

**LIGA DE ENSINO DO RIO GRANDE DO NORTE
FACULDADE NATALENSE PARA O DESENVOLVIMENTO
DO RIO GRANDE DO NORTE**

RAPHAEL VINICIUS DA SILVA

**RICHFACES:
INTEGRAÇÃO DE JAVASERVER FACES E ASYNCHRONOUS JAVASCRIPT
AND XML (AJAX)**

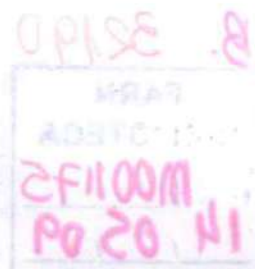
RAPHAEL VINICIUS DA SILVA

RICHFACES:
INTEGRAÇÃO DE JAVASERVER FACES E ASYNCHRONOUS JAVASCRIPT
AND XML (AJAX)

Monografia apresentada ao Curso de Graduação em Sistemas de Informação da Faculdade Natalense para o Desenvolvimento do Rio Grande do Norte como requisito obrigatório para obtenção do grau de Bacharel em Sistemas de Informação.

Orientador: Gleydson de Azevedo F. Lima,
MSc.

Natal
2008



Raphael Vinicius da Silva

Richfaces: Integração de Javasever faces e Asynchronous Javascript and XML (Ajax) / Raphael Vinicius da Silva. – Natal: FARN, 2008.
62f.

Orientador: Gleydson de Azevedo F. Lima (Msc.)

Monografia (Bacharelado em Sistemas de Informação) – Faculdade Natalense para o Desenvolvimento do Rio Grande do Norte.

1. Rich Faces – Monografia. 2. Guia Prático – Monografia. 3. Biblioteca de Componentes – Monografia I.Título.

RN/FARN/BC CDU 004



RAPHAEL VINICIUS DA SILVA

RICHFACES:
**INTEGRAÇÃO DE JAVASERVER FACES E ASYNCHRONOUS JAVASCRIPT
AND XML (AJAX)**

Monografia apresentada ao Curso de Sistemas de Informação da Faculdade
Natalense para o Desenvolvimento do Rio Grande do Norte, aprovada em 16 de
Dezembro de 2008, pela banca examinadora composta pelos seguintes membros:

Gleydson de Azevedo F. Lima

Gleydson de Azevedo F. Lima, MSc.

Orientador

Joseane Alves Pinheiro

Joseane Alves Pinheiro, MSc.

Membro Examinador

Nilda Maria S. de Cerqueira

Nilda Cerqueira, MSc.

Membro Examinador

RESUMO

Este trabalho procurou disponibilizar um referencial teórico, em lingua portuguesa, acerca da biblioteca de componentes RichFaces. É baseado na documentação oficial da Jboss sobre o RichFaces, na qual, realizou-se um processo de tradução dos item mais relevantes para entendimento da funcionalidade, da estrutura e da utilização do framework. Foram eleitos, trabalhados e comentados alguns dos seus principais componentes. O problema a resolver com este trabalho é diminuir a curva de aprendizado deste framework, que possui documentação apenas em lingua inglesa. A pesquisa permiti a disponibilização de um material em português, sobre RichFaces, que serve como guia pratico para o usuário iniciante.

Palavras-chave: Java, JSF, AJAX, Richfaces, Ajax4JSF.

ABSTRACT

This work has sought to make a frame theoretical, in portuguese, about library components richfaces. Is a project object a search exploratory bibliography official JBoss on the richfaces, in which a process of translation of the item most relevant for understanding functionality, structure and use of the framework. They were elected worked and talked some of its main components. To resolve the problem with this work is a smaller learning curve this framework, which has only in English documentation. The search has a material in English, on richfaces that serves as a practical guide to the user.

Key-words: Java, JSF, AJAX, Richfaces, Ajax4JSF.

2.1 JSP	13
2.2 SERVLETS	14
2.3 JSP (JAVASERVER PAGES)	15
2.4 JSF (JAVASERVER FACES)	16
2.5 AJAX/JSF E RICHFACES	18
2.6 AJAX	19
2.7 JAVASCRIPT	20
2.8 XML	21
3 METODOLOGIA	22
4 RESULTADOS	23
4.1 VISÃO GERAL ARQUITETURA	24
4.2 FILTRO DE AJAX	25
4.3 LIMITAÇÕES E REGIMENTO	26
4.4 OTIMIZAÇÃO DA RECURSOS	27
4.5 FILA DE TRÁFEGO E O PRU	28
4.6 OPÇÕES DE PRODUÇÃO	29
4.7 AÇÃO E NAVEGAÇÃO	30
4.8 INTERAÇÕES JAVASCRIPT	31
4.9 ITERAÇÃO DO LAYOUT	32
4.10 COMO GERAR UM ARQUIVO	33
4.11 DECIDIR O QUE VER	34
4.12 DECIDIR O QUE VER	35
4.13 DECIDIR O QUE VER	36
4.14 COMPONENTES RICHFACES	37
4.15 JAVASCRIPT	38

SUMÁRIO

1 INTRODUÇÃO.....	7
1.1 OBJETIVOS GERAIS.....	9
1.2 OBJETIVOS ESPECÍFICOS.....	9
1.3 JUSTIFICATIVA.....	10
2 REFERENCIAL TEÓRICO.....	11
2.1 MVC (MODEL VIEW CONTROLLER)	12
2.2 JAVA.....	13
2.3 J2EE (JAVA ENTERPRISE EDITION)	14
2.4 SERVLETS.....	14
2.5 JSP (JAVASERVER PAGES)	15
2.6 JSF (JAVASERVER FACES)	16
2.7 AJAX4JSF E RICHFACES.....	18
2.8 AJAX.....	18
2.9 JAVASCRIPT.....	19
2.10 XML.....	20
3 METODOLOGIA.....	21
4 RESULTADOS.....	22
4.1 VISÃO GERAL ARQUITETURA RICHFACES.....	23
4.2 FILTRO DE AJAX.....	24
4.3 LIMITAÇÕES E REGIMENTO.....	26
4.4 OTIMIZAÇÃO DA REQUISIÇÃO AJAX.....	27
4.5 FILA DE TRÁFEGO E DE PROTEÇÃO CONTRA INUNDAÇÕES.....	30
4.6 OPÇÕES DE PROCESSAMENTO DE DADOS.....	32
4.7 AÇÃO E NAVEGAÇÃO.....	33
4.8 INTERAÇÕES JAVASCRIPT.....	34
4.9 ITERAÇÃO DOS ATRIBUTOS DOS COMPONENTES AJAX.....	36
4.10 COMO ENVIAR UMA SOLICITAÇÃO AJAX.....	38
4.11 DECIDIR O QUE VAI SER ENVIADO.....	38
4.12 DECIDIR O QUE VAI SER ALTERADO.....	39
4.13 DECIDIR O QUE PROCESSAR.....	40
4.14 COMPONENTES RICHFACES.....	41
4.14.1 < rich:gmap >.....	41

Introdução

4.14.2 < a4j:poll >.....44

4.14.3 < rich:tree >.....46

4.14.4 < a4j:status >.....55

5 CONCLUSÃO.....58

REFERÊNCIA.....59

Trabalho realizado por

Departamento de Engenharia

A primeira parte

Comentário: O trabalho tem como objetivo apresentar os problemas de performance e estabilidade, bem como as regras de negócio com a implementação. Vale salientar que um servidor que usa este tipo de tecnologia pode ter um desempenho muito melhor, uma vez que cada solicitação recebida de uma CGI requer a criação de um novo processo.

Em seguida, vamos ao conteúdo. Similarmente às CGI's, servem as aplicações web, de modo que tenham algumas funcionalidades. Uma aplicação web, diferentemente das CGI's, também não precisa ser executada em outro processo, pois a própria aplicação é executada no mesmo processo do servidor.

No entanto, com esta tecnologia, a implementação das aplicações web é muito mais simples, pois não é necessário criar um novo processo para cada solicitação. Isso resulta em um desempenho muito melhor, uma vez que a aplicação é executada no mesmo processo do servidor.

O JSP (Java Server Pages) é uma tecnologia que permite a criação de páginas web dinâmicas. Ele é baseado no Java e utiliza o modelo MVC (Model-View-Controller). O JSP é executado no servidor e gera HTML dinamicamente para o navegador. Ele suporta o padrão WAP 2.0 (Wireless Application Protocol) e possui uma capacidade de escalabilidade muito alta.

1 INTRODUÇÃO

As aplicações WEB enriqueceram muito desde o início de sua utilização, quando os sites possuíam apenas conteúdo estático, ou seja, o conteúdo de uma página não podia ser modificado em tempo de execução.

Com o tempo os sites passaram a oferecer páginas com conteúdo dinâmico e personalizado trazendo para os usuários da WEB a possibilidade de interagir com dados através de, por exemplo, utilitários de pesquisa, internet bankings, serviços de pagamentos on-line, jogos interativos, lojas virtuais e muito, muito mais.

A primeira tecnologia voltada para a construção de páginas dinâmicas foi Common Gateway Interface (CGI). Eles, porém, apresentam problemas de portabilidade e escalabilidade, além de mesclarem as regras de negócio com a visualização. Vale salientar que um servidor que usa este tipo de tecnologia pode ter seu desempenho comprometido, uma vez que cada solicitação recebida de uma CGI requer a criação de um novo processo.

Em seguida vieram os servlets. Similarmente às CGIs, servlets são pequenos programas feitos em Java que encapsulam algumas funcionalidades inerentes à sua aplicação WEB. Diferentemente das CGIs, servlets são objetos Java que não precisam ser executados em outro processo, pois o processamento é executado dentro de uma thread do processo do servidor.

No entanto, eles ainda não resolvem o problema da separação das regras de negócio da visualização, dificultando a manutenção. Posteriormente surgiram as páginas JSP (JavaServer Pages). Elas são facilmente codificadas e produzem conteúdos reutilizáveis. Assim como os servlets, as JSPs também não resolvem o problema da manutenção das aplicações. Logo em seguida surge o JSF (Java Server Faces), uma tecnologia que incorpora características de um framework MVC (Model View Controller) para WEB e de um modelo de interfaces gráficas baseado em eventos, capaz de desenvolver aplicações web de forma ágil.

O JSF levou a várias empresas desenvolverem frameworks que agregam mais valor ao mesmo. O RichFaces, da Jboss, é um dos mais reconhecidos frameworks disponíveis no mercado para construção de aplicações WEB 2.0 (as chamadas rich client applications), de código aberto, o mesmo, adiciona capacidade de AJAX (Asynchronous Javascript and XML) dentro de aplicações JSF existentes

sem a necessidade de recorrer ao JavaScript e alavanca o framework JSF inclusive no ciclo de vida, validação, na facilidade de conversão e gerenciamento de recursos estático e dinâmico.

O presente trabalho visa criar uma documentação em português sobre o RichFaces com base na documentação oficial de sua mantenedora e apresentar alguns de seus componentes.

OBJETIVOS ESPECÍFICOS

Objetivo Geral

Objetivos Específicos

Apresentar a Tecnologia

Apresentar a Tecnologia

Apresentar o Framework

Apresentar alguns dos

Objetivos

Objetivos Específicos

Objetivos Específicos

1.1 OBJETIVOS GERAIS

Apresentar a integração entre JSF (*JavaServer Faces*) e AJAX (*Asynchronous Javascript and XML*), através do *Framework RichFaces*.

1.2 OBJETIVOS ESPECÍFICOS

Apresentar a Tecnologia JSF (*JavaServer Faces*);

Apresentar a Tecnologia AJAX (*Asynchronous JavaScript and XML*);

Apresentar o *Framework RichFaces*;

Apresentar alguns dos principais componentes *RichFaces*;

1.3 JUSTIFICATIVA

Atualmente *RichFaces* é um dos mais reconhecidos *framework* para se trabalhar na camada de visualização, sendo considerada uma tecnologia de ponta, tanto pela variedade quanto pela riqueza de seus componentes.

O desenvolvedor que utiliza a biblioteca de componentes *RichFaces* no desenvolvimento de seus projetos consegue criar a diferenciação de sua aplicação dentre as outras do mercado, pois a mesma ficará com interfaces ricas e amigáveis sem abrir mão da qualidade e desempenho da aplicação.

Porém, a escassez de material em português a cerca da biblioteca de componentes *RichFaces*, aumenta a curva de aprendizado, tanto para os desenvolvedores que não dominam o inglês, como também para aos estudantes que venham a se interessar.

Baseado nisso, o mercado possui, de fato, necessidade de material gratuito e em português disponível, que apresente a tecnologia, mostre os requerimentos técnicos, conduza os primeiros passos e apresente os componentes.



2 REFERENCIAL TEÓRICO

2.1 MVC (MODEL VIEW CONTROLLER)

MVC (*Model View Controller* – Modelo Visão Controlador) é um padrão de desenvolvimento de aplicações que visa separar a aplicação em camadas. Essa separação em camadas torna as manutenções na aplicação mais fáceis e rápidas, criando cada camada independente uma da outra (SUN, 2002). O padrão MVC é dividido em três camadas como pode ser vista na Figura 1:

- Modelo: responsável por representar os dados e as regras da aplicação.
- Visão: responsável por representar a interface com o usuário.
- Controlador: é responsável por traduzir as interações do usuário com a camada de visão em ações que a camada de modelo irá executar.

