UNIVERSIDADE TIRADENTES

RODRIGO MENEZES DA CONCEIÇÃO

JAVASERVER FACES (JSF):
UM ESTUDO COMPARATIVO ENTRE BIBLIOTECAS DE COMPONENTES.

ARACAJU

2008

RODRIGO MENEZES DA CONCEIÇÃO

JAVASERVER FACES (JSF):
UM ESTUDO COMPARATIVO ENTRE BIBLIOTECAS DE COMPONENTES.

Monografia apresentada à Universidade Tiradentes como um dos pré-requisitos para a obtenção do grau de bacharel em Sistema de Informação.

ORIENTADOR: JAILSON LESSA MARQUES

ARACAJU

2008

RODRIGO MENEZES DA CONCEIÇÃO

JAVASERVER FACES (JSF): UM ESTUDO COMPARATIVO ENTRE BIBLIOTECAS DE COMPONENTES.

Monografia apresentada como exigência parcial para a obtenção do grau de bacharel em Sistemas de Informação, à comissão julgadora da Universidade Tiradentes.

Aprovada em ____ / ____ / _____

BANCA EXAMINADORA

_____

M.Sc. Andrés I. Martínez Menéndez
Universidade Tiradentes

_____

Levi da Costa Mota
Infonet

_____

Prof. Jailson Lessa Marques
Universidade Tiradentes

Dedico este trabalho a todos que me apoiaram neste árduo caminho e a todos que acreditam na honestidade como um dos pilares para a construção de um Brasil melhor.

AGRADECIMENTOS

Primeiramente agradeço aos meus pais por terem sempre me apoiado em meus estudos, sempre se sacrificando para me dar a melhor educação possível, e por terem confiado em minha capacidade, incluindo os momentos difíceis em que eu mesmo duvidava dela.

Agradeço também a Daniel Camara que me indicou para um estagio que mudou muito o rumo de minha vida. Nesse estagio pela primeira vez trabalhei como desenvolvedor e foi quando eu pensei: “Eu consigo fazer isto e eu gosto de fazer isto”.

O estágio citado foi na Secretaria Municipal de Finanças da cidade de Aracaju, que me fez amadurecer tanto profissionalmente quanto pessoalmente e, por isso, não poderia deixar de registrar meus agradecimentos ao pessoal da SEFIN, principalmente a Heverton por ter confiado em minha capacidade, e por ter sido um ótimo chefe.

Agradeço ao pessoal da Universidade Federal de Sergipe, tanto aos meus colegas quantos aos professores, mesmo que um dia eu tenha perdido o estimulo e tenha mudado de curso e universidade. Hoje eu percebo que os anos, assim como tudo o que passei na UFS, nos bons e nos maus momentos, ajudaram muito a construir quem eu sou pessoalmente e profissionalmente.

Agradeço ao pessoal da INFOX, empresa onde trabalho, por ter me acolhido e confiado em mim, e principalmente ao pessoal da Fabrica de Software, pessoas que convivo mais de 8 horas por dia.

Agradeço a todos os meus amigos, que por motivos de espaço não conseguiria citar todos os nomes.

Em relação à UNIT, eu agradeço aos colegas que conheci e aos vários deles que se tornaram bons amigos. Agradeço também aos grandes professores que tive como: Eduardo Bernardes e Jailson Lessa, este ultimo meu orientador, a quem eu preciso agradecer muito por ter me orientado de verdade, e por me ajudar muito a concluir este trabalho.

Up the irons

All of the books in the world contain no more information than is broadcast as video in a single large American city in a single year. Not all bits have equal value.

Carl Sagan

RESUMO

A Internet mudou a maneira como as pessoas se comunicam e obtêm informação, diminuindo o tempo e encurtando as distâncias. Desde seu surgimento aconteceu uma evolução muito grande, onde os sítios com documentos estáticos foram substituídos por aplicações web dinâmicas, que se tornam cada vez mais complexas. Durante esta evolução das aplicações web, a tecnologia Java teve e tem um papel muito importante, sempre evoluindo para resolver os problemas encontrados, e melhorar a experiência, tanto de quem usa a tecnologia no desenvolvimento, quanto para quem é o usuário final. Nessa etapa atual da evolução da tecnologia Java para web, temos o JavaServer Faces que foi idealizado para ser robusto, escalável e produtivo. Este trabalho mostra a evolução da tecnologia Java e como a partir desta constante evolução, foi projetado e o construído o JavaServer Faces, e como os sistemas de componentes de interface do usuário ajudou a tornar o desenvolvimento mais produtivo e menos cansativo. O ponto principal do trabalho é mostrar a importância, em termos de produtividade, do surgimento de bibliotecas de componentes de interface do usuário e fazer um estudo comparativo com três das principais bibliotecas encontradas no mercado, fazendo deste trabalho uma referencia para quem estiver iniciando no desenvolvimento de aplicações, utilizando a tecnologia JavaServer Faces.

PALAVRAS-CHAVE: Java; JavaServer Faces; Internet

ABSTRACT

The Internet has changed the way people communicate and obtain information, reducing the time and shorten the distances. Since its emergence was a very big development, where the sites with static documents were replaced by dynamic web applications, which are becoming increasingly complex. During the development of web applications, the Java technology have had a very important role, always evolving to address the problems encountered and improve the experience of who uses the technology in development, as to who is the end user. In this current stage of the evolution of Java technology for web, we have the JavaServer Faces which was designed to be robust, scalable and productive. This work shows the evolution of Java technology and how from this constant evolution was designed and built the JavaServer Faces and how the systems components of the user interface helped make the development more productive and less stressful. The main point of this work is to show the importance, in terms of productivity, the emergence of user interface component libraries and make a comparative study with three major libraries found in the market, making this a reference work for those who are starting to develop of applications using the JavaServer Faces technology.

KEY-WORDS: Java; JavaServer Faces; Internet

LISTA DE TABELAS

Tabela 1: Termos importantes do JSF. 16

Tabela 2: Classificação proposta para quantificar a qualidade dos atributos de cada componente estudado.....43

Tabela 3: Tabela comparando os componentes de calendário.....46

Tabela 4: Tabela comparando os componentes de upload de arquivo.....49

Tabela 5: Tabela comparando os componentes de abas.....52

Tabela 6: Tabela comparando os componentes de editores WYSIWYG.....55

Tabela 7: Tabela comparando os componentes de árvore.....58

Tabela 8: Tabela comparando os componentes de menu.....61

Tabela 9: Tabela comparando os componentes de tabelas dinâmicas.....64

Tabela 10: Tabela comparando os componentes do Google Maps.....67

Tabela 11: Tabela comparando os componentes de menu de contexto.....70

Tabela 12: Pontuação dos componentes das bibliotecas.....73

LISTA DE FIGURAS

Figura 1: Funcionamento de um Servlet....4

Figura 2: Navegação entre páginas na arquitetura Model 1....10

Figura 3: Funcionamento do modelo MVC....11

Figura 4: A base da arquitetura JSF....15

Figura 5: Os componentes UI são mantidos no servidor em um view, e são renderizados em HTML ou outra linguagem de apresentação. Adaptado de Mann [13]....19

Figura 6: Ciclo de vida de uma requisição JSF....32

Figura 7: Componentes de calendário no modo popup....45

Figura 8: Componente calendário do RichFaces, no modo agenda....46

Figura 9: Componente de envio de arquivo do ICEFaces....47

Figura 10: Componente de envio de arquivo do RichFaces....48

Figura 11: Componente de envio de arquivo do Tomahawk....48

Figura 12: Componente de Abas do RichFaces....51

Figura 13: Componente de Abas do ICEFaces....51

Figura 14: Compoente de Abas do Tomahawk....51

Figura 15: Componente editor WYSIWYG do Tomahawk....54

Figura 16: Componente editor WYSIWYG do ICEFaces....55

Figura 17: Componente de Árvore do Richfaces representando uma coleção de música....56

Figura 18: Componente de árvore do ICEFaces....57

Figura 19: Componente de árvore do Tomahawk....57

Figura 20: Componente de menu do RichFaces....59

Figura 21: Componente de menu do ICEFaces....60

Figura 22: Componente de menu do Tomahawk.....60

Figura 23: Componente de tabela dinâmica do Tomahawk.....63

Figura 24: Componente de tabela dinâmica do ICEFaces mostrando a possibilidade de agrupamento.....63

Figura 25: Componente de tabela dinâmica do RichFaces.....64

Figura 26: Componente do Google Maps do RichFaces.....66

Figura 27: Componente do Google Maps do ICEFaces.....67

Figura 28: Componente de menu de contexto do ICEFaces.....69

Figura 29: Componente de menu de contexto do RichFaces.....69

LISTA DE LISTAGENS

Listagem 1: Fragmento de código de um servlet.....5

Listagem 2: Fragmento de um template processado por um servlet.....6

Listagem 3: Fragmento de código JSP.....7

Listagem 4: Fragmento de código JSP usando JavaBean.....9

Listagem 5: Fragmento de código com uso de JSTL e linguagens de expressão.....9

Listagem 6: Fragmento de código JSF.....20

Listagem 7: Fragmento de código HTML gerado pelo renderizador HTML do JSF.....20

Listagem 8: Script para o Greasymonkey que permite a edição em campos de uma página que estejam em modo somente leitura, ou desabilitados 22

Listagem 9: Fragmento de código JSF mostrando uso de um validador.....23

Listagem 10: Fragmento de código JSP mostrando a associação entre um componente UI e um back bean.....24

Listagem 11: Fragmento de código JSF, mostrando a associação, usando a propriedade binding.....24

Listagem 12: Fragmento de código JSP que mostra o problema de conversão de tipos que um programador enfrenta.....26

Listagem 13: Fragmento de código mostrando um evento de mudança de valor.....28

SUMÁRIO

1 ARQUITETURAS DE APLICAÇÕES JAVA PARA WEB....4

1.1 Servlets....4

1.2 JavaServer Pages (JSP)....6

1.3 JSP Model 1: JavaBeans, Java Standart Tag Library e navegação descentralizada....8

1.4 JSP Model 2: MVC (Model - View - Controller)....10

2 JAVASERVER FACES (JSF)....13

2.1 Introdução....13

2.2 As peças do quebra cabeças....15

2.2.1 Componentes UI....16

2.2.2 Renderizadores....19

2.2.3 Validadores....21

2.2.4 Backing Beans....23

2.2.5 Conversores....24

2.2.6 Eventos e Ouvintes....27

2.2.6.1 Eventos de mudança de valor....27

2.2.6.2 Eventos de Ação....28

2.2.6.3 Eventos de Modelos de Dados....28

2.2.6.4 Evento de Fase....29

2.2.7 Mensagens....29

2.2.8 Navegação....31

2.3 Ciclo de vida....31
2.3.1 Fase 1: Restore View....33
2.3.2 Fase 2: Apply Request Values....33
2.3.3 Fase 3: Process Validation....34
2.3.4 Fase 4: Update Model Values....34
2.3.5 Fase 5: Invoke Application....35
2.3.6 Fase 6: Render Response....35

3 BIBLIOTECAS DE COMPONENTES JSF....37
3.1 Introdução....37
3.2 Escolha das bibliotecas estudadas....38
3.3 RichFaces....39
3.4 ICEfaces....40
3.5 MyFaces Tomahawk....41

4 ESTUDO COMPARATIVO ENTRE AS BIBLIOTECAS....42
4.1 Introdução....42
4.2 Comparativo Entre Componentes....44
4.2.1 Calendário....44
4.2.2 Envio de Arquivo....46
4.2.3 Abas....49
4.2.4 Editor WYSIWYG....52
4.2.5 Árvores....55
4.2.6 Menus....58

4.2.7 Tabelas Dinâmicas.....61

4.2.8 Google Maps.....64

4.2.9 Menu de Contexto.....68

5 CONCLUSÃO.....71

6 REFERÊNCIAS BIBLIOGRÁFICAS.....75

INTRODUÇÃO

A Internet surgiu de um projeto militar, iniciado em plena Guerra Fria. Este projeto possuía a intenção de formar uma rede descentralizada que garantisse a comunicação caso houvesse um ataque ao solo Norte Americano por parte da antiga União Soviética. Com o tempo, mais países foram se conectando a esta rede, e na década de 90, foi permitido acesso ao público.

Em seu início a Internet mostrou ser bastante útil, reunindo e permitindo o compartilhamento de informações. Estas características despertaram o interesse acadêmico, que com sua contribuição participou muito com esta evolução.

No início a informação era basicamente estática, contendo milhares de documentos em hipertexto formando uma base de conhecimento muito grande. Com o tempo foram estudadas maneiras de interagir melhor com esta informação, mudando o foco de apenas texto estático à páginas com texto dinâmico baseadas em operações interativas com o usuário.

Desta maneira surgiram as aplicações Web, que em seu modesto início forneciam uma interação básica com o usuário, permitindo que certos serviços simples pudessem ser executados através de uma rede.

No início, foram surgindo diversas arquiteturas e linguagens de programação que passaram por um desenvolvimento progressivo que pode ser comparado a Teoria da Evolução das Espécies proposta por Charles Darwin, que diz que os animais evoluíam de modo que os

mais aptos sobrevivem. Muitas arquiteturas surgiram para tentar solucionar os problemas de cada época, e algumas tiveram êxito e continuaram a receber melhorias, e muitas fracassaram e deixaram de ser usadas.

Destas tecnologias, uma das mais bem sucedidas foi o Java, que mesmo tendo surgido com um propósito diferente do uso em aplicações Web, evoluiu de modo a atender esta forte necessidade, tento muito sucesso. O Java teve um desenvolvimento gradativo e cada passo dado, contou com uma melhora significativa ao estado anterior da tecnologia.

Primeiro surgiram os servlets, que nada mais eram que classes Java que seguiam um padrão, e eram executados no lado do servidor, de maneira bem eficiente. Os servlets tinham o problema der ser necessário escrever o código de saída HTML na classe Java, dificultando em vários aspectos o trabalho do programador. Com isso foi inevitável a criação de algo novo, e daí surgiu o JSP, que seguiu o caminho inverso, adicionando código Java em páginas HTML, melhorando assim o desenvolvimento e manutenção.

Outras etapas evolutivas aconteceram. Foram criados os conceitos de JavaBeans e Java Standard Tags Library (JSTL), para gerar uma maior separação entre os códigos HTML e Java. Em outro passo tivemos a utilização do padrão MVC (Model - View - Controller), como uma tentativa ainda maior de separação de camadas.

Todas estas tecnologias Java, amadureceram muito, e o padrão MVC se mostrou muito importante, levando à criação de uma especificação oficial MVC, que foi o passo evolutivo anterior à criação do JavaServer Faces. O JSF trouxe muitos conceitos com a intenção de facilitar o trabalho do desenvolvedor e, ao mesmo tempo, criar uma base sólida, robusta e escalável.

Entre estes conceitos, um dos principais foi o de componentes de interface de usuário (UI), que trouxe uma vantagem muito grande em termos de reutilização de código e padronização, permitindo que equipes de desenvolvedores criassem bibliotecas de componentes, que poderiam ser reaproveitadas em muitos projetos. Este reaproveitamento motivou equipes a se especializarem na criação de bibliotecas poderosas e versáteis, tais como RichFaces, IceFaces e a MyFaces Tomahawk.

1 ARQUITETURAS DE APLICAÇÕES JAVA PARA WEB

1.1 Servlets

Os Servlets sugiram de um esforço da Sun Microsystems na criação de uma arquitetura de aplicações WEB padronizada e teve sua primeira versão lançada em junho de 1997 [2]. Servlets são componentes escritos em Java que são executados em servidores que implementem a especificação dos servlets, podendo estes servidores serem escritos em outras linguagens além do Java [3].

