FACULDADE DE TECNOLOGIA DE SÃO PAULO

THIAGO AURELIANO

INTEGRAÇÃO JAVASERVER FACES E AJAX

São Paulo

2011

FACULDADE DE TECNOLOGIA DE SÃO PAULO

THIAGO AURELIANO

INTEGRAÇÃO JAVASERVER FACES E AJAX

Monografia submetida como exigência
parcial para a obtenção do Grau de
Tecnólogo em Processamento de Dados

Orientador: Prof. Irineu Francisco de Aguiar

São Paulo

2011

DEDICATÓRIA

Primeiramente quero agradecer à minha família, em especial à minha amada esposa Dáphini, que esteve sempre presente em minha vida, me dando apoio, incentivando a continuar, persistir, não desistir.

Meus amigos e professores também foram fundamentais para a conclusão desta caminhada.

Thiago Aureliano

AGRADECIMENTOS

Inúmeras pessoas contribuíram para a minha evolução neste período.

Agradeço ao Professor Irineu pelo auxílio na conclusão deste trabalho.

Aos meus amigos Marcelo M. Nóbrega, Rodrigo Torres, Roberto Akio entre muitos outros que fizeram parte da minha vida durante o período que estive na FATEC.

Agradeço também aos meus amigos e colegas de trabalho que sempre me incentivaram e auxiliaram, ajudando-me a crescer pessoal e profissionalmente.

Thiago Aureliano

“O sucesso é ir de fracasso em fracasso sem perder o entusiasmo.”

Winston Churchill

RESUMO

O desenvolvimento de aplicativos Web não é uma tarefa fácil utilizando tecnologia Java. Mas em todo mundo, muitas bibliotecas foram criadas para auxiliar os desenvolvedores nas suas aplicações, os **frameworks**.

Os *frameworks* auxiliam os desenvolvedores deixando mais simples a criação de uma aplicação, pois é um conjunto de bibliotecas com práticas gerais encapsuladas. Para a criação de aplicações Web temos diversos *frameworks* existentes, como por exemplo, o **JavaServer Faces**.

JavaServer Faces é desenhado para o desenvolvimento de aplicações Web baseado em componentes. A construção de aplicações através de componentes de interface com o usuário (GUI), e a conexão desses componentes a objetos de negócios é relativamente fácil usando esse framework.

Outra característica do JSF é a utilização apenas de requisições síncronas, ou seja, o cliente invoca um evento que cria uma requisição, para o qual, o servidor realiza o processamento requisitado. Em seguida a página é totalmente reconstruída, mesmo que esta seja igual à anterior, e exibida no browser do cliente.

Outro projeto que tem igual sucesso é a tecnologia chamada AJAX, e que utiliza principalmente JavaScript e XML, onde juntos são capazes de tornar o navegador mais interativo, utilizando-se de solicitações assíncronas de informações.

Ao longo do desenvolvimento do trabalho, será feito um estudo entre as duas tecnologias comentadas e suas formas de integração, tomando como base idéias de alguns desenvolvedores reconhecidos mundialmente (e citados neste estudo) e bibliotecas de componentes já existentes.

ABSTRACT

The development of Web applications is not an easy task using Java technology. But worldwide, many libraries have been created to assist developers in their applications, frameworks.

Frameworks help developers making it simple to create an application, it is a set of libraries with general practices encapsulated. For the creation of Web applications have many existing frameworks, such as JavaServer Faces.

JavaServer Faces is designed for the development of component-based Web applications. Building applications with components for user interface (GUI), and connecting these components to business objects is relatively easy using this framework.

Another feature of JSF is the only use synchronous requests, ie, the client invokes an event that creates a request, for which the server performs the processing required. Then the page is completely rebuilt, even though this is the same as before, and displayed in the client browser.

Another project that has the same success is the technology called AJAX, which mainly uses JavaScript and XML, which together are able to make the browser more interactive, using asynchronous requests for information.

Throughout the development of work, a study will be done between the two technologies discussed and their forms of integration, based on ideas of some world-renowned developers (and cited in this study) and libraries of existing components.

LISTA DE FIGURAS

Figura 1: Funcionamento do protocolo HTTP 13

Figura 2: Modelo MVC e seus frameworks 16

Figura 3: Representação da Forma 1 de integração AJAX e JSF..... 28

Figura 4: Representação da Forma 1 de integração AJAX e JSF..... 32

Figura 5: Representação da Forma 1 de integração AJAX e JSF..... 35

Figura 6: Representação da Forma 1 de integração AJAX e JSF..... 37

Figura 7: Representação da Forma 1 de integração AJAX e JSF de “Super-Charge Ájax Data Fetch” 40

Figura 8: Representação da forma 1 de integração AJAX e JSF 42

Figura 9: Representação da forma 1 de integração AJAX e JSF 45

Figura 10: Representação da forma 2 de integração AJAX e JSF 49

Figura 11: Representação da forma 3 de integração AJAX e JSF 51

Figura 12: Representação da forma 2 de integração AJAX e JSF 53

LISTA DE SÍMBOLOS, SIGLAS E ABREVIATURAS

AJAX	Asynchronous JavaScript And Xml
DOM	Document Object Model
DWR	Directed Web Remoting
FTP	File Transfer Protocol
GUI	Graphical User Interface
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
JSF	JavaServer Faces
MVC	Model-view-controller
URI	Uniform Resource Identifier
Web	World Wide Web
W3C	World Wide Web Consortion
XML	Extensible Markup Language

SUMÁRIO

1. INTRODUÇÃO 11

1.1. CONTEXTO 11

1.2. MOTIVAÇÃO PARA O TEMA 14

1.3. OBJETIVO 14

1.4. ORGANIZAÇÃO DO TRABALHO 14

2. JAVASERVER FACES 15

2.1. INTRODUÇÃO 15

2.2. CONTEXTO JSF 16

2.2.1. CICLO DE VIDA DO JSF 16

2.2.2. CONTROLADOR FRONTAL 17

2.2.3. VIEW ID 17

2.2.4. FACESCONTEXT 17

2.2.5. BACKING BEANS 18

2.2.6. FACES-CONFIG.XML 18

2.3. CICLO DE VIDA 18

2.3.1. FASES DO CICLO DE VIDA 18

2.3.1.1. RESTORE VIEW (RECUPERAR VIEW) 19

2.3.1.2. APPLY REQUEST VALUES (APLICAR VALORES DA REQUISIÇÃO) 19

2.3.1.3. PROCESS VALIDATIONS (VALIDAÇÕES DE PROCESSO) 19

2.3.1.4. UPDATE MODEL VALUES (ATUALIZAÇÃO DOS VALORES DA MODEL) 19

2.3.1.5. INVOKE APPLICATION (INVOCAR A APLICAÇÃO) 20

2.3.1.6. RENDER RESPONSE (RENDERIZA A RESPOSTA) 20

2.4. VANTAGENS DO JSF 20

2.5. DESVANTAGENS DO JSF 20

2.6. CONCLUSÃO 21

3. AJAX 22

3.1. INTRODUÇÃO 22

3.2. CONTEXTO AJAX 22

3.3. AJAX E SUAS CARACTERÍSTICAS 23
3.4. AJAX: PASSO A PASSO 23
3.5. VANTAGENS DO AJAX 24
3.6. DESVANTAGENS DO AJAX 25
3.7. CONCLUSÃO 25
4. FORMAS DE INTEGRAÇÃO 27
4.1. INTRODUÇÃO 27
4.2. SEM COMPONENTES PERSONALIZADOS 27
4.2.1. USANDO JAVASCRIPT PARA INTEGRAR OS COMPONENTES JSF COM AJAX 27
4.2.2. ALTERANDO A ESPECIFICAÇÃO JSF 30
4.3. COMPONENTES PERSONALIZADOS 30
4.3.1. USANDO APENAS UM SERVLET 31
4.3.1.1. UTILIZANDO UM PHASELISTENER APÓS A QUINTA FASE DO CICLO DE VIDA 31
4.3.1.2. REIMPLEMENTANDO OS RENDERIZADORES DOS COMPONENTES 34
4.3.1.3. UTILIZANDO UM PHASELISTENER APÓS A PRIMEIRA FASE DO CICLO DE VIDA 36
4.3.1.4. UTILIZANDO UM PHASELISTENER ANTES DA PRIMEIRA FASE DO CICLO DE VIDA 39
4.3.1.5. IMPLEMENTANDO UM MÓDULO DE CONTROLE APÓS A TERCEIRA FASE DO CICLO DE VIDA 42
4.3.1.6. UTILIZANDO PHASELISTENER EM TRÊS FASES DO CICLO DE VIDA .. 44
4.3.2. USANDO DOIS SERVLETS 47
4.3.2.1. CRIANDO UM SERVLET PARA GERENCIAR REQUISIÇÕES AJAX 48
4.3.2.2. CRIANDO UM SERVLET PARA GERENCIAR AS REQUISIÇÕES AJAX E IMPLEMENTAR UM NOVO CICLO DE VIDA PARA REQUISIÇÕES AJAX 50
4.3.2.3. CRIANDO UM SERVLET PARA GERENCIAR REQUISIÇÕES AJAX E REIMPLEMENTAR OS RENDERIZADORES DOS COMPONENTES 53
4.4. CONCLUSÃO 55
5. CONCLUSÃO FINAL 56
6. REFERÊNCIAS BIBLIOGRÁFICAS 57

1. INTRODUÇÃO

1.1. CONTEXTO

Com a evolução da Internet, informações puderam ser compartilhadas por usuários, de forma on-line e simultânea, ultrapassando grandes barreiras, distâncias e diferentes culturas. Com essa revolução da Internet, tivemos grandes avanços no desenvolvimento de aplicativos Web. Por isso, grandes montantes de recursos são investidos objetivando a melhoria dos processos de desenvolvimento e manutenção de softwares e a obtenção de maior interatividade de suas interfaces, a fim de deixá-los mais atraentes, eficientes e intuitivos aos usuários.

Dentro desse contexto, a arquitetura MVC (Modelo – Visão – Controle) (FOWLER, Martin., 2003. p.330-332.) vem se tornando cada vez mais comum entre os desenvolvedores de aplicações Web.

A idéia central do MVC é dividir o programa em três conjuntos, cada um com sua respectiva responsabilidade. Assim, a camada de Visão seria a responsável pela apresentação dos dados. O Controle seria incumbido de receber os dados inseridos pelo usuário, manipulá-los utilizando-se da camada de Modelo, e retornar alguma informação para a camada de Visão. Já o Modelo teria a função de definir o modo como os dados são organizados no meio persistente.

Como o modelo MVC gerencia múltiplos visualizadores usando o mesmo modelo é fácil manter, testar e atualizar sistemas múltiplos. Também é muito simples incluir novos clientes apenas incluindo seus visualizadores e controles, torna a aplicação escalável e é possível ter desenvolvimento em paralelo para o modelo, visualizador e controle, pois são independentes.

Com a popularização da internet muitas linguagens de programação para Web já foram inventadas, tais como: **VBScript**, **Php**, **Java** (utilizando o padrão Java EE 6, uma plataforma de programação que faz parte da plataforma Java e utilizada para desenvolvimento de aplicações executadas em um servidor (J2EE, Wikipédia)), entre outras.

A linguagem Java, criada em 1992, foi adotada mais rapidamente do que qualquer outra linguagem de programação na história da computação, sendo hoje uma das linguagens

mais utilizadas para desenvolvimento de aplicativos processados na rede mundial de computadores. Este sucesso deve-se a algumas características: por ser uma linguagem orientada a objetos, multiplataforma e pela facilidade de utilização de protocolos para acesso remoto, como **HTTP** e **FTP**.

Para facilitar o desenvolvimento de aplicativos com a linguagem Java foram criados muitos projetos de frameworks e novas tecnologias como: Struts, Spring, DWR, entre outros.

Um dos frameworks mais utilizados até então no desenvolvimento de programas Web é o Struts. Este utiliza arquivos html, xml e classes Java para a construção das aplicações e facilita principalmente o desenvolvimento das camadas de Visão e Controle do modelo MVC.

Entretanto, o Struts começou a perder seu espaço no mercado com o surgimento, a alguns anos, do framework JavaServer Faces (JSF) que contém inovações que agradam em cheio os desenvolvedores como, por exemplo:

- ✓ Possuir um conjunto de componentes de interface prontos e padronizados;
- ✓ Ter um modelo de programação dirigida a eventos, ou seja, que se assemelha muito ao desenvolvimento Java desktop utilizando Swing (padrão J2SE).
- ✓ Permitir que os desenvolvedores construam de forma simples seus próprios componentes aproveitando os fornecidos pelo JSF.

O framework JSF se baseia no modelo de programação Cliente/Servidor (COULOURIS, George; DOLLIMORE, Jean; KINDBERG, Tim, 1994, p. 10-15), no qual os computadores podem ser divididos em dois grupos: Clientes e Servidores.

Os servidores geralmente são mais poderosos e contêm as informações desejadas pelos computadores clientes. Esses computadores também fazem processamento de dados que podem ser muito pesado para máquinas que não possuam velocidade tão expressiva como as que os servidores possuem.