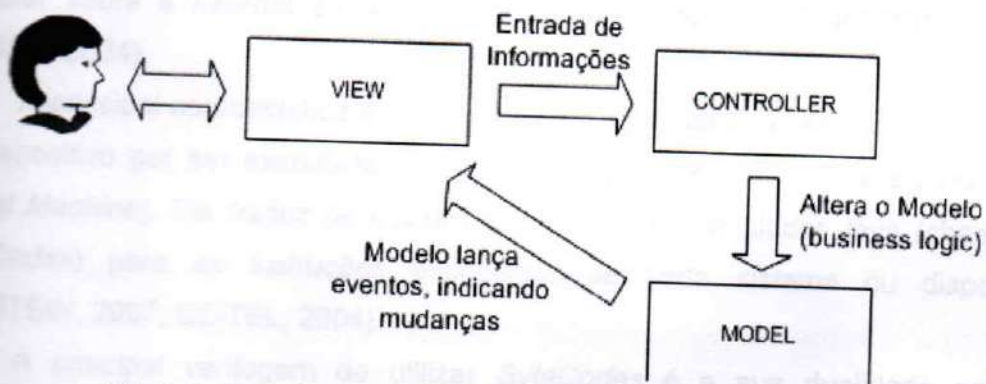


Figura 1 - Fluxo de eventos e informações em uma arquitetura MVC

Fonte: ALMEIDA, 2006

Desenvolvendo utilizando a arquitetura MVC torna o projeto capaz de crescer sem que seja necessário modificar os módulos já desenvolvidos do projeto. No entanto, é necessário um tempo maior para análise do projeto, para que as camadas sejam corretamente divididas. É fundamental o uso dessa arquitetura em ambiente Web, uma vez que a camada de Visão é modificada com certa frequência. Assim, para realizar essa mudança seria necessário modificar apenas a camada de Visão, mantendo as camadas restantes intactas (ALMEIDA, 2006).

2.2 JAVA

A plataforma Java é uma tecnologia voltada ao desenvolvimento de software. Ela teve suas origens em 1991 sob influência de diversas linguagens principalmente C++. Idealizada pela Sun Microsystems, a plataforma era um projeto interno da Sun de codinome *Green* (verde). Os micro-processadores tinham um profundo impacto sobre os dispositivos eletrônicos. Baseado nisso, James Gosling, criador e líder do projeto *Green*, criou uma linguagem chamada Oak, que seria projetada para ser portátil a todos os dispositivos eletrônicos (O'REILLY, 2008).

O projeto *Green* esteve perto de ser cancelado quando a explosão dos dispositivos eletrônicos não ocorreu quando a Sun havia previsto. Então em 1993 a *World Wide Web* (Rede Mundial de Computadores, também conhecida como Internet) explodiu em popularidade. A equipe da Sun observou o potencial de se usar Java para adicionar conteúdo dinâmico nesse ambiente, então em 1995 a linguagem Oak foi rebatizada para Java, então a plataforma foi adaptada para executar sobre a Internet através do navegador Netscape (ECKSTEIN, 2007), (DEITEL, 2004).

A principal característica é ser portátil, ou seja, ser independente de sistema ou dispositivo por ser executada sobre uma máquina virtual chamada JVM (*Java Virtual Machine*). Ela traduz os códigos gerados pelo compilador Java (chamados *ByteCodes*) para as instruções específicas de cada sistema ou dispositivo (ECKSTEIN, 2007; DEITEL, 2004).

A principal vantagem de utilizar *ByteCodes* é a sua dualidade entre a portabilidade e a ausência da necessidade do pré-processamento típico dos compiladores, onde, ele irá produzir o mesmo resultado em qualquer sistema ou dispositivo que possua a JVM instalada (ECKSTEIN, 2007; DEITEL, 2004).

2.3 J2EE (JAVA ENTERPRISE EDITION)

J2EE (*Java Enterprise Edition*) é uma plataforma Java para desenvolvimento e execução de aplicações servidoras baseadas em componentes. Aplicações desenvolvidas com essa tecnologia tendem a ser robustas, escaláveis e extensíveis. Possui ainda as vantagens de ser de fácil implementação seguindo o padrão MVC, portátil a diversas plataformas e reuso de código (MONNOX, 2005).

A plataforma tem o foco de manter o desenvolvimento de soluções mais produtivo, fornecendo uma série de serviços baseados em segurança, gerenciamento de transações e de recursos, entre outros. Provê ainda uma série de APIs para essas aplicações como JSP (*JavaServer Pages*) e *Java Servlets*. Os recursos da plataforma J2EE aumentam a produtividade no desenvolvimento de software provendo uma arquitetura robusta na qual o desenvolvedor teria que implementar por si próprio de outra forma, mantendo o seu foco na aplicação (MONNOX, 2005).

2.4 SERVLETS

A API (*Application Programming Interface* - *Interface de Programação de Aplicativos*) *Java Servlet* é um conjunto de classes Java que definem a interface padrão entre o cliente e o servidor Web. Ela permite que requisições HTTP (*HyperText Transfer Protocol* - *Protocolo de Transferência de Hipertexto*) sejam processadas e retornadas ao cliente pelo servidor Web. Ao serem acionados, os objetos *Servlets* podem enviar a resposta na forma de uma página HTML ou qualquer outro conteúdo MIME¹. Na verdade os *Servlets* podem trabalhar com vários tipos de servidores e não só servidores Web, uma vez que a API

¹ Os tipos Mime representam os tipos de arquivos que são apresentados a seu browser para que ele saiba o que está sendo baixado e apresentar o arquivo de forma apropriada. Por exemplo, um arquivo HTML que é baixado pelo browser é indicado pelo servidor web com o tipo Mime "text/html". Já arquivos GIF recebem o tipo "image/gif". A função Mime Type Mime Type Mime Type (tipo Mime) do painel de controle de sua conta permite atribuir tipos Mime a arquivos de seu site para que os browsers de seus visitantes possam tratar esses arquivos de forma apropriada.

(Application Programming Interface) dos Servlets não assume nada a respeito do ambiente do servidor, sendo independentes de protocolos e plataformas (WAHLI, 2000).

Os Servlets são tipicamente usados no desenvolvimento de sites dinâmicos. Sites dinâmicos são sites onde algumas de suas páginas são construídas no momento do atendimento de uma requisição HTTP. Assim é possível criar páginas com conteúdo variável, de acordo com o usuário, tempo, ou informações armazenadas em um banco de dados (WAHLI, 2000).

Servlets não possuem interface gráfica e suas instâncias são executadas dentro de um ambiente Java denominado de Container. O container gerencia as instâncias dos Servlets e provê os serviços de rede necessários para as requisições e respostas. O container atua em associação com servidores Web recebendo as requisições reencaminhadas por eles. Tipicamente existe apenas uma instância de cada Servlet, no entanto, o container pode criar vários threads de modo a permitir que uma única instância Servlet atenda mais de uma requisição simultaneamente (WAHLI, 2000).

A API é independente de protocolo e plataforma, sendo carregada e executada pela JVM de um container Web. Ela foi uma resposta para a programação CGI (*Common Gateway Interface*), porém possui algumas vantagens sobre ele:

- Eficiência: a cada requisição ao servidor com CGI, um novo processo é criado, necessitando um demasiado poder de processamento do servidor. Já com *Servlets*, independentemente de quantas requisições for realizado, somente um processo será executado pelo servidor;
- Portabilidade: por ser escrito em Java, os *Servlets* são portáveis a qualquer Sistema Operacional;
- Poderoso: os *Servlets* possuem uma série de recursos nos quais os CGI's não possuem como acesso direto ao servidor. Além de tudo, tem acesso a todas as API's da plataforma Java.

Servlets é uma poderosa ferramenta para produzir aplicações complexas, possuindo acesso à todas as API's Java, como JDBC (*Java Database Connectivity*), que provê uma interface para acesso a Bancos de Dados (WAHLI, 2000).

2.5 JSP (JAVASERVER PAGES)

JSP consiste na integração de código HTML (*Hypertext Markup Language*) com código Java para customizar e dinamizar o conteúdo da página Web. O código JSP é compilado em um *Servlet*, executado e então a resposta é retornada em forma de HTML. Dentre suas características, destacam-se a separação do conteúdo estático do dinâmico, ou seja, a lógica da aplicação é separada do código HTML, que é responsável pela interface com o usuário. Assim, é possível alterar o design da página Web sem influenciar na lógica da aplicação. Para facilitar o desenvolvimento do conteúdo dinâmico, o JSP utiliza tags (estruturas de marcação que consistem em breves instruções, tendo uma marca de início e outra de fim) que permitem a inserção de conteúdo que deve ser gerado dinamicamente na página Web. As páginas JSP são traduzidas para um *Servlet*, então é compilado pelo servidor Web e é gerada uma resposta no browser cliente. Tanto o conteúdo dinâmico, quanto o conteúdo estático é processado pelo servidor (WAHLI et al, 2000).

A primeira vez que uma página JSP é carregada pelo container JSP o código Java é compilado gerando um *Servlet* que é executado, gerando uma página HTML que é enviada para o navegador. As chamadas subseqüentes são enviadas diretamente ao *Servlet* gerado na primeira requisição, não ocorrendo mais às etapas de geração e compilação do *Servlet* (WAHLI et al, 2000).

A figura 2 mostra um esquema das etapas de execução de uma página JSP na primeira vez que é requisitada. Na etapa 1 a requisição é enviada para um servidor Web que reencaminha a requisição etapa 2 para o container *Servlet/JSP*. Na etapa 3 o container verifica que não existe nenhuma instância de *Servlet* correspondente à página JSP. Neste caso, a página JSP é traduzida para código fonte de uma classe *Servlet* que será usada na resposta à requisição. Na etapa 4 o código fonte do *Servlet* é compilado, e na etapa 5 é criada uma instância da classe. Finalmente, na etapa 6 é invocado o método *service()* da instância *Servlet* para gerar a resposta à requisição (WAHLI et al, 2000).

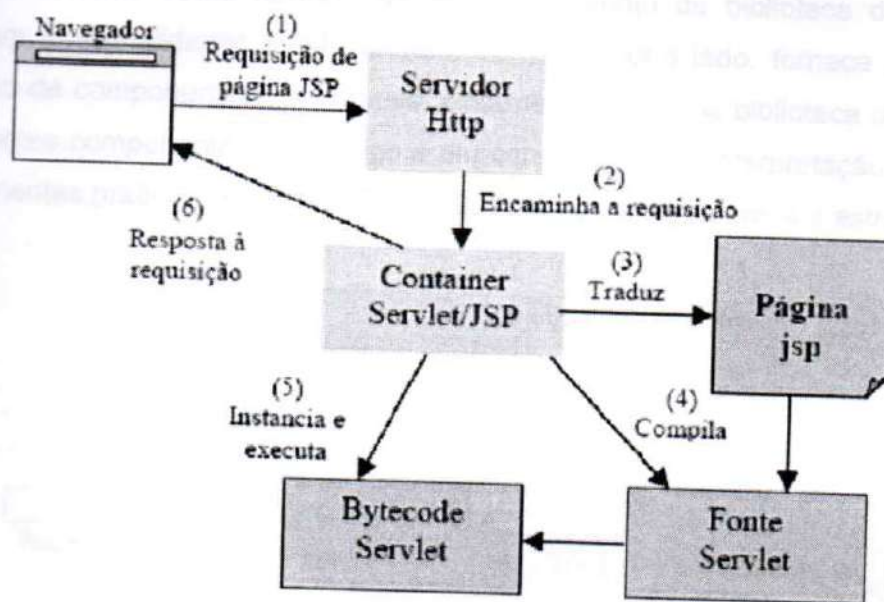


Figura 2 - Etapas da primeira execução de uma página JSP
 Fonte: WAHLI et al, 2000.

2.6 JSF (JAVASERVER FACES)

JSF é um framework para a construção de interfaces de usuário para aplicações da WEB. É construído sobre o conceito introduzido pelo Struts e traz consigo os benefícios de uma estrutura que separa claramente a lógica de negócio em um padrão baseado em componentes de interface do usuário parecido em muitas maneiras com o Swing (SUN, 2008). Abaixo está a figura detalhando como o sistema funciona:

Como se pode ver, JSF também tem uma clara separação entre os componentes para a camada *Model*, *View* e *Controller*. Como Struts, JSF tem uma *servlet front controller*, chamado *FacesServlet*, responsável por receber solicitações de clientes e então executar as ações apropriadas necessárias que são ditadas pelo sistema (SUN, 2008).

Outra semelhança entre Struts e JSF é que ambos fazem uso das *Actions Handlers* separada do *servlet front controller* embora JSF reconheça essa pequena diferença quando comparada ao Struts. JSF e Struts agem similarmente em relação

à camada View. Struts fornece apenas um conjunto de biblioteca de tags que agregam funcionalidades HTML padrão. JSF, por outro lado, fornece seu próprio conjunto de componentes e interfaces, juntamente com uma biblioteca de tags para exibir estes componentes como tags e um componente de interpretação que traduz componentes gráficos em HTML (SUN, 2008). A figura 3 apresenta a estrutura.