A API Servet foi de grande importância, pois consolidou o uso de conceitos fundamentais para aplicações WEB como Request, Response, Session, etc [2]. Na figura 1 podemos observar o funcionamento de um servlet, observando seu ciclo de vida.

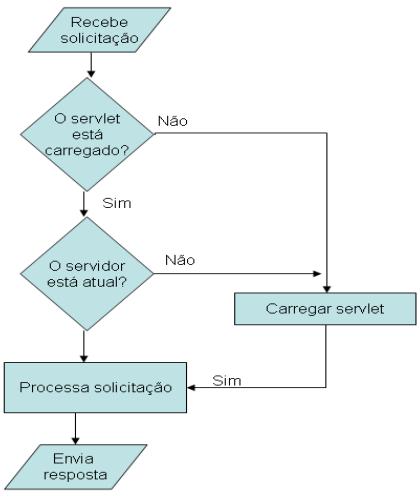

Figura 1: Funcionamento de um Servlet

Fonte: KURNIAWAN, 2002. 7

O servlet permite que por programação em Java se defina todo o código HTML que será mostrado ao cliente. O programador precisa preparar todo o texto que será mostrado ao cliente, sendo este texto estático ou não, como podemos ver no trecho de código na listagem 1.

```
PrintWriter out = response.getWriter();
int x = 50;
out.println("<body>");
out.println("<p>A raiz quadrada de " + x +
                    " é " + Math.sqrt(x) + ".</p>");
out.println("</body>");
```

Listagem 1: Fragmento de código de um servlet

Analisando a listagem 1 fica claro que apesar dos benefícios que a API Servlet trouxe, surgiram alguns problemas:

1. Todo o código HTML precisa ser escrito em Java, mesmo as partes que são totalmente estáticas, isto trazia um trabalho desnecessário em muitos momentos, pois é fácil perceber que é mais trabalhoso gerar o código HTML mediante a comandos Java do que escrever o próprio código em um arquivo HTML.

2. O programador precisa fazer o papel de web designer sem nenhuma ferramenta adequada e para cada ajuste, precisa recompilar a classe do servlet, realizar o *deploy* e executar o teste, dificultando o desenvolvimento e a manutenção.

3. Dificuldade na visualização da aparência da aplicação, pois só é possível vê-la depois, quando o servlet é executado.

4. Falta de separação entre camada de negócio e apresentação.

Uma solução para contornar estes problemas foi o uso de *templates* (modelos) que

mantinham o código HTML. Os *templates* são escritos com marcações especiais, para identificar os locais onde deve ser gerado texto de maneira dinâmica, de modo que o Servlet vai processar este arquivo de *template* e substituirá as marcações especiais pelo conteúdo dinâmico.

Podemos notar o ganho, analisando um possível *template*, na listagem 2, para o exemplo do nosso Servlet.

```
<body>
<p>A raiz quadrada de ${x} %> +
        é ${sqrtX}.</p>
</body>
```

Listagem 2: Fragmento de um *template* processado por um servlet

Neste caso nosso *template* poderia ser construído por um web designer, que colocaria as marcações que seriam processadas e substituídas pelo seu devido valor no servlet. Como o *template* é um arquivo texto comum com o código HTML e as marcações especiais, ele pode ser facilmente editado, usando algum editor HTML, e visualizado sem precisar executar o sistema.

Surgiram vários *frameworks* para facilitar o desenvolvimento com *templates* e entre eles o mais usado foi o Velocity, desenvolvido pelo grupo Apache.

1.2 JavaServer Pages (JSP)

Segundo a Sun Microsystems, a indústria precisava de uma melhor solução para o desenvolvimento de aplicações, e esta deveria suprir as lacunas impostas pelas limitações das

tecnologias alternativas [4]:

- Trabalhar em qualquer servidor WEB ou de aplicação.
- Separar a lógica de negócios da camada de apresentação.
- Permitir um rápido desenvolvimento e teste.
- Simplificar o desenvolvimento.

Como uma evolução natural da abordagem de *templates* para servlets, foi criado o JSP, que foi introduzido na forma de uma extensão da API Servlet 2.1, sendo uma abstração de alto nível para Servlet [2]. Por ser apenas uma camada de abstração, uma página JSP, é transformada em um Servlet automaticamente pelo servidor e é processada pela API Servlet.

Uma página JSP se parece com uma página padrão escrita em HTML ou XML, com elementos adicionais que a *engine* JSP processa antes de enviar a página para visualização. Como podemos observar na listagem 3, aconteceu uma grande mudança, pois agora o código Java foi inserido em uma página HTML, justamente o inverso do que ocorria com os servlets, tornando uma página JSP similar ao que era feito com os *templates*.

```
<%int x = 50; %>
<body>
<p>A raiz quadrada de <%= x %> +
 é <%= Math.sqrt(x) %>.</p>
</body>
```

Listagem 3: Fragmento de código JSP

Apesar da evolução, ainda se mantiveram alguns problemas como por exemplo a mistura de código de apresentação com de negócio. Em situações de páginas mais complexas

a vantagem da arquitetura JSP não fica tão evidente, pois o código acaba ficando menos legível e com manutenção cada vez mais difícil.

Nestas páginas complexas temos novamente o problema de definição entre os papéis do web designer e do programador. Outro problema é a depuração de código, pois como o JSP é convertido automaticamente em um servlet pelo servidor, qualquer exceção que ocorra, será em cima deste servlet que fica de certa forma inacessível ao programador.

1.3 JSP Model 1: JavaBeans, Java Standart Tag Library e navegação descentralizada.

Seguindo a linha da evolução da arquitetura, a Sun criou os JavaBeans e os definiu como sendo componentes reutilizáveis de software que podem ser manipulados visualmente com a ajuda de uma ferramenta de desenvolvimento [2].

Um JavaBean, consiste em uma classe Java comum que precisa ser escrita seguindo algumas convenções [9]:

- A classe deve possuir um construtor público sem argumentos: Isto facilita a instanciação da classe para os *frameworks*.
- As propriedades da classe devem ser acessíveis usando get's e set's: Isto permite aos *frameworks* uma fácil inspeção automática do estado do Bean, bem como a atualização do seu estado.
- A classe deve ser serializável, permitindo às aplicações e *frameworks* a facilidade de

salvar, armazenar e recuperar os Beans, de maneira independente da JVM (máquina virtual Java).

Com o uso do JavaBean no JSP a Sun Microsystems, a criadora do Java, trouxe a possibilidade de uma melhor separação entre as responsabilidades de apresentação e negócio, separando ainda mais o código Java do código HTML. Todo código de negócio estará no JavaBean que será referenciado no JSP por meio de tags, como podemos observar na listagem 4.

```
<jsp:useBean id="bean"
      class="bean.MyBean" scope="request" />
```

Listagem 4: Fragmento de código JSP usando JavaBean

Depois foram criados os sistemas de tags personalizadas, o JSTL (Java Standart Tag Library), aumentando ainda mais a separação de responsabilidades. Essas tags trouxeram recursos de programação como iteração e expressões condicionais, permitindo uma maior independência da utilização de código Java na forma de *scriplets* na página JSP.

Também foi criada a linguagem de expressão que permite recuperar informações dos Bean, bem como realizar operações aritméticas e relacionais. Podemos ver na listagem 5, como a linguagem de expressão junto com as JSTL trouxe mais simplicidade e clareza para o código JSP.

```
<c:if test="${pessoa.sexo == 'M'}">
    <a href="esporte.html">Entrar</a>
</c:if>
```

Listagem 5: Fragmento de código com uso de JSTL e linguagens de expressão

Nesta arquitetura Model 1, a navegação é feita de modo descentralizado através de links entre as páginas, fazendo com que cada página conheça e seja responsável pelo acesso a próxima página. Esta característica da arquitetura acaba trazendo o problema da mistura entre as lógicas de apresentação e navegação, dificultando a reutilização das páginas, pois será preciso fazer operações lógicas condicionais para controlar a navegação de acordo com cada operação diferente.

Figura 2: Navegação entre páginas na arquitetura Model 1

Fonte: COSTA, RUIZ e MELO, 2006. 24

1.4 JSP Model 2: MVC (Model - View - Controller)

O padrão MVC foi criado para resolver o problema do fluxo de navegação não centralizado encontrado no Model 1. Com o MVC temos uma separação distinta entre as camadas:

1. Model: Lógica de negócios e persistência de dados. Representa os dados e as regras de negócio de acesso e modificação destes dados.

2. View: Interface com o usuário. Acessa os dados através do model e especifica como os dados devem ser apresentados. É de sua responsabilidade manter a consistência nessa apresentação quando há mudanças no model, sendo obrigado, quando necessário, a se

comunicar com o model para garantir que esteja sempre com os dados mais atualizados.

3. Controller: Controla os detalhes que envolvem a comunicação das ações do usuário, como pressionamento de teclas e movimento do mouse, à camada Model [9]. Baseado nas iterações com usuário e nas ações do model, o controller responde selecionando o view apropriado.

Conseqüências do desenvolvimento no modelo MVC [11]:

1. Reaproveitamento dos componentes Model: Esta separação entre as camadas de model e view, permite que um mesmo model seja utilizado por múltiplos view.

2. Facilidade em atender a novos tipos de clientes: Para adicionar suporte a um novo tipo de cliente, basta escrever um view e alguma lógica de controller e adicionar na aplicação existente.

3. Aumento da complexidade: É preciso algumas classes extras devido a separação entre model, view e controller.

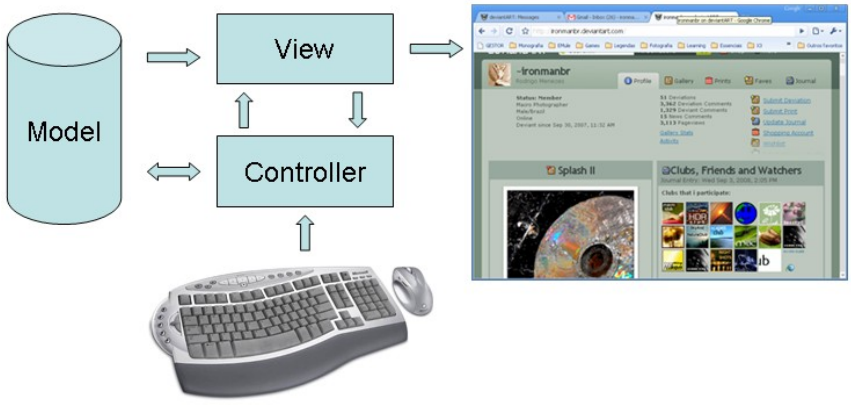

Figura 3: Funcionamento do modelo MVC.

Este padrão geralmente é implementado utilizando um servlet que passa a ser o alvo de todas as requisições, e passa a cuidar do controle de fluxo, redirecionando estas requisições para o model e manipulando a seqüência de apresentação das páginas, tornando o controle de navegação centralizado e resolvendo o problema do Model 1 [2]. Na figura 3 podemos observar o funcionamento do modelo MVC.

Esta arquitetura MVC trouxe enormes benefícios para o desenvolvimento de aplicações em Java e logo surgiram alguns *frameworks* que a implementaram. Estes *frameworks* sugiram com o objetivo de facilitar a construção de aplicações, tentando minimizar o trabalho repetitivo, melhorando a produtividade do desenvolvedor, permitindo que ele se concentrasse mais na lógica de negócio de sua aplicação. O mais popular destes *frameworks* foi o Jakarta Struts, que se tornou praticamente referência em termos de MVC [2]. Mesmo não sendo uma implementação padronizada JEE (Java Enterprise Edition), o Struts foi inclusive recomendado pela Sun na categoria de boas práticas para o desenvolvimento de aplicações Web com Java.

2 JAVASERVER FACES (JSF)

2.1 Introdução

O advento da arquitetura MVC no desenvolvimento de aplicações web utilizando Java foi um caminho sem volta. Ficou muito claro que esta arquitetura trouxe ganhos muito grandes e que era necessária a criação de uma especificação oficial MVC para Java. A comunidade Java através do JCP (Java Community Process), criou um pedido de especificação (JSR 127) para atender esta necessidade.

De acordo com a primeira proposta de especificação para o modelo JSF, foram definidas oito metas que representam o foco principal desta proposta [14]:

1. Criar um conjunto padrão de componentes GUI (interface gráfica do usuário), que possam permitir a criação de ferramentas de desenvolvimento que, facilitem aos usuários a criação tanto de interfaces de alta qualidade quanto, das ligações entre os componentes da interface e o comportamento da aplicação.

2. Definir um conjunto simples de classes base para os componentes GUI, estados de componentes e eventos de entrada. As classes abordarão as questões de ciclo de vida, gerenciando a persistência do estado do componente durante seu tempo de vida.

3. Prover um conjunto comum de componentes GUI, incluindo os elementos de entrada padrão dos formulários HTML. Esses componentes serão derivados do conjunto simples de classes (descritas no item 1) que podem ser usados para definir novos

componentes.

4. Prover um modelo JavaBeans para despachar os eventos invocados, do lado dos componentes de interface do cliente, para o servidor que controlará o comportamento da aplicação.

5. Definir APIs para validação de entrada, incluindo suporte para validações do lado do cliente.

6. Especificar um modelo de internacionalização e localização da GUI. Têm o intuito de adaptar a aplicação para varias linguagens e regiões sem que seja preciso reescrever a mesma. Sendo muito importante, porque não há somente uma mudança de língua, entre as regiões havendo mudanças mais sutis como formato de representação de data e números.

7. Gerar de forma automática a saída adequada ao tipo do cliente, tendo em conta todos os dados disponíveis de configuração do cliente, tais como versão do navegador, etc.

8. Gerar de forma automática a saída contendo os itens obrigatórios para fornecer suporte a acessibilidade, conforme define a Web Accessibility Iniciative (WAI).

A especificação JSF foi desenvolvida por uma comunidade de grandes mentes no campo de desenvolvimento de interfaces do usuário (UI) de aplicações Web. Tentou-se colher as melhores idéias e abordagens dos vários *frameworks* e uni-las de maneira coerente em uma especificação sólida. Um problema comum com as padronizações é que elas acabam se tornando muito complexas para poderem resolver o problema que eles se propuseram.

Para o JSF, o problema é prover um *framework* de componentes de interface de

usuário de fácil utilização, tendo como base um conjunto de tecnologias que não tinham sido desenvolvidas com este propósito (Figura 4). Isto culminou em uma especificação complexa de se implementar, porém bem abrangente que possui muitas definições entre as quais podemos destacar o modelo de componentes que têm seu estado mantido do lado do servidor, tratamento dos eventos disparados na interface pelo cliente, conversão e validação de dados, suporte a internacionalização, definição de ciclo de vida de cada requisição, possibilidade de diversificação de interface com o usuário, entre outros [2].