O funcionamento das aplicações baseadas no modelo Cliente/Servidor se dá da seguinte maneira: o programa requisitante (cliente), quando necessário, envia uma mensagem de requisição (request messages) ao servidor, indicando à URI (Uniform Resources Identifiers) e a informação desejada por intermédio dos métodos de requisição (request methods), por exemplo: post, get, delete, entre outros, requisitando dessa forma algum processamento do servidor. Ao finalizar o serviço solicitado, o servidor elabora uma

mensagem de resposta (response messages) ao cliente informando os dados solicitados pelo computador Cliente ou ainda se houve alguma falha durante a execução.

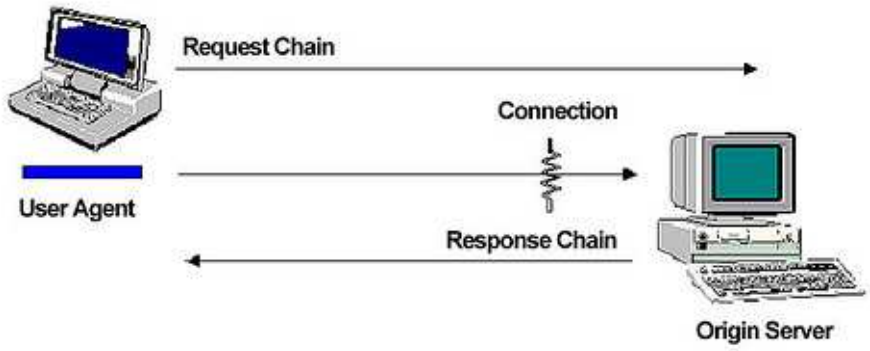

Figura 1: Funcionamento do protocolo HTTP

No caso de aplicações Web, o cliente, com base na resposta do servidor, reconstrói uma página totalmente nova, mesmo se esta for igual à primeira, e a exibe ao usuário. Como dito anteriormente, o JSF é um framework que se baseia no modelo de programação Cliente/Servidor e por utilizar apenas requisições síncronas, a cada evento realizado pelo cliente, é criada uma requisição que vai até o servidor para realizar um processamento. Uma nova página deve ser totalmente construída, mesmo que esta seja igual à anterior e exibida no browser do cliente, que espera todos esses acontecimentos sem usar o sistema.

Uma possível solução para este problema seria a utilização de AJAX (Asynchronous JavaScript and XML), uma tecnologia hoje em grande evidência no contexto da programação Web, sendo utilizada pelo GMail, GoogleMaps entre outras.

Pelo fato de que as páginas não precisam mais ser totalmente reconstruídas quando ocorrer algum evento, o AJAX tem impressionado bastatnte. Utilizando esta tecnologia, as páginas atualizam apenas os dados que foram modificados. Desse modo os programas ficam muito mais interativos e mais rápidos na visão do usuário.

Ainda assim existe pouca integração do AJAX com os frameworks que atuam nas camadas de Visão e Controle do modelo MVC, que são as camadas mais interessadas nas vantagens oferecidas por essa tecnologia.

Isso pode ser observado, por exemplo, no caso do framework JSF contido na especificação *Java Enterprise Edition 6 (Java EE 6)*, que permite a criação de componentes personalizados, mas que ainda possui poucos recursos que facilitam a integração com a tecnologia AJAX.

O objetivo principal deste trabalho é analisar as diversas maneiras de se efetuar uma integração entre JavaServer Faces e AJAX, de forma a facilitar o desenvolvimento das páginas WEB e também sua utilização.

Outro objetivo deste trabalho é um estudo para verificar se o framework JSF poderia incorporar em seus componentes a tecnologia AJAX, oferecendo assim os serviços desta tecnologia aos seus usuários de maneira natural, eliminando dessa forma trabalho adicional aos desenvolvedores.

1.2. MOTIVAÇÃO PARA O TEMA

Por atuar na área de desenvolvimento a 2 anos, utilizando a linguagem JAVA como principal ferramenta de trabalho, identifiquei a oportunidade de aprimorar meus conhecimentos técnicos desenvolvendo o tema escolhido para conclusão do meu curso.

Desta forma, poderei aplicar as informações adquiridas em minha vida profissional, trazendo como conseqüência a evolução da minha carreira.

1.3. OBJETIVO

Estudo comparativo entre formas de integração entre o framework JavaServer Faces e a tecnologia AJAX.

1.4. ORGANIZAÇÃO DO TRABALHO

Inicialmente será feito um estudo aprofundado do framework JSF e da tecnologia AJAX. Depois dessa etapa pretende-se analisar, comparar e verificar as formas de integração dessas duas tecnologias.

2. JAVASERVER FACES

2.1. INTRODUÇÃO

JavaServer Faces é uma ferramenta em Java para construir interfaces usuária para aplicações Web. Seu ponto forte é a maneira fácil para desenvolver a interface usuária, na qual é frequentemente uma das mais tediosas partes do desenvolvimento Web. Embora seja possível construir interface usuária usando tecnologia Java como Servlet e JavaServer Pages, sem auxílio de qualquer ferramenta de auxílio, isso pode conduzir a uma variedade de problemas de desenvolvimento e de manutenção. JavaServer Faces previne esses problemas oferecendo um robusto framework (ferramenta) com padrões de desenvolvimento bem estabelecidos, construído encima da experiência de vários outros frameworks pré-existente no desenvolvimento de aplicações Web em Java.

JavaServer Faces foi criado através da Comunidade Java (JCP) por um grupo de líderes de tecnologia, incluindo a Sun Microsystems, Oracle, Borland, BEA, e IBM junto com um conjunto de empresas que conhecem Java e "Experts" da Web. A especificação original (JSR 127) para JavaServer Faces foi iniciado no meio de 2001 com Amy Fowler como o responsável da especificação inicial. Em 2002,o responsável por dirigir a especificação foi transferida para Ed Burns e Craig McClanahan. Você pode identificar Craig McClanahan como o criador do popular framework para aplicações Web de código livre – Struts. Depois da revisão detalhada e dos votos de aprovação, a especificação e a implementação do JavaServer Faces foram formalmente lançadas a público em março de 2004.

JavaServer Faces tem como objetivo simplificar o desenvolvimento de interface usuária para aplicações Web em Java na seguinte maneiras:

- ✓ Fornecer o desenvolvimento central do componente, independente do cliente, assim melhorando a produtividade do programador e o fácil uso.
- ✓ Simplificar o acesso e o gerenciamento dos dados da aplicação, da interface usuária Web.
- ✓ Gerenciar automaticamente o estado da interface usuária entre as várias requisições e clientes de uma maneira simples e não obstrutiva.

✓ Fornecer um framework de desenvolvimento amigável.

O JavaServer Faces atua principalmente nas camadas de Visão e Controle do modelo MVC (FOWLER, Martin., 2003. p.330-332.), conforme visualizado na figura abaixo.

Figura 2: Modelo MVC e seus frameworks

2.2. CONTEXTO JSF

O mais importante aspecto do desenvolvimento da especificação JavaServer Faces é que sua base trabalha sobre uma tecnologia já existente, J2EE. Isto significa que uma aplicação JSF é realmente apenas um padrão de aplicação J2EE com poucos arquivos de configuração.

Veremos alguns tópicos importantes do JSF.

2.2.1. CICLO DE VIDA DO JSF

É o caminho percorrido por uma requisição JSF, desde a sua criação, com a realização de um evento, até a exibição da página ao usuário.

O ciclo de vida do JSF é divido em seis fases, onde podemos passar por todas essas fases ou por nenhuma, dependendo do tipo de pedido de erros que ocorrem durante as validações, conversões e do tipo de resposta.

2.2.2. CONTROLADOR FRONTAL

FacesServlet é conhecido como “Controlador Frontal” do framework JSF, onde toda requisição é recebida e prossegue a passagem pelas fases até uma resposta ser retornada ao cliente.

Depois de passar pelo Controlador Frontal, a requisição começa a executar a primeira fase do ciclo de vida das requisições JSF.

2.2.3. VIEW ID

É uma árvore de informações de todos os componentes de uma página que instanciou uma requisição.

As View IDs de cada página podem se encontrar em três estados:

- ✓ Novo (New View): View que acaba de ser criada e que possui todas as informações dos componentes vazias.

- ✓ Inicial (Initial View): Na primeira vez em que uma página é carregada as informações dos componentes desta são preenchidas. Portanto, o estado Inicial View é um estado subseqüente ao New View.

- ✓ PostBack: Ocorre quando a View de uma página já existe e precisa apenas ser restaurada para o usuário.

2.2.4. FACESCONTEXT

O objeto FacesContext contém toda a informação de estado que o JSF precisa para gerenciar o estado da GUI de componente para a **requisição e sessão corrente**. O FacesContext armazena a view na propriedade viewRoot, que por sua vez possui todos os componentes JSF da view corrente, e seus estados.

2.2.5. BACKING BEANS

São classes as quais contém parte ou todas as informações dos valores de uma página (GEARY, D.; CAY, Horstmann., 2005. p.1-234.).

2.2.6. FACES-CONFIG.XML

Este é o principal arquivo de configuração do framework JSF, no qual se deve definir todos os backing beans e as regras de navegação que serão utilizadas na aplicação.

2.3. CICLO DE VIDA

O ciclo de vida do framework JSF merece destaque, pois este assunto é essencial para entender em quais momentos de uma requisição JSF a tecnologia AJAX pode ou não atuar.

2.3.1. FASES DO CICLO DE VIDA

Conhecer o ciclo de vida ajuda a compreender o processo de aplicação de valores, conversões, validações, manipulação de eventos (event handling), exceptions e a renderização das páginas.

Este ciclo compreende todas as fases que ocorrem desde a requisição até o fornecimento da respectiva resposta de determinado usuário. Abaixo é apresentada uma visão esquemática deste ciclo.

2.3.1.1. RESTORE VIEW (RECUPERAR VIEW)

O JSF mantém uma cópia da página (view) do cliente no servidor. Durante esta fase cria-se uma nova página (view) caso ela ainda não exista no servidor, ou restaura a página (view) do servidor.

2.3.1.2. APPLY REQUEST VALUES (APLICAR VALORES DA REQUISIÇÃO)

Atualiza os valores da página (view) do servidor com os novos valores dos componentes da página do cliente, como por exemplo: valores de campos de formulários, cookies.

2.3.1.3. PROCESS VALIDATIONS (VALIDAÇÕES DE PROCESSO)

Realiza a conversão e ou a validação dos novos valores dos componentes, por exemplo, na página do cliente quando ele digita o valor 5 para um <input type=’text’>, este valor é atribuído como String (texto) ao componente, antes deste valor ser atribuído para a sua classe Java correspondente é necessário convertê-lo para Integer (número). A validação do campo é efetuada aqui também, por exemplo, a validação de CPF ou se determinado valor está entre 1 e 10.

2.3.1.4. UPDATE MODEL VALUES (ATUALIZAÇÃO DOS VALORES DA MODEL)

Aqui os valores dos componentes (por exemplo, campos de um formulário) enviados pelo cliente são atribuídos aos atributos das classes Java correspondente.

2.3.1.5. INVOKE APPLICATION (INVOCAR A APLICAÇÃO)

Invoca os métodos (action) dos botões e links que estão configurados nos formulários da página do cliente. Repare que nesta fase todos os valores já estão convertidos e validados, logo é possível manipular todos os dados inseridos pelo usuário.

2.3.1.6. RENDER RESPONSE (RENDERIZA A RESPOSTA)

Nesta fase todos componentes renderizam (montam) a página (view) que será exibida para o cliente.

2.4. VANTAGENS DO JSF

- ✓ Conversão e validação de formulários.
- ✓ Arquivo de configuração centralizado representado em um XML.
- ✓ Encoraja o uso consistente de MVC na aplicação.
- ✓ Possuir inúmeras IDEs e plugins que ajudam e facilitam o desenvolvimento de sistemas utilizando este framework.
- ✓ Possuir um modelo de programação orientado a eventos.
- ✓ Oferecer facilidade de construção de novos componentes.
- ✓ É uma ferramenta completa para aplicações Web, apropriada para construção de “front-end” com qualidade.
- ✓ Reutiliza componentes da página.

2.5. DESVANTAGENS DO JSF

- ✓ Pouca documentação em português e maior curva de aprendizado
- ✓ Executar apenas requisições síncronas.

2.6. CONCLUSÃO

O framework JSF desenvolve aplicações web de forma ágil, atuando nas camadas de Visão e Controle. Além disso, é um framework que obteve boa aceitação no mercado e muita disseminação entre os desenvolvedores.

No entanto, um de seus maiores problemas é a ausência de recursos que permitam que as páginas não precisem ser totalmente reconstruídas ao sofrerem algum evento, mesmo que apenas parte de seu conteúdo tenha sido modificado. Tal problema, além de degradar o desempenho dos sistemas, deixa a aplicação mais lenta, desagradando o usuário.

Na continuidade do trabalho vamos estudar AJAX para buscarmos recursos para a integração da tecnologia AJAX com o JSF para solucionar o problema exposto acima.