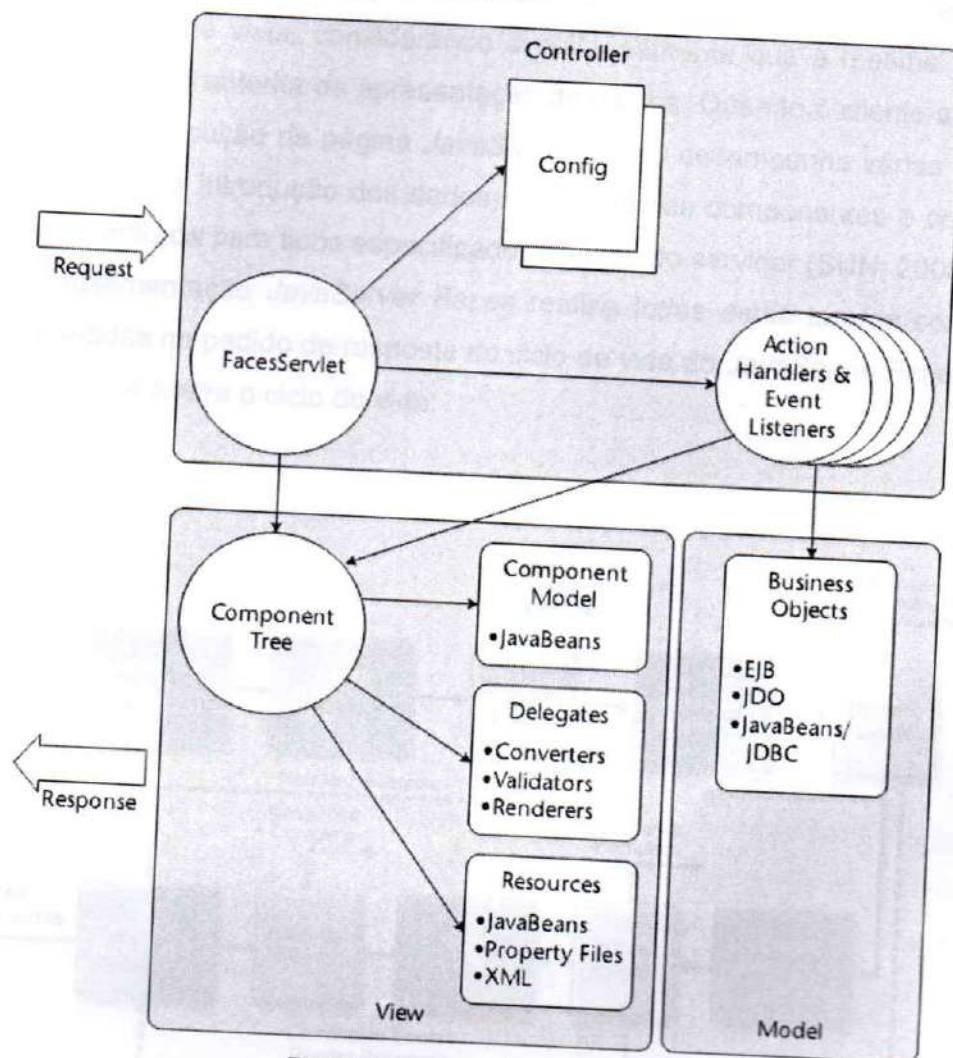


Figura 3: Framework JavaServerFaces

Fonte: SUN, 2008

O ciclo de vida de uma página *JavaServer Faces* é algo semelhante ao de uma página JSP: O cliente faz uma solicitação HTTP para a página, e o servidor responde com a página traduzida para HTML. Contudo, o ciclo de vida do *JavaServer Faces* difere do ciclo de vida de uma JSP, o mesmo é dividido em várias fases, a fim de apoiar o modelo sofisticado dos componentes UI. Este modelo exige

que os dados sejam convertidos e validados, os eventos dos componentes sejam manuseados e, os dados dos componentes a ser propagados estejam em *beans* de forma ordenada (SUN, 2008).

As páginas *JavaServer Faces* também são diferentes de uma página JSP, na medida em que é representado por uma árvore de componentes IU, chamado de vista. Durante o ciclo de vida, a execução da página *JavaServer Faces* deve construir o ponto de vista, considerando simultaneamente que a mesma salvou a partir de um estado anterior da apresentação da página. Quando o cliente apresenta uma página, a execução da página *JavaServer Faces* desempenha várias funções, tais como validar a introdução dos dados, na opinião de componentes e conversão de dados de entrada para tipos especificados do lado do servidor (SUN, 2008).

A implementação *JavaServer Faces* realiza todas estas tarefas como uma série de medidas no pedido de resposta do ciclo de vida do *JavaServer Faces* (SUN, 2008). A figura 4 ilustra o ciclo de vida:

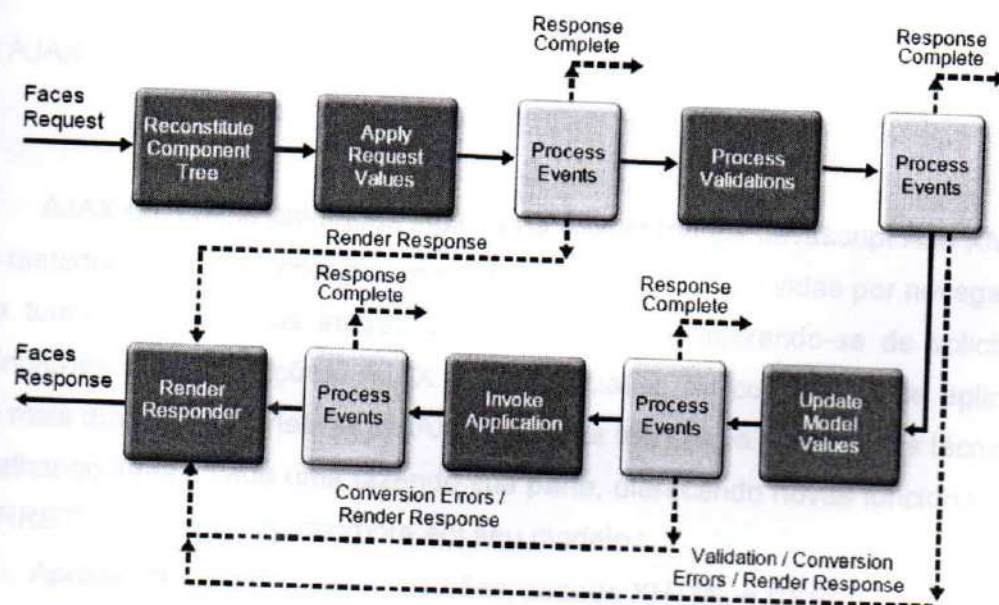


Figura 4 – Ciclo de vida (requisição-processamento) JSF

Fonte: SUN, 2008

2.7 AJAX4JSF E RICHFACES

Em Março de 2007, o código-fonte do Exadel, Ajax4jsf, RichFaces, e os produtos Studio Pro contribuiu para a Comunidade de JBoss.org. Estes produtos de Web 2.0 agora estão disponíveis sob os projetistas da Jboss.org (RED HAT, 2008a).

Ajax4JSF e *RichFaces* são dois *frameworks* que têm como funcionalidade adicionar aos componentes JSF, a capacidade de fazer requisições e renderização de conteúdo via Ajax (*Asynchronous Javascript And XML*), além de possuir seus próprios componentes visuais, enriquecendo a aplicação Web (RED HAT, 2008a).

O *framework RichFaces* se utiliza do *Ajax4JSF* para renderizar seus componentes. Ele era mantido pela Exadel e possuía uma licença comercial, no entanto, ele foi adquirido pela Red Hat, passando a ter seu código livre. Como a Red Hat também é a mantenedora do *Ajax4JSF*, os dois *frameworks* foram incorporados e passaram a ser um só projetos sob o nome de *RichFaces* (RED HAT, 2008a).

2.8 AJAX

AJAX (acrônimo em língua inglesa de *Asynchronous Javascript And XML*) é o uso metodológico de tecnologias como Javascript e XML, providas por navegadores, para tornar páginas mais interativas com o usuário, utilizando-se de solicitações assíncronas de informações. AJAX é uma iniciativa na construção de aplicações web mais dinâmicas e criativas. AJAX não é uma tecnologia e sim várias tecnologias trabalhando juntas, cada uma fazendo sua parte, oferecendo novas funcionalidades. (GARRETT, 2005). AJAX incorpora em seu modelo.:

- Apresentação baseada em padrões, usando XHTML e CSS;
- Exposição e interação dinâmica usando o DOM;
- Intercâmbio e manipulação de dados usando XML e XSLT;
- Recuperação assíncrona de dados usando o objeto XMLHttpRequest;
- e JavaScript unindo todas elas em conjunto.

AJAX (*Asynchronous Javascript And XML*) é uma tecnologia criada para aumentar a agilidade das páginas Web entre o servidor e o cliente. Através desta tecnologia é possível criar páginas mais interativas, onde existe uma simulação de um ambiente semelhante às aplicações desenvolvidas para o ambiente *desktop*. As páginas se tornam mais dinâmicas, permitindo a atualização de todo o conteúdo ou apenas trechos específicos eliminando a necessidade de atualização da página (GARRETT, 2005).

Para isto, a tecnologia utiliza a linguagem de script Javascript e a linguagem de marcação XML. Através dessas duas tecnologias são realizadas requisições assíncronas ao servidor que responde e via Javascript o conteúdo é atualizado (GARRETT, 2005).

O modelo clássico de aplicação web trabalha assim: A maioria das ações do usuário na interface dispara uma solicitação HTTP para o servidor web. O servidor processa algo, recuperando dados, realizando cálculos, conversando com vários sistemas legados, e então retorna uma página HTML para o cliente. É um modelo adaptado do uso original da Web como um agente de hipertexto, porém o que faz a Web boa para hipertexto não necessariamente faz ela boa para aplicações de software (GARRETT, 2005).

Esta aproximação possui muito dos sentidos técnicos, mas não faz tudo que um usuário experiente poderia fazer. Enquanto o servidor está fazendo seu trabalho, o que o usuário estará fazendo? O que é certo, esperando. E a cada etapa em uma tarefa, o usuário aguarda mais uma vez (GARRETT, 2005).

A maior vantagem das aplicações AJAX é que elas rodam no próprio navegador web. Então, para estar hábil a executar aplicações AJAX, basta possuir algum dos navegadores modernos, ou seja, lançados após 2001 (GARRETT, 2005).

2.9 JAVASCRIPT

JavaScript é uma linguagem de script multi-plataforma orientada a objetos. Desenvolvida pela Netscape, inicialmente com o nome de *LiveScript*, a linguagem tinha dois propósitos. Um deles visava ser uma linguagem de script para que os administradores de servidores Web pudessem gerenciar seus servidores e conectar

a outros serviços. O outro propósito visava ser usado no lado do cliente em páginas Web para validar formulários de cadastro e melhorar o design dos sites (GOODMAN, 2001).

Como a linguagem é interpretada, é executada no lado do cliente, sendo assim, se o software cliente não possuir suporte à linguagem ou o mesmo estiver desabilitado, o usuário não poderá visualizar o resultado do *script*. JavaScript não deve ser confundida com Java, são linguagens distintas, possuindo interpretadores diferentes para executar suas linhas de código. Apesar de esse fato ter sido divulgado pela Netscape desde a criação da linguagem, ainda hoje existe confusão entre as duas linguagens. A linguagem foi utilizada nesse projeto como apoio, não sendo requerido pelo cliente possuir suporte à linguagem para permitir que a o portal seja visualizado corretamente (GOODMAN, 2001).

2.10 XML

XML (*Extensible Markup Language*) é uma linguagem de marcação desenvolvida pela W3C em 1996. Foi desenvolvida com o intuito de ser flexível e permitir a troca de dados entre aplicações pela internet, porém com o seu uso, ela pode desde formatar documentos até filtrar dados (RAY, 2001).

Ela provê um formato padrão para documentos, sendo bastante flexível. Os dados são inseridos no documento como texto e são marcados com *tags* que descrevem seus valores. XML não tem um conjunto de *tags* predefinidas para nenhum tipo de aplicação. A linguagem permite que seja criada qualquer tag para qualquer necessidade. A linguagem é muito flexível neste aspecto, porém é muito restrita em relação à sintaxe. Assim como na linguagem HTML (*HyperText Markup Language*), XML também possui um DTD (*Document Type Definition*) que diz onde as *tags* devem ser colocadas, como devem se parecer, quais tipos de caracteres pode ser aceitos para nomear as *tags*, entre outros. Por motivos de interoperabilidade muitas empresas e organizações padronizam tags e valores para que possam trocar dados através de arquivos XML. É uma linguagem muito simples, e por se tratar de apenas texto, pode ser lida por qualquer programa que possa ler um arquivo de texto (RUSTY, 2002).

3 METODOLOGIA

Trabalho objeto de uma pesquisa exploratória da bibliografia oficial da Jboss sobre o RichFaces, na qual, realizou-se um processo de tradução e exploração da possibilidade dos item mais relevantes para entendimento da funcionalidade e utilização do framework. Foram eleitos, trabalhados e comentados alguns dos seus principais componentes, e outros foram apenas citados e apresentados através de figuras.