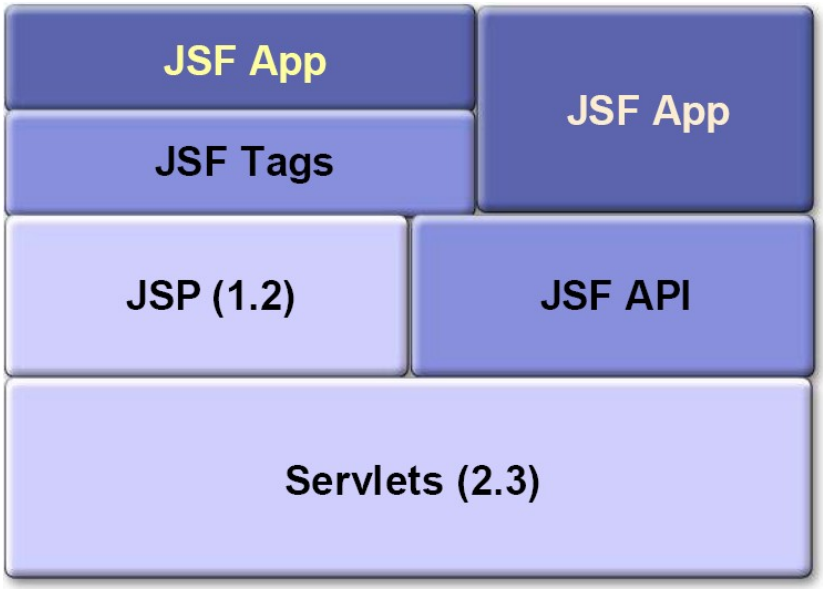

Figura 4: A base da arquitetura JSF

Fonte: GOYAL e VARMA. 8

2.2 As peças do quebra cabeças

Como a maioria das tecnologias, o JSF possui um conjunto próprio de termos, que formam a base conceitual para os recursos que proporciona [14]. São muitos termos como componentes de interfaces (UI), validores, renderizadores, backing beans, entre outros, que

serão encontrados por quem começar a desenvolver utilizando JSF.

Antes de nos aprofundarmos em cada peça deste quebra cabeça, podemos ter uma idéia geral da função de cada um observando a tabela 1.

Tabela 1: Termos importantes do JSF.

Termo	Descrição
Componente UI (também chamado de control ou simplesmente componente)	Um objeto sem estado de conversação associado, mantido no servidor, que fornece uma funcionalidade específica para interagir com um usuário final. Componentes UI são JavaBeans com propriedades, métodos e eventos. Elas são organizadas em uma estrutura, que é uma árvore de componentes geralmente apresentados como uma página.
Renderizador	Responsável por exibir um componente da UI e transformar uma entrada do usuário em um valor do componente. Renderizadores podem ser projetados para funcionar com um ou mais componentes UI, um componente UI pode ser associado a diversos renderizadores.
Validador	Responsável por garantir que o valor digitado por um usuário é aceitável. Um ou mais validadores podem ser associados com um componente da UI.
Backing beans	Uma especialização do JavaBeans que coleta valores a partir de componentes da UI e implementam métodos dos eventos dos ouvintes. Eles também podem possuir referências a elementos da UI.
Conversor	Converte o valor de um componente para String e também executa a operação reversa. Um componente UI pode ser associado com apenas um conversor.
Eventos e ouvintes(listeners)	JSF usa o modelo JavaBean de evento/ouvinte(também usado pelo Swing). Componentes UI (e outros objetos) geram eventos e ouvintes podem ser registrados para lidar com esses eventos.
Mensagens	Informações que são exibidas de volta para o usuário. Em qualquer parte da requisição (backing beans, validadores, conversores, e assim por diante) mensagens de erro podem ser geradas ou podem ser exibidas informações para o usuário.
Navegação	A capacidade de passar de uma página para a próxima. JSF possui um poderoso sistema de navegação que é integrado com os ouvintes dos eventos.

Fonte: Traduzido de MANN, 2005. 39

2.2.1 Componentes UI

Segundo a Wikipedia, interface do usuário é um conjunto de características com o

qual as pessoas (usuários), interagem com o sistema (uma máquina, dispositivo, software ou alguma outra ferramenta complexa). Para interagir em um sistema, os usuários necessitam ter controle sobre o sistema, e também ter indicativos do estado atual do mesmo [15].

Como exemplos, ao dirigir um carro o motorista usa o volante para controlar a direção do carro, acelerador, freio e marchas para controlar velocidade. Para obter a posição do veículo em relação ao mundo exterior, o motorista olha através das janelas e pelos retrovisores. Ele pode avaliar sua autonomia pelo indicador de combustível e sua velocidade pelo velocímetro. Então a UI do automóvel, consiste em todos instrumentos que permitem ao usuário conseguir utilizar o automóvel.

No JavaServer Faces os componentes UI, têm a função de interagir com o usuário final do sistema. Como estão construídos sobre a arquitetura JavaBeans, possuem propriedades, métodos e eventos e suporte das IDE, suporte este que facilita muito o desenvolvimento, permitindo ao desenvolvedor ter acesso a todas as características do componente.

Componentes UI podem ser um conjunto de subcomponentes que permite que o desenvolvimento se torne muito mais fácil e rápido, pois este componente irá utilizar tudo que está pronto nos outros componentes. Um exemplo prático, seria a criação de um componente de endereço, que poderia ser composto por campos de entrada para os dados como rua, CEP e bairro e por uma combo para representar a unidade da federação.

Para o desenvolvedor deste componente de endereço, todo o trabalho na construção dos elementos básicos como entrada de texto e combos, já estará feito, permitindo que este desenvolvedor se concentre na arquitetura de seu componente. Pensando mais a

frente, outro desenvolvedor que for utilizar este componente, irá abstrair todo o trabalho que foi feito para a construção do mesmo, e concentrará seus esforços apenas na construção da página.

Esta arquitetura, além de permitir a reutilização de código, permite também que o trabalho seja melhor dividido, permitindo que um desenvolvedor se especialize em uma parte do trabalho. Para o desenvolvedor que utilizará o componente pronto, não haverá o pré-requisito que o mesmo possua domínio em HTML e Javascript. Ele precisará apenas saber quais as propriedades deste componente, para poder fazer o uso do mesmo.

Os componentes possuem uma separação entre seu funcionamento e sua aparência que é definida pela maneira como ele é renderizado. Esta separação permite a modificação da aparência, sem modificar seu comportamento e funcionamento, permitindo uma flexibilidade muito grande, pois uma aplicação pode ter sua aparência toda mudada sem precisar modificar a maneira como os componentes são utilizados na programação.

Mesmo tendo funcionamento e modelos de arquiteturas parecidos, há uma diferença chave entre o funcionamento dos componentes UI JSF (Web) e Swing (Desktop). Uma aplicação desktop possui um comportamento mais local, e a aplicação interage direto com a máquina do cliente, fazendo com que todos os dados permaneçam na máquina do usuário. Em uma aplicação web há um comportamento diferente, pois os dados informados pelo usuário são trasportados ao servidor. Devido a está diferença, os componentes no lado do servidor precisam lembrar seus valores, pois em um erro de preenchimento de formulário por exemplo, a página é renderizada novamente informando a mensagem de erro.

O *framework* JSF mantém uma árvore com os componentes UI, chamada de view,

que permite que cada componente lembre seus valores entre as requisições [13]. A view é a representação interna do JSF para a página e mantém uma estrutura hierárquica de pai e filho (figura 5), como por exemplo um formulário contêm um uma combo e esta combo possui ítens.

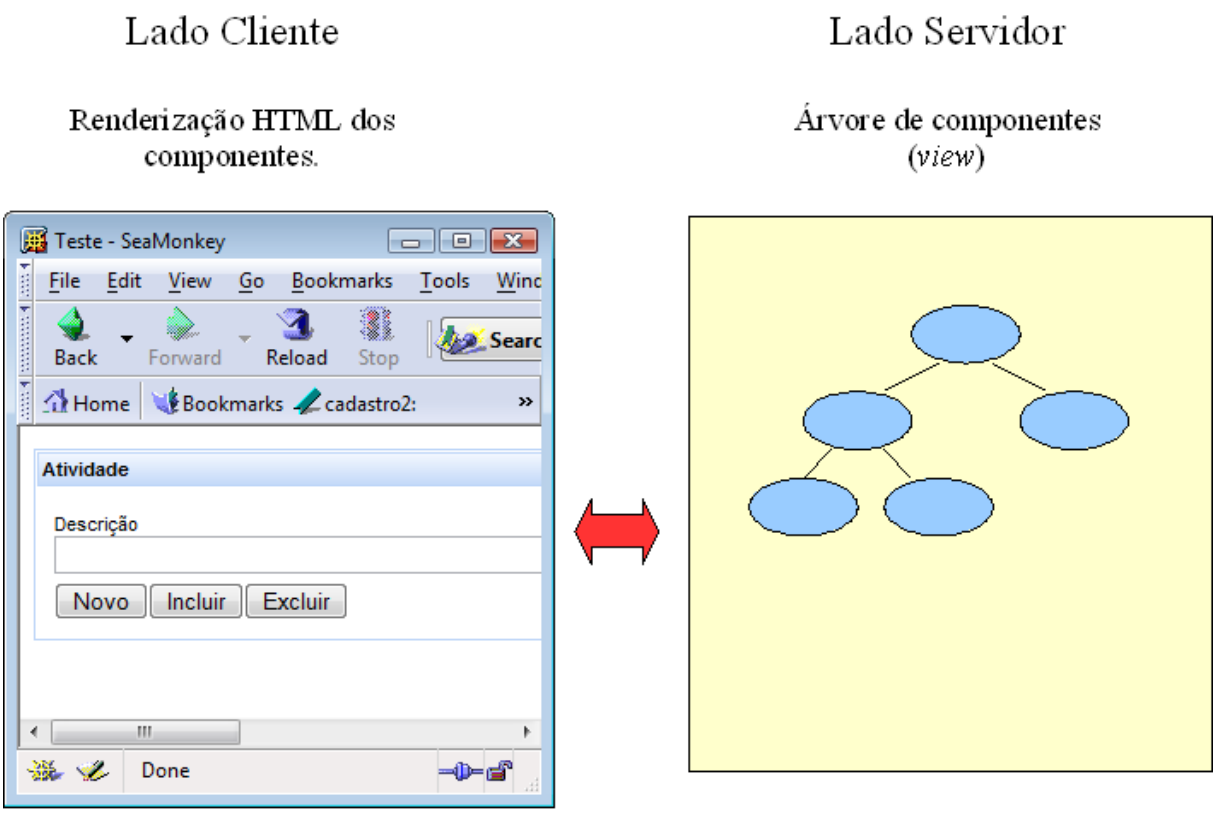

Figura 5: Os componentes UI são mantidos no servidor em um view, e são renderizados em HTML ou outra linguagem de apresentação. Adaptado de Mann [13]

Cada componente possui uma propriedade de nome ID que o identifica unicamente e caso o desenvolvedor não defina um valor para esta propriedade de identificação, será definido um ID automaticamente para o componente.

2.2.2 Renderizadores

O JSF suporta dois modelos de renderização, o modelo de implementação direta

no qual os componentes são responsáveis pela própria renderização e o modelo de implementação delegada, em que classes distintas manipulam o processo de renderização.

Essas classes recebem o nome de renderizadores e estão organizadas em render kits, que costumam focar em um tipo especifico de saída. A implementação JSF disponibiliza um Render Kit HTML, mas outros kits podem ser implementados para interfaces diferentes, como dispositivos móveis.

O renderizador trabalha como um tradutor que opera entre o cliente e o servidor. Na direção do servidor para o cliente, ele trabalha codificando a representação dos componentes em uma linguagem que o cliente entenda. Na direção oposta, o renderizador recebe os parâmetros que foram passados pela requisição e aplica esses valores aos componentes. Na listagem 6 temos um exemplo JSF que representa um Radio para seleção de sexo em um formulário e na listagem 7 podemos observar o código HTML equivalente que foi gerado pelo renderizador.

```
<h:selectOneRadio id="sexo" required="true">
   <f:selectItem itemLabel="Masculino" itemValue="M"/>
   <f:selectItem itemLabel="Feminino" itemValue="F"/>
</h:selectOneRadio>
```

Listagem 6: Fragmento de código JSF

```
<input type="radio" name="formAtividade:sexo"
       id="formAtividade:sexo:0" value="M" />
         <label for="formAtividade:sexo:0"> Masculino</label>
<input type="radio" name="formAtividade:sexo"
       id="formAtividade:sexo:1" value="F" />
         <label for="formAtividade:sexo:1"> Feminino</label>
```

Listagem 7: Fragmento de código HTML gerado pelo renderizador HTML do JSF.

2.2.3 Validadores

Um dos grandes problemas ao desenvolver aplicações, é a validação dos dados informados pelo usuário, que pode ir desde uma simples validação de um campo obrigatório a uma validação mais complexa que envolve regras de negócio. Há duas abordagens clássicas, que são: usar validação por javascript no cliente e com lógica Java no servidor.

Usar javascript no cliente tem seus benefícios, fazendo com que o cliente gaste processamento nas validações economizando este processamento no lado do servidor, mas trás inúmeros problemas. Existem técnicas de se modificar o conteúdo de uma página, então não se pode confiar apenas em uma validação Javascript, precisando existir também uma validação do lado do servidor. Um bom exemplo do que pode acontecer, é o desenvolvedor marcar um campo como somente leitura e um usuário com maior conhecimento técnico, alterar as propriedades deste campo, permitindo modificar o valor dele. Atualmente realizar algo assim é bastante simples, utilizando o navegador FireFox com alguma extensão como Greasemonkey e FireBug como pode ser observado no *script* para o Greasemonkey da listagem 8.

```
// ==UserScript==
// @name           Teste segurança
// @namespace      http://diveintogreasemonkey.org/download/
// @description    Permite edição em todos os campos que estão em somente
leitura ou debilitados
// @include        *
// ==/UserScript==

var allImputs = document.getElementsByTagName('input');

for (var i = 0; i < allImputs.length; i++) {
    var thisInput = allImputs[i];
      thisInput.readOnly = false;
      thisInput.disabled = false;
      thisInput.setAttribute('onblur', '');
}
```

Listagem 8: Script para o Greasymonkey que permite a edição em campos de uma página que estejam em modo somente leitura, ou desabilitados

Outro problema que aflige a parte de validação tanto por javascript quanto por lógica em Java, é que muitas vezes acaba-se criando uma lógica cheia de comandos condicionais, tornando o código confuso e de difícil manutenção.

Por estes problemas citados, fica muito clara a importância de um framework para ajudar na validação de dados e o JSF lida com validação de dados de três maneiras: no componente, através de métodos de validação nos backing beans ou com classes de validação que são os chamados validadores. Na validação no componente, temos geralmente uma validação básica como a de obrigatoriedade de um campo, ou checar se o usuário informou uma data válida. Métodos de validação são úteis quando a validação depende de mais um componente, ou há uma lógica mais complexa envolvida e está lógica é especifica e não precisa ser compartilhada com outras páginas.

As classes de validação, são muito úteis para validações genéricas pontuais, como checar o tamanho de uma String, ou chegar se a data é menor que atual. São validações pontuais e específicas que podem ser encadeadas, adicionando mais de um validador por

componente. Esta abordagem é muito eficiente e garante um reuso muito grande de código, alem de ser de fácil utilização, como podemos observar na listagem 9, onde é usado um validador que verifica se a String informada no campo possui tamanho maior que nove e menor que duzentos. Ainda no exemplo da listagem 9, poderíamos adicionar outros validadores como por exemplo algum que verificasse se há caracteres proibidos no texto.

```
<h:inputText id="nome" size="150">
      <f:validateLength minimum="10" maximum="200" />
</h:inputText>
```

Listagem 9: Fragmento de código JSF mostrando uso de um validador.

Pela especificação JSF toda validação é feita no lado do servidor e quando um validador encontra um erro como uma String maior que o permitido ou um CPF inválido, é adicionada uma mensagem de erro à lista de mensagens atual, tornando fácil mostrar para o usuário estas mensagens.