3. AJAX

3.1. INTRODUÇÃO

AJAX é um conjunto de tecnologia que juntas trazem um estimulo para a construção de aplicações Web. Neste capítulo vamos comentar algumas de suas características mais importantes.

Esse conjunto de tecnologias não era muito conhecido até a Google começar a investir e a utilizar o AJAX em seus aplicativos como: Google Earth, Google Maps, entre outros.

Depois disso, o AJAX começou a ganhar destaque e ser objeto de estudo de muitos desenvolvedores e centros de pesquisa, e começou a ser integrado a diversos aplicativos no mundo inteiro.

3.2. CONTEXTO AJAX

Com a implementação em todos os navegadores do XMLHTTPRequest e do DOM, e a utilização do XML e do JavaScript, os desenvolvedores puderam implementar recursos nas aplicações Web para atualizar algum conteúdo de uma página sem ter que reconstruí-la integralmente, algo que só era possível de ser feito através de artifícios de programação (utilizando iFrames).

Utilizando o XMLHTTPRequest, as novas informações que as aplicações deveriam mostrar ao seu usuário poderiam ser colocadas em algum ramo da árvore e a página poderia carregar apenas o conteúdo alterado.

Em fevereiro de 2005, Jesse James Garret, funcionário de uma empresa de consultoria e desenvolvimento de softwares chamada Adaptive Path, nomeou de AJAX a integração de todas as tecnologias citadas acima, tais como, XML, JavaScript, DOM e outras, no intuito de acrescentar mais dinamismo aos programas. (GARRETT, James J, 2005**)**.

A sigla AJAX significa Asynchronous JavaScript and XML. O primeiro A da sigla significa Assíncrono (Asynchronous), ou seja, o cliente pode solicitar ações ao servidor, porém esse pode continuar interagindo com o sistema, não precisando ficar parado enquanto o

servidor processa a informação. O resto da sigla, “JAX”, demonstra os dois principais componentes utilizados: JavaScript e XML.

3.3. AJAX E SUAS CARACTERÍSTICAS

Segundo o livro AJAX in Action (CRANE, Dave; PASCARELLO, Eric, 2006. 650 p.), a tecnologia AJAX segue três princípios:

1. Os browsers apenas acessam a aplicação: Os browsers são apenas terminais que possuem acesso ao sistema, portanto, cada vez que o usuário necessita requisitar uma informação nova, deve solicitar tal informação ao servidor. Para diminuir esse tráfego de requisições, o conjunto de tecnologias AJAX transfere uma parte da lógica de aplicação para o browser do cliente.

2. Servidores enviam dados: A cada nova requisição o servidor deve enviar ao seu cliente uma resposta contendo, normalmente, um conjunto de dados solicitados.

3. Interação com usuário pode ser contínua: No desenvolvimento de sistemas sem a utilização de AJAX, a cada ação o usuário deveria esperar o processamento do sistema e a renderização da página para continuar a utilizar o programa.

Como o AJAX utiliza o modo de atualização de dados assíncrono, o sistema pode executar processamentos, como por exemplo, validação de campos, sem ter que deixar o usuário esperando que a página seja renderizada.

3.4. AJAX: PASSO A PASSO

Utilizando a tecnologia AJAX temos as seguintes interações entre cliente e servidor descritas em 7 passos:

1. O usuário executa um evento na página, como por exemplo, digitar alguma palavra num determinado campo.

2. Nesta etapa um objeto XMLHTTPRequest é criado e configurado, ou seja, esta instância recebe o método que deve executar no servidor e determinar se esta ação deve ser realizada de maneira assíncrona.

3. O objeto XMLHTTPRequest criado no passo 2 executa a ação configurada também na etapa anterior.

4. Neste passo pode-se fazer a validação de alguns valores do programa, como verificar se o usuário tem permissão para executar esta ação. Esta é uma etapa opcional da tecnologia AJAX.

5. Os resultados da ação do XMLHTTPRequest, executada no Passo 3, são colocados em um XML que será retornado ao cliente.

6. Nesta etapa é chamada uma função JavaScript no cliente, o qual verifica se o método foi executado com sucesso e, em caso afirmativo, são atualizados alguns dados no DOM.

7. Neste último passo os dados que foram modificados na etapa anterior são exibidos ao usuário.

3.5. VANTAGENS DO AJAX

Segue abaixo algumas vantagens na utilização da tecnologia AJAX:

- ✓ **Aumento da performance:** A utilização do AJAX de maneira correta pode aumentar o desempenho dos aplicativos, já que algumas páginas não precisam ser totalmente recarregadas a todo o momento (CRANE, Dave; PASCARELLO, Eric., 2006. 650 p.), diminuindo assim a quantidade de dados trafegados na rede.

  Embora o número de dados que trafegam entre o cliente e o servidor de uma aplicação que utiliza o conjunto de tecnologias AJAX serem maior no início do uso do sistema, a função que representa esta quantidade é quase constante.

Enquanto isso, a função que representa a quantidade de dados de um sistema desenvolvido sem AJAX possui um coeficiente angular muito maior, portanto, apresenta um crescimento do número de informações igualmente superior.

- ✓ **Maior interação com o usuário:** Utilizando AJAX percebe-se também que os aplicativos apresentam uma maior interação com o usuário, pela diminuição do tempo de espera. O usuário pode fazer uso de outras partes da página enquanto aguarda o processamento do servidor e utilizar este conjunto de tecnologias para validar e/ou autocompletar campos.

3.6. DESVANTAGENS DO AJAX

- ✓ **Aumento de uso de JavaScript:** Utilizando AJAX, temos por conseqüência, o aumento do uso da tecnologia JavaScript, muito eficiente, porém causadora de muitos transtornos na manutenção de sistemas. Isso acontece porque usando essa linguagem uma parte da lógica é colocada na camada de Visão do sistema, dificultando assim a compreensão do código do programa.

  Adicionalmente, os navegadores trabalham com esta linguagem de diferentes formas, ou seja, o que funciona no Mozzila Firefox pode não funcionar em outro browser como o Internet Explorer.

  Outro item a ser destacado, é que o usuário pode desabilitar o serviço de JavaScript de seu browser e com isso qualquer página que contiver AJAX não funcionará corretamente.

3.7. CONCLUSÃO

Constatamos que as principais características de AJAX é a interatividade nos sistemas, reduzindo assim um dos maiores problemas das aplicações Web, que obriga o cliente, depois de solicitar algum processamento ao servidor, esperar pelo resultado para continuar seu trabalho.

Concluindo, o AJAX permite também a atualização de apenas algum conteúdo nas páginas dos sistemas, sem a necessidade de carregar toda página.

4. FORMAS DE INTEGRAÇÃO

4.1. INTRODUÇÃO

Começaremos neste capítulo uma análise aprofundada das formas de integração da tecnologia AJAX e do framework JSF. Serão analisadas uma série de publicações, entre as quais se destaca o livro “Pro JSF and AJAX” (JACOBI, Jonas.; FALLOWS, John., . 2006. 435p.), escrito por Jonas Jacobi e John R.Fallows.

Podemos dividir a integração entre a tecnologia AJAX e o framework JavaServer Faces de duas formas: com o uso dos componentes JSF, adicionando na página as funcionalidades AJAX ou com a construção de componentes personalizados.

4.2. SEM COMPONENTES PERSONALIZADOS

Esta é a forma de integração na qual o desenvolvedor não precisa construir componentes JSF para que esses disponibilizem as funções da tecnologia AJAX aos seus usuários. As idéias de integração que tem este princípio são apresentadas abaixo:

4.2.1. USANDO JAVASCRIPT PARA INTEGRAR OS COMPONENTES JSF COM AJAX

Nesta estratégia de integração AJAX e JSF, o desenvolvedor pode utilizar componentes disponibilizados pelo próprio framework JavaServer Faces, não necessitando portanto, criar o seu próprio componente.

Nessa forma de integração o desenvolvedor é o responsável por todos os mecanismos de integração AJAX e JSF, como por exemplo: escrever funções JavaScript, criar objetos XMLHttpRequest, executar requisições assíncronas e atualizar dados de uma página após o DOM sofrer alguma alteração.

A forma de integração comentada é ilustrada na figura abaixo, com as ações realizadas por uma requisição de aplicação.

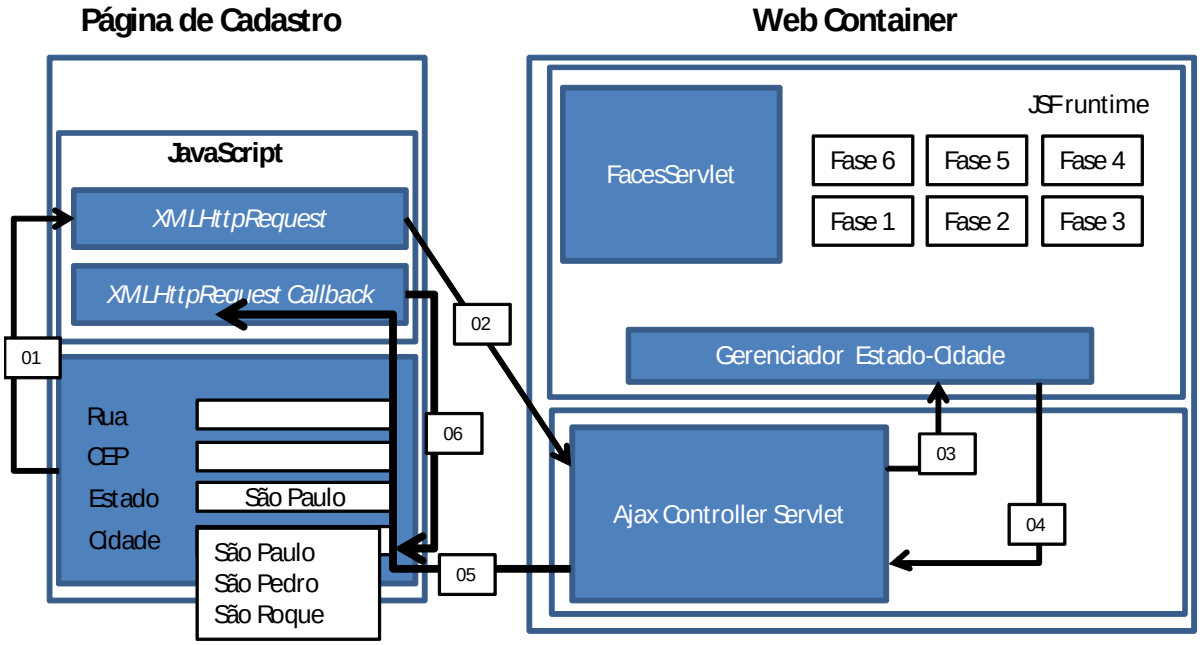

Figura 3: Representação da Forma 1 de integração AJAX e JSF

Os passos executados pela requisição desde a ação do usuário até a renderização da página são:

1. Toda vez que o usuário selecionar um valor para o campo Estado, um novo objeto XMLHttpRequest é instanciado.

2. Neste momento o objeto XMLHttpRequest invoca uma instância de AjaxControllerServlet, que deve ser criada pelo desenvolvedor, passando o método que deve ser executado e os parâmetros necessários para sua realização.

3. A instância de AjaxControllerServlet repassa a responsabilidade de executar o método, passado anteriormente pelo XMLHttpRequest, para a classe capaz de realizá-lo, neste caso a classe GerenciadorEstadoCidade. Uma instância da classe GerenciadorEstadoCidade irá filtrar todas as cidades que pertencem ao valor de Estado selecionado pelo usuário.

4. Depois de o objeto da classe GerenciadorEstadoCidade selecionar as cidades no passo anterior, esse transfere à instância de AjaxControllerServlet todos esses valores filtrados para serem empacotados em um arquivo XML e repassados ao browser do usuário.

5. Nesse momento a função Callback da instância XMLHttpRequest é invocada, repassando assim os registros contidos no XML para a página do usuário.

6. A função Callback da instância de XMLHttpRequest atualiza o DOM da página do cliente com os dados recebidos no XML criado no passo anterior. Nessa forma de integração são utilizados dois Servlets: FacesServlet, que gerencia as requisições JSF e AjaxControllerServlet, que administra as solicitações utilizadas com a tecnologia AJAX.

Como neste exemplo não foi desenvolvido nenhum componente JSF, apesar de reduzir o trabalho de desenvolvimento inicial, esta solução só resolve problemas específicos, neste caso a disponibilização das cidades do estado (UF) selecionado pelo cliente.

Caso o desenvolvedor deseje ter esta mesma funcionalidade em outra página, adotando esta forma de integração AJAX e JSF, ele deverá duplicar todo o código utilizado na primeira página.

Além das desvantagens apresentadas, nessa forma de integração as funções JavaScript, criadas pelo desenvolvedor, ficam nas páginas “jsp”, fazendo com que qualquer usuário do sistema possa analisar facilmente este código, deixando o sistema vulnerável em termos de segurança.