4 RESULTADOS

O framework RichFaces é implementado como uma biblioteca de componentes que adiciona capacidade Ajax nas páginas existentes, para que não necessite escrever qualquer código JavaScript, ou para substituir componentes existentes por novos já com Ajax. RichFaces permite a página amplo apoio Ajax em vez do tradicional apoio apenas a nível de componente. Assim, você pode definir um evento na página que invoca um pedido Ajax para áreas específicas da página que devem ser sincronizadas com o componente JSF Tree, por exemplo, após o pedido Ajax muda os dados no servidor, de acordo com os acontecimentos que o cliente disparou (RED HAT, 2008b). A figura 5 mostra como funciona:

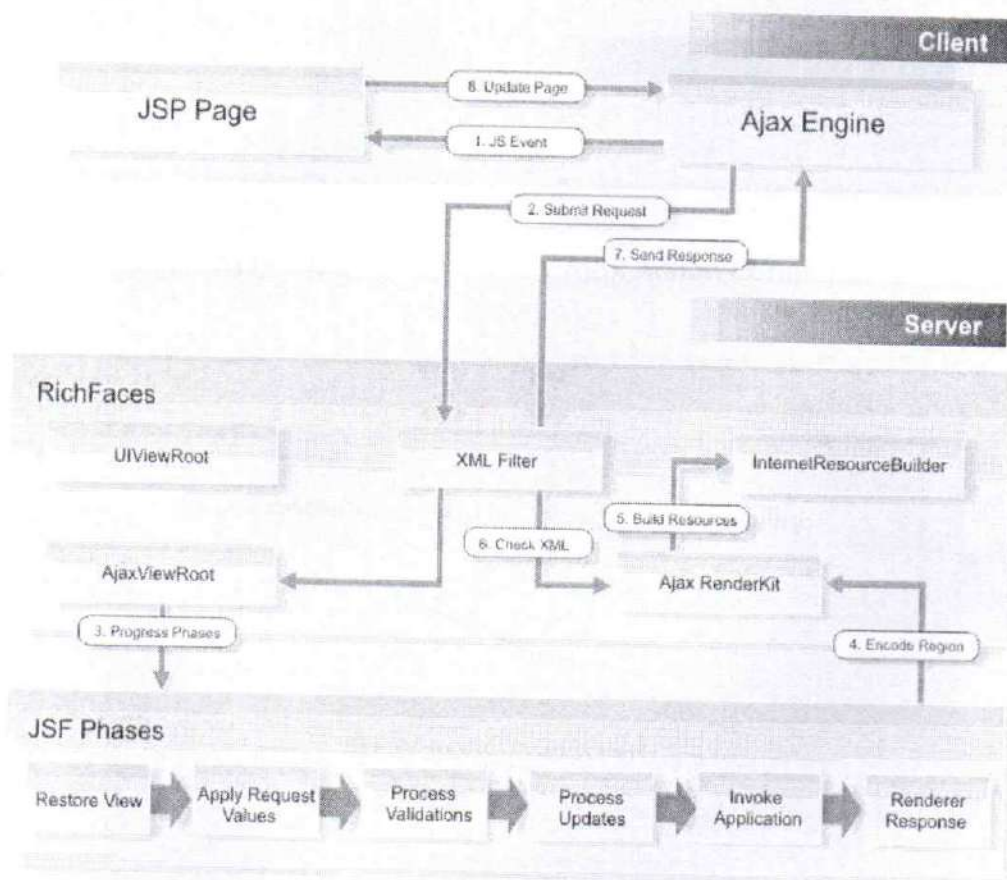


Figura 5. Pedido de processamento de fluxo

Fonte: RED HAT, 2008b

RichFaces permite definir (por meio de tags JSF) diferentes partes de uma página JSF que você deseja atualizar com um pedido Ajax e fornecer algumas opções para enviar os pedidos para o servidor de Ajax. A página JSF também não muda a partir de uma página JSF "regular" e você não precisa ter o JavaScript para escrever quaisquer objeto XMLHttpRequest a mão, tudo é feito automaticamente (RED HAT, 2008b).

4.1 VISÃO GERAL ARQUITETURA RICHFACES

A figura 6 enumera vários elementos importantes do quadro RichFaces:

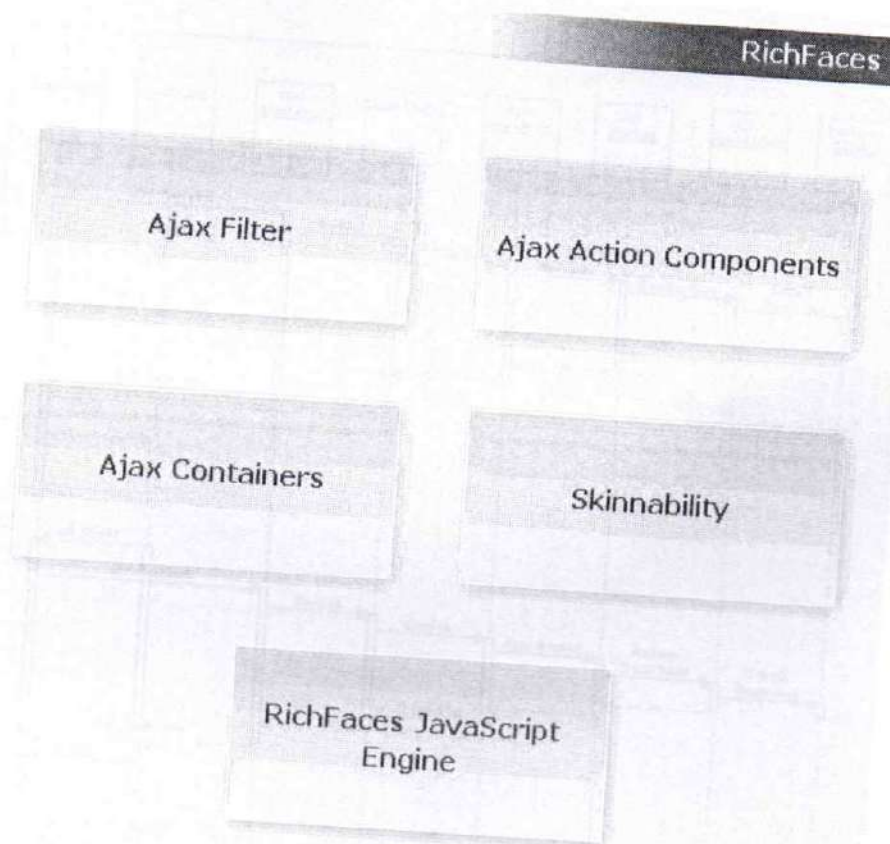


Figura 6. Estrutura do componente Core Ajax
Fonte: RED HAT, 2008b

4.2 FILTRO DE AJAX

Para obter todos os benefícios do RichFaces, você deverá registrar um filtro no arquivo web.xml de sua aplicação. O Filtro reconhece vários tipos de solicitação. As informações necessárias sobre configuração de filtro podem ser encontradas na seção "Configuração de Filtro". O diagrama sequencial na Figura 6 mostra a diferença na transformação de um JSF "regular" e solicitar um pedido Ajax (RED HAT, 2008b).

No primeiro caso, a árvore inteira JSF será codificada, na segunda uma opção que depende do "tamanho" da região Ajax. Como você pode ver, no segundo caso, o filtro analisa o conteúdo de uma resposta Ajax antes de enviá-lo ao lado do cliente (RED HAT, 2008b). Observe a figura 7 para compreender estas duas formas:

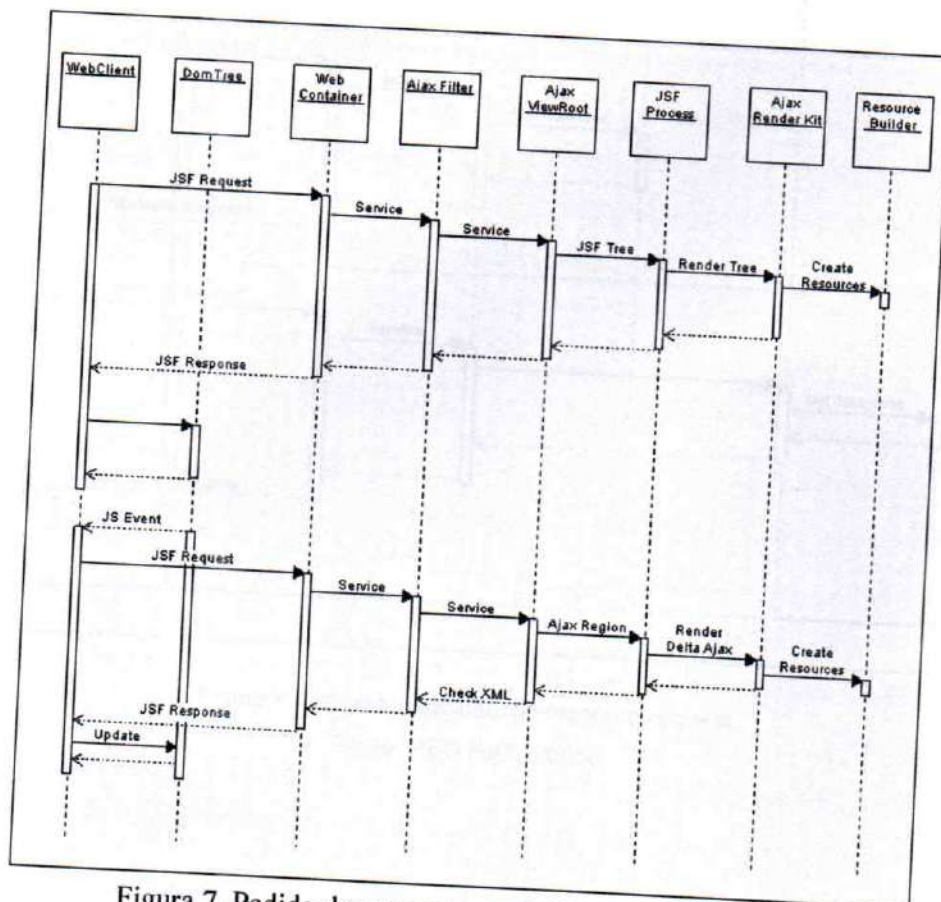


Figura 7. Pedido de processamento diagrama sequencial

Fonte: RED HAT, 2008b

Em ambos os casos, as informações requeridas sobre estática ou dinâmica dos recursos que o aplicativo é registrado solicitações no ResourceBuilder classe (RED HAT, 2008b).

Quando uma solicitação para um recurso vem, o filtro verifica o RichFaces Resource Cache para este recurso, e, se for lá, o recurso é enviado ao cliente. Caso contrário, as pesquisas para o filtro de recursos entre aqueles que são registrados pelo ResourceBuilder. Se o recurso for cadastrado, o RichFaces filtro irá enviar um pedido à ResourceBuilder para criar (entregar) o recurso. A figura 8 mostra as formas de recurso pedido de transformação (RED HAT, 2008b).

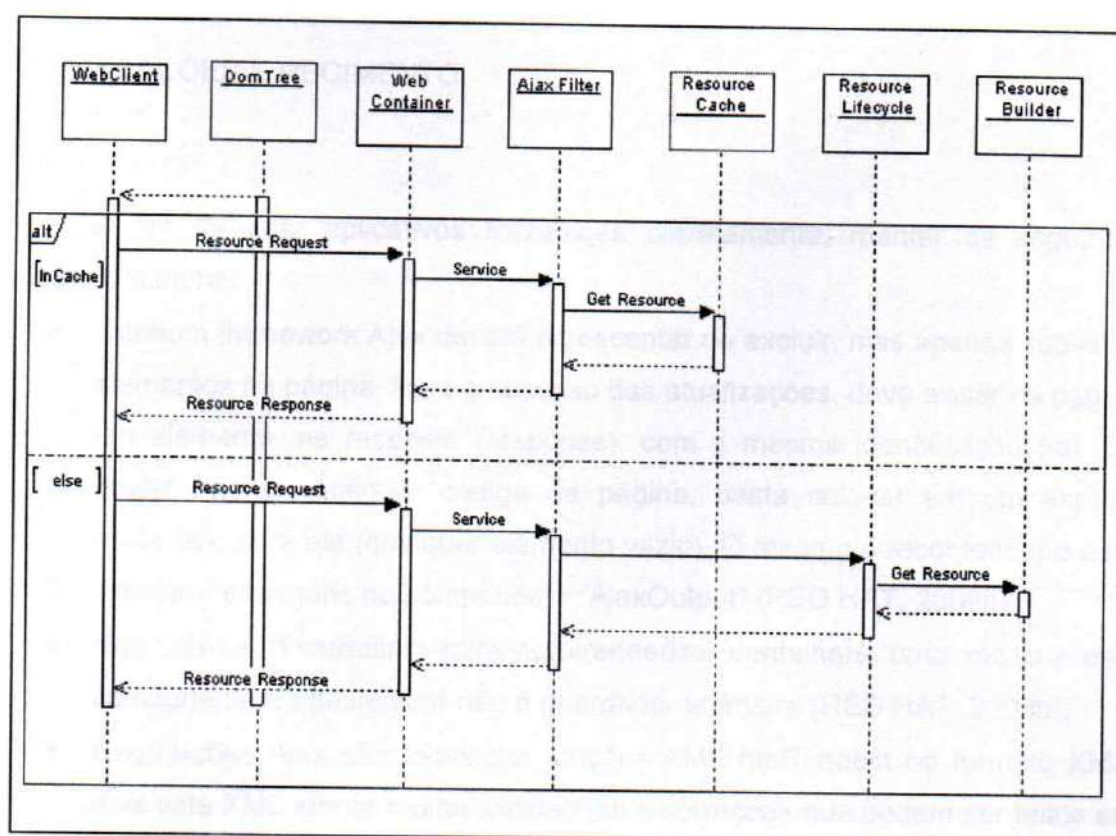


Figura 8. Diagrama sequencial resource request

Fonte: RED HAT, 2008b

Os componentes AJAX Action são, `<a4j:commandButton>`, `<a4j:commandLink>`, `<a4j:poll>`, `<a4j:support>` etc, Você pode usá-los para enviar solicitações de Ajax lado do cliente (RED HAT, 2008b).

- **AJAX Containers:** Ajax Container é uma interface que descreve uma área em sua página JSF que devem ser decodificados (decoded) durante um pedido Ajax. AjaxViewRoot e AjaxRegion são implementações desta interface (RED HAT, 2008b).
- **JavaScript Engine:** O motor de JavaScript RichFaces é executado no lado do cliente. Ela sabe bem como realizar atualização de áreas diferentes na sua página JSF baseada nas informações da resposta Ajax. Não use este código JavaScript diretamente, pois ele está disponível automaticamente (RED HAT, 2008b).

