transferido para desenvolvimento de outros processos no desenvolvimento da aplicação.

REFERÊNCIA

- [1] – SUN MICROSYSTEMS. <http://java.sun.com/javase/5/docs/tutorial/doc/bnaaw.html> - **Overview - The Java EE 5 Tutorial** - acessado em 17 de fevereiro de 2010
- [2] - DEITEL, H.M.; DEITEL, P. J. : **Java: como programar**; trad. Carlos Arthur Lang Lizbôa. - 4.ed. - Porto Alegre: Bookman, 2003
- [3] - <http://disciplinas.ist.utl.pt/leic-es/2008-2009/labs/stripes-tutorial/request-response.jpg> - acessado em 17 de fevereiro de 2010
- [4] - Gamma, Erich: **Padrões de projeto: soluções reutilizáveis de software orientado a objetos** / Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides; trad. Luiz A. Meirelles Salgado. – Porto Alegre; Bookman, 2000.
- [5] - Don Roberts, Ralph Johnson. <http://www.dsc.ufcg.edu.br/~jacques/cursos/map/html/frame/evolve.html> - **Evolving Frameworks** - acessado em 17 de fevereiro de 2010
- [6] - wingS Project. <http://wingsframework.org/cms/comparisons.html> - **wingsframework.org** - acessado em 17 de fevereiro de 2010
- [7] - Silva, Maurício Samy Jquery: **a biblioteca do programador JavaScript** / Maurício Samy Silva. – São Paulo : Novatec Editora, 2008.

- José Augusto Navarro Garcia Manzano, Suely Alvez de Toledo, - São Paulo; Érica, 2008.
- [8] - <http://www.tutorialspoint.com/images/html-dom.jpg> - acessado em 17 de fevereiro de 2010
- [9] - Dall'Oglio, Pablo PHP: **programando com orientação a objetos** / Pablo Dall'Oglio. – São Paulo : Novatec Editora, 2007.
- [10] - <http://technology-related.com/javaee/5/docs/tutorial/doc/bnaqq.html> - **The Life Cycle of a JavaServer Faces Page** - acessado em 17 de fevereiro de 2010
- [11] - SUN MICRO SYSTEMS. <http://java.sun.com/javaee/5/docs/tutorial/doc/bnaay.html#bnabb> - **Distributed Multitiered Applications** - acessado em 17 de fevereiro de 2010
- [12] – BIBEAULT, Bear e KATZ, Yehuda, **Jquery In Action**, Manning. Manning Publication Co. 2008