Tais funções JavaScript podem ser construídas automaticamente, como feitas na biblioteca DWR. No entanto, tal biblioteca ainda possui algumas limitações, como a quantidade de funcionalidades disponíveis, por exemplo.

Concluindo, destaca-se que a maior deficiência dessa forma de integração é a necessidade da duplicação de código fonte a cada utilização da funcionalidade que utiliza AJAX, refletindo na manutenção dos programas.

4.2.2. ALTERANDO A ESPECIFICAÇÃO JSF

O principal objetivo é atualizar apenas o conteúdo da página que foi modificado, não necessitando para tal carregar toda a página novamente, nem criar classe adicional para a utilização de requisições assíncronas.

Esta idéia, conhecida como AVATAR, surgiu em janeiro de 2006 apresentada no blog de Jacob Hookom (HOOKOM, Jacob. Blog), desenvolvedor que contribui com idéias e implementações para a Sun para a melhoria do framework JSF.

No JSF, cada componente tem controle sobre si durante todo o ciclo de vida das requisições, com isso, no final de uma requisição invocada por um usuário, todos os componentes de uma página se atualizam.

Assim, as atualizações da árvore DOM ocorreriam utilizando-se a tecnologia AJAX e só seriam atualizados os ramos que sofreram alguma alteração.

Todos os componentes herdariam a capacidade de utilizar a tecnologia AJAX se modificarmos as classes citadas da especificação JSF, sem nenhum esforço adicional do desenvolvedor.

Dentro desse preceito, o desenvolvedor não precisa saber JavaScript. Porém, caso necessário, pode utilizar tal linguagem para sobrescrever algumas funções, adicionar ou melhorar algumas funcionalidades existentes.

A maior vantagem dessa forma de integração é o fato de que o usuário, ao construir seus próprios componentes, não precisa saber e nem se preocupar em como irá integrar esses componentes com a tecnologia AJAX. Após essa análise, a dificuldade e o tempo de desenvolvimento de componentes JSF que utilizam esta tecnologia devem cair consideravelmente.

4.3. COMPONENTES PERSONALIZADOS

Nesta forma de integração AJAX e JSF, o desenvolvedor é obrigado a construir componentes personalizados, o que inicialmente necessita de mais trabalho que a forma anterior.

Mas existem algumas vantagens para o desenvolvedor que personaliza seus próprios componentes, como: reutilizar componentes e dificultar a visualização, por parte dos usuários do sistema, das funções JavaScript.

Existem duas formas de integração da tecnologia AJAX com o framework JSF que envolvem esta idéia, a qual depende do número de Servlets usados para isso, conforme explicado nas seções abaixo:

4.3.1. USANDO APENAS UM SERVLET

Nesta forma, a idéia é que todas as requisições que utilizam ou não o AJAX devam passar pelo Servlet disponibilizado pelo framework JavaServer Faces, conhecido como FacesServlet, e alterar em algum momento o fluxo do ciclo de vida das requisições JSF para que o sistema se beneficie da tecnologia AJAX.

A seguir serão apresentadas algumas idéias de integração entre AJAX e JSF que utilizam apenas um Servlet:

4.3.1.1. UTILIZANDO UM PHASELISTENER APÓS A QUINTA FASE DO CICLO DE VIDA

Esta é uma forma de integração que pode ser representada pela figura abaixo:

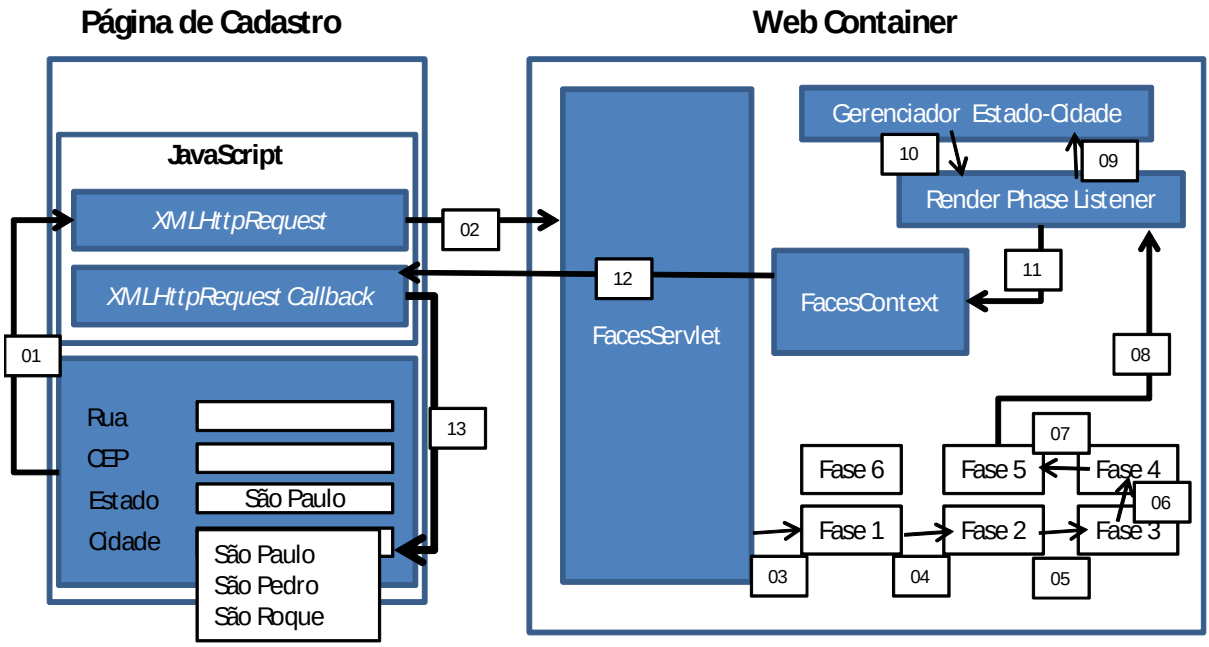

Figura 4: Representação da Forma 1 de integração AJAX e JSF

O roteiro a ser seguido pela requisição, desde a ação do usuário até a renderização da página, é comentado a seguir:

1. Toda a vez que o usuário selecionar um estado na página, um objeto XMLHttpRequest é instanciado.

2. Depois de ser criado, o objeto XMLHttpRequest invoca a instância da classe FacesServlet repassando a esta o método a ser executado juntamente com os parâmetros necessários para que ele seja realizado. No exemplo, o parâmetro é o valor do estado selecionado.

3. Ao receber a requisição, a instância de FacesServlet a repassa para que assim seja executada a primeira fase do ciclo de vida das requisições JSF, chamada de “Restaurar Visão”.

4. Neste momento a requisição já passou pela primeira fase do ciclo de vida das requisições JSF e então começa a executar a segunda fase, chamada de “Aplicar Valores de Requisição”.

5. Ao finalizar a fase anterior, a requisição começa a executar a terceira fase do ciclo, conhecida como “Processar Validações”.

6. Depois da terceira fase do ciclo de vida das requisições JSF, a requisição executa a quarta fase do ciclo, chamada de “Atualizar Valores do Modelo”.

7. Neste momento a requisição inicia a realização da quinta fase do ciclo, conhecida como “Invocar Aplicação”.

8. Ao finalizar a quinta fase do ciclo de vida das requisições JSF, o fluxo da requisição é desviado, em vez de passar para a sexta fase chamada de “Renderizar Resposta” uma instância nomeada de RenderPhaseListener é invocada.

9. (Inclui etapas 9 e 10 do processo) Neste momento, a instância de RenderPhaseListener invoca um objeto da classe GerenciadorEstadoCidade, a fim de que esta filtre todas as cidades que pertencem ao estado selecionado pelo usuário. Depois dessa pesquisa todas as cidades selecionadas são repassadas à instância de RenderPhaseListerner.

11. Ao receber o resultado da pesquisa realizada, o objeto da classe RenderPhaseListerer empacota todos estes registros em um arquivo XML. Ao empacotar os dados a instância de FacesContext é invocada.

12. Ao ser invocado, o objeto de FacesContext executa o método responseComplete() finalizando assim o ciclo de vida das requisições JSF e invocando a função CallBack da instância XMLHttpRequest.

13. A função XMLHttpRequest Callback atualiza o DOM da página de acordo com o XML retornado, mostrando assim as opções de cidades que pertencem ao estado selecionado pelo usuário.

Uma observação importante a respeito dessa forma de integração é o fato de que as requisições que se utilizam da tecnologia AJAX modificam o fluxo de execução antes da

sexta e última fase do ciclo de vida de uma requisição JavaServer Faces, chamada de “Renderizar Resposta”.

Essa forma de integração também é apresentada no artigo “Super-Charge JSF AJAX Data Fetch” (JACOBI, Jonas; FALLOWS, John).

Um dos problemas que pode ser gerado é o fato que se vários PhaseListeners forem adicionados ao ciclo de vida e configurados para serem executados na mesma fase, o desenvolvedor não pode garantir a ordem de sua execução.

Outro problema dessa idéia é que o desempenho de um sistema é inversamente proporcional ao número de PhaseListeners nele adotado, dado ao fato de que todas as instâncias de cada PhaseListener são invocadas nas requisições.

4.3.1.2. REIMPLEMENTANDO OS RENDERIZADORES DOS COMPONENTES

Esta forma de integração AJAX e JSF é uma das apresentadas no artigo “Super-Charge JSF AJAX Data Fetch” (JACOBI, Jonas; FALLOWS, John), conhecida como “The Renderer Approach”.

São adicionadas funcionalidades ao Renderizador, que é disponibilizado pelo framework JSF, a fim de que consiga carregar alguns componentes no usuário sem ter que reconstruir toda a página.

Na ilustração a seguir são exibidas as ações realizadas pela requisição de uma aplicação que faz uso da forma de integração comentada.

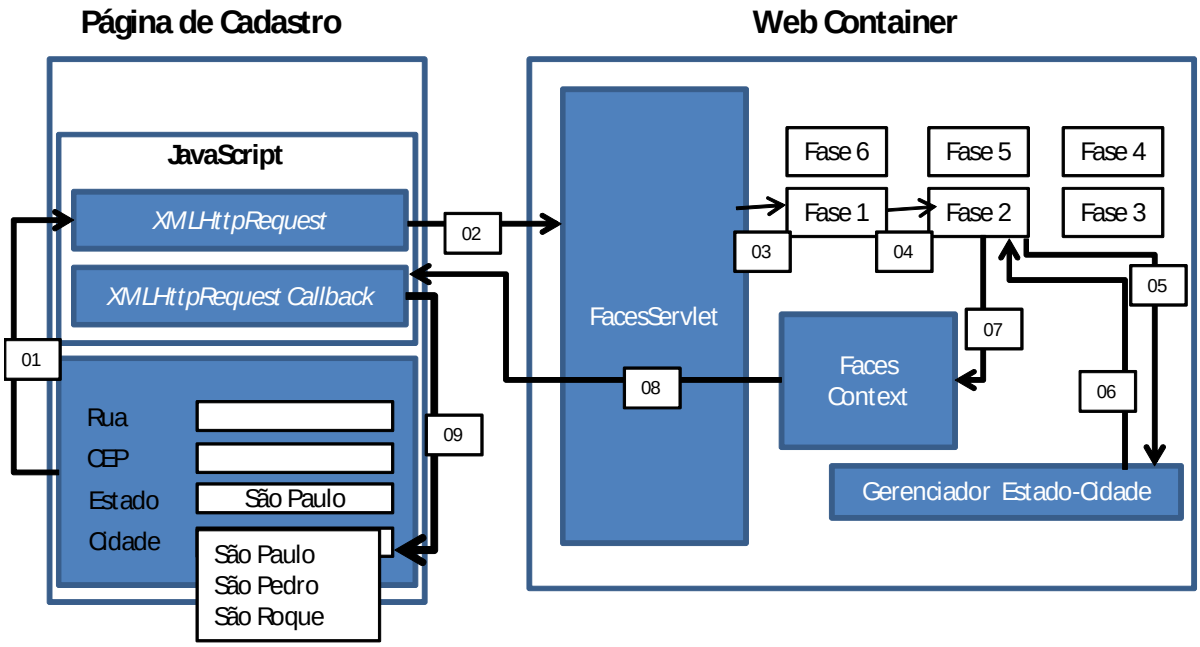

Figura 5: Representação da Forma 1 de integração AJAX e JSF

Os passos executados pela requisição desde a ação do usuário até a página ser reexibida ao usuário são:

1. Toda vez que o usuário selecionar um valor do campo Estado na página, irá ser construído um objeto XMLHttpRequest.

2. O objeto XMLHttpRequest irá invocar então uma instância da classe FacesServlet passando para esta o método que deseja executar juntamente com seus parâmetros.

3. Neste momento a instância da classe FacesServlet repassa a requisição para a fase chamada de “Restaurar Visão”, que é a primeira etapa do ciclo de vida de requisições JSF.

4. Ao finalizar a primeira fase do ciclo de vida das requisições JSF começa a segunda etapa chamada de “Aplicar Valores de Requisição”, a qual é o grande diferencial desta idéia de integração comparando-se com as requisições JSF.