4.3 LIMITAÇÕES E REGIMENTO

A fim de criar aplicativos RichFaces corretamente, manter os seguintes pontos em mente:

- Nenhum framework Ajax deverá acrescentar ou excluir, mas apenas substituir elementos na página. Para o sucesso das atualizações, deve existir na página um elemento, na resposta (response), com a mesma identificação (id). Se quiser anexar qualquer código na página, basta colocar em um espaço reservado para ele (qualquer elemento vazio). O mesmo é recomendado para colocar mensagens no componente "AjaxOutput" (RED HAT, 2008b).
- Não usa-se <f.verbatim> para auto-renderizar containers, uma vez que este componente é transiente e não é guardado na árvore (RED HAT, 2008b).
- Requisições Ajax são feitas por funções XMLHttpRequest no formato XML, mas este XML ignora muitas validações e correções que podem ser feitos em um navegador. Desta forma, criar apenas uma estrita norma de compilação com o sistema de código para HTML e XHTML, sem pular quaisquer elementos ou atributos exigidos. Quaisquer correções necessárias no XML são feitas automaticamente pelo filtro XML no servidor, mas o lote dos efeitos inesperados pode ser produzido por um código HTML incorreto (RED HAT, 2008b).

- O RichFaces ViewHandler coloca-se em frente da cadeia do Facelets ViewHandlers (RED HAT, 2008b).
- Utiliza os seus próprios componentes RichFaces renderers. Sobre o Render Response Fase RichFaces quadro faz uma Traversal do componente árvore, solicita a sua própria renderer e colocou o resultado em resposta a Caras (RED HAT, 2008b).

4.4 OTIMIZAÇÃO DA REQUISIÇÃO AJAX

Componentes Ajax como `<a4j:support>`, `<a4j:commandButton>`, `<a4j:jsFunction>`, `<a4j:poll>`, `<a4j:push>` possuem atributos Ajax comuns, como o `re-renderizar`. Além disso, a maioria dos componentes RichFaces com built-in Ajax apoia esses atributos para ter um efeito semelhante. RichFaces ajuda a expor as suas funcionalidades dos atributos dos componentes Ajax. A maioria dos atributos têm valores padrão. Assim, você pode começar a trabalhar com RichFaces sem saber o uso destes atributos. No entanto, a sua utilização permite afinar o comportamento Ajax exigido tornando-o muito mais harmonioso (RED HAT, 2008b).

"reRender" é um atributo chave essencial. O atributo permite apontar para a(s) área(s) em uma página que deveria ser atualizado como uma resposta sobre uma interação Ajax. O valor do "reRender" é um atributo id do componente JSF ou uma lista id (RED HAT, 2008b). A figura 9 é um exemplo simples:

```
...
<a4j:commandButton value="update" reRender="infoBlock"/>
...
<h:panelGrid id="infoBlock">
...
</h:panelGrid>
...
```

Figura 9. Exemplo do atributo reRender

Fonte: RED HAT, 2008b

O valor do atributo "reRender" da tag <a4j:commandButton> define qual parte (s) da sua página for (em) a ser atualizado. Neste caso, a única parte da página para atualizar é a tag <h:panelGrid> porque o valor do seu ID corresponde ao valor do atributo "reRender". Como é possível vê, não é difícil de atualizar vários elementos da página, bastando apenas listar os seus IDs como o valor de "reRender" (RED HAT, 2008b).

"reRender" usa o algoritmo `UIComponent.findComponent()` (com algumas exceções adicionais) para localizar o componente na árvore de componentes. Como pode se ver, o algoritmo pressupõe várias etapas. Outro passo será usado se o passo anterior não for bem-sucedido. Por conseguinte, você pode definir, mais precisamente, a velocidade para localizar o componente (RED HAT, 2008b). A figura 10 mostra a diferença entre as abordagens (dois botões funcionarão com êxito):

```
...
<h:form id="form1">
...
<a4j:commandButton value="Usual Way" reRender="infoBlock, infoBlock2" />
<a4j:commandButton value="Shortcut" reRender=":infoBlock1,sv:infoBlock2" />
...
</h:form>
<h:panelGrid id="infoBlock">
...
</h:panelGrid>
...
<f:subview id="sv">
<h:panelGrid id="infoBlock2">
...
</h:panelGrid>
...
</f:subview>
...
```

Figura 10. Definido a velocidade para localizar o componente no reRender
Fonte: RED HAT, 2008b

Também possível utilizar a expressão de JSF EL como um valor do atributo reRender. Pode ser uma propriedade de tipos Set, Collection, Array ou String simples. O EL para reRender é resolvido direito antes da fase de resposta render.

Por conseguinte, você pode calcular qual deve ser re-rendered anteriormente a qualquer fase durante o processamento de solicitação do AJAX (RED HAT, 2008b).

Os problema mais comum com o uso de reRender estão em apontá-lo para o componente que tem um atributo "rendered". Observe, que JSF não marca o lugar no navegador DOM onde o resultado do componente deve ser colocado no caso, no qual, a condição "rendered" retorna false. Por conseguinte, depois que o componente torna-se renderizado durante a solicitação do AJAX, o RichFaces fornece o código processado para o cliente, mas não atualiza a página, porque o local para a atualização é desconhecido. Você precisa apontar para um dos componentes pai que não possuem nenhum atributo "rendered". Como alternativa, você pode encapsular o componente com o layout de `<a4j:outputPanel> = "NONE"` (RED HAT, 2008b).

O atributo "ajaxRendered" do `<a4j:outputPanel>` definido como "true" permite definir a área da página que será re-rendered mesmo que ele não seja definido no atributo reRender explicitamente. Ele pode ser útil se você tiver a área de uma página que deve ser atualizada em resposta a qualquer pedido de AJAX. Por exemplo, a figura 11 permite ver a saída da mensagens de erro independentemente da falha, na fase de validação, que o pedido AJAX causa (RED HAT, 2008b).

```
...  
<a4j:outputPanel ajaxRendered="true">  
  <h:messages />  
</a4j:outputPanel>  
...
```

Figura 11. Atributo "ajaxRendered" do `<a4j:outputPanel>` definido como "true"

Fonte: RED HAT, 2008b

O atributo "limitToList" permite ignorar o comportamento do atributo "ajaxRendered" dos `<a4j:outputPanel>`. O `limitToList = "false"` significa apenas a atualização para o(s) domínio(s) mencionado(s) no atributo "reRender" explicitamente. Todos os painéis com saída `ajaxRendered = "true"` é ignorado (RED HAT, 2008b). Um exemplo é colocado na figura 12:

```
...  
<h:form>  
  <h:inputText value="#{person.name}">  
    <a4j:support event="onkeyup" reRender="test" limitToList="true"/>  
  </h:inputText>  
  <h:outputText value="#{person.name}" id="test"/>  
</form>  
...
```

Figura 12. Atributo "limitToList"

Fonte: RED HAT, 2008b

4.5 FILA DE TRÁFEGO E DE PROTEÇÃO CONTRA INUNDAÇÕES

O atributo "eventsQueue" define o nome da fila que será usado para ordenar a futuros pedidos de AJAX. Por padrão, RichFaces não faz fila de requisições de AJAX. Se os eventos são produzidos simultaneamente, viram para o servidor também simultaneamente. Implementações de JSF não garantem que o pedido que está em primeiro lugar será servido ou passado para o ciclo de vida JSF como primeiro. A ordem como os dados do lado servidor serão modificados em caso de pedido simultâneo pode ser imprevisível. Uso do atributo eventsQueue permite evitar a possível confusão. Define-se o nome da fila explicitamente, quando se espera tráfego de AJAX intenso na aplicação (RED HAT, 2008b).

As próximas solicitações fixadas na mesma fila aguardaram até as anteriores que não são respostas AJAX serem processadas, é atualizada a posição na fila e o atributo "eventsQueue" é definido. Além disso, a RichFaces começa a remover os pedidos "semelhantes" da fila. "Semelhantes" são os pedidos produzidos pelo mesmo evento. Por exemplo, na figura 13, só a solicitação mais recente será enviada ao servidor, se você já tiver digitado alguns caracteres e digitar muito rápido os mesmos caracteres sem aguardar a resposta AJAX anterior (RED HAT, 2008b).

```

...
<h:inputText value="#{userBean.name}">
  <a4j:support event="onkeyup" eventsQueue="foo" reRender="bar" />
</h:inputText>
...

```

Figura 13. Verificando pedidos "semelhantes" da fila

Fonte: RED HAT, 2008b

O atributo "requestDelay" define o tempo (em ms), que o pedido irá esperar na fila, antes que fique pronto para enviar. Quando a hora de espera é longo, a solicitação será enviada para o servidor ou removida se o pedido "semelhante" mais recente já estiver em uma fila (RED HAT, 2008b).

O atributo "ignoreDupResponses" determina ordens para ignorar a resposta AJAX produzida pelas requisições, se um pedido "semelhante" mais recente já estiver em uma fila. Com a linha, IgnoreDupResponses = "true", permite não cancelar a solicitação enquanto ela é processada no servidor, mas apenas evita atualizações desnecessárias no lado do cliente, caso a resposta perde a realidade (RED HAT, 2008b).

Definir o eventsQueue juntamente com o "requestDelay" permite proteger a aplicação contra inundações de tráfego desnecessárias e sincroniza a ordem de solicitações de AJAX. Se você tiver várias fontes de solicitações de AJAX, pode definir o mesmo nome de uma fila existente. Isso pode ser muito útil se você tiver componentes AJAX que invocam a solicitação assíncrona dos eventos produzidos pelos usuários. Por exemplo, <a4j:poll> ou <a4j:push>. No caso dos pedidos desses componentes para modificar os mesmos dados, a sincronização pode ser muito útil (RED HAT, 2008b).

O atributo "timeout" é usado para definir o tempo de espera da resposta em um pedido específico. Se não for recebida uma resposta durante este tempo, o pedido é abortado (RED HAT, 2008b).

4.6 OPÇÕES DE PROCESSAMENTO DE DADOS

RichFaces usa abordagem de base para envio de solicitação do AJAX. Isso significa que sempre ao clicar em um botão de AJAX, ou `<a4j:poll>` irá produzir uma solicitação assíncrona, os dados do formulário JSF mais próximo são apresentados com o objeto `XMLHttpRequest`. Os dados do formulário contém os valores a partir do elemento input do formulário e de informações auxiliares como estado salvo dos dados (RED HAT, 2008b).

Quando o valor do atributo "ajaxSingle" for "true", ele ordena incluir apenas um valor do componente atual (juntamente com valores de parâmetros `<f:param>` ou `<a4j:action>` se houver) para o mapa de pedido. No caso do `<a4j:support>`, é o valor do componente pai (RED HAT, 2008b). Um exemplo é colocado na figura 14:

```
...
<h:form>
  <h:inputText value="#{person.name}">
    <a4j:support event="onkeyup" reRender="test" ajaxSingle="true"/>
  </h:inputText>
  <h:inputText value="#{person.middleName}" />
</form>
...
```

Figura 14. Opções de processamento de dados
Fonte: RED HAT, 2008b

Neste exemplo, a requisição contém apenas o componente de entrada, fazendo com que a geração do pedido não ocorra para todos os componentes contidos no formulário, devido o uso da linha `ajaxSingle = "true"` (RED HAT, 2008b).

Com o `ajaxSingle = "true"` se reduz o tráfego futuro, mas não impede a decodificação de outros componentes de entrada do lado do servidor. Alguns componentes JSF, tais como `<h:selectOneMenu>` reconhece os dados que faltam mapear no pedido, o valor como nulo por exemplo, e tentam passar o processo de validação com um resultado de falha. Assim, usa-se `<a4j:region>` para limitar a parte na árvore de componentes que será processada no lado do servidor, quando é necessário (RED HAT, 2008b).

O atributo "immediate" tem a mesma finalidade que qualquer outro componente não-JSF. O padrão "ActionListener" deve ser executado imediatamente (ou seja, durante a fase do ciclo de vida da solicitação de processamento, aplicar os valores da solicitação), em vez de esperar até a fase de invocação do aplicativo. O uso do immediate = "true" é um meio usado quando alguns valores do modelo de dados atualizados não pode ser atualizado com sucesso por causa de um problema ao passar pela fase de validação com êxito. Isso pode ser importante dentro da <h:dataTable> por usar <a4j:region> onde é impossível devido a arquitetura do componente <h:dataTable> (RED HAT, 2008b).

O atributo "bypassUpdates" permite ignorar a fase de atualização do modelo. Ele pode ser útil se você precisa verificar a sua entrada contra o validador disponível, mas não quer atualizar o modelo com esses dados. Observe, que a ação será chamada no final da fase de validação apenas se a fase de validação é passada com êxito (RED HAT, 2008b).

4.7 AÇÃO E NAVEGAÇÃO

O componente AJAX é semelhante a qualquer outro componente não-AJAX JSF como <h:commandButton>, que permite enviar o formulário. É possível usar atributos de "actionListener" e "action" para invocar o método de ação e definir o evento dessa ação (RED HAT, 2008b).

O método "action" deve retornar nulo se você quiser ter uma resposta de AJAX com uma atualização em parte da página. Isto é chamado modo regular (solicitação AJAX gera resposta não-AJAX). Nos casos, nos quais, a ação não retornar null, mas o resultado da ação corresponde a uma das regras de navegação, o RichFaces começa trabalhar no modo regular (RED HAT, 2008b). Este modo pode ser útil em dois casos principais:

- RichFaces permite organizar o fluxo da página dentro do componente <a4j:include>. Este é um cenário típico para o assistente de comportamento. O novo conteúdo é processado dentro da área <a4j:include>. O conteúdo é obtido do estado de navegação do arquivo de configuração faces (normalmente, faces-config.xml). O conteúdo do assistente não é isolado do

resto da página. A página incluída não deve ter sua própria <f:view> (não importa se usar facelets). Você precisa ter um componente de AJAX dentro da <a4j:include> para navegar entre as páginas do assistente. Assim especificada, a atualização de página inteira será executada (RED HAT, 2008b).