Ficou claro que o JSF trouxe facilidade na área de validação, que sempre foi algo tedioso para o programador. Junto com a implementação do JSF vêm um conjunto de validadores, mas o programador pode escrever os seus próprios validadores, e estes provavelmente serão muito utilizados em outras páginas do projeto ou em até muitos outros projetos, permitindo a criação de bibliotecas de componentes de validação.

2.2.4 Backing Beans

Dá-se o nome de backing beans aos objetos que fazem a ligação entre os componentes UI e a camada model. Eles geralmente contêm propriedades que precisam ser obtidas do usuário e ouvintes de eventos. Estes ouvintes, processam as propriedades dos

backing beans e podem, além disto, manipular o componente UI ou executar algum processamento na aplicação.

Um backing bean pode ser associado a um componente UI de maneira declarativa. A associação é feita usando linguagem de expressão e pode-se apontar diretamente a alguma propriedade do backing bean, como podemos ver na listagem 10, onde esta sendo feita uma associação entre o componente e o atributo 'nome' do bean. Com esta associação, temos uma sincronia, caso o valor seja mudado no componente, a propriedade do bean terá seu valor modificado e recíproca também ocorre.

```
<h:inputText id="nome" size="150" value="#{atividadeBean.nome}">
      <f:validateLength minimum="10" maximum="200" />
</h:inputText>
```

Listagem 10: Fragmento de código JSP mostrando a associação entre um componente UI e um back bean

Outra operação que pode ser feita, é associar diretamente uma propriedade do back bean com uma instância do componente no lado do servidor, utilizando a propriedade 'binding' como mostra a listagem 11. Isto permite que o componente seja manipulado através de código Java.

```
<h:panelGrid id="controlPanel"
      binding="#{atividadeBean.controlPanel}"
      columns="20" border="1" cellspacing="0"/>
```

Listagem 11: Fragmento de código JSF, mostrando a associação, usando a propriedade binding

2.2.5 Conversores

Os usuários do sistema interagem com ele por meio dos componentes UI. Os

renderizadores agem de modo que o componente seja renderizado de maneira adequada e faça sentido. Mas os componentes podem estar associados também a propriedades do back beans que podem ser de qualquer tipo, desde tipos padrão da linguagem Java (tais como String, Date, int) até algum tipo definido pelos desenvolvedores da aplicação [13].

Para quem já trabalhou com tecnologias web, fica mais claro o problema que os conversores vieram resolver. Ao se trabalhar com JSP por exemplo um programador, ao capturar um valor que foi informado pelo usuário em um campo de entrada de texto, teria de se preocupar com a conversão desse tipo, caso o campo não representasse uma String.

Esse problema ocorre porque, em uma requisição, os valores informados e submetidos em um formulário são do tipo String, sendo o programador responsável por fazer a conversão da String para o tipo apropriado depois da submissão. O problema não para por aí, pois o programador precisa fazer a operação inversa, quando for mostrar os dados processados, precisando transformar esse dado em uma String que represente algo que o usuário entenda. Isto pode ser observando analisando o código JSP da listagem 12, onde o programador precisou fazer as conversões do objeto dtNascimento de String para java.util.Date para obter o dado e realizou a operação inversa para mostrar o dado ao usuário.

```
<%
	String nome = request.getParameter("nome");
	Date dtNascimento = null;
	SimpleDateFormat dateFormat = new SimpleDateFormat("dd/MM/yyyy");
	if (request.getParameter("dtNascimento") != null) {
		dtNascimento = dateFormat.parse(
				request.getParameter("dtNascimento"));
	}
%>
<body>
	<form action="resposta.jsp" name="formCadastro">
		Nome:
		<input type="text" name="nome"
			value="<%= nome != null ? nome : "" %>"
			size="150" maxlength="150"/><br/>
		Data de Nascimento (dd/mm/yyyy):
		<input type="text" name="dtNascimento"
			value="<%= dtNascimento != null ?
dateFormat.format(dtNascimento) : ""%>"
			size="10" maxlength="10"/>
		<br/>
		<br/>
		<input type="button" value="Atualizar Dados"
			onclick="document.forms['formCadastro'].submit()"/>
	</form>
</body>
```

Listagem 12: Fragmento de código JSP que mostra o problema de conversão de tipos que um programador enfrenta.

Para resolver o problema acima, surgiu o *converter* (conversor) que tem a função de fazer essas duas operações de conversão. O JSF por padrão já vem com conversores para os tipos comuns do Java tais como java.util.Date, String, Integer, entre outros, mas também permite que o programador desenvolva seus próprios conversores para os tipos que criar.

Outro problema que podemos observar na listagem 12 é que a data informada está no padrão de formatação que usamos no Brasil (dd/mm/aaaa), então nossa aplicação seria confusa para alguém de outro país, como um norte americano que usa data no formato (mm/dd/aaaa). Para resolver isto o programador precisaria se preocupar em obter do navegador qual é a localização do usuário e fazer as conversões no formato correspondente. Os conversores no JSF podem lidar com estes problemas de formatação e localização,

facilitando em mais outro aspecto a tarefa de programar.

2.2.6 Eventos e Ouvintes

Nas interfaces de usuário, os eventos capturam a interação do usuário com os componentes, sendo este evento algo como um simples clique em um botão ou link, ou algo mais complicado que execute alguma regra de negócio. No JSF o modelo de eventos tem seu funcionamento bastante semelhante ao modelo de eventos do Swing e fica sobre responsabilidade de um JavaBean lidar com os eventos, usando objetos de eventos e ouvintes e um componente pode possuir um, vários ou nenhum ouvinte para manipular seus respectivos eventos.

O tratamento de eventos sempre foi algo complexo na programação web, precisando o programador tratar as submissões e requisições geradas pelas ações do usuário ao interagir com a aplicação, gerando eventos, e aplicar a lógica de negócios correta. No JSF não se pensa em submissões e requisições, mas somente nos eventos que são divididos em quatro tipos: eventos de mudança de valor, eventos de ação, eventos de modelos de dados e evento de fase.

2.2.6.1 Eventos de mudança de valor

São eventos gerados por campos de entrada de dados quando o usuário entra como um novo valor em algum campo e um ouvinte fica responsável por este evento.

Na listagem 13, podemos observar como se associa um evento de mudança de

valor a um componente UI utilizando a *tag* valueChangeListener. A expressão desta *tag* faz referência a um método ouvinte em um back bean.

```
<h:inputText id="nome" size="150"
    valueChangeListener="#{atividadeBean.processChange}">
</h:inputText>
```

Listagem 13: Fragmento de código mostrando um evento de mudança de valor.

2.2.6.2 Eventos de Ação

São os eventos acionados quando um usuário interage com um controlador que representa um comando, tais como um botão ou *hiperlink*.

A responsabilidade de lidar com estes eventos fica a cargo dos ouvintes de ação que se dividem em dois tipos: os que afetam e os que não afetam a navegação. Os que afetam a navegação costumam realizar algum processamento e obter uma condição lógica que determina qual a próxima página que será selecionada pelo sistema de navegação JSF. Os que não afetam a navegação comumente manipulam componentes ou são responsáveis por processamentos que mudam valores dos objetos model ou back bean [13].

2.2.6.3 Eventos de Modelos de Dados

São disparados quando um componente que, alimentado diretamente com múltiplas linhas de dados, processa uma destas linhas. O uso mais comum é feito em um componente que mostra uma lista selecionável de itens como uma combo, onde, ao ser processada cada linha, é disparado um evento de modelo de dados.

Eles funcionam de maneira ligeiramente diferente porque eles são disparados por uma instância no DataModel em vez de um componente UI. Este DataModel é utilizado internamente por estes componentes e são *wrappers* para fontes de dados como listas, vetores e *result sets.*

2.2.6.4 Evento de Fase

Uma aplicação JSF possui um ciclo de vida de requisição que é dividido em seis fases. Cada fase possui uma responsabilidade e finalidade tais como restaurar o view requisitado, executar validações, fazer a conversão dos valores dos campos de entrada em valores nos componentes, entre outras.

Um evento de fase é gerado antes e depois de cada um destes passos e é necessário implementar uma interface Java para registrar os eventos. Estes eventos são utilizados geralmente internamente pelo JSF, mas estão ao dispor do programador para que implemente algo necessário a uma aplicação.

Entre algumas possibilidades de utilizações, podemos mensurar o grande beneficio de se usar eventos de fase para identificar gargalos de desempenho em um sistema, registrando a hora de entrada e saída de cada uma das fases do ciclo de vida da requisição. Desde modo podemos identificar qual a fase que está demorando mais ao ser processada.

2.2.7 Mensagens

Em uma aplicação web é necessário o envio de mensagens de texto em resposta às

interações do usuário com a aplicação. Estas mensagens podem ter muitas finalidades, desde meramente informativas para auxiliar o usuário, até mensagens de alerta, informando ao usuário que o mesmo não está utilizando corretamente o sistema.

Este problema cresce junto com o tamanho das aplicações e número de programadores envolvidos, pois é desejável que mensagens sigam um padrão consistente por toda a aplicação.

Outro problema se dá ao fato que atualmente muitos sistemas precisam atender a usuários que falam idiomas diferentes e seguem padrões diferentes como o de representação de data e números aumentando o problema com a padronização das mensagens.

As mensagens mais comuns são as de erro e se dividem em erros de aplicação e erros de entrada de dados do usuário. Os erros de aplicação, geralmente são mais graves e redirecionam para alguma página informando o problema e pedindo para tentar novamente mais tarde, ou entrar em contato com o suporte do sistema. Quando é um erro de entrada de dados, o comum é mostrar a página novamente, e informar qual foi o erro, como uma data inválida, ou algum campo requerido não preenchido.

O JSF possui o sistema de mensagens para auxiliar nestes problemas e podem ser automaticamente geradas no idioma do usuário. Uma mensagem nada mais é que um conjunto de dados tais como um texto resumido, um texto detalhado e um nível de severidade que nas mensagens de erro informa a gravidade do erro.

As mensagens podem ser associadas a um componente especifico, caracterizando os erros de entrada de dados ou não possuir associação e assumir o papel de uma mensagem

de erro de aplicação.

As mensagens fazem parte do sistema de validadores e conversão de tipos de dados do JSF, e onde ocorrer um erro em alguma destas fazes, será gerada uma mensagem.

2.2.8 Navegação

Em uma aplicação normalmente existem muitas páginas e de alguma maneira, ao operar o sistema, o usuário navegará entre diversas páginas. Nós vimos que antes do MVC o sistema de navegação era descentralizado, cabendo a cada página saber qual a próxima página que será mostrada ao usuário. Este sistema descentralizado trás alguns problemas e o mais obvio é quando se faz necessária alguma regra de navegação, implicando na utilização de testes condicionais para decidir qual será a próxima página.

O JSF resolveu este problema criando um controlador de navegação que possui a responsabilidade de decidir qual será a próxima página, baseado nas ações do usuário. Para qualquer página, um sistema de regras de navegação define como será feita a navegação, sendo estas regras configuradas em arquivos específicos e/ou programadas na aplicação.

2.3 Ciclo de vida

Como foi comentado nos eventos de fase, o JSF possui um ciclo de vida de requisição baseado em fases, que são executadas seguindo uma ordem. Para um desenvolvedor não é obrigatório que ele conheça a fundo o funcionamento interno do JSF, mas é um diferencial muito importante, ter um conhecimento sobre o ciclo de vida, sabendo a

ordem e o que acontece em cada uma destas fases. Este conhecimento permite que se sejam construídas aplicações melhores, pois o desenvolvedor sabe com detalhes que operações estão acontecendo em cada fase, tornando mais fácil inclusive detectar um erro, ao se identificar em qual fase ele ocorreu. Podemos entender melhor este ciclo de vida observando a figura 6.

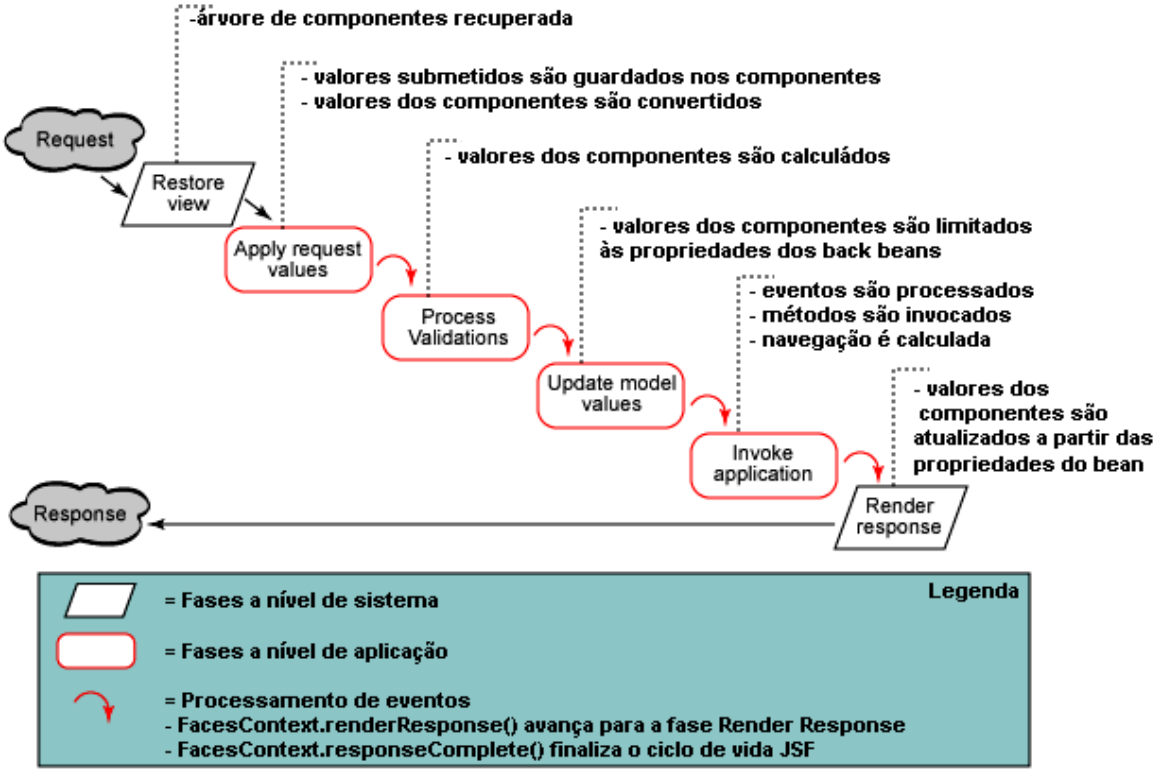

Figura 6: Ciclo de vida de uma requisição JSF

Fonte: HIGHTOWER, 2008.

Uma das vantagens deste mecanismo de fases é permitir que um desenvolvedor concentre seus esforços em determinadas fases. Por exemplo, quem for trabalhar mais com desenvolvimento da aplicação, estará mais concentrado nas fases: Apply resquest values, process validations, update model values e invoque aplication. Já para quem for trabalhar mais com desenvolvimento de componentes as fases mais envolvidas são: Restore view e render response.

2.3.1 Fase 1: Restore View

No JSF a view representa uma árvore com todos os componentes de uma página e a principal função da fase Restore View é inicializar o ciclo de vida de processamento da requisição, tentando obter a informação de estado para construir a árvore de componentes que representa a requisição.

Uma requisição chega através do controlador FacesServlet, que examina e extrai o ID da view. Com este ID é feita um busca, na sessão do usuário, pela view e caso não seja encontrada, é criada uma.