5. (Inclui etapas 5 e 6 do processo) Durante a segunda fase do ciclo de vida das requisições JSF é invocada uma instância da classe GerenciadorEstadoCidade para que esta filtre as cidades pertencentes ao estado (UF) selecionado pelo usuário.

7. Agora a aplicação já contém a lista de cidades do estado (UF) selecionado, portanto o objeto da classe Renderizador invoca o método responseComplete() da classe FacesContext e com isso todas as demais fases do ciclo são ignoradas.

8. A função Callback da instância de XMLHttpRequest é invocada e os dados alterados podem ser renderizados na página do cliente.

9. A função do XMLHttpRequest Callback atualiza o DOM da página colocando neste os registros vindos do XML. Com isso a página mostra ao usuário todas as cidades que pertencem ao estado selecionado.

Uma observação importante refere-se ao fato de que o ciclo de vida das requisições JSF é interrompido após a segunda fase para todos os componentes que tiverem seus renderizados sobrescritos.

Um dos possíveis problemas que podem ocorrer ao fazer uso dessa forma de integração é quando há algum componente que esteja com o valor do atributo immediate igual a true na página em que a requisição foi criada. Dessa forma a lógica do problema é chamada antes da segunda fase do ciclo, denominada “Aplicar Valores de Requisição”, danificando assim o correto funcionamento.

4.3.1.3. UTILIZANDO UM PHASELISTENER APÓS A PRIMEIRA FASE DO CICLO DE VIDA

Tem como idéia principal a de alterar o ciclo de vida das requisições JSF incluindo neste um PhaseListener que atue após a primeira fase do ciclo, conhecida como “Restaurar Visão”.

Todos os passos realizados por uma requisição que se utiliza desta idéia de integração AJAX e JSF podem ser observados na figura abaixo.

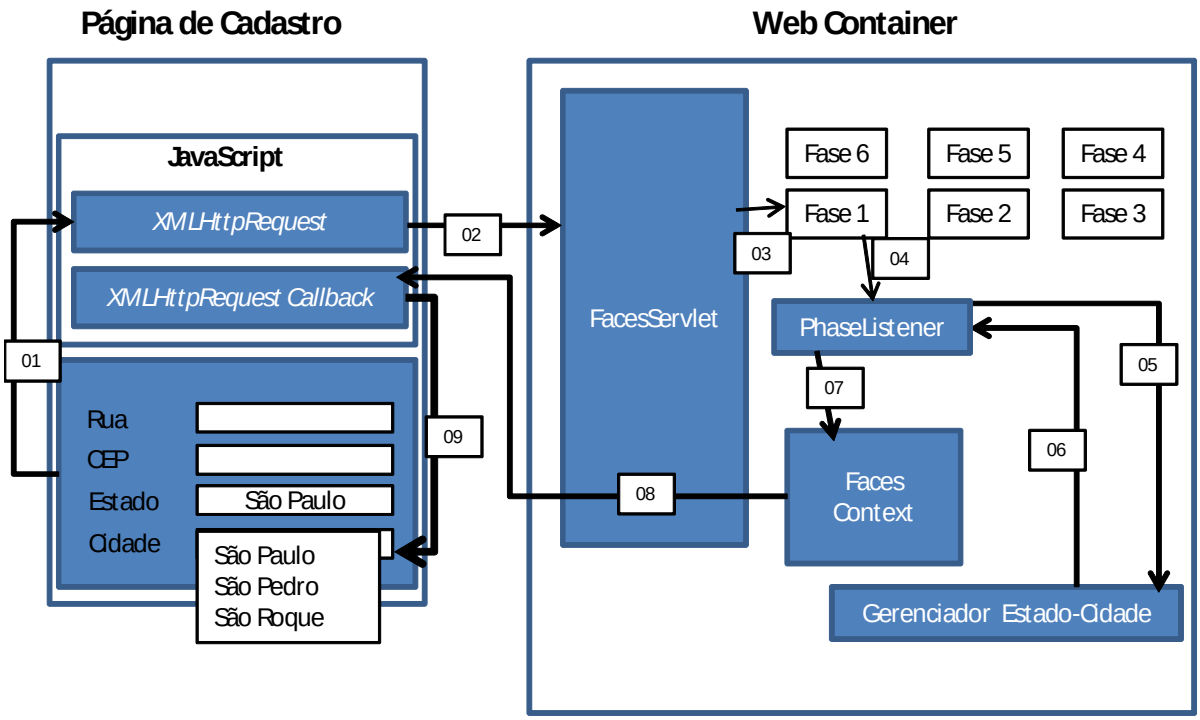

Figura 6: Representação da Forma 1 de integração AJAX e JSF

De acordo com a ilustração apontada, as etapas realizadas por uma requisição que utiliza esta forma de integração são:

1. Sempre que o usuário selecionar um valor para o Estado no exemplo mostrado acima, um novo objeto XMLHttpRequest é criado.

2. Após ser criado, o objeto XMLHttpRequest invoca a instância da classe FacesServlet repassando para esta o nome do método a ser executado e os parâmetros necessários para sua realização

3. A requisição, ao chegar na instância da classe FacesServlet, inicia a primeira fase do ciclo de vida das requisições JSF, chamada de “Restaurar Visão”.

4. Depois da fase “Restaurar Visão”, a requisição passa a executar a fase “PhaseListener”, adicionada ao ciclo de vida das requisições JSF.

5. (Inclui etapas 5 e 6 do processo) Durante a fase “PhaseListener” é invocada uma instância da classe GerenciadorEstadoCidade para que esta filtre as cidades pertencentes ao estado selecionado pelo usuário. O “PhaseListener” também tem a função de inserir todas as cidades filtradas num arquivo XML, que deve ser encaminhado para o browser do usuário, atualizando dessa forma os dados a serem mostrados.

7. Neste momento o sistema já contém um arquivo XML no qual existe uma lista com todas as cidades que devem ser exibidas ao usuário. Uma instância da classe FacesContext é invocada.

8. Na instância da classe FacesContext é executado o método responseComplete(). Com isso todas as fases seguintes do ciclo de vida das requisições JSF não são realizadas. Depois dessas atividades o método CallBack da classe XMLHttpRequest é invocado e o DOM da página é atualizado de acordo com o XML criado nas etapa anterior.

9. Nesta etapa o sistema exibe ao usuário todas as cidades do estado selecionado anteriormente.

Esta idéia de integração é muito similar à apresentada na seção “*4.3.1.1UTILIZANDO UM PHASELISTENER APÓS A QUINTA FASE DO CICLO DE VIDA*”, pois desviam o fluxo do ciclo de vida das requisições JSF incluindo uma classe PhaseListener durante este ciclo. Porém, enquanto esta idéia apresentada por Mark Basler executa apenas a primeira fase do ciclo de vida das requisições JSF, as soluções apresentadas anteriormente executam todas, com exceção da última fase, chamada de “Renderizar Resposta”.

Esta forma de integração possui o mesmo problema que a apresentada na seção “*4.3.1.1UTILIZANDO UM PHASELISTENER APÓS A QUINTA FASE DO CICLO DE*

VIDA”, ou seja, se houver muitos Listeners para serem executados na mesma fase o desenvolvedor não pode garantir a ordem de execução.

Pelo fato de serem invocados em todas as requisições, o aumento da utilização de PhaseListeners faz com que o desempenho do sistema decaia, constituindo um outro grande problema.

No último livro lançado por Chris Schalk, Ed Burns e James Holmes essa forma de integração apresentada acima também é apoiada (SCHALK, Chris; BURNS, Ed; HOLMES, James., 2006. p. 287-320.).

4.3.1.4. UTILIZANDO UM PHASELISTENER ANTES DA PRIMEIRA FASE DO CICLO DE VIDA

O artigo “Including AJAX Functionality in a Custom JavaServer Faces Component” (MURRAY, Gregory; BALL Jeniffer) apresenta uma nova forma de integração da tecnologia AJAX com o framework JSF.

O artigo sugere utilizar apenas o Servlet disponibilizado pelo JavaServer Faces chamado de FacesServlet. Porém, ao contrário das idéias exibidas anteriormente, nenhuma fase do ciclo de vida das requisições JSF é executada.

Todos os passos executados pela requisição que utiliza a tecnologia AJAX podem ser observados na figura que segue.

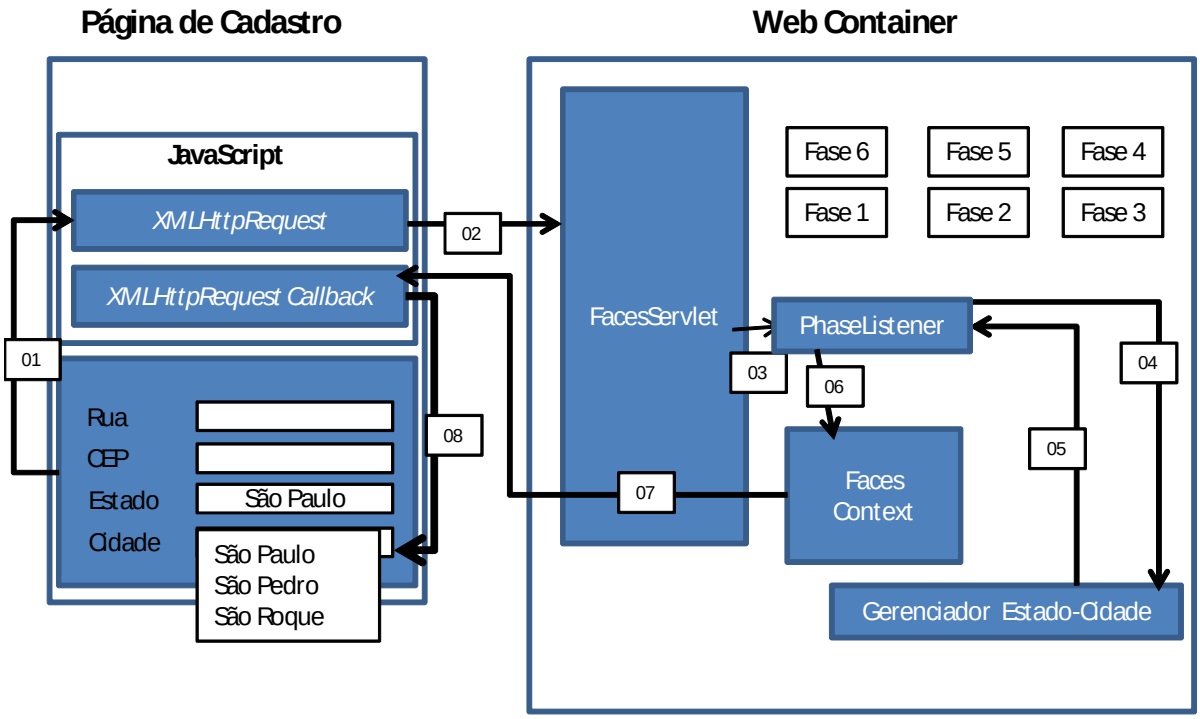

Figura 7: Representação da Forma 1 de integração AJAX e JSF de “Super-Charge Ájax Data Fetch”

Como ilustrado, as etapas de uma requisição que utiliza esta forma de integração executa são:

1. Cada vez que o usuário selecionar na página um valor para o Estado, uma instância da classe XMLHttpRequest é criada.

2. Neste momento a instância da classe FacesServlet é invocada pelo XMLHttpRequest, que passa para esse o método a ser executado juntamente com os parâmetros necessários.

3. Depois de chamado, o FacesServlet deveria invocar a primeira fase do ciclo de vida das requisições JSF, chamada de “Restaurar Visão”, porém nesta idéia é inserida uma classe PhaseListener que deve ser executada antes desta fase.

4. (Inclui etapas 4 e 5 do processo) Ao ser invocada a instância da classe PhaseListener, a classe GerenciadorEstadoCidade, responsável por filtrar todas as cidades que pertencem ao estado selecionado pelo usuário, delega a função. Depois

de obter as cidades, a instância da classe GerenciadorEstadoCidade repassa essas informações ao PhaseListener.

6. De posse de todas as cidades que pertencem ao estado selecionado pelo usuário, a instância de PhaseListener empacota todas essas informações num arquivo XML para repassá-las ao browser do usuário. Com essas etapas finalizadas, o PhaseListener invoca a instância FacesContext.

7. Neste momento a instância de FacesContext é chamada para que execute o método responseComplete() a fim de que nenhuma fase do ciclo de vida das requisições JSF seja executada. A função Callback do objeto XMLHttpRequest é invocada.

8. Neste passo a função do XMLHttpRequest Callback atualiza o DOM da página, colocando neste os registros vindos do XML. A página mostra ao usuário todas as cidades que pertencem ao estado selecionado.

Apesar de utilizar o FacesServlet, nesta forma de integração as requisições que usam a tecnologia AJAX não passam por nenhuma fase do ciclo de vida das requisições JSF, visto que uma classe PhaseListener é colocada para ser executada antes da primeira fase do ciclo, conhecida como “Restaurar Visão”.