- Se quiser envolver os validadores do lado do servidor e navegar para outra página apenas se a fase de validação passar com êxito. Basta substituir <h:commandButton> e <a4j:commandButton> e apontar para o método de ação que aponta para a próxima página. Se o processo de validação falhar, a atualização parcial da página ocorrerá e você verá uma mensagem de erro. Assim especificado, o aplicativo prosseguirá para a próxima página. Certificando-se que a regra de navegação no <redirect/> esta correta é possível evitar "vazamentos" de memória (RED HAT, 2008b).

4.8 INTERAÇÕES JAVASCRIPT

RichFaces permite habilitar AJAX em aplicações JSF sem escrever nenhum código JavaScript. No entanto, é possível invocar código JavaScript se for necessário. Há vários atributos ajax que ajudam a fazê-lo (RED HAT, 2008b).

O atributo "onsubmit" permite chamar código JavaScript antes que uma solicitação AJAX seja enviada. Se o "onsubmit" retornar "false", o pedido de AJAX será cancelado. O código "onsubmit" é inserido antes da chamada de AJAX RichFaces. O onsubmit não deve, por consequência, ter uma declaração "return" caso se deseje que o pedido AJAX seja enviado. Quando se faz necessário invocar uma função JavaScript que retorna "true" ou "false", usa-se a instrução condicional para retornar a algo, apenas quando quando se precisa cancelar a solicitação (RED HAT, 2008b). Por exemplo na figura 15:

```
...  
onsubmit="if (mynosendfunct()==false){return false}"  
...
```

Figura 15. Interações JavaScript

Fonte: RED HAT, 2008b

O atributo "onclick" é semelhante ao "onsubmit", mas apenas para componentes "clicáveis" tais como o <a4j:commandLink> e <a4j:commandButton>. Se ele retornar "false", o pedido AJAX será cancelado também (RED HAT, 2008b).

O atributo "oncomplete" permite invocar o código JavaScript após a resposta do AJAX ser retornada e voltar a árvore DOM atualizada do navegador. O RichFaces registra o código para posterior invocação do objeto antes de uma solicitação XMLHttpRequest e antes da solicitação AJAX seja enviada. Isso significa que o código não será alterado durante o processamento do pedido no servidor e se usar o valor JSF EL vinculativo. Além disso, não é possível usar "este" dentro o código, porque ele não apontará o componente onde a solicitação AJAX foi iniciada (RED HAT, 2008b).

O atributo "onbeforeDOMupdate" define o código JavaScript para realizar a chamada após recepção da resposta AJAX e antes de atualizar a árvore DOM do um lado cliente (RED HAT, 2008b).

O atributo "data" permite obter os dados adicionais do servidor durante uma chamada AJAX. Usando o JSF EL para apontar a propriedade do managed bean seu valor será serializado no formato JSON e estará disponível do lado do cliente. É possível realizar consultas ao mesmo usando a variável "data" (RED HAT, 2008b). Por exemplo na figura 16 e 17:

```
...  
<a4j:commandButton value="Update" data="#{userBean.name}"  
oncomplete="showTheName(data.name)" />  
...
```

Figura 16. O atributo "data"

Fonte: RED HAT, 2008b

RichFaces permite serializar não só tipos primitivos em formato JSON, mas também tipos complexos incluindo matrizes e coleções, para isso o bean deve também ser serializável fazendo referência através do atributo "data" (RED HAT, 2008b). Há funções úteis que podem ser utilizadas em JavaScript:

- rich:clientId('Id') - retorna a id do cliente ordenada por id ou null caso o componente com o id especificado não foi encontrado;
- rich:Element('Id') - é um atalho para document.getElementById("#{rich:clientId('id')}")

- rich:component('id') - é um atalho para #{rich:clientId('id')} .Component
- rich:findComponent('id') - retorna uma instância do UIComponent tendo a ordenação pela identificação do componente como um parâmetro.

```
...
<h:inputText id="myInput">
    <a4j:support event="onkeyup" reRender="outtext"/>
</h:inputText>
<h:outputText id="outtext" value="#{rich:findComponent('myInput').value}" />
...
```

Figura 17. O atributo "data"

Fonte: RED HAT, 2008b

4.9 ITERAÇÃO DOS ATRIBUTOS DOS COMPONENTES AJAX

O atributo "ajaxKeys", figura 18 define cadeias de caracteres que são atualizadas após uma solicitação AJAX. Ele permite atualizar vários componentes filhos separadamente sem atualizar a página inteira (RED HAT, 2008b).

```
...
<a4j:poll interval="1000" action="#{repeater.action}" reRender="text">
    <table>
        <tbody>
            <a4j:repeat value="#{bean.props}" var="detail"
ajaxKeys="#{repeater.ajaxedRowsSet}">
                <tr>
                    <td>
                        <h:outputText value="detail.someProperty" id="text"/>
                    </td>
                </tr>
            </a4j:repeat>
        </tbody>
    </table>
</a4j:poll>
...
```

Figura 18. O atributo "ajaxKeys"

Fonte: RED HAT, 2008b

Outros atributos úteis, como o atributo "status" para componentes AJAX (<a4j:commandButton>, <a4j:poll> etc) aponta para um ID de componente de <a4j:status>. Usa-se esse atributo para compartilhar o componente <a4j:status> entre os diferentes componentes AJAX localizados em diferentes regiões (RED HAT, 2008b). O exemplo na figura 19 mostra.

```
...
<a4j:region id="extr">
  <h:form>
    <h:outputText value="Status:" />
    <a4j:status id="commonstatus" startText="In Progress...." stopText="" />
    <h:panelGrid columns="2">
      <h:outputText value="Name"/>
      <h:inputText id="name" value="#{userBean.name}">
        <a4j:support event="onkeyup" reRender="out" />
      </h:inputText>
      <h:outputText value="Job"/>
      <a4j:region id="intr">
        <h:inputText id="job" value="#{userBean.job}">
          <a4j:support event="onkeyup" reRender="out" status="commonstatus"/>
        </h:inputText>
      </a4j:region>
    </h:panelGrid>
    <a4j:region>
      <h:outputText id="out" value="Name: #{userBean.name}, Job: #{userBean.job}" />
      <br />
      <a4j:commandButton ajaxSingle="true" value="Clean Up Form" reRender="name,
job, out" status="commonstatus">
        <a4j:actionparam name="n" value="" assignTo="#{userBean.name}" />
        <a4j:actionparam name="j" value="" assignTo="#{userBean.job}" />
      </a4j:commandButton>
    </a4j:region>
  </h:form>
</a4j:region>
...
```

Figura 19. O atributo "status"

Fonte: RED HAT, 2008b

No exemplo o `<a4j:support>` e o `<a4j:commandButton>` são definidos em diferentes regiões. Os valores atribuídos no "status" para estes componentes apontam para o ID do `<a4j:support>`. Assim, o componente `<a4j:support>` é compartilhado entre dois componentes em diferentes regiões (RED HAT, 2008b).

Outro atributo útil é "focus". Ele aponta para o ID de um componente onde o foco será definido após uma solicitação AJAX (RED HAT, 2008b).

4.10 COMO ENVIAR UMA SOLICITAÇÃO AJAX

Há diferentes maneiras de enviar solicitações AJAX das páginas JSF. Por exemplo, você pode usar as tags `<a4j:commandButton>`, `<a4j:commandLink>`, `<a4j:poll>` ou `<a4j:support>` (RED HAT, 2008b).

Todas estas tags escondem as atividades JavaScript habituais que são necessárias para construção de objetos XMLHttpRequest e um envio de solicitações AJAX. Além disso, eles permitem decidir quais componentes da página JSF serem novamente renderizados em consequência da resposta AJAX, apenas listando as identificações desses componentes no atributo "reRender" (RED HAT, 2008b).

As tags `<a4j:commandButton>` e `<a4j:commandLink>` são usadas para enviar uma solicitação AJAX no evento JavaScript do "onclick".

A tag `<a4j:poll>` é usada para enviar uma solicitação AJAX periodicamente usando um timer.

A tag `<a4j:support>` permite adicionar funcionalidade de AJAX ao padrão de componentes JSF e enviar solicitações AJAX para um evento JavaScript escolhido ("onkeyup", "onmouseover" etc).

4.11 DECIDIR O QUE VAI SER ENVIADO

Existe a possibilidade de descrever qual região na página que vai ser enviada ao servidor, desta forma também é possível controlar que parte do modo de exibição

JSF será decodificado no lado do servidor quando é enviada uma solicitação AJAX (RED HAT, 2008b).

A maneira mais fácil para descrever uma região AJAX na página JSF é não fazer nada, pois o conteúdo entre as tags `<f:view>` e `</f:View>` já é considerado como a região AJAX padrão (RED HAT, 2008b).

Pode-se definir várias regiões AJAX na página JSF (que podem ser aninhadas) usando a tag `<a4j:region>` (RED HAT, 2008b).

Quando for necessário processar o conteúdo de uma resposta AJAX fora da região ativa então o valor do atributo "renderRegionOnly" deve ser definido como "false" ("false" é o valor padrão). Caso contrário, as atualizações AJAX serão limitadas a elementos da área ativa (RED HAT, 2008b).

4.12 DECIDIR O QUE VAI SER ALTERADO

Usando IDs no atributo "reRender" para definir "zonas AJAX" para atualização funciona bem em muitos casos, porém, não será possível usar essa abordagem caso a página contenha, por exemplo, uma tag `<f:verbatim>` e for necessário atualizar seu conteúdo em uma resposta AJAX (RED HAT, 2008b).

O problema com a marca `<f:verbatim/>` como descrito é relacionado com o valor da transientFlag dos componentes JSF. Se o valor deste sinalizador for "true", o componente não deve participar no estado salvar ou restaurar do processo (RED HAT, 2008b).

A fim de fornecer uma solução para este tipo de problemas, o RichFaces usa o conceito de um painel de saída que é definido pela tag `<a4j:outputPanel>`. Ao colocar a tag `<f:verbatim>` dentro do painel de saída, então o conteúdo da tag `<f:verbatim/>` e o conteúdo das outras tags "filhas" do painel podem ser atualizadas com a resposta AJAX. Há duas maneiras de realizar o controle, definindo o valor do atributo "ajaxRendered" como "true" ou definir o valor do atributo reRender de um componente de ação com o ID do painel saída (RED HAT, 2008b).

4.13 DECIDIR O QUE PROCESSAR

O atributo "process" permite definir as identificações dos componentes que serão processados em conjunto com o componente que está marcado como ajaxSingle ou acondicionado a uma região (RED HAT, 2008b).

O uso do atributo "process" ocorre quando precisa se processar apenas dois componentes que estão em regiões diferentes do modo de exibição (view) (RED HAT, 2008b).

e.g. quando é necessário processar apenas dois campos de entrada, e não todos os outros da exibição, e encapsular a primeira entrada a região ou configurar o componente <a4j:support> com ajaxSingle = "true" a segunda entrada não será processada (RED HAT, 2008b). Na figura 20 está uma solução simples:

```
...
<h:inputText value="#{bean.name}" id="name">
  <a4j:support ajaxSingle="true" process="email" event="onblur" reRender="someOut"/>
</h:inputText>
<h:inputTextarea value="#{bean.description}" id="desc" />
<h:inputText value="#{bean.email}" id="email">
  <a4j:support ajaxSingle="true" process="name" event="onblur" reRender="someOut"/>
</h:inputText>
...
```

Figura 20. O atributo "process"

Fonte: RED HAT, 2008b

No exemplo acima quando o campo de entrada com o id = "name" perde o foco, uma solicitação AJAX é enviada. Assim apenas dois campos de entrada (com id = "name" e adicionalmente com id = "e-mail") são processados: decodificação, conversão/validação, as fases de aplicação de valor são executadas (RED HAT, 2008b).

4.14 COMPONENTES RICHFACES

4.14.1 < rich:gmap >

Componente que apresenta o mapa do Google nas aplicações JSF, figura 21. Apresenta todos os a funcionalidade do mapa da Google, é altamente personalizável através de atributos e não é necessário desenvolver JavaScript para usar.