Caso seja a primeira vez que o usuário faz a requisição da página, não existirão dados submetidos pelo usuário, então o JSF vai pular as próximas fases, indo direto para a Render Response que é a ultima fase. Caso não seja a primeira requisição da página, o JSF aplicará na view os dados informados pelo usuário e seguirá para a próxima fase, Apply Request Values.

2.3.2 Fase 2: Apply Request Values

Nesta fase o valor local de cada componente é atualizado pela requisição atual. Os componentes precisam ser restaurados ou criados e seus valores são recuperados, na maioria das vezes através dos parâmetros da requisição, mas também podem ser recuperados de *cookies,* cabeçalhos e da sessão HTTP.

Outra função desta fase é fazer a conversão de dados, de modo que os valores passados na requisição, sejam convertidos em seu tipo esperado. Nesta fase atuam os

conversores, que são responsáveis por este trabalho e caso ocorra um erro é criada uma mensagem que é colocada na fila de mensagens para ser mostrada ao usuário na fase Render Response.

Esta fase pode se comportar de maneira um pouco diferente se a propriedade immediate estiver com o valor 'true'; neste caso, além da conversão dos dados é assumida a tarefa de validação dos dados que é de responsabilidade da fase Process validation. Caso ocorra um erro na conversão ou validação, também é gerada e enfileirada uma mensagem na fila de mensagens.

2.3.3 Fase 3: Process Validation

Nesta fase, a árvore de componentes é percorrida e é feito um teste para saber se o valor é aceitável. A responsabilidade de validação pode ser atribuída ao próprio componente, ou a um ou mais validadores registrados.

Caso todos os componentes sejam validados com sucesso, o processo avança para fase update model, mas caso um ou mais componentes possuam valores inválidos, é gerada uma mensagem de erro para cada componente e o JSF pula para a fase render response que mostrará a view atual com todas as mensagens de erro.

2.3.4 Fase 4: Update Model Values

Aqui são atualizadas as propriedades, dos back beans, que possuem ligação com algum componente. Como esta fase só é alcançada caso não ocorra nenhum erro de validação

ou conversão, pode-se assegurar que os valores que serão atualizado estão válidos [16].

Vale ressaltar que mesmo os valores estando válidos com as regras de conversão e validação, nada impede que eles estejam inválidos para alguma regra de negócio da aplicação. Um exemplo prático é uma validação de CPF, que pode comumente usar um algorítimo para checar se o número informado é um numero de CPF no formato válido, mas para nossa regra de negócio pode não bastar o número possuir um formato válido, sendo obrigatório que este número represente um CPF existente.

2.3.5 Fase 5: Invoke Application

Ao chegar nesta fase temos a certeza que nossos dados foram convertidos, validados e usados para atualizar os back beans, então é agora que o JSF executa as lógicas de negócio da aplicação.

Os eventos de ação padrão de cada componente e os que foram restaurados na primeira fase, serão agora enviados aos ouvintes de ação correspondentes.

Primeiro serão executados os eventos de ação recuperados na fase restore view, depois o JSF executará os eventos padrão e depois que tudo isso é feito, a requisição segue para a próxima fase.

2.3.6 Fase 6: Render Response

Chegamos na última fase do ciclo de vida da requisição sabendo que todo o processamento de nossa requisição foi feito, faltando apenas mostrar o resultado ao usuário.

Mostrar o resultado ao usuário é a principal função desta fase, mas ela também grava o estado da view, para que ela possa ser recuperada na fase restore view caso seja necessário.

Podemos ressaltar que nesta fase os conversores precisam trabalhar de maneira inversa ao que ocorre na fase apply request values. Aqui os dados são convertidos em String, utilizando uma formatação que o usuário entenda.

3 BIBLIOTECAS DE COMPONENTES JSF

3.1 Introdução

Como foi visto durante este trabalho, a especificação JSF foi criada para prover um *framework* de programação web poderoso, confiável e escalável, construído em cima da tecnologia Java, que tem se mostrado ao longo dos anos ser bastante confiável, o que explica sua maciça adoção.

Mesmo os requisitos acima sendo fundamentais, era claro que o JSF precisava ter ainda uma outra característica, a de ser bastante produtivo. E entre os pilares que fornecem esta produtividade, podemos destacar os componentes UI.

Como foi visto na evolução da tecnologia Java, criar a interface com o usuário, sempre foi uma tarefa complicada e isto fica claro pelo número de evoluções que foram necessárias neste sentido.

O JSF definiu um modelo de componentes, visando facilidade de uso e produtividade, e na sua implementação padrão trouxe implementação para todos os componentes básicos HTML. Este conjunto básico de componentes serve como pontapé inicial no desenvolvimento das aplicações e em muitos casos são suficientes, mas também é permitindo ao desenvolvedor criar seus próprios componentes, que depois de criados poderão ser utilizados em muitos outros projetos.

Em uma fábrica de software, é comum que alguns desenvolvedores fiquem

responsáveis por alguma tarefa específica, e em muitos casos há desenvolvedores responsáveis pela criação e manutenção de componentes. A partir desta grande necessidade por componentes sofisticados, foi esperado o surgimento de projetos que direcionassem seus esforços para a construção de bibliotecas de componentes, o que levou ao surgimento de várias alternativas bastante maduras e com muita qualidade.

3.2 Escolha das bibliotecas estudadas

Existem vários projetos de criação de bibliotecas de componentes ao redor do mundo e muitas fabricas de software possuem inclusive suas próprias bibliotecas. Mas como é de esperar, alguns projetos sempre se destacam em sua área, sejam por ter uma equipe excelente, gerenciamento competente e boas idéias; eles acabam atingindo um patamar de qualidade acima do padrão.

Na área de bibliotecas de componentes não foi diferente, e objetivo principal deste trabalho foi selecionar e fazer um estudo comparativo destes projetos. Entre os critérios utilizados para a escolha das bibliotecas, ficam destacados:

1. Aceitação pela comunidade Java.
2. Acabamento visual.
3. Documentação.
4. Número de componentes.

3.3 RichFaces

A RichFaces é uma biblioteca bastante popular que foi desenvolvida inicialmente pela Exadel, que em março de 2007 firmou uma parceria com a Red Hat, tornando open source alguns de seus produtos tais como RichFaces, Ajax4jsf e Studio Pro e deixando-os sob responsabilidade da JBoss que pertence ao grupo Red Hat [17].

O Ajax4jsf é um framework que permite a utilização de Ajax de forma fácil e integrada com o JSF. Neste estudo, vamos considerar o Ajax4jsf algo integrante da biblioteca Richfaces, pois na prática é isso o que acontece, sendo o RichFaces construído em cima do Ajax4jsf.

Em meados de 2005 Alexander Smirnov, procurando por algo novo, observou com bastante atenção duas tecnologias, o AJAX e o JSF e imaginou como seria interessante unir as duas tecnologias, podendo então tornar fácil o uso de funcionalidades AJAX em uma aplicação JSF. Ele inciou o projeto no sourceforge.net, o maior portal de software de código aberto do mundo, e o chamou de Telamon [18].

Ao final de 2005 Alexander se tornou funcionário da Exadel e continuou o desenvolvimento de seu *framework,* e tinha como meta criar algo que fosse fácil de usar e que pudesse ser usado com qualquer biblioteca de componentes JSF. Em março de 2006 a Exadel lançou um produto completo chamado Exadel RichFaces, que possuía como um de seus componentes o *framework* de Alexander. Ainda em 2006 ocorreu a fusão com o Exadel RichFaces e nasceu o Ajax4jsf [18].

O Ajax4jsf nasceu como um projeto de software livre hospedado na java.net

enquanto o RichFaces se tornou uma biblioteca de componentes comercial e em 2007, como já foi dito, os dois projetos foram incorporados a JBoss.

Segundo a Jboss, o RichFaces possui um diferencial em relação aos concorrentes por permitir uma abordagem de AJAX orientada por página ao invés, do mais comum, que é orientado a componente [17]. Este diferencial permite que seja definido um evento que faça uma requisição AJAX em particular e podem ser definidas áreas nesta página que devem ser sincronizadas com a árvore de componentes JSF, ao contrário da outra abordagem, no qual tudo é centralizado no componente.

3.4 ICEfaces

A biblioteca ICEFaces, seguiu um caminho parecido com a RichFaces, sendo desenvolvida pela ICESoft, inicialmente como um produto fechado em 2005, mas mudou sua licença para código aberto em novembro de 2006.

Tal como o RichFaces/Ajax4jsf, o ICEFaces consiste de um *framework* AJAX para JSF somado a uma biblioteca de componentes. Sua comunidade conta com mais de trita mil desenvolvedores ao redor do mundo e tem como base seu *forum* e seu sítio, icefaces.org, sendo administrado pelos seus criadores [19].

Entre suas características, podemos destacar a Ponte Ajax, que facilita o mecanismo de atualização incremental da camada de aplicação, usando um mecanismo de submissão parcial. O mecanismo de submissão parcial está presente na arquitetura dos componentes, permitindo ao desenvolvedor ter controle deste mecanismo a nível de componente [21].

3.5 MyFaces Tomahawk

Diferente das bibliotecas RichFaces e ICEfaces, o MyFaces Tomahawk já nasceu como software de código aberto e depois acabou sendo incorporado como um projeto do grupo Apache, que sempre foi muito respeitado na comunidade Java por seus inúmeros projetos tais como o servidor Web Apache, o servidor Web Java Tomcat, e *framework webservice* Axis, entre outros.

O Tomahawk é um subprojeto do MyFaces, que é a implementação JSF feita pelo grupo Apache. Segundo a Apache [22], mesmo o projeto se comprometendo a ser 100% compatível com a implementação JSF da Sun Microsystems entretanto, nem todas as versões do Tomahawk são compatíveis com a implementação JSF da Sun ou até mesmo com versões mais antigas da MyFaces. É garantido que toda versão do Tomahawk é sempre compatível com a ultima versão do MyFaces.

Além de adicionar novos componentes, foram adicionadas funcionalidades a alguns dos componentes básicos da especificação JSF.

4 ESTUDO COMPARATIVO ENTRE AS BIBLIOTECAS

4.1 Introdução

Este trabalho se propõe a fazer um estudo comparativo estre as três bibliotecas selecionadas e, para isso, adotará uma metodologia comparativa, definindo os critérios que serão analisados.

Primeiramente selecionaremos a maior quantidade possível de componentes que possuem correspondentes semelhantes entre as três bibliotecas, como calendários, envio de arquivos, entre outros. Feita esta seleção inicial, serão escolhidos outros componentes que não sejam implementados por todos, usando como critério a utilidade deste componente.

No nosso teste foram utilizados projetos de demonstração que são fornecidos para download pelos fabricantes, ou disponibilizados para uso pela Internet [24] [25] [26]. Neste inicio houve uma decepção quando ao Tomahawk, que não possuía um demonstrativo oficial de seus componentes, então foi usado um demonstrativo encontrado na jsfmatrix.com, que é um sitio que trás uma tabela comparativa entre bibliotecas de componentes JSF.

Neste comparativo foram usadas as seguintes versões das bibliotecas: RichFaces 3.2, ICEFaces 1.7 e Tomahawk 1.1.7.

Na análise dos componentes vários critérios serão avaliados tais como:

1. Usabilidade.

2. Uso de AJAX.

3. Compatibilidade com os navegadores: Microsoft Internet Explorer 7 e Firefox 3.

4. Acabamento e apelo visual.

5. Diferenciais perante os concorrentes.

Com a premissa de enriquecer o comparativo, foi feita uma qualificação dos componentes. O critério de pontuação foi elaborado seguindo os itens citados anteriormente, escolhendo uma classificação para cada um dos itens que pode ser observada na tabela 2.

Tabela 2: Classificação proposta para quantificar a qualidade dos atributos de cada componente estudado.

Termo	Significado	Pontuação
★★★	Representa o nível desejado de qualidade. Esta classificação é dada quando todas as características necessárias são plenamente atendidas e com um nível bom de qualidade.	3
★★	Esta classificação é dada quando as características necessárias são atendidas de maneira satisfatória.	2
★	Esta classificação é dada quando as características necessárias não são totalmente atendidas, ou são atendidas sem um nível bom de qualidade.	1

Com base na nota atribuída a cada critério será calculada uma nota para cada componente analisado e no final do estudo estas notas serão somadas para se obter as notas gerais das três bibliotecas. As notas serão calculadas pela porcentagem da soma de pontos da biblioteca em relação a pontuação máxima possível que é de cento e trinta e cinco pontos. Deste modo teremos uma pontuação que irá de de zero a cem.

4.2 Comparativo Entre Componentes

4.2.1 Calendário

Os componentes de calendário foram criados e utilizados bem antes do surgimento do JSF e sua finalidade principal é auxiliar o usuário na seleção de uma data, ou servir apenas como informativo. Como já existiam muitos calendários disponíveis antes do JSF, era de se esperar que fossem incluídos em todas as bibliotecas de componentes.

Os calendários contam com duas abordagens, na primeira e mais usada ao se clicar em um botão o calendário surge em um *popup* interno na página, que é uma pequena janela que surge por sobre o conteúdo da página, e ao usuário escolher uma data, um campo texto é preenchido. Na segunda abordagem, a padrão, o calendário não é aberto com *popup*, ele esta fixo na página e caso não utilize AJAX é comum ocorrer a submissão da página ao se selecionar outro mês ou ano.

Comparativo:

1. Usabilidade: Os três componentes possuem usabilidade agradável ao usuário, mas o Tomahawk é menos funcional no modo padrão, já que ele não permite que o usuário escolha um mês ou ano especifico diretamente, ele somente permite que o usuário avance ou retroceda os meses um a um, o que pode ser muito cansativo ao usuário que precise usar uma data com mês ou ano distante da atual.

2. Uso de AJAX: Apenas o RichFaces e ICEFaces utilizam, o que é uma grande vantagem principalmente quando não é usado o modo *popup.* Com o Tomahawk, toda

a página precisou ser submetida ao se avançar ou retroceder o mês no modo normal.

3. Compatibilidade: Os três se mostraram perfeitamente compatíveis com o Internet Explorer e Firefox.

4. Visual: Neste quesito o Tomahawk deixa a desejar frente aos concorrentes, que possuem um visual mais bonito, moderno e bem acabado. Isto fica bem claro observando a figura 7.

5. Diferencial: Entre os três concorrentes, o RichFaces possui modo extra, em que o componente, pode ser utilizado como uma agenda de compromissos bastante interativa, permitindo ao usuário inserir anotações para cada dia, como pode ser observado na figura 8.

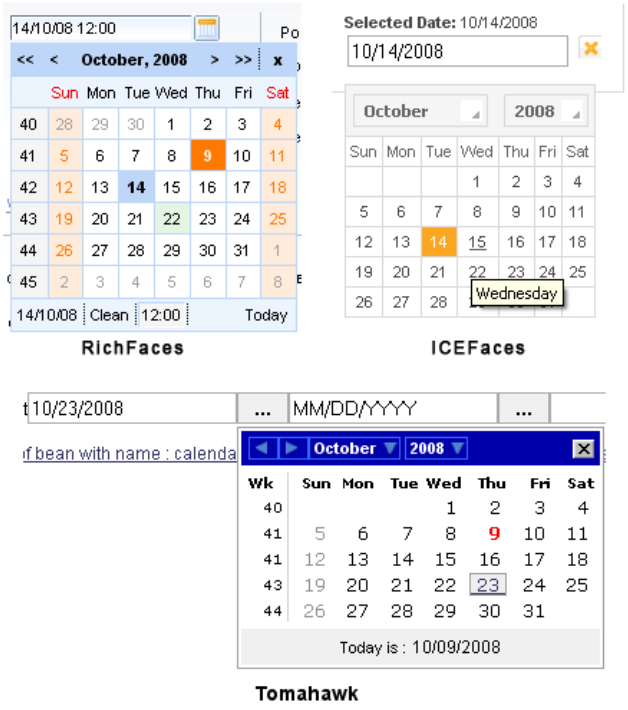

Figura 7: Componentes de calendário no modo popup

Pela analise dos componentes, ficou claro que o RichFaces e o ICEFaces são as melhores opções, o que fica refletido na tabela 3, ressaltando que o modo agenda do RichFaces é um bom diferencial.