Possui o mesmo defeito das idéias apresentadas nas seções “*4.3.1.1 UTILIZANDO UM PHASELISTENER APÓS A QUINTA FASE DO CICLO DE VIDA*” e “*4.3.1.3 UTILIZANDO UM PHASELISTENER APÓS A PRIMEIRA FASE DO CICLO DE VIDA*”, pois não garantem a ordem de execução dos PhaseListeners se houver muitos a serem executados na mesma fase.

Além disso, outro defeito dessa idéia, comum também às outras que utilizam PhaseListerner, é o fato de que o desempenho das aplicações decai, quando são utilizados muitas instâncias dessa classe, levando em consideração que todas são invocadas em todas as requisições JSF.

4.3.1.5. IMPLEMENTANDO UM MÓDULO DE CONTROLE APÓS A TERCEIRA FASE DO CICLO DE VIDA

Esta forma é defendida e utilizada pela biblioteca Backbase (BACKBASE,2003) que fornece componentes JSF com funcionalidades da tecnologia AJAX.

As idéias principais são: manter no servidor uma árvore de componentes DOM da página exibida ao usuário e reimplementar a última fase do ciclo de vida das requisições JSF para que a cada requisição o servidor possa enviar ao browser do cliente apenas os componentes que foram alterados.

Já no browser do cliente são colocadas inúmeras funções JavaScript para que a cada requisição a página atualize apenas os componentes recebidos pelo servidor.

A figura abaixo representa todas as etapas cumpridas por uma requisição que se utiliza desta idéia.

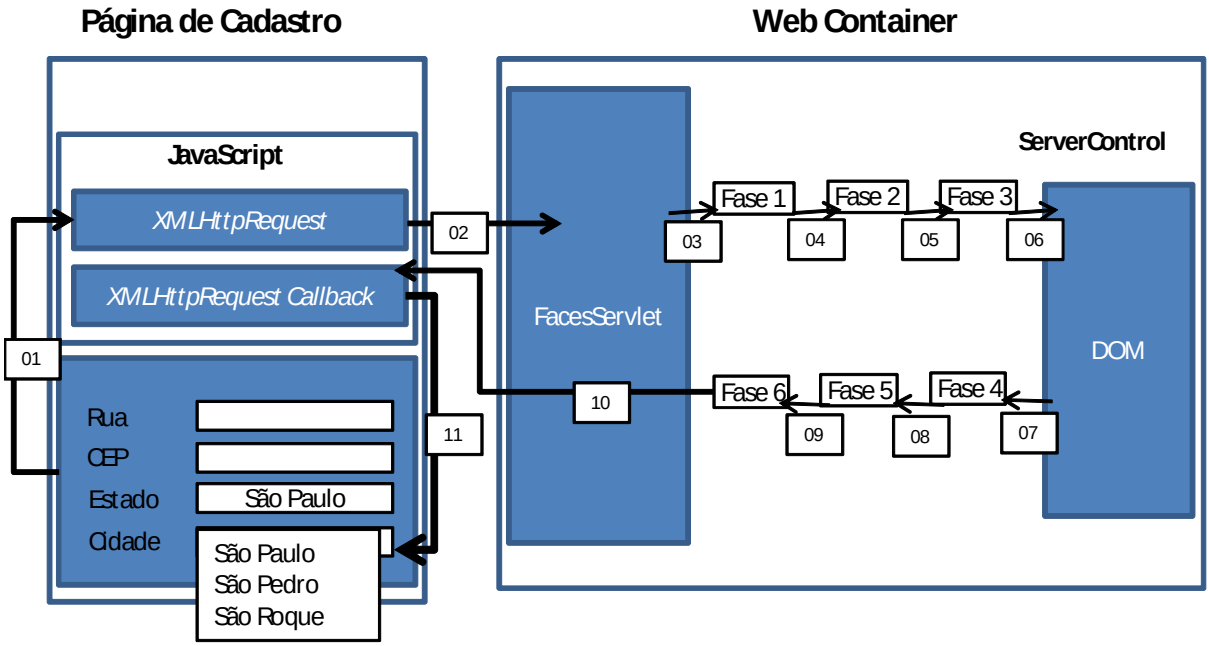

Figura 8: Representação da forma 1 de integração AJAX e JSF

Essa requisição obedece a seguinte seqüência:

1. Toda vez que o usuário selecionar um valor para o campo Estado na página é criado um objeto XMLHttpRequest.

2. O objeto XMLHttpRequest invoca uma instância da classe FacesServlet, repassando para esta o método que deve ser executado e todos os parâmetros necessários para sua execução.

3. Neste momento a instância da classe FacesServlet invoca a primeira fase do ciclo de vida das requisições JSF, chamada de “Restaurar Visão”.

4. Inicia-se a segunda fase, chamada de “Aplicar Valores de Requisição”.

5. Neste passo a requisição começa a executar a terceira fase do ciclo de vida das requisições JSF, denominada “Processar Validações”.

6. Neste momento o fluxo do ciclo de vida das requisições JSF deveria seguir para a quarta fase, contudo, o fluxo é desviado para um módulo de controle encontrado no servidor que armazena uma árvore DOM de componentes exibidos no browser do usuário. O servidor deve então neste passo comparar quais componentes foram modificados desde a última requisição e armazenar tais informações para serem repassadas ao browser do usuário.

7. Depois da verificação no passo anterior de quais componentes foram modificados o fluxo do ciclo de requisições JSF volta ao normal, executando então a quarta etapa, conhecida como “Atualizar Valores do Modelo”.

8. Na seqüência a requisição começa a executar a quinta fase do ciclo de vida: “Invocar Aplicação”.

9. A sexta e última fase do ciclo de vida das requisições JSF, a de “Renderizar Resposta”, é executada. Deve-se notar que esta é a única fase que foi alterada pela biblioteca Backbase a fim de que seus componentes pudessem disponibilizar os benefícios da tecnologia AJAX.

10. Com o ciclo de vida finalizado, a função Callback do objeto XMLHttpRequest é invocada e recebe um arquivo XML contendo informações de todos os componentes da página que tiveram alguma modificação.

11. A instância XMLHttpRequest atualiza a árvore de componentes DOM da página, inserindo no campo Cidade todas aquelas pertencentes ao estado selecionado pelo usuário.

Pode-se observar que nesta forma de integração AJAX e JSF, todas as etapas do ciclo de vida das requisições JSF foram executadas, sendo que a última fase, a de "Renderizar Resposta", foi alterada para o correto funcionamento dos componentes com a tecnologia AJAX.

Nota-se também a complexidade no desenvolvimento de uma biblioteca que utiliza essa forma de integração, considerando a necessidade de armazenar no servidor um mecanismo para guardar uma cópia da árvore DOM de componentes exibidos no browser e também sobrescrever a última fase do ciclo de vida das requisições JSF. Essas ações são executadas a fim de que sejam enviados para o cliente somente os componentes modificados.

4.3.1.6. UTILIZANDO PHASELISTENER EM TRÊS FASES DO CICLO DE VIDA

Esta é a forma de integração AJAX e JSF utilizada na biblioteca conhecida como Ajax4Jsf.

A idéia principal é colocar um PhaseListener atuando em três fases durante o ciclo de vida das requisições JSF:

- ✓ Depois da fase "Restaurar Visão" o PhaseListener é chamado e define se apenas uma região ou toda a página deve ser atualizada após a execução da requisição. Para isso, o Listener deve verificar se o componente que gerou a requisição está contido entre alguma tag <a4j:region>.no caso da biblioteca Ajax4Jsf.
- ✓ O PhaseListener criado por esta idéia de integração é invocado pela segunda vez após a fase "Processar Validações". Neste momento o PhaseListener verifica se o valor de um atributo chamado de bypassUpdate, no caso de Ajax4Jsf, for igual a true.

Em caso afirmativo, o ciclo de vida das requisições JSF passa a executar a quinta fase, conhecida como “Invocar Aplicação”; caso contrário, o fluxo do ciclo continua normalmente.

- ✓ A terceira e última vez em que o PhaseListener é chamado antes da última fase do ciclo de vida das requisições JSF é a chamada fase de “Renderizar Resposta”. Aqui, se a requisição que está sendo executada utilizar a tecnologia AJAX, o PhaseListener empacota os dados que devem ser enviados ao usuário para ter o seu browser atualizado. Nesse caso, a última fase do ciclo de vida das requisições JSF não é executada.

A figura abaixo ilustra as ações realizadas por uma requisição de uma aplicação que utiliza esta forma de integração.

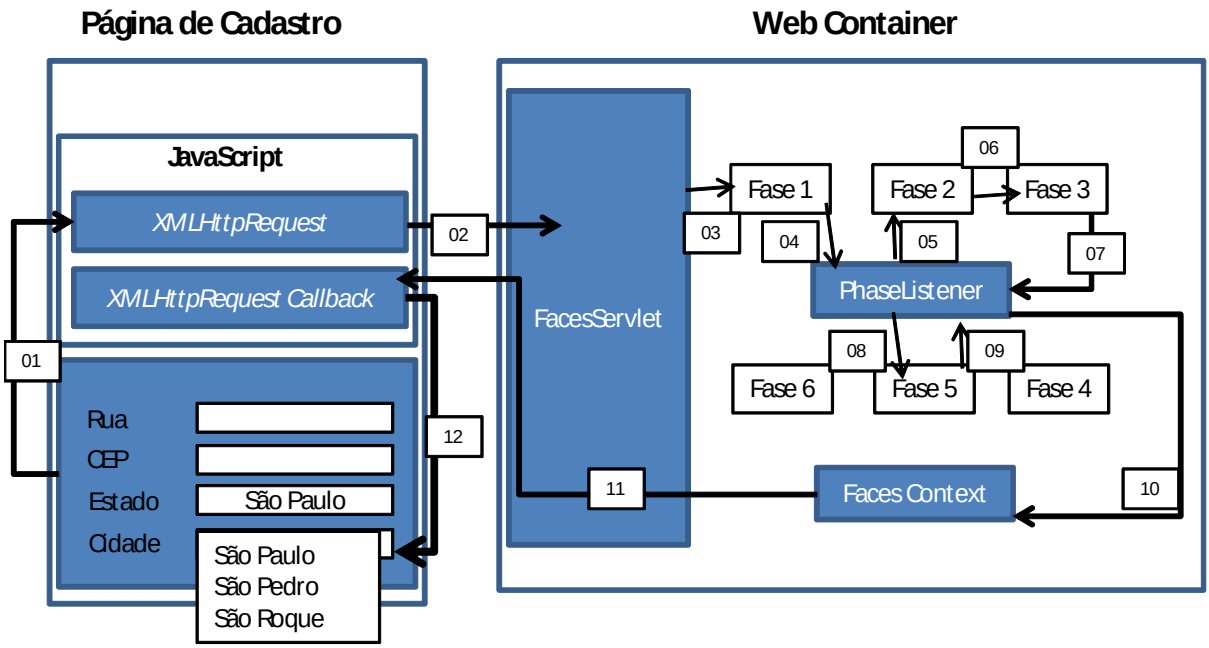

Figura 9: Representação da forma 1 de integração AJAX e JSF

Como podemos observar na figura acima, os passos realizados por uma requisição que utiliza esta forma de integração são:

1. Toda vez que o usuário selecionar um valor para o campo Estado, uma instância de XMLHttpRequest é criada.

2. Depois de criada, a instância de XMLHttpRequest invoca um objeto da classe FacesServlet que recebe o método que deve ser executado juntamente com os parâmetros necessários para sua realização.

3. A requisição começa a executar a primeira fase do ciclo de vida das requisições JSF, conhecida como “Restaurar Visão”.

4. Ao acabar a primeira fase do ciclo de vida das requisições JSF, a requisição deveria começar a realizar a segunda fase, porém nesta idéia o fluxo é desviado. Um PhaseListener é invocado para verificar se o componente gerador da requisição está entre alguma tag <a4j:region>. Caso esteja, o PhaseListener conclui que apenas alguns componentes da página devem ser atualizados.

5. Passada a primeira intervenção do PhaseListener, a requisição deve executar a segunda fase do ciclo de vida, chamada de “Aplicar Valores de Requisição”.

6. Ao finalizar a segunda etapa do ciclo de vida, a requisição passa para fase “Processar Validações”, a terceira do ciclo.

7. Ao término da terceira fase, a requisição deveria executar a quarta fase, “Atualizar Valores do Modelo”. Ressalta-se que com essa idéia a requisição deve invocar o PhaseListener novamente para que este verifique o atributo bypassUpdate do componente que gerou a requisição.

8. Neste momento o PhaseListener verifica se o valor do atributo bypassUpdate do componente que gerou a requisição é igual a true. Neste exemplo foi utilizada esta condição como verdadeira, portanto, a requisição não executa a quarta fase do ciclo e passa a executar a quinta fase diretamente.

9. No final da quinta fase do ciclo de vida das requisições JSF, o fluxo da requisição é modificado e o PhaseListener é invocado novamente. Dessa vez o

trabalho deste PhaseListener é de empacotar os dados que devem ser enviados ao usuário a fim de que sejam atualizados no browser.