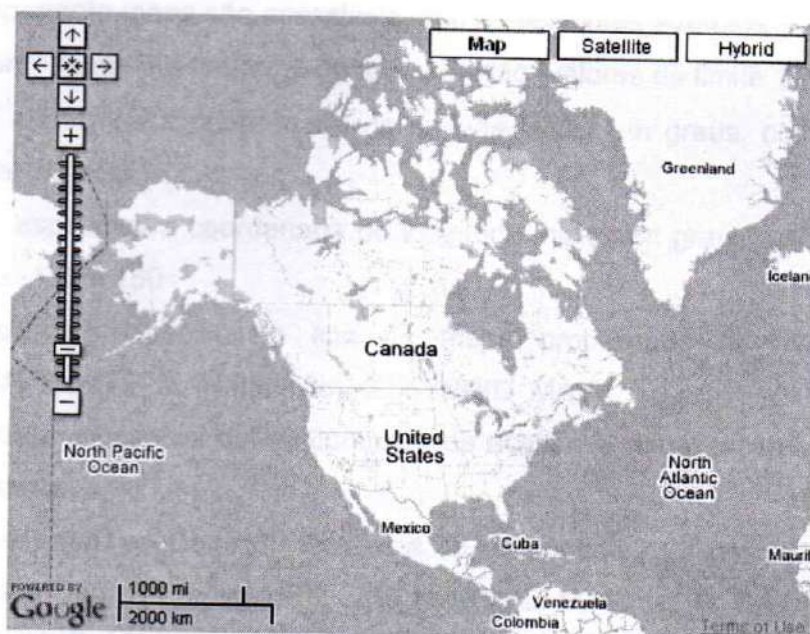


Figura 21. Componente <rich:gmap>

Fonte: RED HAT, 2008b

Para criar o componente, figura 22, com uma tag de página utiliza-se a simples sintaxe:

```
...  
<rich:gmap gmapKey="..." />  
...
```

Figura 22. Criando componente <rich:gmap>

Fonte: RED HAT, 2008b

E para criar o componente dinamicamente usando Java, figura 23:

```
import org.richfaces.component.html.HtmlGmap;  
...  
HtmlGmap myMap = new HtmlGmap();  
...
```

Figura 23. Criando dinamicamente componente <rich:gmap>

Fonte: RED HAT, 2008b

Detalhes de uso: As principais configurações da renderização inicial realizado com o componente mapa são acessíveis com os seguintes atributos:

- "zoom" define um tamanho de aproximação (valores de limite 1 -18);
- "lat" especifica a coordenada de latitude inicial em graus, com um número entre -90 e 90;
- "lng" especifica a coordenada de longitude inicial em graus, com um número entre -180 e 180 ;
- "mapType" especifica o tipo de mapa processado (G_NORMAL_MAP, G_SATELLITE_MAP (padrão), G_HYBRID_MAP) ;

Também é possível definir controles de acesso e visualização sobre o mapa com os atributos:

- "showGMapTypeControl" determina se os controles da definição de tipo de mapa são ativados (G_NORMAL_MAP, G_SATELLITE_MAP, G_HYBRID_MAP);
- "showGScaleControl" determina se os controles de escala estão ativados;
- "showGLargeMapControl" determina se o controle de renderização de escala do mapa é renderizado;

Na figura 24, é mostrada a cidade de Paris após renderização com as seguintes configurações iniciais: lat = "48.44", lng = "2.24" e zoom = "5".

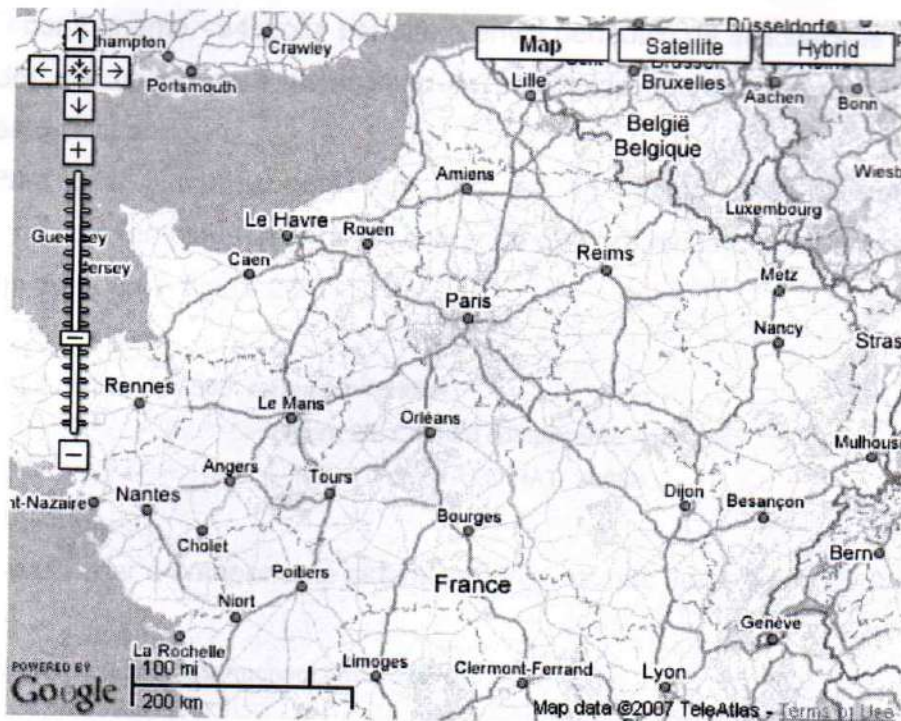


Figura 24. Renderização inicial <rich:gmap>

Fonte: RED HAT, 2008b

Nota: Google map não suporta o formato de página XHTML. Assim, ao usar Facelets e JSF 1.2, deve-se configurar a pagina. Segue a tag na figura 25:

```
<f:view contentType="text/html">...</f:view>
```

Figura 25. Criando dinamicamente componente <rich:gmap>

Fonte: RED HAT, 2008b

4.14.2 <a4j:poll>

O componente <a4j:poll> permite o envio periódico de solicitações AJAX para o servidor e é usado para atualização da página de acordo com um intervalo de tempo especificado.

Para criar o componente com uma tag de página utiliza-se uma simples sintaxe, como exemplo uma linha de comando que renderiza o componente grid em intervalos de 500ms, figura 26:

```
<a4j:poll interval="500" reRender="grid"/>
```

Figura 26. Criando componente <rich:poll>

Fonte: RED HAT, 2008b

E para criar o componente dinamicamente usando Java, figura 27:

```
import org.ajax4jsf.component.html.AjaxPoll;  
...  
AjaxPoll myPoll = new AjaxPoll();  
...
```

Figura 27. Criando dinamicamente componente <rich:poll>

Fonte: RED HAT, 2008b

Atributos chave e formas de uso: O componente <a4j:poll> é usado para realizar uma sondagem periódica dos dados do servidor e para usar o componente, é necessário definir um intervalo de atualização. O atributo "interval" define um intervalo em milissegundos entre a resposta anterior e a próxima solicitação sendo o valor padrão para o atributo "interval" definido como "1000" milissegundos (1 segundo).

O atributo de "timeout" define o tempo de espera da resposta em milissegundos. Se não for recebida nenhuma resposta durante o período definido é anulada a conexão e a próxima solicitação é enviada. O valor padrão para o atributo "timeout" não está definido.

O atributo "enabled" define quando o <a4j:poll> deve enviar um pedido ou não. Ele é necessário para renderizar o <a4j:poll> e aplicar o valor atual do atributo

"enabled". É possível utilizar uma expressão para o atributo "enabled" para apontar para uma propriedade de um bean. Um exemplo, na figura 28, de uso de atributos acima mencionados é colocado abaixo:

```
...
<a4j:region>
  <h:form>
    <a4j:poll id="poll" interval="1000" enabled="#{userBean.pollEnabled}"
reRender="poll,grid"/>
  </h:form>
</a4j:region>
<h:form>
  <h:panelGrid columns="2" width="80%" id="grid">
    <h:panelGrid columns="1">
      <h:outputText value="Polling Inactive" rendered="#{not
userBean.pollEnabled}"></h:outputText>
      <h:outputText value="Polling Active"
rendered="#{userBean.pollEnabled}"></h:outputText>
    <a4j:commandButton style="width:120px" id="control"
      value="#{userBean.pollEnabled?'Stop':'Start'} Polling"
      reRender="poll, grid">
    <a4j:actionParam name="polling" value="#{!userBean.pollEnabled}"
      assignTo="#{userBean.pollEnabled}"/>
    </a4j:commandButton>
  </h:panelGrid>
  <h:outputText id="serverDate" style="font-size:16px" value="Server Date:
#{userBean.date}"/>
</h:panelGrid>
</h:form>
...
```

Figura 28. O atributo "enabled" componente <rich:poll>

Fonte: RED HAT, 2008b

A figura 28 mostra como data e hora são atualizados em uma página em conformidade com os dados obtidos a partir de um servidor. O componente <a4j:poll> envia solicitações para o servidor a cada segundo. O atributo "reRender" contém o mesmo valor ID do <a4j:poll>.

4.14.3 <rich:tree>

O componente é projetado para apresentação de dados hierárquicos e é aplicado para a criação de uma estrutura de árvore com uma capacidade de arrastar-e-soltar, vide figura 29.



Figura 29. <Rich:tree> expandido com elementos filho

Fonte: RED HAT, 2008b

Os recursos chave são:

- Altamente personalizável (look-and-feel);
- Possui capacidade de arrastar e soltar o que habilita a realocação dos nós na árvore;
- Processamento AJAX interno;
- Possibilidade de definir uma representação visual por tipo de nó;
- Suporte a vários elementos raiz em uma árvore;

Para criar o componente com uma tag de página utiliza-se a seguinte estrutura, pois existem duas formas de configurar uma árvore:

1. Através do <rich:recursiveTreeNodesAdaptor> ou do <rich:treeNodesAdaptor>;

2. E sem eles. O primeiro método permite omitir a definição de atributos de "value" e "var" como mostra figura 30:

```
...
<rich:tree>
    <rich:recursiveTreeNodesAdaptor roots="#{fileSystemBean.sourceRoots}" var="item"
nodes="#{item.nodes}" />
</rich:tree>
...
```

Figura 30. Criando componente <Rich:tree>

Fonte: RED HAT, 2008b

A segunda forma requer a definição de alguns atributos, como que é mostrado na figura 31:

```
...
<rich:tree value="#{library.data}" var="item" >
    <rich:treeNode icon="/images/tree/singer.png" >
        <h:outputText value="#{item.name}" />
    </rich:treeNode>
</rich:tree>
...
```

Figura 31. Criando componente <Rich:tree>

Fonte: RED HAT, 2008b

Já para criar o componente dinamicamente usando Java, figura 32:

```
import org.richfaces.component.html.HtmlTree;
...
HtmlTree myTree = new HtmlTree();
...
```

Figura 32. Criando dinamicamente componente <Rich:tree>

Fonte: RED HAT, 2008b

Detalhes de uso: Como foi referido acima o componente `<rich:tree>` permite renderizar qualquer modelo de dados em uma árvore.

O componente interage com o modelo de dados por meio da interface "TreeNode" que é usada para representação em nós de árvore. O atributo "value" do componente `<rich:tree>` contém uma estrutura de nós definida em uma propriedade de um bean. A propriedade mantém uma estrutura de objetos que implementa a interface "TreeNode".

`<rich:treeNode>` tem uma propriedade "Data", na qual, os dados contidos da propriedade são colocados em uma variável no escopo da requisição, sendo seu nome definido com o atributo "var".

É possível desenvolver e usar um padrão próprio de interface "TreeNode" definido pelo desenvolvedor ou usar uma implementação padrão, que é definida com a classe padrão "TreeNodeImpl".

Existe uma classe chamada "XmlTreeDataBuilder" que permite transformar XML em estruturas de objetos que contenham as instâncias de "XmlNodeData" como dados, e que podem ser representados pelo componente `<rich:tree>`.

É possível também, definir uma representação visual de um modelo de dados no nó (para definir um ícone de nó) e seu comportamento em correspondência com os dados contidos neste nó (com o valor do atributo "var"). O comportamento do nó é definido pelos componentes aninhados no `<rich:treeNode>` (e.g. links ou botões). Para estes fins deve-se usar o atributo "nodeFace". Para cada nó da árvore com valor no atributo "nodeFace" é feita previa avaliação e no componente `<rich:treeNode>` seta o valor do atributo "type" igual ao valor do "nodeFace". Veja a figura 33.

```

...
<h:form>
  <rich:tree style="width:300px" value="#{library.data}" var="item" nodeFace="#{item.type}">
    <rich:treeNode type="artist" iconLeaf="/images/tree/singer.png"
icon="/images/tree/singer.png">
      <h:outputText value="#{item.name}" />
    </rich:treeNode>
    <rich:treeNode type="album" iconLeaf="/images/tree/disc.png"
icon="/images/tree/disc.png">
      <h:outputText value="#{item.title}" />
    </rich:treeNode>
    <rich:treeNode type="song" iconLeaf="/images/tree/song.png"
icon="/images/tree/song.png">
      <h:outputText value="#{item.title}" />
    </rich:treeNode>
  </rich:tree>
</h:form>
...

```

Figura 33. O atributo "nodeFace"

Fonte: RED HAT, 2008b

E o resultado da figura 33, na figura 34:

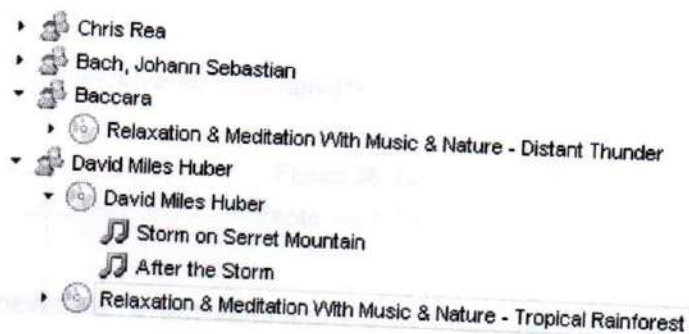


Figura 34. O atributo de uso "nodeFace"

Fonte: RED HAT, 2008b

No exemplo acima, quando cada nó do modelo de dados é processado, os dados contidos na propriedade "data" da interface "TreeNode" são atribuídos a uma variável no escopo da requisição e seu nome é definido com o atributo "var". O valor do atributo "nodeFace" é avaliado em correspondência com os dados atribuídos ao

atributo "var". O componente <rich:treeNode> correspondente (com um valor de atributo "type" igual ao valor do "nodeFace") é usado para a representação de nó. Por exemplo, durante o processamento do modelo de dados, um objeto com o nome de "Chris Rea" foi inserido no atributo "var". Em seguida, o valor do atributo "nodeFace" foi avaliado como "artist". Portanto, para a representação do nó foi usado a <rich:treeNode> com o "type" igual a "artist".

Também é possível atribuir uma expressão como valor, para o atributo "nodeFace". Veja um exemplo na figura 35:

```
nodeFace="#{data.name != 'param-value' ? 'artist' : 'album'}"
```

Figura 35. Atribuindo uma expressão "nodeFace"

Fonte: RED HAT, 2008b

Existem alguns pontos essenciais no uso do atributo "nodeFace": é preciso definir uma notação para os nós sem especificação de tipo e para os nós padrão.