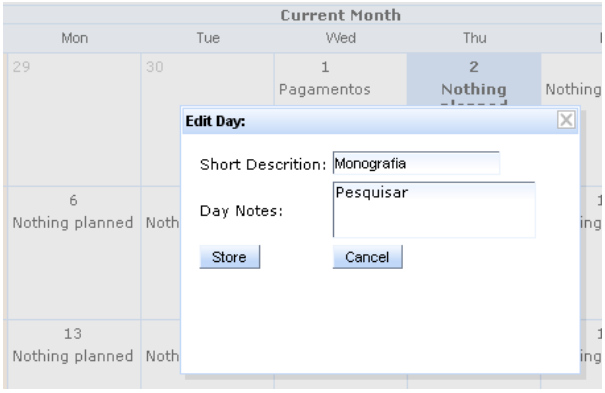

Figura 8: Componente calendário do RichFaces, no modo agenda

Tabela 3: Tabela comparando os componentes de calendário

Atributo	Bibliotecas		
	RichFaces	ICEFaces	Tomahawk
Usabilidade	★★★	★★★	★★
AJAX	★★★	★★★	-
Compatibilidade	★★★	★★★	★★★
Acabamento	★★★	★★★	★★
Diferencial	★★	★	★

4.2.2 Envio de Arquivo

O envio de arquivos é algo muito comum em sistemas web e existe no próprio HTML uma tag própria para isto, que permite ao usuário escolher um arquivo para o envio.

Muita coisa mudou nos últimos anos, as conexões de Internet melhoraram e os usuários enviam arquivos cada vez maiores.

Em um envio comum de arquivo em uma requisição web, por padrão, depois da requisição, a página fica parada esperando o término do envio do arquivo antes de prosseguir. Pensando em melhorar isto, começaram a surgir junto com o AJAX, sistemas que aperfeiçoaram este envio, fornecendo ao usuário mais dados, como velocidade, tempo estimado para o fim do envio, entre outros. Esta mudança trouxe um ganho de usabilidade muito grande, pois em muitos casos o envio poderia demorar muitos minutos e os usuários ficavam sem saber quando este envio iria terminar.

Comparativo:

1. Usabilidade: Neste quesito o RichFaces e ICEFaces se mostraram bem superiores ao Tomahawk que não mostra nenhum indicativo do progresso do envio.

2. Uso de AJAX: O RichFaces e ICEFaces fazem um bom uso do AJAX e fazem o envio sem submeter toda a página, como ocorre com o Tomahawk.

3. Compatibilidade: Todos se mostraram compatíveis com o FireFox e Internet Explorer.

4. Visual: O RichFaces e ICEFaces possuem um visual muito melhor que o Tomahawk, que fica bem evidente observando as figuras 9, 10 e 11.

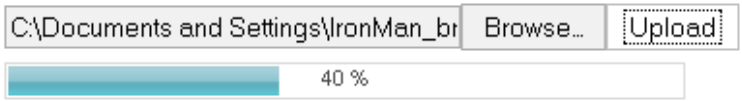

Figura 9: Componente de envio de arquivo do ICEFaces

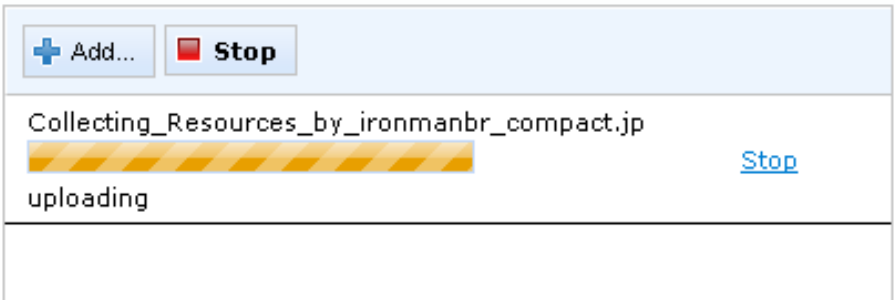

Figura 10: Componente de envio de arquivo do RichFaces

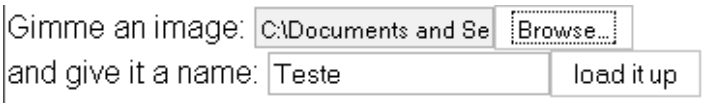

Figura 11: Componente de envio de arquivo do Tomahawk

5. Diferencial: O RichFaces possui como diferencial uma opção em que o mecanismo de seleção do arquivo é feito usando a tecnologia Flash da Adobe, que permite um filtro de arquivos por extensão que não é suportado pelo padrão HTML.

Pela analise feita o RichFaces e ICEFaces se mostraram superiores em muitos quesitos o que pode ser observado na tabela 4. O RichFaces têm a seu favor a possibilidade de utilizar a tecnologia Flash para seleção de arquivo, que permite um filtro das extensões permitidas do lado do cliente, mas também obriga que o usuário possua o Adobe Flash instalado o que pode ser um ponto negativo em determinadas situações. Outro ponto a ser considerado é o uso de AJAX que torna a experiência mais agradável ao usuário.

Tabela 4: Tabela comparando os componentes de upload de arquivo

Atributo	Bibliotecas		
	RichFaces	ICEFaces	Tomahawk
Usabilidade	★★★	★★★	★★
AJAX	★★★	★★★	-
Compatibilidade	★★★	★★★	★★★
Acabamento	★★★	★★★	★★
Diferencial	★★★	★★	★

4.2.3 Abas

O uso de abas para organizar informações sempre foi algo muito utilizado na computação e há alguns anos um dos programas ajudou a difundir, mais ainda, esse uso foi o navegador Opera, que mostrou que era muito mais produtivo e confortável usar várias abas em uma mesma janela do navegador do que ter muitas janelas abertas. Outros programas desktop também adotaram o uso de abas por dar melhorar a usabilidade e este conceito logo foi implementado nas aplicações web.

Como era esperado as três bibliotecas estudadas incluíram componentes de abas.

Na implementação de abas, existem três modos de uso:

1. Carregando todo o código no cliente: neste modo o abrir da abas será instantâneo, pois todo conteúdo é previamente carregado quando o cliente abre a página. Só deve ser usado quando a informação das Abas é estática ou não precisa ser atualizada sempre, pois os dados só serão atualizados se o cliente abrir a página novamente.

2. Carregamento incremental com AJAX: neste modo só é carregado o código da Aba que é aberta como padrão e quando o cliente selecionar a próxima Aba, os dados serão trazidos para montar o código da aba. É o modo ideal quando os dados precisam ser mostrados sempre da maneira mais atualizada possível, pois a cada mudança de aba os dados são coletados novamente. Esta forma de carregamento reduz o tempo inicial de carga da pagina, pois neste carregamento inicial, somente os dados da aba padrão serão carregados.

3. Carregamento incremental sem AJAX: funciona igual ao modo anterior, mas tem a séria desvantagem de ter de submeter toda a página.

Comparativo:

1. Usabilidade: Neste quesito os três possuem as mesmas características, mas o Tomahawk, não vem com suporte a AJAX em seu componente, o que pode ser considerado um ponto negativo.

2. Uso de AJAX: O RichFaces e ICEFaces fazem um bom uso do AJAX e fazem a requisição do que será mostrado em cada aba sem submeter toda a página, como ocorre com o Tomahawk.

3. Compatibilidade: Todos se mostraram compatíveis com o FireFox e Internet Explorer.

4. Visual: O RichFaces e ICEFaces possuem um visual padrão muito melhor que o Tomahawk, que fica bem evidente observando as figuras 12, 13 e 14.

5. Diferencial: O ICEFaces possui como diferencial já possuir um componente para que as abas sejam formadas de forma dinâmica no carregar da página.

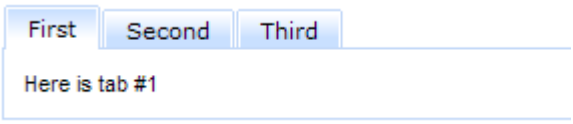

Figura 12: Componente de Abas do RichFaces

Figura 13: Componente de Abas do ICEFaces

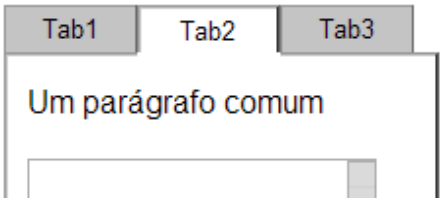

Figura 14: Compoente de Abas do Tomahawk

As três bibliotecas permitem a customização da aparência das abas de maneira simples para quem possui domínio em css, pois toda a aparência da aba é definida utilizando classes css.

Pelo comparativo ficou evidente que os três componentes são bem construídos, mas entre as características estudadas apontamos uma vantagem para o ICEFaces e Richfaces, principalmente pelo uso do AJAX que melhora a usabilidade. O resultado do comparativo pode ser observado na tabela 5.

Tabela 5: Tabela comparando os componentes de abas

Atributo	Bibliotecas		
	RichFaces	ICEFaces	Tomahawk
Usabilidade	★★★	★★★	★★
AJAX	★★★	★★★	-
Compatibilidade	★★★	★★★	★★★
Acabamento	★★★	★★★	★★
Diferencial	★	★★	★

4.2.4 Editor WYSIWYG

A sigla WYSIWYG significa "**W**hat **Y**ou **S**ee **I**s **W**hat **Y**ou **G**et" que pode ser traduzida para o português como "O que você vê é o que você tem" e isto significa a capacidade de abstrair a codificação interna do documento e manipulá-lo da maneira que ele será utilizado. No geral WYSIWYG implica na possibilidade de manipulação direta da aparência de elementos de um documento, sem precisar digitar ou se lembrar dos comandos de formatação do *layout* do mesmo. O exemplo mais comum de editor WYSIWYG é o Microsoft Word, que permite que se edite e visualize o texto da maneira que ele será impresso, sem o usuário ser obrigado a conhecer a formatação interna do mesmo [23].

Como a Web utiliza a linguagem de marcação HTML, foi inevitável o surgimento de editores WYSIWYG que facilitassem a edição de texto formatado em HTML de modo transparente como se o usuário estivesse editando um texto em um editor como Microsoft Word ou OpenOffice Writer.

Estes editores WYSIWYG, foram aprimorando sua qualidade e permitindo uma total transparência ao usuário final, podendo o mesmo editar um texto com formatações complexas de maneira fácil. Outra novidade interessantes destes editores é o reconhecimento da formatação do texto de editores Desktop, como o Microsoft Word, na área de transferência do sistema operacional (Clipboard), permitindo que se copie o conteúdo de um arquivo e se cole no editor web e este reconheça e converta as formatações visuais do editor Desktop para o equivalente em código HTML.

Entre os editores mais populares e completos, dois que se destacam por sua qualidade e recursos são: o FCKEditor e O TinyMCE que são ambos gratuitos e de código aberto.

Como a dificuldade de se implementar um editor deste tipo é grande, algumas bibliotecas de componentes preferiram usar algum editor gratuito existente, para desenvolver seus componentes.

Infelizmente apenas o ICEFaces e Tomahawk possuem este componente e eles são construídos com base no editores FCKEditor e KuPu respectivamente.

Comparativo:

1. Usabilidade: Neste quesito os dois possuem características semelhantes, com o ICEFaces fornecendo mais opções, dando um poder maior ao usuário, permitindo ao mesmo operações mais complexas, como substituição de texto, além de permitir o uso de modelos, visualização em tela cheia, entre outras.

2. Uso de AJAX: Esse não é um componente que o uso do AJAX seja um grande

diferencial, pois todo o processamento no editor é feito por JavaScript no lado do cliente.

3. Compatibilidade: No Internet Explorer 7, utilizando o Windows XP, o componente do Tomahawk solicitou a instalação de um complemento ActiveX chamado MSXML que é da própria Microsoft. Esta solicitação poderia confundir um usuário menos experiente e poderia ser um problema caso ele não possua privilégios de administrador.

4. Visual: O ICEFaces possui um visual padrão semelhante ao Tomahawk, que pode ser observado nas figuras 15 e 16; mas o componente do ICEFaces por herdar as características do FCKeditor, permite a utilização de *skins*, que podem melhorar a aparência do mesmo.

5. Diferencial: O ICEFaces possui como diferencial a utilização de um editor bastante maduro, conhecido e com muitos recursos.

Ficou claro que os dois componentes estudados possuem uma boa qualidade, mas o componente do ICEFaces se destaca por possuir mais recursos. O resultado do comparativo pode ser visto na tabela 6.

Figura 15: Componente editor WYSIWYG do Tomahawk

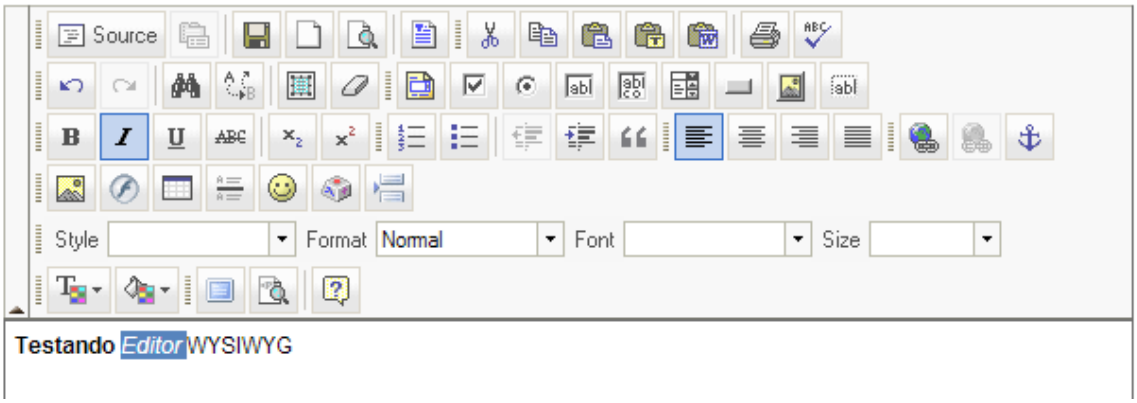

Figura 16: Componente editor WYSIWYG do ICEFaces

Tabela 6: Tabela comparando os componentes de editores WYSIWYG

Atributo	Bibliotecas		
	RichFaces	ICEFaces	Tomahawk
Usabilidade	-	★★★	★★
AJAX	-	★	-
Compatibilidade	-	★★★	★
Acabamento	-	★★★	★★
Diferencial	-	★★	★

4.2.5 Árvores

Os componentes UI de árvores são úteis, quando se quer exibir informações agrupadas, como por exemplo em uma árvore que represente uma coleção de música; a raiz seria a coleção e teria como nós filhos, os artistas, que teriam seus nós filhos representando os álbuns, e dentro do álbum as músicas, como mostra a figura 17.