10. Ao empacotar os dados, o PhaseListener invoca uma instância da classe FacesContext para que ela conclua o ciclo de vida das requisições JSF sem executar a última fase deste ciclo.

11. Ao ser invocado, o objeto de FacesContext executa o método responseComplete() finalizando assim o ciclo de vida das requisições JSF e invocando a função CallBack da instância XMLHttpRequest.

12. A função Callback do XMLHttpRequest atualiza o DOM da página de acordo com o XML retornado no passo anterior, mostrando assim as opções de cidades que pertencem ao estado selecionado pelo usuário.

Observa-se que esta forma de integração é similar à apresentada na seção "*4.3.1.1 UTILIZANDO UM PHASELISTENER APÓS A QUINTA FASE DO CICLO DE VIDA*", mas apresenta alguns detalhes adicionais.

Um detalhe observado pela primeira vez nas formas de integração AJAX e JSF pesquisados foi o atributo bypassUpdate, no qual o desenvolvedor pode escolher durante o desenvolvimento se deseja ou não executar a quarta fase, conhecida como "Atualizar Valores do Modelo".

Tal atributo, quando bem utilizado, pode trazer algumas vantagens, como a melhora no desempenho. Para isso o desenvolvedor deve possuir um conhecimento bom sobre o ciclo de vida das requisições JSF para que não ocorram problemas nos sistemas por eles desenvolvidos.

4.3.2. USANDO DOIS SERVLETS

A solução básica aqui é criar um novo Servlet para gerenciar todas as requisições que utilizam a tecnologia AJAX.

Enquanto isso, o Servlet disponibilizado pelo framework JavaServer Faces, o FacesServlet, continua sendo utilizado, porém apenas para administrar as requisições JSF que não usam AJAX.

Entre as vantagens estão a menor dependência do framework JSF, pois não atua durante o ciclo de vida das requisições JSF, possuindo assim uma clara separação no tratamento entre requisições assíncronas e síncronas; e a ausência de problemas com PhaseListeners e Renderizadores (como visto na seção "*4.3.1* USANDO APENAS UM SERVLET").

A dificuldade de implementação é basicamente o desenvolvimento de um Servlet.

Essa idéia apresenta dois defeitos principais: a baixa colaboração entre os componentes e o esforço adicional (desenvolvimento de um Servlet).

Essas limitações ocorrem pois a lógica é executada neste novo Servlet, ou seja, fora do contexto do framework JSF.

No projeto ICEfaces (ICESOFT), o primeiro defeito foi contornado utilizando-se um artifício conhecido como AJAX Bridge, que faz com que todo o tráfego das requisições assíncronas passem por este artifício, aumentando assim a colaboração entre os componentes.

A seguir serão apresentadas algumas idéias de como pode ser implementada essa forma de integração.

4.3.2.1. CRIANDO UM SERVLET PARA GERENCIAR REQUISIÇÕES AJAX

As ações executadas por uma requisição que utiliza esta idéia são apresentadas na figura a seguir.

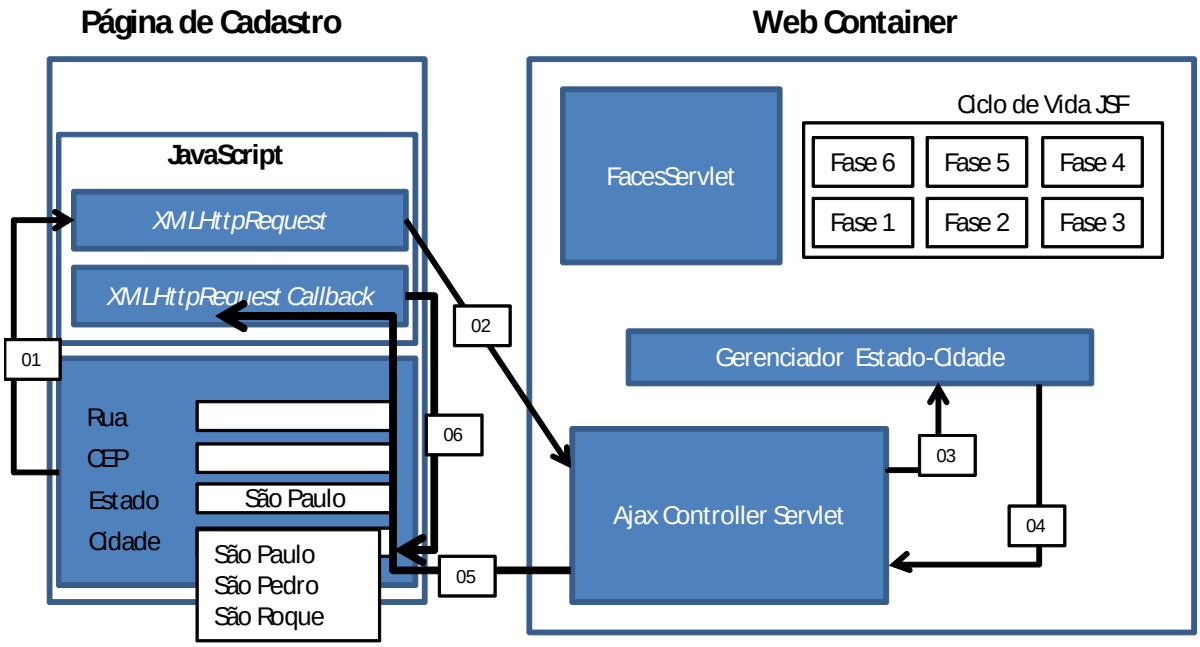

Figura 10: Representação da forma 2 de integração AJAX e JSF

Os passos executados pela requisição desde a ação do usuário até a página ser reexibida são:

1. Cada vez que o usuário selecionar um valor na página para o campo Estado, um novo objeto XMLHttpRequest é criado.

2. Neste momento, o objeto XMLHttpRequest, em vez de invocar uma instância de FacesServlet como é feito na integração na seção Erro! Fonte de referência não encontrada, deve chamar uma instância da classe AjaxControllerServlet, um Servlet que deve ser criado pelo desenvolvedor.

3. Ao ser requisitada, a instância de AjaxControllerServlet deve repassar a requisição à classe responsável pela execução do método passado pelo XMLHttpRequest. Nesse caso, a classe responsável por filtrar todas as cidades do estado selecionado pelo usuário é GerenciadorEstadoCidade.

4. Depois de filtrar todas as cidades que pertencem ao estado selecionado pelo usuário a classe GerenciadorEstadoCidade devolve todos esses registros à

instância de AjaxControllerServlet, para que estes sejam empacotados em um arquivo XML.

5. Assim, a instância de AjaxControllerServlet envia o arquivo XML para o browser do cliente e invoca a função Callback do objeto XMLHttpRequest.

6. A função do XMLHttpRequest Callback atualiza o DOM da página, colocando neste os registros vindos do XML. A página mostra então ao usuário todas as cidades que pertencem ao estado selecionado.

Esta é a forma de integração AJAX e JSF mais fácil de ser desenvolvida, bastante eficiente e mais comum de ser encontrada em sites que explicam como unir esta tecnologia e este framework.

4.3.2.2. CRIANDO UM SERVLET PARA GERENCIAR AS REQUISIÇÕES AJAX E IMPLEMENTAR UM NOVO CICLO DE VIDA PARA REQUISIÇÕES AJAX

A terceira forma de integração apresentada pelo artigo “Super-Charge JSF AJAX Data Fetch” (JACOBI, Jonas; FALLOWS, John) chama-se “The Lifecycle Approach” e tem como idéia principal criar um novo ciclo de vida para as requisições que utilizam a tecnologia AJAX, para que somente algumas fases sejam executadas.

A figura abaixo apresenta todos os passos realizados por uma requisição que utiliza esta forma de integração.

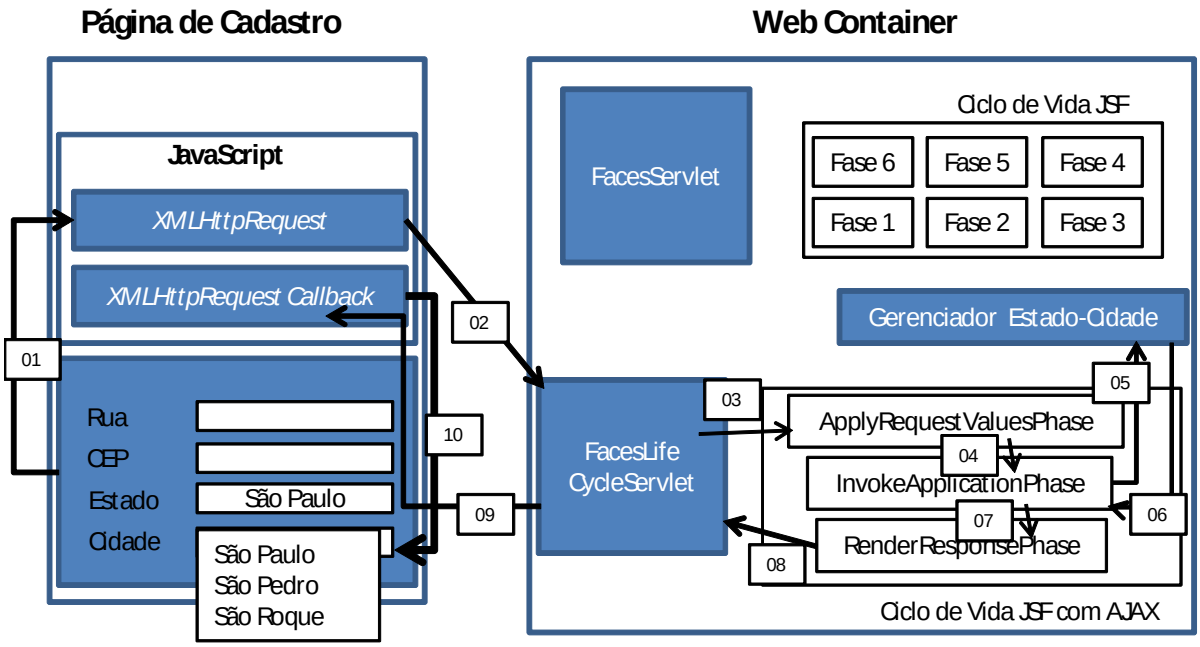

Figura 11: Representação da forma 3 de integração AJAX e JSF

Como podemos observar na figura anterior, os passos executados pela requisição são:

1. Toda vez que o usuário escolher um valor para o campo Estado do exemplo, um objeto XMLHttpRequest é criado.

2. Ao ser instanciado o objeto XMLHttpRequest recebe o nome do método que deve ser executado juntamente com os seus parâmetros. Depois dessa etapa, diferentemente das outras formas de integração AJAX e JSF apresentadas neste artigo, a instância XMLHttpRequest não invoca o FacesServlet e sim FacesLifeCycleServlet.

3. Depois de o FacesLifeCycleServlet ser invocado pelo objeto XMLHttpRequest, este repassa a requisição juntamente com os seus parâmetros para a primeira fase do ciclo de vida das requisições JSF que utilizam a tecnologia AJAX, chamada de “ApplyRequestValuesPhase”. Entre as principais funções desta fase estão: encontrar a referência da classe que deve executar o método e verificar os parâmetros necessários para sua execução.

4. Ao terminar a fase “ApplyRequestValuesPhase”, a requisição passa então para a etapa chamada de “InvokeApplicationPhase” que é a segunda fase do ciclo de vida das requisições JSF que utiliza a tecnologia AJAX. É nesta fase que o método, juntamente com seus argumentos que foram passados desde a instância XMLHttpRequest, é executado.

5. (Inclui etapas 5 e 6 do processo) Ainda na segunda fase do ciclo de vida, a classe GerenciadorEstadoCidade, capaz de realizar o método passado desde o XMLHttpRequest, é invocada para que possa filtrar todas as cidades que pertencem ao estado selecionado pelo usuário.

7. Ao fim da segunda fase, a requisição passa a executar a terceira fase, denominada “RenderResponsePhase”. Tem como principais funções coletar o resultado do método anterior e terminar o ciclo.

8. Ao final do ciclo o controle do fluxo da aplicação retorna novamente ao FacesLifeCycleServlet.

9. Neste momento, o Servlet FacesLifeCycleServlet invoca a função Callback do objeto XMLHttpRequest para que os dados da página possam ser renderizados.

10. Para finalizar o processo, a função Callback do XMLHttpRequest atualiza o DOM da página, mostrando ao usuário todas as cidades que pertencem ao estado selecionado.

Neste exemplo, as requisições JSF que utilizam a tecnologia AJAX percorrem um novo ciclo de vida, constituído de três fases: ApplyRequestValuesPhase, InvokeApplicationPhase e RenderResponsePhase.

Apesar da requisição se tornar mais eficiente por passar apenas pela metade do número de fases do ciclo de vida comparando-se com o do framework JSF, o desenvolvimento é mais complexo, considerando a necessidade de que o desenvolvedor implemente e faça o controle sobre essas três fases.