O nó sem especificação de tipo é o primeiro componente <rich:treeNode> (de todas os nós filhos aninhados para o componente <rich:tree>) com atributo "type" não definido e o atributo "rendered" definido. O nó sem especificação de tipo é usado para a representação quando atributo "nodeFace" é nulo.

Nó padrão tem a seguinte apresentação interior, vide figura 36:

```
...  
<h:outputText value="#{varAttributeName}">  
...
```

Figura 36. Nó padrão

Fonte: RED HAT, 2008b

"varAttributeName" é um valor para o atributo "var". Nó padrão é usado nos seguintes casos:

- O atributo "nodeFace" é definido, mas seu valor não é igual a qualquer valor do atributo de "type" de todos os nós filhos;
- O atributo "nodeFace" é definido e seu valor é igual ao valor de alguns atributos "type" de todos os nós filhos, mas o valor do atributo "rendered" para este nó é "false".

Também deve-se lembrar que usando os atributos "type" e "rendered" é possível definir vários componentes <rich:treeNode> com valores do atributo "type" iguais e diferentes valores do atributo "rendered". Ele fornece a possibilidade de definir estilos de representação diferentes para os mesmos tipos de nó. É possível representarmos álbuns que estão disponíveis para venda e de álbuns que não estão disponíveis no exemplo com "artist" e seus álbuns. Veja a figura 37:

```
...
<h:form>
  <rich:tree style="width:300px" value="#{library.data}" var="item" nodeFace="#{item.type}">
    ...
    <rich:treeNode type="album" iconLeaf="/images/tree/album.gif"
icon="/images/tree/album.gif"
      rendered="#{item.exist}">
        <h:outputText value="#{item.name}" />
      </rich:treeNode>
    <rich:treeNode type="album" iconLeaf="/images/tree/album_absent.gif"
icon="/images/tree/album_absent.gif"
      rendered="#{not item.exist}">
        <h:outputText value="#{item.name}" />
      </rich:treeNode>
    ...
  </rich:tree>
</h:form>
...
```

Figura 37. Os atributos "type" e "rendered"

Fonte: RED HAT, 2008b

Este é um resultado do código, figura 38:



Figura 38. Os atributos de uso "type" e "rendered"

Fonte: RED HAT, 2008b

No exemplo, os componentes de `<rich:treeNode>` tem valores iguais ao atributo "type". Consoante ao valor do atributo "rendered" o componente `<rich:treeNode>` correspondente é selecionado para representação do nó. Se um álbum estiver disponível para venda, e o valor do "rendered" para o primeiro componente `<rich:treeNode>` é "true", para a segunda opção é "false". Assim, a primeira `<rich:treeNode>` é selecionada para representação do nó.

O nó também pode ser executado no modo de árvore. Modos podem ser especificados com o atributo "switchType" para o componente de `<rich:tree>`.

- AJAX (valor padrão) – É usada a submissão do AJAX para executar a funcionalidade. Observe, que para as operações de recolher/expandir é enviada uma solicitação AJAX para o servidor (pode ocorrer um atraso curto);
- Server – É usado um formulário regular para a submissão das requisições;
- Client – Todas as operações são executadas totalmente no cliente; sem envolver interação com um servidor. O conteúdo da página é recarregado completamente após cada ação.

Os atributos "icon", "iconCollapsed", "iconExpanded", "iconLeaf" definem as imagens dos ícones para o componente. Também é possível definir ícones usando facetas com os mesmos nomes. Se as "facet" forem definidas, os atributos correspondentes são ignorados e o conteúdo dos "facet" é usado como ícones. Por padrão, a largura de uma zona de "facet" renderizada é 16px, vide figura 39.

```

...
<rich:tree value="#{library.data}" var="item">
  ...
  <f:facet name="icon">
    <h:graphicImage value="/images/tree/singer.png" />
  </f:facet>
  <f:facet name="iconCollapsed">
    <h:graphicImage value="/images/tree/singer.png" />
  </f:facet>
  <f:facet name="iconExpanded">
    <h:graphicImage value="/images/tree/singer.png" />
  </f:facet>
  <f:facet name="iconLeaf">
    <h:graphicImage value="/images/tree/song.png" />
  </f:facet>
  ...
</rich:tree>
...

```

Figura 39. Os atributos "icon", "iconCollapsed", "iconExpanded", "iconLeaf"

Fonte: RED HAT, 2008b

O componente `<rich: tree>` pode ser usado juntamente com `<rich: treeNodeAdaptor>`. Neste caso, não há nenhuma necessidade para especificar os atributos "value" e "var". Além disso, não deve ser definida a representação visual direto na árvore. Neste caso uma tag `<rich: tree>` é aplicada principalmente para definir atributos comuns tais como "ajaxSubmitSelection" etc.

Considerando agora a capacidade arrastar-e-soltar dentro de uma árvore. Todas as zonas, que são consideradas para ser arrastadas, devem ser marcadas. Em termos, na `<rich:tree>` estas zonas correspondem completamente a nós de árvore. Assim, todos os nós arrastáveis devem ser marcados com o atributo "dragType". Em seguida, para as zonas marcadas, onde o nó arrastável poderia ser suprimido, passa-se o tipo de arrasto que o nó terá para o atributo "acceptedTypes" da zona de soltar. O ideal é que cada nó da árvore na estrutura do componente `<rich:tree>` tivesse sua própria chave. Dependendo de como o componente é usado, essas chaves podem ser geradas pelo próprio componente ou podem ser retiradas do modelo de dados do componente. Chaves ajudam a identificar cada nó na árvore.

Por último, a ligação do método, que processará a operação de arrastar e soltar, deve ser referida por meio do atributo "dropListener" do <rich:tree>, figura 40.

```
<h:form>
  <rich:tree style="width:300px" value="#{libraryAjaxTree.data}" nodeFace="#{item.type}"
    var="item" dragIndicator=":treeDragIndicator" dropListener="#{libraryAjaxTree.processDrop}">
    <rich:treeNode type="artist" icon="/images/tree/group.png"
      iconLeaf="/images/tree/group.png" acceptedTypes="album">
      <h:outputText value="#{item.name}" />
    </rich:treeNode>
    <rich:treeNode type="album" icon="/images/tree/cd.png"
      iconLeaf="/images/tree/cd.png" dragType="album" acceptedTypes="song">
      <h:outputText value="#{item.title}" />
      <rich:dndParam name="label" type="drag" value="Album: #{item.title}" />
    </rich:treeNode>
    <rich:treeNode type="song" icon="/images/tree/music.png"
      iconLeaf="/images/tree/music.png" dragType="song">
      <h:outputText value="#{item.title}" />
      <rich:dndParam name="label" type="drag" value="Song: #{item.title}" />
    </rich:treeNode>
  </rich:tree>
</h:form>
```

Figura 40. Capacidade arrastar-e-soltar

Fonte: RED HAT, 2008b

Figura 41 mostra resultado da figura 40:



Figura 41. Operações de arrastar-e-soltar

Fonte: RED HAT, 2008b

4.14.4 < a4j:status >

O componente <a4j:status> gera elementos de exibição do status atual das solicitações AJAX. Existem dois modos de status: solicitação AJAX está no processo ou concluído.

Existem duas formas para definir elementos indicando um status de solicitação, com "StartText" e "StopText" ou com os atributos "Start" / "Stop" definidos no "facet":

A figura 42 e 43 mostram um exemplo:

```
<h:outputText value="Ajax Status:"/>
```

```
<a4j:status id="commonstatus" startText="In progress..." stopText="Complete"/>
```

```
<br />
```

```
<rich:spacer height="5" />
```

```
<h:form>
```

```
  <a4j:commandButton eventsQueue="foo" value="Ajax Request"/>
```

```
</h:form>
```

```
<br />
```

Figura 42. O componente <a4j:status>

Fonte: RED HAT, 2008b

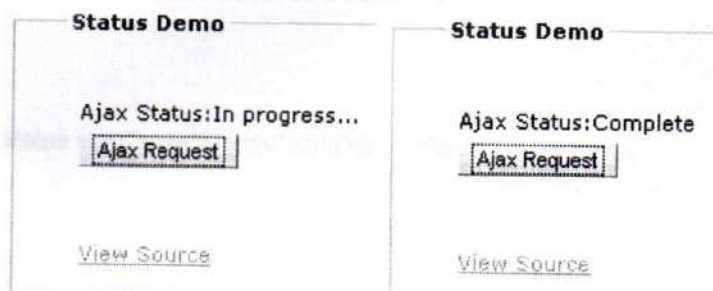


Figura 43. Operações de status

Fonte: RED HAT, 2008b

Para criar o componente dinamicamente usando Java, vide figura 44:

```
import org.ajax4jsf.component.html.HtmlAjaxStatus;  
...  
HtmlAjaxStatus myStatus = new HtmlAjaxStatus();  
...
```

Figura 44. Criar componente <a4j:status> dinamicamente usando Java

Fonte: RED HAT, 2008b

Existem duas formas para a definição de componentes ou outros recipientes, que status das solicitações AJAX é controlado por um componente.

- Definição com o atributo "for" no componente <a4j:status>. Aqui o atributo "for" deve apontar o id do componente <a4j:region>, para que as solicitações sejam controladas pelo componente.
- Definição com o atributo "status" obtido por qualquer componente de ação da biblioteca RichFaces. O atributo deve apontar para o id do componente <a4j:status>. Em seguida, este componente <a4j:status> mostra o status para a solicitação acionada pelo componente de ação.

O componente cria dois elementos, o ou o <div> dependendo o atributo "layout" com conteúdo definido para cada status, um dos elementos (start) é inicialmente oculto. No início de uma solicitação AJAX, o status do elemento é invertido, por conseguinte, o segundo elemento é mostrado e o primeiro é oculto. No final de uma solicitação de resposta, os elementos exibem seu status e ao retornar a solicitação voltam ao seus valores iniciais. Veja na figura 45:

```
<a4j:status startText="Started" stopText="stopped" />
```

Figura 45. Componente <a4j:status>

Fonte: RED HAT, 2008b

O código mostrado na figura 45 é decodificado como mostra figura 46:

```
<span id="_id20:status.start" style="display: none">
  Started
</span>
<span id="_id20:status.stop">
  Stopped
</span>
```

Figura 46. Componente <a4j:status>

Fonte: RED HAT, 2008b

e, após a geração da resposta AJAX, é alterada para, vide figura 47:

```
<span id="_id20:status.start">
  Started
</span>
<span id="_id20:status.stop" style="display: none">
  Stopped
</span>
```

Figura 47. Componente <a4j:status>

Fonte: RED HAT, 2008b

Existe uma forma de agrupar o teor dos elementos de <a4j:status> em elementos de <div>, em vez de . Para usá-lo, apenas define-se o atributo "layout" de "inline" (padrão) para "block".

5 CONCLUSÃO

Mesmo sem contemplar tudo que existe sobre RichFaces é possível através da documentação levantada o entendimento do framework RichFaces em virtude dos exemplos de componentes comentados e de parte significativa de sua documentação oficial se encontrar agora na língua portuguesa.

Os modelos de componentes apresentados podem ser assimilados e de forma genérica podem servir de modelo para outros componentes não contemplados.

REFERÊNCIA

ALMEIDA, Rodrigo Rebouças de. **Model view controller**. 2006. Disponível em: < <http://www.dsc.ufcg.edu.br/~jacques/cursos/map/html/arqu/mvc/mvc.htm> >. Acesso em: 05 abr. 2008.

DEITEL, H.M; DEITEL, P.J. **Java: How to program**. 6.ed. Indianápolis: Prentice Hall, 2004.

ECKSTEIN, Robert. **Java SE application design with MVC**. 2007. Disponível em: < <http://java.sun.com/developer/technicalArticles/javase/mvc> >. Acesso em: 17 abr. 2008.

GARRETT, Jesse James. **AJAX: A new approach to web applications**. 2005. Disponível em: < <http://www.adaptivepath.com/ideas/essays/archives/000385.php> >. Acesso em: 03 mai. 2008.

GOODMAN, Danny. **JavaScript bible**. 4.ed. New York: Hungy Minds, 2001.

MONNOX, Alan. **Rapid J2EE development: an adaptive foundation for enterprise applications**. New Jersey: Pearson PTR, mar. 2005.

O'REILLY Media, Inc. **History of programming languages**. 2008. Disponível em: < http://oreilly.com/pub/a/oreilly/news/languageposter_0504.html >. Acesso em: 12 abr. 2008.

RAY, Erik T. **Aprendendo XML**. Rio de Janeiro: Campus, 2001.

Red Hat, Inc. **Exadel: red hat's enterprise web 2.0 center for excellence**. 2008a. Disponível em: < <http://www.redhat.com/developers/partners/exadel/> >. Acesso em: 19 mai. 2008.

Red Hat, Inc. **RichFaces developer guide**. 2008b. Disponível em: < http://www.jboss.org/fileaccess/default/members/jbossrichfaces/freezone/docs/devguide/en/html_single/index.html >. Acesso em: 19 mai. 2008.

RUSTY, Elliotte; MEANS, W. Scott. **XML in a nutshell**. 2.ed. Sebastopol: O'Reilly, 2002.

Sun Microsystems. **Model view controller**. 2002. Disponível em:
< <http://java.sun.com/blueprints/patterns/MVC.html> > Acesso em: 12 abr. 2008.

Sun Microsystems. **The Java EE 5 tutorial**. 2008. Disponível em:
< <http://java.sun.com/javaee/5/docs/tutorial/doc/index.html> > Acesso em: 10 mai. 2008.

WAHLI, Ueli et al. **Servlet and JSP programing with IBM webSphere studio and visualAge for Java**. San Jose: Red Books, 2000.