São componentes que podem ser usados para representar muitas coisas que possuem relacionamento hierárquico, como país-estados-cidades, empresa-setores-

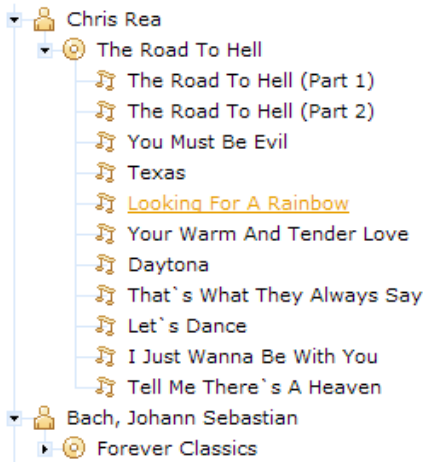

Figura 17: Componente de Árvore do Richfaces representando uma coleção de música

departamentos, entre outros.

A exemplo dos componentes de abas, as implementações possuem as três opções de carregamento: Carregando todo o código no cliente antecipadamente, carregamento incremental com AJAX, e carregamento incremental sem AJAX.

Comparativo:

1. Usabilidade: Neste quesito os três componentes possuem as mesmas características, mas a usabilidade fica comprometida no Tomahawk pela falta de AJAX.

2. Uso de AJAX: O RichFaces e ICEFaces fazem um bom uso do AJAX e fazem a requisição do que será mostrado em cada nó sem submeter toda a página, como ocorre com o Tomahawk.

3. Compatibilidade: Todos se mostraram compatíveis com o FireFox e Internet Explorer.

4. Visual: O RichFaces e ICEFaces possuem um visual padrão melhor que o Tomahawk,

que pode ser observado nas figuras 17, 18 e 19.

5. Diferencial: O RichFaces e ICEFaces se destacam principalmente pelo uso do AJAX, pois em um componente deste tipo, as respostas às ações do usuário devem ser as mais rápidas possíveis, para que a utilização do componente não se torne tediosa e cansativa.

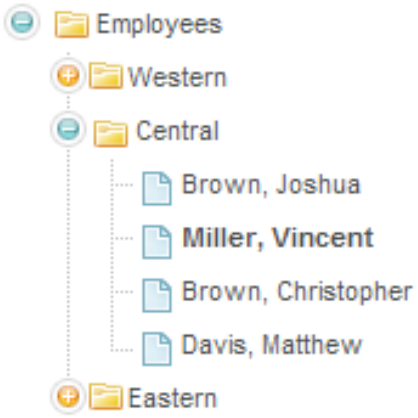

Figura 18: Componente de árvore do ICEFaces

Figura 19: Componente de árvore do Tomahawk

Pelo comparativo ficou evidente que os três componentes são bem construídos, mas a falta de AJAX por parte do Tomahawk é um ponto negativo muito forte e isto ficou refletido na tabela 7.

Tabela 7: Tabela comparando os componentes de árvore

Atributo	Bibliotecas		
	RichFaces	ICEFaces	Tomahawk
Usabilidade	★★★	★★★	★
AJAX	★★★	★★★	-
Compatibilidade	★★★	★★★	★★★
Acabamento	★★★	★★★	★★
Diferencial	★★★	★★★	★

4.2.6 Menus

Em qualquer programa de computador que possua UI, os menus são fundamentais para a navegação do usuário pelo sistema, e podem, inclusive, definir o sucesso ou o fracasso de um software.

Nos sistemas web há uma variedade muito grande em estilos de menus e surgiram muitos componentes para facilitar a construção dos mesmos, que se especializaram em criar verdadeiras bibliotecas de menus, com poucas variações nos tipos, mas com muita variação de design.

As bibliotecas de componentes JSF também produziram seus sistemas de menus, criando inclusive sistemas de *skins*, que são na verdade modelos visuais que podem ser

implementados criando um uma aparência totalmente nova para o modelo do menu.

Estes componentes de menu, permitem que a aplicação possua menus que podem ser estáticos ou montados dinamicamente, o que é muito importante em sistemas que possuem perfis de usuário, sendo o menu montado mostrando ao usuário apenas o que ele tem permissão de acesso.

Comparativo:

1. Usabilidade: Neste quesito três possuem características semelhantes.
2. Uso de AJAX: Como são componentes em que todo o código é montado no cliente, não é usado AJAX.
3. Compatibilidade: Todos se mostraram compatíveis com o FireFox e Internet Explorer.
4. Visual: O RichFaces e ICEFaces possuem um visual padrão melhor que o Tomahawk, que pode ser observado nas figuras 20, 21 e 22.
5. Diferencial: O RichFaces e ICEFaces se destacam por possuir um acabamento melhor, simulando muito bem os menus de programas desktop. O Tomahawk se destaca por possui mais tipos de menu.

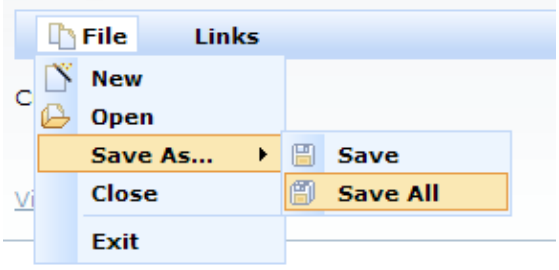

Figura 20: Componente de menu do RichFaces

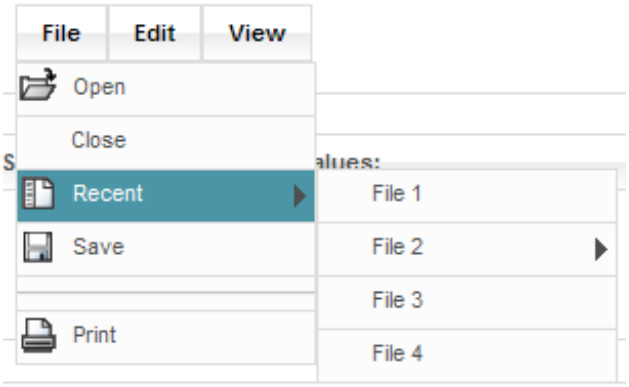

Figura 21: Componente de menu do ICEFaces

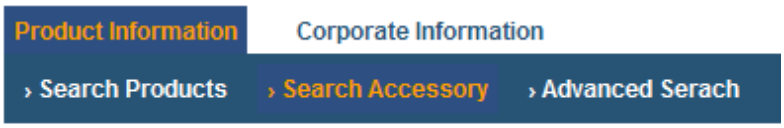

Figura 22: Componente de menu do Tomahawk

Os três componentes permitem customização através de css, mas o RichFaces, em seu demonstrativo, apresentou uma documentação melhor, com uma lista completa dos atributos do componente de menu que podiam ter sua aparência modificada através de classes css.

Os três componentes mostraram possuir bastante qualidade, o que ficou refletido na tabela 8.

Tabela 8: Tabela comparando os componentes de menu

Atributo	Bibliotecas		
	RichFaces	ICEFaces	Tomahawk
Usabilidade	★★★	★★★	★★★
AJAX	★	★	-
Compatibilidade	★★★	★★★	★★★
Acabamento	★★★	★★★	★★
Diferencial	★★	★★	★★

4.2.7 Tabelas Dinâmicas

Seja em um relatório complexo ou uma listagem simples de uma tabela de um banco de dados, sempre é preciso mostrar dados ao usuário formatados como uma tabela.

Parar gerar esta tabela em uma aplicação web, os dados são trazidos do banco de dados, ou de alguma outra fonte de dados como arquivo texto, arquivo xml, entre outros, e é feita uma iteração para formar uma linha da tabela para cada objeto da coleção que vamos representar.

Fica fácil imaginar como era trabalhoso fazer isso em uma aplicação JSP e mais ainda usando servlet.

Para abstrair estas dificuldades e aumentar a produtividade, as bibliotecas de componentes JSF possuem implementações bastantes versáteis de tabelas dinâmicas. Essas implementações são projetadas e construídas para que resolvam muitos dos problemas que os programadores sempre tiveram de lidar.

Entre os problemas já solucionados por estas implementações, podemos destacar alguns como: a paginação de dados que é limitar o número de registros que será mostrado em uma tabela, dividindo em grupos, e permitir que o usuário possa ir navegando por grupos de registros. Outro problema é o de ordenação por colunas, dando flexibilidade ao usuário escolher por qual coluna quer ordenar os dados, e inclusive, se a ordenação será crescente ou decrescente.

Cabe ao programador, ao escrever o código, configurar o componente informando a coleção de dados, e outros parâmetros como as colunas utilizadas, os conversores, entre outros, dando um ganho de produtividade muito grande.

Comparativo:

1. Usabilidade: Neste quesito, os três possuem características semelhantes, possuindo muitos recursos.

2. Uso de AJAX: O AJAX tem um uso bastante interessante, pois ao se paginar a listagem, ou mudar a ordenação, é muito interessante que seja atualizada apenas a área da tabela, sendo mais agradável ao usuário e gastando menos recursos na rede. O RichFaces e ICEFaces fazem isto muito bem.

3. Compatibilidade: Todos se mostraram compatíveis com o FireFox e Internet Explorer.

4. Visual: O RichFaces e ICEFaces possuem um visual padrão melhor que o Tomahawk, que pode ser observado nas figuras 23, 24 e 25, mas este visual é definido por css e nas três bibliotecas pode ser facilmente modificado.

5. Diferencial: Todos os três componentes possuem muitas funcionalidades, mas o

RichFaces, logo seguido pelo ICEFaces, se destaca pela quantidade de recursos que pode facilitar em muito o trabalho do desenvolvedor. Um recurso bastante interessante, que apenas o ICEFaces e Tomahawk possuem, é o de agrupamento, que pode ser melhor entendido observando a figura 24.

Os três componentes mostraram possuir bastante qualidade, o que ficou refletido na tabela 9.

ID	Car type	Car color
1	car B	blue
6	car J	blue
11	car L	dark blue
7	car I	gray
4	car C	green
8	car M	lightGray
9	car N	magenta
5	car E	orange
2	car A	red
10	car K	unknown
3	car D	yellow

Figura 23: Componente de tabela dinâmica do Tomahawk

ID	Region	Office	Contact Info		
			First Name	Last Name	Phone
10	Western	Calgary	Ethan	Smith	555-4562
15			Jacob	Smith	555-4563
20			Logan	Smith	555-4564
25			Benjamin	Smith	555-4565
30			Jack	Smith	555-4566
35			Noah	Johnson	555-4567
40			William	Johnson	555-4568
45			Andrew	Johnson	555-4569
46			Samuel	Johnson	555-4570
47		Victoria	Joseph	Johnson	555-4571
50			Daniel	Williams	555-4572
16			Anthony	Williams	555-4573
17			Angel	Williams	555-4574
18			Jacob	Williams	555-4575
100			David	Williams	555-4576

Figura 24: Componente de tabela dinâmica do ICEFaces mostrando a possibilidade de agrupamento

Sorting Example		
State Name	State Capital	Time Zone
New York	Albany	GMT-5
Maryland	Annapolis	GMT-5
Georgia	Atlanta	GMT-5
Maine	Augusta	GMT-5
Texas	Austin	GMT-6
Louisiana	Baton Rouge	GMT-6
North Dakota	Bismarck	GMT-6
Idaho	Boise	GMT-8
Massachusetts	Boston	GMT-5
Nevada	Carson City	GMT-8
West Virginia	Charleston	GMT-5
Wyoming	Cheyenne	GMT-7
South Carolina	Columbia	GMT-5
Ohio	Columbus	GMT-5
New Hampshire	Concord	GMT-5

Figura 25: Componente de tabela dinâmica do RichFaces

Tabela 9: Tabela comparando os componentes de tabelas dinâmicas

Atributo	Bibliotecas		
	RichFaces	ICEFaces	Tomahawk
Usabilidade	★★★	★★★	★★★
AJAX	★★★	★★★	-
Compatibilidade	★★★	★★★	★★★
Acabamento	★★★	★★★	★★
Diferencial	★★	★★	★★

4.2.8 Google Maps

Em fevereiro de 2005 a Google lançou mais um de seus serviços. O Google Maps

foi grande sucesso ao permitir que pessoas comuns tivessem acesso gratuito a fotos de satélite, e por ter a ambição de formar uma mapa virtual com imagens de satélite em alta definição e mapas.

Junto com o serviço o Google lançou uma API que permitia a desenvolvedores utilizarem os serviço de mapa e imagens de satélite em suas aplicações. Se tornou algo muito útil, para aplicações que necessitam de georeferenciamento, dando a possibilidade de ter, nas imagens de satélite, pontos marcados a partir de coordenadas de latitude e longitude.

Além do GoogleMaps, surgiram outros concorrentes como o Yahoo Maps e o Live Earth e Visual Earth da Microsoft, possuindo funcionalidades semelhantes. Outra funcionalidade bastante interessante que a Google agregou ao serviço, foi a possibilidade de, dados alguns pontos, o sistema traçar as rotas nos mapas, respeitando o sentido das ruas e mostrando a distância entre os pontos.

Percebendo a grande utilidade deste serviço, algumas bibliotecas de componentes JSF, criaram componentes que utilizam a API do Google maps, permitindo uma fácil utilização por parte do desenvolvedor.

Das três bibliotecas estudadas, somente o RichFaces e a ICEFaces possuem este componente, e a RichFaces vai além possuindo suporte para o Virtual Earth da Microsoft.

Comparativo:

1. Usabilidade: Neste quesito os dois possuem as mesma características, pois eles funcionam basicamente da mesma maneira que o próprio Google Maps.

2. Uso de AJAX: O RichFaces possui a opção de utilizar sua implementação com AJAX

e JSON.

3. Compatibilidade: Todos se mostraram compatíveis com o FireFox e Internet Explorer.

4. Visual: Não há praticamente nenhuma diferença visual, já que as imagens são fornecidas pelo Google.

5. Diferencial: O RichFaces, por possuir como opção usar sua implementação ao invés da API do Google, pode ser um diferencial, mas não foi notada nenhuma melhoria em relação a esta implementação própria.

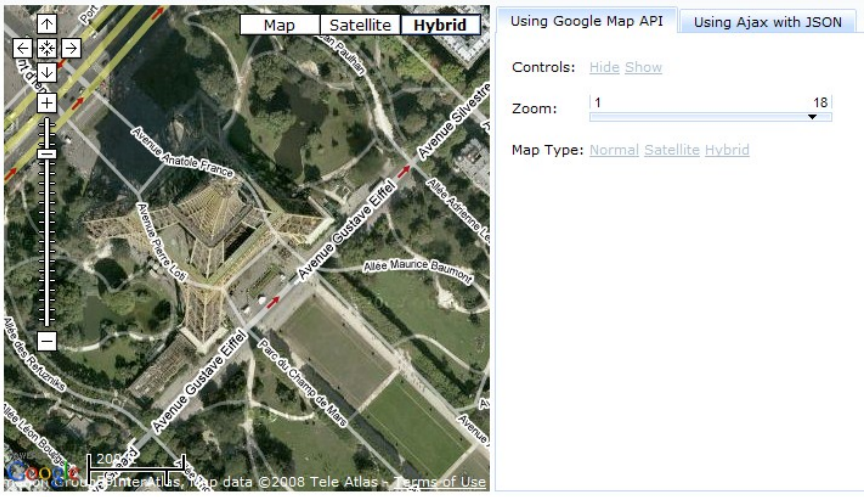

Figura 26: Componente do Google Maps do RichFaces

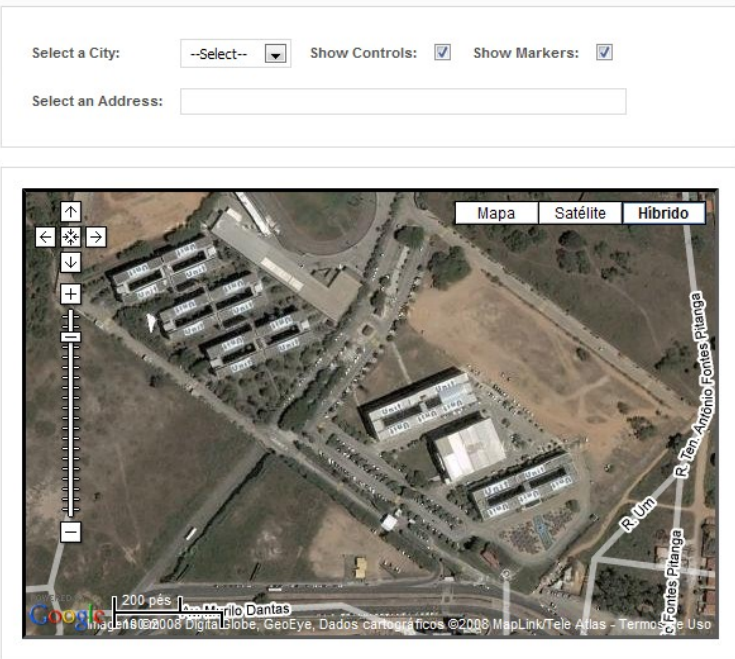

Figura 27: Componente do Google Maps do ICEFaces

Os dois componentes possuem bastante qualidade, o que pode ser observado na tabela 10.