Outra desvantagem refere-se ao fato que os componentes só podem buscar informações no servidor, não podendo alterar dados dos objetos ou modificar a hierarquia dos componentes, porque as fases “Restaurar Visão”, “Processar Validações” e “Atualizar Valores no Modelo” não são executadas por requisições que utilizam a tecnologia AJAX.

4.3.2.3. CRIANDO UM SERVLET PARA GERENCIAR REQUISIÇÕES AJAX E REIMPLEMENTAR OS RENDERIZADORES DOS COMPONENTES

A segunda técnica para integração AJAX e JSF tem como base a criação de um Servlet que gerencie todas as requisições que utilizam a tecnologia AJAX.

Todos os passos realizados por uma requisição que se utiliza desta forma de integração AJAX e JSF são exibidos na figura abaixo:

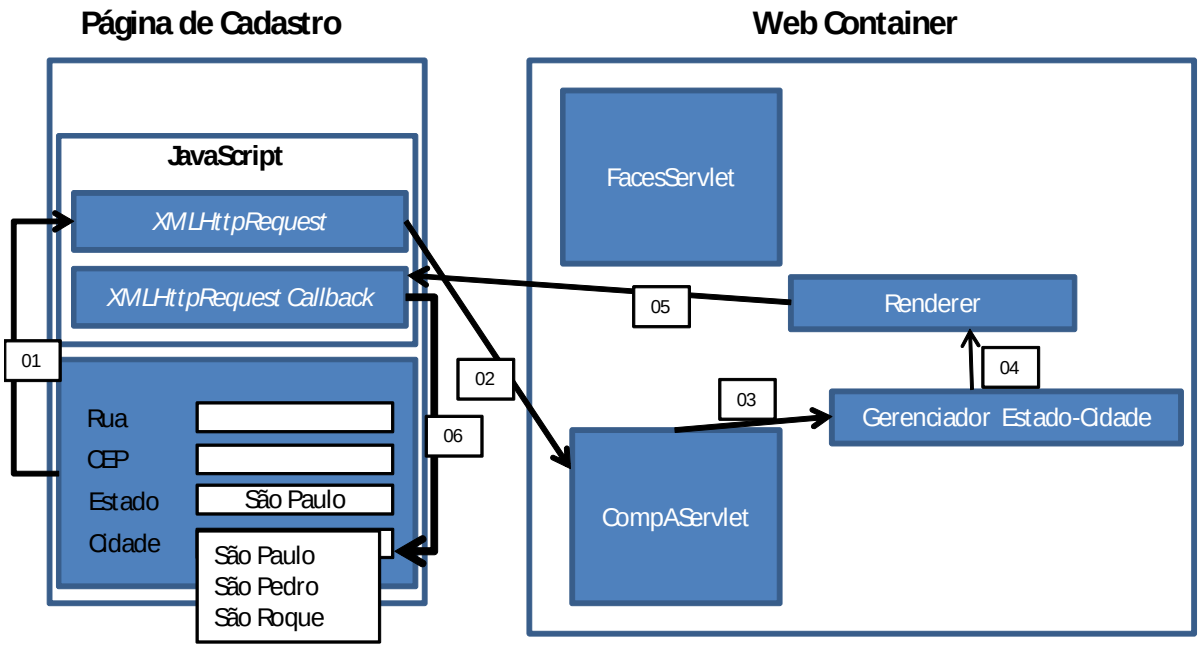

Figura 12: Representação da forma 2 de integração AJAX e JSF

Como ilustrado, o roteiro dessa requisição executa as seguintes ações:

1. Toda vez que o usuário escolher um valor para o campo Estado, um objeto XMLHttpRequest é criado.

2. Depois de sua criação, o objeto XMLHttpRequest repassa o método a ser executado e os parâmetros necessários para realização ao Servlet, chamado de CompAServlet, criado com o objetivo de gerenciar as requisições que utilizam a tecnologia AJAX.

3. Ao ser notificado, o CompAServlet repassa a responsabilidade de execução do método para uma instância da classe GerenciadorEstadoCidade, que possui a capacidade de filtrar todas as cidades do campo Estado selecionado pelo usuário.

4. Posteriormente à seleção de todas as cidades que pertencem ao estado selecionado, a requisição invoca uma instância da classe Renderer, que deve ser reimplementada para que esta forma de integração funcione corretamente. A principal função desta nova classe Renderer é a de empacotar todas as cidades filtradas na etapa anterior e colocá-las num arquivo XML, por exemplo, para que este possa ser enviado ao browser do cliente de maneira assíncrona.

5. Com o arquivo XML do passo anterior pronto, a função Callback da instância de XMLHttpRequest é invocada e os dados alterados podem ser renderizados na página do cliente.

6. Neste momento, a função do XMLHttpRequest Callback atualiza o DOM da página, colocando neste os registros vindos do XML. A página mostra então ao usuário todas as cidades que pertencem ao estado selecionado.

Em todos os componentes em que o desenvolvedor deseja obter os benefícios da tecnologia AJAX, deve ser desenvolvido uma nova classe Renderer para cada um deles, para que nesta sejam empacotados os dados retornados ao cliente. Por essa razão, esta idéia é pouco aceita e utilizada pelos desenvolvedores.

4.4. CONCLUSÃO

Podemos verificar neste capítulo que existem várias maneiras para realizar a integração entre AJAX e JSF. Verificamos ainda, que muitas dessas maneiras de integração possuem falhas, constatando que não há a melhor maneira de efetivar essa integração.

A idéia de integração que encontra maior aceitabilidade é a apresentada por Edward Burns (representante da Sun Microsystems), Jacob Hookom (funcionário da McKesson Medica-Surgical) e Adam Winer (trabalhador da Oracle) e que foi discutida na seção "*4.2.2 ALTERANDO A ESPECIFICAÇÃO JSF*".

Essa idéia é baseada na modificação da especificação JSF para que todos os componentes possam usufruir de requisições assíncronas sem a necessidade de nenhum trabalho adicional do desenvolvedor.

A segunda melhor forma de integração AJAX e JSF é a apresentada na seção *"4.3.2.1 CRIANDO UM SERVLET PARA GERENCIAR REQUISIÇÕES AJAX*". Apesar de possuir algumas falhas, ainda é muito utilizada devido à facilidade de uso e eficiência.

5. CONCLUSÃO FINAL

O *framework* JavaServer Faces (JSF), objeto de estudo neste trabalho, atua principalmente nas camadas de Visão e Controle do modelo MVC (Modelo – Visão – Controle).

Esse *framework* possui um modelo de programação dirigida a eventos como uma das suas principais características, deixando assim o desenvolvimento Web mais fácil e rápido. Mas comparando este framework com seus concorrentes, por exemplo, o Struts (considerado um padrão de mercado), todos apresentam limitações relacionadas à interatividade dos aplicativos desenvolvidos com base em tais frameworks, limitação esta advinda da utilização de requisições síncronas.

Essa limitação faz com que modificando apenas um conteúdo da página, as páginas dos aplicativos tenham que ser totalmente carregadas ao sofrerem algum evento, deixando, assim, a aplicação mais lenta e desagradando o usuário.

O foco deste estudo está dirigido aos benefícios de cada integração do framework JSF com a tecnologia AJAX, que promete mais interatividade nas aplicações Web.

Avaliando todas as formas de integração entre o framework JSF e a tecnologia AJAX, a melhor integração avaliada foi a apresentada na seção "*4.2.2* ALTERANDO A ESPECIFICAÇÃO JSF".

A maior vantagem desta integração citada acima foi o fato de que os componentes JSF poderão realizar requisições assíncronas, fazendo uso da tecnologia AJAX, sem nenhum esforço adicional dos desenvolvedores.

Concluindo, considera-se que os objetivos inicialmente traçados para este trabalho, de compreender, analisar e comparar as formas de integração entre JavaServer Faces e AJAX, foram alcançados. Entretanto, apesar das evidências apresentadas ao longo do trabalho, constata-se também que não existe um consenso da melhor forma de integração do framework e da tecnologia em evidência. Mas como o desenvolvimento de tecnologias atualmente cresce muito rápido, esperamos nos próximos anos, o surgimento de novas maneiras de integração entre JSF e AJAX.

6. REFERÊNCIAS BIBLIOGRÁFICAS

AJAXFACES, **AjaxFaces:General and Complete Integration Solution for JSF and AJAX**.
Disponível em <http://www.ajaxfaces.com>. Acesso em: 03 set. 2011.

BACKBASE,
Disponível em: <http://www.backbase.com>. Acesso em: 01 nov 2011.

COULOURIS, George; DOLLIMORE, Jean; KINDBERG, Tim. **Distributed Systems**. 2. ed. Addison Wesley. 1994. p. 10-15.

CRANE, Dave; PASCARELLO, Eric. **Ájax in Action.Greenwich**. 2006. 650 p.

DEITEL, H.; DEITEL, P. **Java Como Programar**. 4. ed. Bookman. 2003. 1386 p.

DURANT Brian; BENZ John. **XML Programming Bible**. Wiley Publishing Inc. 2003. p. 3-28.

DWR. **DWR Easy AJAX for JAVA**.
Disponível em: <http://directwebremoting.org/dwr/index.html >. Acesso em: 21 set. 2011.

EXADEL. **RichFaces**.
Disponível em: < http://www.exadel.com/web/portal/jrf >. Acesso em: 27 out. 2011.

FOWLER, Martin. **Patterns of Enterprise Application Architecture**. 2. ed. Addison Wesley. 2003. p.330-332.

GARRETT, James J., Adaptive Path >> **Ajax: A New Approach to Web Applications, 2005**.
Disponível em:
< http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications >. Acesso em: 10 jun. 2011.

GEARY, D.; CAY, Horstmann. **Core JavaServer Faces**. Alta Books. 2005. p.1-234.

GOODMAN, Danny. **JavaScript Bible Gold**. Ed Gold. Hungry Minds. 2001. p. 40-53.

GOOGLE, **Google Maps**.
Disponível em: <http://maps.google.com.br/>. Acesso em: 12 jun. 2011.

GOOGLE. **Google Suggest**.
Disponível em:
<http://www.google.com.br/ >. Acesso em: 6 aug. 2011.

GONÇALVES, Edson. **Dominando JavaServer Faces e Facelets utilizando Spring 2.5, Hibernate e JPA**. Rio de Janeiro: Editora Ciência Moderna Ltda., 2008.

HOOKOM, Jacob. **Jacob Hookom Blog**.
Disponível em:
<http://weblogs.java.net/blog/jhook/archive/2005/09/jsf_avatar_vs_m_1.html>. Acesso em: 02 set. 2011.

ICESOFT. **ICEfaces the rich web application**.
Disponível em:
<http://www.icesoft.com>. Acesso em: 02 jul. 2011.

J2EE, Wikipédia.

Disponível em: <http://pt.wikipedia.org/wiki/J2EE>. Acesso em: 02 dez. 2006.

JACOBI, Jonas.; FALLOWS, John. **PRO JSF and AJAX.** Apress.1. ed.Apress. 2006. 435p.

JACOBI, Jonas; FALLOWS, John. **Super-Charge JSF AJAX Data Fetch**.
Disponível em: <http://java.sys-con.com/read/192418_1.htm>. Acesso em: 13 set. 2006.

MANN, K. **JavaServer Faces in Action.** 2.ed. Manning. 2005. 1038p.

MURRAY, Gregory; BALL Jeniffer. **Including Ajax Functionality in a Custom JavaServer Faces Component**.
Disponível em:
< http://www.oracle.com/technetwork/java/javaee/tutorial-jsp-140089.html >. Acesso em: 09 out. 2011.

ORACLE,
Disponível em <http://www.oracle.com/us/sun/index.htm>. Acesso em: 11 jul. 2011.

ORACLE. **ADF Faces Oracle ADF's JSF Components**.
Disponível em:
< http://www.oracle.com/technetwork/developer-tools/adf/overview/index-092391.html
>. Acesso em: 26 set. 2011.

ORACLE, **Java Enterprise Edition 6**.
Disponível em:
<http://www.oracle.com/technetwork/java/javaee/tech/index.html>. Acesso em: 1º jun. 2011.

ORACLE, **JavaServer Faces Technology**.
Disponível em:

<http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html > Acesso em: 1º jun. 2011.

ORACLE, **Asynchronous JavaScript and XML**.
Disponível em:
<http://www.oracle.com/technetwork/articles/javaee/ajax-135201.html>. Acesso em: 15 jun. 2011.

ORT, Ed; BASLER, Mark.
Disponível em:
<http://java.sun.com/developer/technicalArticles/J2EE/AJAX/DesignStrategies>. Acesso em: 12 out. 2011.

SCHALK, Chris; BURNS, Ed; HOLMES, James. **JavaServer Faces: The complete reference**. 1.ed. McGraw-Hill Osborne. 2006. p. 287-320.
W3C, **World Wide Web Consortium.**
Disponível em: <http://www.w3.org>. Acesso em: 15 jun. 2006.

W3C. **Document Object Model (DOM)**.
Disponível em: <http://www.w3.org/DOM>. Acesso em: 1º dez. 2006.