Tabela 10: Tabela comparando os componentes do Google Maps

Atributo	Bibliotecas		
	RichFaces	ICEFaces	Tomahawk
Usabilidade	★★★	★★★	-
AJAX	★★★	★★★	-
Compatibilidade	★★★	★★★	-
Acabamento	★★★	★★★	-
Diferencial	★★	★	-

4.2.9 Menu de Contexto

O menu de contexto, também chamado de menu *popup*, é basicamente um menu que está associado a um elemento de um componente UI, e que surge na forma de uma pequena janela, quando o usuário interage com o mouse.

Um bom exemplo é o menu de contexto em editores de texto como o Microsoft Word, onde o usuário pode selecionar um texto e ao clicar com o botão direito do mouse, surgirá o menu com ações que serão executadas apenas para o texto selecionado, como formatar a fonte do texto, copiar, recortar entre outras.

As bibliotecas RichFaces e ICEFaces, implementam componentes que permitem o uso desta funcionalidade, e têm uma utilidade prática muito boa, pois permite uma interação muito grande entre o usuário e o sistema.

Um bom exemplo de uso, pode ser visto na figura 29, em que o usuário tem a possibilidade de selecionar por meio de um menu de contexto, várias ações, que serão aplicadas à linha selecionada, com o clique do mouse. Este exemplo ilustra como o sistema ficou mais funcional e intuitivo, pois sem o menu de contexto o programador teria de colocar botões, ou links para executar estas ações o que poderia diminuir a usabilidade alem de ocasionar perda de espaço sem necessidade.

Comparativo:

1. Usabilidade: Neste quesito os dois possuem as mesma características, sendo muito parecidos com um menu de contexto de uma aplicação desktop.

2. Uso de AJAX: Ambos usam AJAX de maneira indireta, pois eles apenas executam uma ação, essa sim é que pode usar AJAX ou não.

3. Compatibilidade: Todos se mostraram compatíveis com o FireFox e Internet Explorer.

4. Visual: Os dois possuem um visual padrão bastante elegante e moderno, como pode ser visto nas figuras 28 e 29.

5. Diferencial: Pelo que foi visto os dois possuem as mesmas qualidades, não ficando evidente um diferencial mensurável.

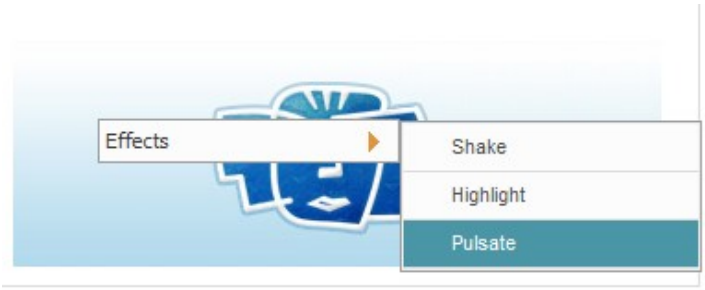

Figura 28: Componente de menu de contexto do ICEFaces

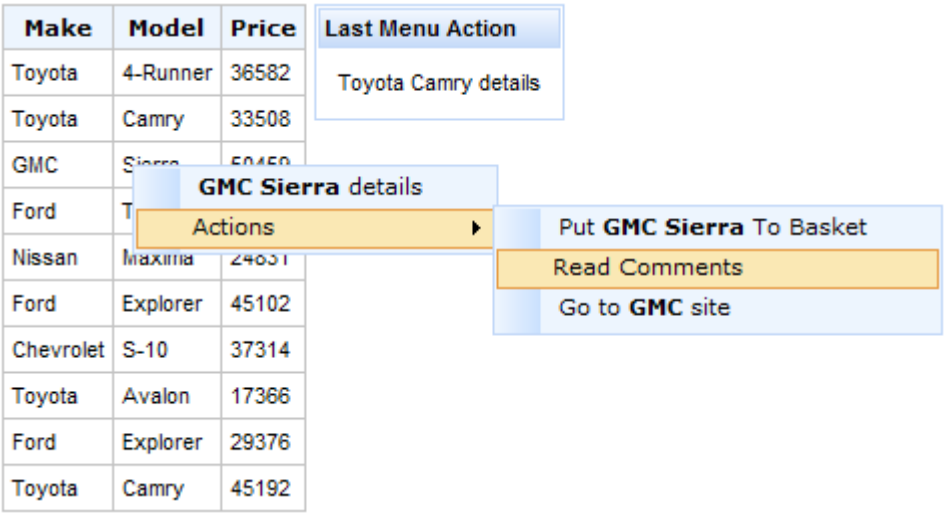

Figura 29: Componente de menu de contexto do RichFaces

Os dois componentes mostraram possuir bastante qualidade, o que ficou refletido na tabela 11.

Tabela 11: Tabela comparando os componentes de menu de contexto

Atributo	Bibliotecas		
	RichFaces	ICEFaces	Tomahawk
Usabilidade	★★★	★★★	-
AJAX	★★★	★★★	-
Compatibilidade	★★★	★★★	-
Acabamento	★★★	★★★	-
Diferencial	-	-	-

5 CONCLUSÃO

Neste estudo, foi apresentada a evolução das tecnologias Java para aplicações web sempre enfatizando como as deficiências de cada tecnologia foram sanadas a cada etapa desta constante evolução que levou a criação do JavaServer Faces.

Foi estudada toda teoria envolvendo o funcionamento do JavaServer Faces, mostrando suas vantagens em relação às tecnologias passadas, e procurando sempre entender os impactos destas vantagens para o desenvolvimento de uma aplicação web, e também mostrando como a tecnologia JSF foi arquitetada para ser segura, robusta, escalável e produtiva.

Uma das caraterísticas que trouxeram produtividade foi o sistema de componentes de interface do usuário, principalmente com o uso de bibliotecas de componentes, facilitando a construção das interfaces, algo que sempre se mostrou um problema no desenvolvimento de aplicações web. Muitos programadores se especializaram na construção e manutenção destes componentes UI, sendo inevitável o surgimento de projetos dedicados exclusivamente à construção bibliotecas de componentes.

O uso de uma boa biblioteca de componentes se mostrou algo muito benéfico, permitindo que uma parte do esforço gasto com a construção das interfaces, fosse abstraída, dando aos programadores mais tempo para se dedicarem às lógicas de negócios envolvidas nos projetos.

O estudo abrangeu a análise e comparação de três bibliotecas de componentes

bastante populares e utilizadas, sendo que duas delas (RichFaces e ICEFaces) nasceram como produtos pagos e depois se transformaram em projetos gratuitos de código aberto e a MyFaces Tomahawk sempre foi um projeto gratuito de código aberto.

Logo no inicio do estudo foi percebido que a RichFaces e ICEFaces tem um nível mais profissional que a MyFaces Tomahawk, possuindo uma documentação muito mais completa.

Um dos problemas encontrados no estudo, foi a falta por parte da Apache Tomahawk de uma página com um demonstrativo ou um projeto JSF, detalhando seus componentes com indicações de uso e código fonte de exemplo para o uso de cada componente. Foi necessária a utilização de uma página não oficial, que fornecia um bom demonstrativo dos componentes da biblioteca Tomahawk.

As bibliotecas RichFaces e ICEFaces, ao contrário da Tomahawk, possuem uma ótima página com um demonstrativo de seus componentes, detalhando seu uso e fornecendo tanto o código fonte de uso do componente para a página, bem como o código fonte dos Beans utilizados.

No estudo foi executado um comparativo detalhado entre vários componentes das três bibliotecas, e ficou claro que a RichFaces e ICEFaces são superiores a Tomawank. Entre suas principais vantagens estão: o suporte nativo a AJAX, um acabamento visual muito mais refinado e moderno e uma maior quantidade de componentes.

Nosso estudo atribuiu uma nota para cada atributo dos componentes estudados, obtendo assim uma nota para cada componente, quantificando sua qualidade perante aos

concorrentes. De posse das notas dos componentes, foi criada a tabela 12, que mostra as notas realizando um somatório, e esta pontuação foi dividida pelo total de pontos possíveis e multiplicada por cem, obtendo assim as notas referentes a cada biblioteca estudada. As notas obtidas vieram a corroborar com tudo que foi exposto neste trabalho, mostrando que o RichFaces e ICEFaces possuem uma qualidade melhor que o Tomawank.

Tabela 12: Pontuação dos componentes das bibliotecas

Componente	Pontuação		
	RichFaces	ICEFaces	Tomahawk
Calendário	14	13	9
Envio de arquivo	15	15	9
Abas	13	14	9
Editor WYSIWYG	-	12	6
Árvore	15	15	7
Menu	12	12	10
Tabelas dinâmicas	14	14	10
Google Maps	14	13	-
Menu de contexto	12	12	-
Somatório de Pontos	**109**	**120**	**60**
Nota	**80,74**	**88,88**	**44,44**

Como as três bibliotecas possuem seu código aberto, e são de uso gratuito, podemos nos basear apenas no lado técnico, e afirmar que pelo nosso estudo as bibliotecas RichFaces e ICEFaces são mais indicadas tanto em uso corporativo, quanto em uso pessoal. O uso corporativo pode ser beneficiado ainda, com a possibilidade de cursos e suporte oficial por parte de seus criadores, o que pode ser considerada uma boa garantia para as empresas que

estão iniciando o desenvolvimento de software em JavaServer Faces, utilizando umas das duas bibliotecas.

6 REFERÊNCIAS BIBLIOGRÁFICAS

[1] GEARY, David e HORSTMANN, Cay. **Core JavaServer Faces**. Addison Wesley, 2004. 637p.

[2] COSTA, Elenildes, RUIZ, Luiz e MELO, Marcio. **Interface Web Para Manipulação de Banco de Dados: DBAJAX**. Ano: 2006. Universidade Tiradentes.

[3] **The Java Servlet API White Paper**. Disponível em: <http://java.sun.com/products/servlet/whitepaper.html>. Acessado em: 16/08/08

[4] **JavaServer Pages[ym] Tecnology – White Paper**. Disponível em: <http://java.sun.com/products/jsp/whitepaper.html>. Acessado em: 16/08/08

[5] MAHMOUD, Qusay. **Developing Web Applications with JavaServer Faces**. Disponível em: <http://java.sun.com/developer/technicalArticles/GUI/JavaServerFaces/>. Acessado em: 17/08/2008

[6] **RichFaces Developer Guide**. Disponível em: <http://www.jboss.org/file-access/default/members/jbossrichfaces/freezone/docs/devguide/en/html_single/index.html>. Acessado em 18/08/2008

[7] **ICEfaces Architecture**. Disponível em: <http://www.icefaces.org/main/product/architecture.iface>. Acessado em 18/08/2008

[8] **MyFaces Wiki**. Disponível em: <http://wiki.apache.org/MyFaces/>. Acessado em 18/08/2008

[9] **Wikipedia - JavaBean**. Disponível em: <http://en.wikipedia.org/wiki/JavaBeans>. Acessado em 11/09/2008

[10] KURNIAWAN, Budi. **Java para Web com Servlets, JSP e EJB**. Editora Ciência Moderna, 2002. 807p.

[11] **Java BluePrints - Model-View-Controller.** Disponível em: <http://java.sun.com/blueprints/patterns/MVC-detailed.html>. Acessado em 18/08/2008

[12] GOYAL, Deepak, and VARMA, Vikas, **Introduction do Java Server Faces (JSF)**, Sun Microsystems presetation.

[13] MANN, Kito D. **JavaServer Faces In Action.** Editora Manning, 2005. 1038 pg.

[14] JSR 127: **JavaServer Faces - Java Specification Requests.** Disponível em: <http://www.jcp.org/en/jsr/detail?id=127>. Acessado em 20/09/2008

[15] **Wikipedia - User Interface**. Disponível em: <http://en.wikipedia.org/wiki/User_interface>. Acessado em 25/09/2008.

[16] HIGHTOWER, Richard. **JSF for nonbelievers: The JSF application lifecycle**. Disponível em: <http://www.ibm.com/developerworks/library/j-jsf2/>. Acessado em 03/10/2008.

[17] **Sítio Jboss RichFaces**. Disponível em: <http://www.jboss.org/jbossrichfaces/>. Acessado em 08/10/2008.

[18] **Ajax4jsf and RichFaces - historical perspective**. Disponível em: <http://www.jsfone.com/blog/max_katz/2008/08/ajax4jsf_and_richfaces__historical_perspect

ive.html>. Acessado em 08/10/2008.

[19] **Wikipedia - Icefaces**. Disponível em: <http://en.wikipedia.org/wiki/Icefaces>. Acessado em 08/10/2008.

[20] **Wikipedia - RichFaces**. Disponível em: <http://en.wikipedia.org/wiki/Richfaces>. Acessado em 08/10/2008.

[21] Maryka, Stephen. **Enterprise Ajax Security with ICEfaces**. ICEsoft, 2007.Disponível em: <http://www.icefaces.org/main/resources/whitepapers.iface>. Acessado em 08/10/2008.

[22] **Sítio MyFaces Tomahawk**. Disponível em: <http://myfaces.apache.org/tomahawk/index.html>. Acessado em 08/10/2008.

[23] **Wikipedia - WYSIWYG**. Disponível em: <http://pt.wikipedia.org/wiki/WYSIWYG>. Acessado em 08/10/2008.

[24] **Demonstrativo não oficial de componentes do MyFaces Tomahawk**. Disponível em: <http://www.irian.at/myfacesexamples/home.jsf>. Acessado em 07/10/2008.

[25] **Demonstrativo oficial de componentes do Jboss RichFaces**. Disponível em: <http://livedemo.exadel.com/richfaces-demo/index.jsp>. Acessado em 01/10/2008.

[26] **Demonstrativo oficial de componentes do ICEFaces**. Disponível em: <http://component-showcase.icefaces.org/component-showcase/showcase.iface>. Acessado em 01/10/2008.

[27] **AJAX JSF Matrix**. Disponível em: <http://www.jsfmatrix.net/>. Acessado em 07/10/2008.

[28] **Dive Into Greasemonkey**. Disponível em: <http://diveintogreasemonkey.org/>. Acessado em 16/10/2008.